A handheld optical sensing device for assessing nitrate concentration in maize leaves using reflectance spectroscopy

by

Albert Ansah Benson

B.Sc., Kwame Nkrumah University of Science and Technology, Ghana, 2021

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Biological and Agricultural Engineering
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2026

Approved by:

Major Professor
Naiqian Zhang

Copyright

© Albert Ansah Benson 2026.

Abstract

Nitrogen fertilizer is one of the largest variable input costs in maize production, yet much of the nitrogen applied is not taken up by the crop and is lost through leaching, denitrification, volatilization, and runoff, at a cost to both the farmer and water quality. Because crop nitrogen need varies across a field and from season to season, a single rate decided at planting is often poorly matched to it. A rapid in-season measure of tissue nitrate-N would allow fertilizer to be applied where and when the crop needs it, while a sidedress application can still be made.

Healthy green leaves have a distinctive optical response: chlorophyll absorbs much of the red light that reaches the leaf, whereas internal leaf structure responds strongly in the near-infrared part of the electromagnetic spectrum. Multispectral sensors use this contrast to detect changes in crop condition. However, common vegetation indices often reduce the response to two or three wavelengths, and their interpretation can be affected by crop development, ambient light, and leaf-surface reflection. This study therefore developed a handheld active optical sensor that expanded the measurement to six wavelengths and evaluated its ability to estimate tissue nitrate-N as an indicator of nitrogen status in maize.

The sensor sequentially illuminated maize leaves at 660, 670, 700, 750, 780, and 850 nm. A silicon photomultiplier detected the reflected light, which was amplified and digitized with a 16-bit analog-to-digital converter. Measurements of a 99% reflectance Spectralon reference confirmed a measurable response at all six wavelengths. The instrument was then evaluated in a randomized complete block field experiment with six nitrogen treatments, three blocks, and two planting dates. At V10, R1, and R3, spectral readings were collected from individual leaves under outdoor ambient light and inside a dark enclosure. Tissue nitrate-N determined in the laboratory served as plant-level ground truth.

Under controlled illumination, the sensor recovered the expected lower red and higher near-infrared responses of healthy green leaves. Nevertheless, spectral profiles and NDVI overlapped substantially among nitrogen treatments, so the optical measurements did not consistently separate treatment levels. A senesced-leaf test produced a stronger red response and much lower NDVI than healthy green leaves. Thus, the sensor detected a large change in leaf optical condition, although this did not demonstrate sensitivity to the smaller physiological differences among healthy plants.

Tissue nitrate-N ranged from 10 to 13,937 ppm and was strongly right-skewed; therefore, it was modeled as log(1 + nitrate-N) variable. The data were divided by plot into 174 training plants and 59 test plants. Because growth stage affected both nitrate-N and spectral response, growth stage was included as contextual information. A feed-forward neural network using the six wavelengths and growth stage achieved $R^2 = 0.752$ and $r = 0.902$ on the transformed scale in the dark enclosure. Outdoors, performance decreased to $R^2 = 0.526$ and $r = 0.810$. Withholding growth stage reduced R^2 to 0.471 in the dark enclosure and 0.096 outdoors, confirming that developmental context was especially important. Individual wavelengths carried little predictive information, whereas performance improved as wavelengths were combined. When the six wavelengths were replaced by nine NDVI combinations formed from the same six wavelengths, the dark-enclosure model reached $R^2 = 0.542$ and $r = 0.771$, and the outdoor model reached $R^2 = 0.130$ and $r = 0.470$, which indicates that reducing the bands to NDVI discarded information useful for estimating nitrate-N.

Finally, a controlled angle experiment with six leaves showed that leaf orientation significantly affected the response at all six wavelengths and suggested a small, wavelength-dependent specular component, largest in the red region, although this attribution could not be confirmed and its contribution to nitrate-N prediction error could not be quantified. Overall, the prototype provides a functional platform for controlled, relative measurements of maize leaf reflectance when the six-wavelength response is interpreted with growth stage. The results do not establish an absolute nitrate-N calibration or support field-ready fertilizer recommendations. Further work would improve ambient-light exclusion, use cross-polarization to suppress surface reflection, validate performance across seasons and locations, and col-

lect chlorophyll and total leaf nitrogen measurements, which should help improve the sensor performance.

Table of Contents

List of Figures ix

List of Tables xiv

List of Equations xvii

Acknowledgements xix

1 Introduction 1

1.1 Active Optical Sensing for Nitrogen Assessment 3

1.2 Specular and Diffuse Reflection 4

1.3 Importance of the Sensor for Nitrogen Fertilizer Management 6

1.4 Research Gap and Study Rationale 8

1.5 Research Objectives 9

2 Sensor Structure, Mode of Operation, Circuit Design and Optics 10

2.1 Sensor Mode of Operation 10

2.2 Power Supply Circuit 14

2.3 Light Detection and Amplification 19

2.4 Boost Regulator Circuit 26

2.5 Light-Emitting Diode (LED) Array Module 30

2.5.1 Dual-Ring Geometry for LEDs Array Module 30

2.5.2 Spectral Bandwidth of Each LED Wavelength 32

2.5.3 Sequential Activation Control 34

2.6 LED Strings Constant-Current Driver Circuit Design 35

2.6.1 Solid-State Relay Switching Architecture 35

2.6.2 Constant-Current Driver 37

2.6.3 Constant-Current Driver Circuit Implementation 37

2.6.4 Reference Voltage Derivation 38
2.6.5 Regulated LED Current 38
2.7 Signal Acquisition, ADC Configuration, and Timing 40
2.7.1 Oversampling Ratio Selection and Resolution-Throughput Trade-off 40
2.7.2 Total Signal Acquisitions 42
2.7.3 Timing Summary 43
2.8 Micro-Controller Units 46
2.9 Optical System Design 49
2.10 Device Overall Internal and External Structure 54
3 Experimental Design - Materials and Method 56
3.1 Field Experimental Setup 56
3.2 Method Used for Data Collection 59
3.3 Methods Used for Data Analysis 62
3.4 Specular-Reflection Angle Experiment 70
4 Results and Discussion 77
4.1 Optical Reflectance Characterization Using a 99% Spectralon Reflectance Target 78
4.2 Dark-Enclosure and Outdoor Results by Growth Stage 82
4.3 Exploratory Senesced-Leaf Case 86
4.4 Within-Plot NDVI Variation Across Growth Stage: Rationale for Growth Stage as a Model Feature 89
4.5 Log-Transformed Nitrate-N Regression 93
4.6 Neural Network Using NDVI Combinations as Inputs 97
4.7 Neural Network Using Six Raw Bands and Three NDVI Combinations as Inputs 100
4.8 Regression Model Ablation Experiment 103
4.9 Effect of Leaf Angle and Evidence of Specular Reflection 108

5 Conclusion and Recommendations . . . 112
5.1 Conclusion . . . 112
5.2 Recommendations . . . 116
References . . . 118
A Derivation of the General Expression for Detected Optical Power . . . 122
Find Total Gain After Transimpedance Amplification (TIA) Stage . . . 122
Recover the TIA Output Voltage . . . 123
Recovering the SiPM Output Current . . . 123
Estimate Photon Energy . . . 124
Estimate Effective Responsivity for Each Wavelength . . . 124
Estimate Detected Optical Power . . . 125
General Expression . . . 125
B Complete Plant-Level Spectral Dataset . . . 127
C Exhaustive Wavelength-Combination Diagnostics . . . 144
D Firmware Source Code . . . 157
LED Control Unit Firmware (STM32H725VGT6) . . . 157
Display Unit Firmware (STM32F746ZET6, FreeRTOS and TouchGFX) . . . 209
E Data Processing and Analysis Source Code . . . 253
Data Processing: Raw Readings to Detected Optical Power and Relative Reflectance . . . 254
Statistical Analysis and Nitrate-N Regression Models . . . 296
Specular-Reflection Angle Experiment . . . 311

List of Figures

1 Sensor Operational Chart. 13

2 Main Power Supply Schematic (12V is stepped down to 5 V and 3.3 V). . . 17

3 Main Power Supply Board. 18

4 Design schematic of the light detection and amplification stage. The figure shows the first stage transimpedance amplification stage and the second stage voltage amplification. 20

5 Photo Detection Efficiency of the silicon photomultiplier (SIPM) (Retrieved from the onsemi SiPM datasheet). 22

6 Step-Up Converter Schematic. 26

7 Step-Up Converter Regulator. 29

8 LEDs array concentric-ring layout. Each wavelength pair comprises one LED on the inner ring (Ø 50 mm) and its counterpart diametrically opposite on the outer ring (Ø 65 mm), ensuring beam convergence at the SiPM detector center. 31

9 Idealized relative spectral emission of the six array LEDs, reconstructed as Gaussian profiles from the manufacturer-specified peak wavelength. Curves are illustrative of spectral separation and bandwidth, not measured values. 33

10 Schematic of the LED strings constant-current driver circuit, showing shift-register-controlled relays for individually switching LED strings and a closed-loop regulator maintaining a constant 700 mA supply current. 36

11 Oversampling ratio trade-off analysis (Retrieved from the STM32 ADC datasheet). 43

12 Acquisition timing diagram. 45

13 Cross-sectional schematic of the dual-lens optical system. Outer lens: plano-convex, diameter(Ø) 75 mm, focal length (f) = 150 mm. Inner lens: converging, diameter(Ø) 10 mm, focal length (f) = 20 mm. (Sectioned through the front plane). 50

14 The Left image is the Outer plano-convex lens (Ø 75 mm) in its 3D-printed spoke mount. Right, optical window for protecting inner lens and outer lens from dust and scratch. 50

15 Inner converging lens barrel (Ø 10 mm, f = 20 mm) mounted on the SIPM detector PCB, positioning the lens 20 mm in front of the SiPM. 52

16 Overall Sensor with Connections and Circuit Board Modules. 55

17 Sensor with LCD Display ON. 55

18 Field map with treatments listed. Plots are colored by planting date. Black pin at the top left corner corresponds to location on map. 61

19 White Spectralon 99 % Reflectance Target (Retrieved from the Labsphere Spectralon target datasheet). 62

20 Spectralon 99% diffuse Reflectance Spectral Profile (Retrieved from the Labsphere calibration datasheet supplied with the Spectralon target). 63

21 Distribution of tissue nitrate-N on the raw ppm scale, with skewness annotated. 68

22 The existing six-band sensor mounted above the bench setup used for the specular-reflection angle experiment. 75

23 Detail of the servo-driven tilt mechanism used to control the angle for the leaf-mounting platform. 76

24 Sensor optical power gain by LED wavelength for the 99% Spectralon reference. 79

25 Calculated effective SiPM responsivity across the six measurement wavelengths (660-850 nm). 80

26 Corrected sensor voltage and recovered SiPM photocurrent by LED wavelength. 81
27 Relative Reflectance for each treatment under dark-enclosure conditions, all growth stages and planting dates. 83
28 Relative Reflectance for each treatment under outdoor conditions, all growth stages and planting dates. 84
29 Distribution of laboratory tissue nitrate-N, all available rounds pooled (n = 233 plants). 85
30 Senesced-leaf NDVI under dark-enclosure conditions (Plot 306, 17 August 2026). 87
31 Senesced-leaf spectral profiles and NDVI under outdoor conditions (Plot 306, 17 August 2026). 87
32 Dark-enclosure NDVI by growth stage (V10, R1, R3), all sampled plants, April planting. 90
33 Dark-enclosure NDVI by growth stage (V10, R1, R3), all sampled plants, May planting. 91
34 Dark-enclosure NDVI by growth stage (V10, R1, R3) for three randomly selected plots from the April planting. 92
35 Dark-enclosure NDVI by growth stage (V10, R1, R3) for three randomly selected plots from the May planting. 92
36 Distribution of tissue nitrate-N on the raw ppm scale(Left) and after a log1p transform (Right), with skewness annotated for each. 93
37 Architecture of the selected feed-forward neural network: eight predictors (six relative reflectance features, two growth-stage features), three hidden layers (16, 12, and 8 units; ReLU activation, dropout 0.3), and a single linear output unit predicting log1p(Nitrate_ppm). 95

38 Held-out observed versus predicted log1p(Nitrate_ppm) from the selected neural network model. (left panel) under dark-enclosure conditions and (right panel) outdoor/ambient-light conditions. 96

39 Held-out test R^2 of the neural network comparing the six raw relative-reflectance bands model performance to the nine NDVI combinations inputs model, each with growth stage variable (dark-enclosure and outdoor measurements). 98

40 Test-set R^2 of the 84 neural-network models that combined the six relative-reflectance bands, three NDVI combinations, and growth stage, dark-enclosure measurements, ranked by R^2. The dashed line marks the six bands with growth stage, and the dotted line marks the mean of the 84 models. 101

41 Test-set R^2 of the 84 neural-network models that combined the six relative-reflectance bands, three NDVI combinations, and growth stage, outdoor measurements, ranked by R^2. The dashed line marks the six bands with growth stage, and the dotted line marks the mean of the 84 models. 102

42 Held-out R^2 of every one of the 63 neural-network models trained on all possible combinations of the six wavelengths (dark-enclosure readings, growth stage withheld). Top: one bar per model, grouped by the number of wavelengths combined; dashed black lines show each group's mean and the dashed red line marks the best model. Bottom: filled dots show which wavelengths each model used. Values are those listed in Tables 22–27 (Appendix C). . . 105

43 Measured relative reflectance versus leaf tilt angle at each wavelength (in a dark enclosure). Thin lines show individual leaves; thick lines show the mean ± standard error across leaves. The p-value in each panel is the FDR-corrected p-value for the angle effect (Table 20). 109

44 Cross-polarization concept for the next prototype: a linear polarizer on the LED, a crossed analyzer followed by a depolarizer in front of the SiPM, and the specular and diffuse reflection. 117

List of Tables

1 Parameters of LEDs used in the project (Retrieved from the LED datasheets). 32

2 LED String Activation Sequence. 34

3 Oversampling ratio trade-off for a 50 ms acquisition window (Retrieved from the STM32 ADC datasheet). 42

4 ADC acquisition timing parameters for direct DC detection with background subtraction. 44

5 Summary of field experimental factors. 57

6 Nitrogen treatment structure. 58

7 Target growth stages and nominal sampling windows by planting date. . . 59

8 Spectral and ground-truth measurement plan. 60

9 Rows configuration for each 8-row plot. 60

10 Optical Power calculated from the retrieved corrected digital voltage, after illuminating the 99% diffuse reflectance target. 78

11 Realized sampling dates by growth stage and planting date, with the approximate calendar gap between the two planting dates at each stage. . . . 82

12 Randomized-complete-block ANOVA of tissue nitrate-N by nitrogen treatment and stage-planting combination. 85

13 Performance of five models compared for predicting log1p(Nitrate_ppm) under both dark-enclosure and outdoor illumination, using relative reflectance from the six raw spectral bands and the growth-stage indicator (8 features total), trained on 174 plants and evaluated once on the same held-out test set of 59 plants for both conditions. 94

14 The nine NDVI combinations used as neural-network inputs, formed from the sensor's red (660, 670 nm), red-edge (700 nm), and near-infrared (750, 780, 850 nm) relative-reflectance bands. 97

15 Held-out test performance of the neural network using the nine NDVI combinations and growth stage as inputs, for dark-enclosure and outdoor measurements. 98

16 Held-out test performance of the neural network using the six relative-reflectance bands with and without three added NDVI combinations, with growth stage included in every model (n = 59 test plants). 100

17 Performance of the five models using only the six raw spectral bands with growth stage withheld. 103

18 Share of variance in log1p-transformed tissue nitrate-N explained on the held-out test set (n = 59 plants) by growth stage alone, by the six spectral bands alone, and by both combined (held-out R^2 expressed as a percentage). 104

19 Mean held-out R^2 and r by number of wavelengths combined using dark room readings, neural network model; each value is the mean over all possible combinations of that number of wavelengths (6, 15, 20, 15, 6, and 1 combinations for one to six wavelengths), not the result of a single selected combination. 106

20 Repeated-Measures Mixed-Effects Model Results for the Angle Effect, per Wavelength. 108

21 Wavelength-specific diffuse and specular scale coefficients, their sum, and the specular fraction at zero tilt ($\theta = 0°$). 110

B.1 Complete plant-level spectral dataset (n = 234), all growth stages, plantings, and nitrogen-treatment rates, dark-enclosure measurements. 128

22 Held-out test performance of the neural-network architecture, dark room only, using every possible single-band model (n = 6), sorted by R^2. 144
23 Held-out test performance of the neural-network architecture, dark room only, using every possible two-band combination (n = 15), sorted by R^2. . 145
24 Held-out test performance of the neural-network architecture, dark room only, using every possible three-band combination (n = 20), sorted by R^2. . 146
25 Held-out test performance of the neural-network architecture, dark room only, using every possible four-band combination (n = 15), sorted by R^2. . 147
26 Held-out test performance of the neural-network architecture, dark room only, using every possible five-band combination (n = 6), sorted by R^2. . . 147
27 Held-out test performance of the neural-network architecture, dark room only, using the six-band combination (n = 1). 147
28 Held-out test performance of the neural network using the six relative-reflectance bands, three NDVI combinations, and growth stage as inputs, dark room dataset, for every possible set of three NDVI combinations (n = 84), sorted by R^2. The best combination is shown in bold. 148
29 Held-out test performance of the neural network using the six relative-reflectance bands, three NDVI combinations, and growth stage as inputs, outdoor dataset, for every possible set of three NDVI combinations (n = 84), sorted by R^2. The best combination is shown in bold. 152

List of Equations

1 Voltage regulation feedback resistance calculation (buck stage) 15

2 Voltage regulation feedback resistance calculation (LDO stage) 15

3 Power dissipated by sensor under normal operation 16

4 Effective Response of SIPM 20

5 Photon Energy 21

6 Effective responsivity at 850 nm as an example 21

7 Output voltage after first-stage amplification 22

8 Maximum Output Voltage 23

9 Output voltage after second Stage Amplification Stage 23

10 Corrected digital voltage 25

11 Boost Regulator Inductor 27

12 Boost Regulator Duty Ratio 28

13 Boost Regulator Feedback Resistance 28

14 Reference Voltage Calculation 38

15 Regulated LED current calculation 38

16 Total signal acquisition time 40

17 Effective number of bits due to oversampling 40

18 Thin Lens 51

19 Thin Lens Approximation for a Distant Object 51

20 General Expression for Detected Optical Power 62

21 Relative Reflectance 64

22 NDVI Equation 65

23 Log1p transformation of tissue nitrate-N concentration 69

24 Repeated-Measures Mixed-Effects Model for Angle Effect 71
25 Fresnel-Corrected Specular Reflection Model 72
26 Snell's Law at the Cuticle–Air Interface 73
27 Fresnel Reflection Coefficients, s- and p-Polarization 73
28 Unpolarized Fresnel Reflectance 73
29 Specular Angular-Width Envelope 73
30 Total gain after TIA amplification stage 123
31 Retrieval of voltage output after Transimpedance amplification stage 123
32 Estimating photo current produced 123
33 Photon Energy 124
34 Effective Response of SIPM 124
35 Detected Optical Power 125
36 General expression for estimating optical power detected 125

Acknowledgements

I would like to thank my major professor, Dr. Naiqian Zhang, and Dr. Paul Armstrong, who co-advised this research, for their guidance and support throughout this work. I would also like to thank my committee members, Dr. Daniel Flippo and Dr. Mark Wilkins, for their time and feedback on this work.

I thank the Department of Biological and Agricultural Engineering at Kansas State University for the resources and support that made this research possible, and the FogAg Project team for their help with the field experiment. Finally, I thank my family for their encouragement throughout my studies.

Chapter 1

Introduction

Nitrogen is essential to maize growth because it is closely associated with chlorophyll, photosynthetic capacity, and the development of metabolically active leaf tissue. Changes in plant nitrogen status can therefore alter the way leaves absorb and reflect electromagnetic radiation. In the visible region, chlorophyll strongly absorbs red light, whereas reflectance rises sharply through the red-edge region and remains comparatively high in the near-infrared (NIR), where scattering by internal leaf structure becomes increasingly important. These wavelength-dependent optical responses provide the physical basis for using reflectance measurements and vegetation indices to assess crop condition and, indirectly, nitrogen status (Ustin & Jacquemoud, 2020).

Optical nitrogen sensing can be implemented using passive or active instruments. Passive sensors depend on sunlight as the illumination source and can provide broad spectral information, but their measurements are affected by illumination intensity, solar angle, cloud cover, time of day, canopy geometry, and background contributions. These factors can complicate comparison among measurements collected under changing field conditions. Previous studies have nevertheless demonstrated that carefully selected spectral indices can estimate maize nitrogen-related traits from passive reflectance data (Hasituya et al., 2020). Active sensors address part of this limitation by supplying their own illumination, allowing the measured response to be referenced to a controlled source rather than to sunlight alone.

Padilla et al. (2019), for example, showed that active canopy sensing can be less sensitive to time-of-day variation than passive measurements.

1.1 Active Optical Sensing for Nitrogen Assessment

Active optical sensors have been used successfully to estimate crop biophysical and biochemical properties by illuminating vegetation at selected wavelengths and measuring the returned signal. Cao et al. (2013) demonstrated nitrogen-related discrimination using the multi-waveband Crop Circle ACS-470, illustrating the value of combining red, red-edge, and NIR information. However, the performance of an active sensor is not determined only by wavelength selection. Measurement geometry, sensor-to-target distance, illumination angle, leaf orientation, optical collection efficiency, ambient-light contamination, and canopy position can all influence the recorded signal (He et al., 2016; Li et al., 2021). Nitrogen is also distributed non-uniformly through maize canopies, meaning that measurements of individual leaves and whole canopies do not necessarily present the same physiological information (Winterhalter et al., 2012).

These considerations create an important engineering problem for a compact leaf-level instrument. A handheld sensor must generate sufficient narrowband illumination, recover a small reflected optical signal, account for wavelength-dependent detector sensitivity, and remain stable when measurements are transferred from controlled conditions to the field. It must also be established that any observed spectral differences are associated with biological nitrogen status rather than simply with electronic noise, illumination variability, or changes in leaf optical properties unrelated to nitrogen. Consequently, sensor development and biological validation must be considered together rather than treating the optical hardware and the crop response as separate problems.

1.2 Specular and Diffuse Reflection

Light returned from a leaf to an optical sensor is a mixture of two components with different physical origins. Diffuse reflectance arises from light that enters the leaf, is scattered repeatedly at cell wall–air interfaces within the mesophyll, is partly absorbed by chlorophyll and other pigments, and re-emerges in many directions. This component carries the biochemical information that nitrogen sensing depends on (Woolley, 1971; Ustin & Jacquemoud, 2020). Specular, or surface, reflectance arises at the cuticle–air interface before light enters the tissue. It is governed by the Fresnel equations, depends on the refractive index of the cuticle, the angle of incidence, and polarization, and varies only weakly with wavelength across the visible and near-infrared region, so it carries little information about leaf pigments or nitrogen status (Grant, 1987; Hecht, 2017). Because specular reflection is concentrated around the mirror direction, its contribution to a measurement depends on the relative orientation of the light source, the leaf surface, and the detector, and surface features such as epicuticular wax, hairs, and cuticle roughness broaden the angular range over which it appears (Grant et al., 1993).

This mixing of specular and diffuse reflection is particularly relevant to a handheld leaf-level active sensor. The illumination source and detector are placed close to each other and close to the leaf, and the sensor is designed to view the leaf at near-normal incidence, which is the geometry in which surface-reflected light is most directly returned toward the detector. Small changes in leaf tilt during handheld placement, curvature around the midrib, or leaf rolling can therefore change the specular fraction of the signal from one reading to the next, adding variability that is unrelated to nitrogen. Because surface reflection is approximately constant across wavelength while diffuse reflectance is low in the red, where chlorophyll absorbs strongly, and high in the near-infrared, the same surface-reflected signal forms a larger fraction of the red reflectance than of the near-infrared reflectance. This can distort band ratios and vegetation indices such as NDVI. Previous studies have shown that viewing angle affects leaf nitrogen estimates (He et al., 2016), that leaf-surface reflection can weaken nitrogen-related spectral estimates, and that multi-angle and polarimetric measurements can

help separate surface and tissue responses (Liu et al., 2022; Li et al., 2022). Illumination and reflection geometry is therefore treated in this thesis as both a sensor design issue and a source of measurement uncertainty. A controlled leaf-tilt experiment was used to quantify how reflection angle affects the six-band response.

1.3 Importance of the Sensor for Nitrogen Fertilizer Management

Nitrogen fertilizer is one of the largest variable input costs in maize production, yet much of the nitrogen applied is never taken up by the crop. Raun and Johnson (1999) estimated that only about 33% of the nitrogen applied to cereal crops worldwide is recovered in the harvested grain. The remainder is lost through leaching, denitrification, volatilization, and runoff, which is a direct economic loss to the farmer and a source of nitrate in ground and surface water. A major cause of this low efficiency is the poor match between the nitrogen supplied, which is usually decided before or at planting as a single rate for the whole field, and the nitrogen the crop actually needs, which varies across a field and from one season to the next with soil, weather, and crop growth (Shanahan et al., 2008).

Sensor-based, in-season nitrogen management addresses this mismatch by measuring the crop itself and adjusting the fertilizer rate to the crop's measured nitrogen status at sidedress. In 55 on-farm demonstrations, nitrogen rates directed by crop reflectance sensors reduced nitrogen use by an average of 16 kg N ha^{-1} compared with rates chosen by the producers, while increasing partial profit (Scharf et al., 2011). The handheld sensor developed in this thesis is intended to support this type of decision at the scale at which farmers and crop advisors scout their fields. Because it supplies its own illumination, it can be used at any time of day and under changing sky conditions. Because it measures individual leaves, it can be used to check plants in specific parts of a field where nitrogen loss is suspected, such as low-lying areas, eroded slopes, or areas that received heavy rainfall. A rapid, non-destructive reading in the field also avoids the delay and cost of collecting tissue samples for laboratory analysis, so the information is available while a sidedress or rescue application can still be made around the V10 stage, an important nitrogen management window for maize (Abendroth et al., 2011).

The six wavelengths of the sensor extend beyond the red and near-infrared bands of established two-band meters into the red-edge region, which remains sensitive to chlorophyll and

nitrogen in maize at the higher chlorophyll levels where red reflectance saturates (Schlemmer et al., 2013). Used in this way, the sensor could help a farmer apply additional nitrogen only where and when the crop shows a need and withhold it where the crop already has enough, reducing fertilizer cost and nitrogen losses to the environment without reducing yield. The present study is a first step toward that goal: it develops the sensor hardware and tests how well its six-band readings track tissue nitrate-N. Turning the readings into fertilizer rate recommendations would require further field calibration against the yield response of maize to nitrogen.

1.4 Research Gap and Study Rationale

Existing handheld multispectral active optical sensors for crop nitrogen assessment typically use only a small number of spectral bands. GreenSeeker measures one red and one near-infrared reflectance, making two wavelengths (Raun et al., 2001), and other widely used active meters add at most a red-edge channel beyond that. It remains unclear whether a handheld device capturing substantially richer spectral information across the red absorption region, red-edge transition, and near-infrared plateau would perform better than, worse than, or comparably to established few-band instruments.

This thesis addresses that gap by developing a handheld six-band active sensor, extending the wavelength coverage well beyond the one or two bands typical of existing handheld meters, and evaluating whether that added spectral information translates into a meaningful advantage for nitrogen-related sensing. Because a leaf-level active sensor views the leaf at close range and near-normal incidence, this thesis also examines whether specular reflection from the leaf surface contributes measurably to the six-band signal, and how that contribution varies with wavelength.

1.5 Research Objectives

The overall objective of this research was to design, implement, and evaluate a handheld, six-band multispectral active optical sensor as a platform for assessing nitrogen-related spectral response in maize leaves. The specific objectives were to:

1. Design and implement a portable multispectral sensing system that sequentially illuminates maize leaves at 660 nm, 670 nm, 700 nm, 750 nm, 780 nm, and 850 nm, and measures the reflected signal using a Silicon Photomultiplier (SiPM).

2. Characterize the wavelength-dependent response of the complete sensing system using a 99% Spectralon reference, including recovery of SiPM photocurrent and estimation of detected optical power.

3. Evaluate whether the six-band spectral response can predict tissue nitrate-N, and if the six bands have any advantage over one or two bands in predicting tissue nitrate-N.

4. Quantify how leaf tilt angle affects the six-band response and estimate the relative contributions of specular and diffuse reflection to the measured reflectance at each wavelength.

Together, these objectives establish a progression from sensor construction and optical characterization to biological evaluation and sensor performance assessment.

Chapter 2

Sensor Structure, Mode of Operation, Circuit Design and Optics

2.1 Sensor Mode of Operation

The sensor operates on the principle of diffuse reflectance spectroscopy, exploiting the wavelength-dependent interaction between incident light and foliar biochemical constituents to enable non-destructive, in situ estimation of leaf nitrogen concentration. When plant tissue is illuminated at spectrally discrete wavelengths, characteristic reflectance and absorption signatures emerge as a direct consequence of the optical properties of photosynthetic pigments, primarily chlorophyll a and b, whose concentrations are stoichiometrically coupled to foliar nitrogen status (Schlemmer et al., 2013). The developed system interrogates these spectral features through sequential LED illumination across six discrete wavelength bands centered at 660 nm, 670 nm, 700 nm,750 nm, 780 nm, 850 nm. These bands were selected to capture key reflective properties associated with chlorophyll and accessory pigments, as well as the red-edge inflection region, which is particularly sensitive to nitrogen-driven changes in pigment loading. The reflectance intensity at each band is subsequently acquired, digitized, and processed by an embedded inference model to yield quantitative foliar nitrogen estimates in real time.

A measurement cycle is initiated when a user activates the scan function via the capacitive LCD touchscreen, whereupon a synchronization command is transmitted from the display processing unit to the LED array driver board over a full-duplex Serial Peripheral Interface (SPI) bus. The SPI protocol was selected for its low latency and high-speed clocking capability, ensuring precise temporal alignment between LED activation events and Analog to Digital Convertor (ADC) acquisition windows. This alignment is a requirement for reliable multi-band spectral sampling. Upon receipt of the command, the driver board activates each LED sequentially in a predefined order, with inter-band settling delays of 50 ms imposed to prevent optical crosstalk and allow photocurrent transients to decay prior to signal acquisition. A plano-convex collimating lens positioned at the exit aperture of the LED assembly reduces beam divergence and enforces spatially uniform irradiance across the measurement footprint on the leaf lamina, thereby minimizing edge-scattering artifacts.

The interaction of each narrowband illumination pulse with the leaf surface gives rise to wavelength-dependent reflectance governed by the reflectance and absorption properties of the leaf mesophyll and epidermis. In spectral regions coinciding with strong pigment absorption, notably the red (660 nm, 670 nm, 700 nm, 750 nm) bands, reflectance is attenuated in proportion to pigment concentration, providing an indirect measure of nitrogen content. The diffuse reflected light is collected by a converging lens, optically aligned to focus the diffuse reflectance signal onto the active area of a silicon photomultiplier (SiPM) detector. The SiPM is reverse-biased beyond its avalanche breakdown voltage to operate in Geiger-mode multiplication, therefore each absorbed photon triggers a self-sustaining avalanche of charge carriers, yielding single-photon sensitivity and a high dynamic range (Gundacker & Heering, 2020). This detection modality is particularly advantageous for resolving low-intensity reflectance signals in strongly absorbing spectral bands, where conventional photodiodes would exhibit prohibitively low signal-to-noise ratios.

The analog photocurrent generated by the SiPM is routed through a two-stage signal conditioning chain designed to maximize signal fidelity prior to digitization. In the first stage, a transimpedance amplifier (TIA) employing a feedback resistor (330 ohms) converts the photocurrent to a proportional output voltage while maintaining low input-referred current

noise. This is a critical design consideration given the small-signal nature of the optical reflectance measurement. The second stage applies fixed voltage gain of (1.1 V/V) to scale and invert the conditioned signal to the full positive input range of the downstream analog-to-digital converter (ADC), thereby maximizing the effective use of the converter's dynamic range. To extend the TIA's linear operating region and accommodate the wide reflectance dynamic range across the six measurement bands, a charge pump circuit generates a stable negative supply rail. This dual-supply configuration enables near-rail-to-rail output swing, preventing saturation at high reflectance levels and clipping at low-reflectance extremes. Both of these failure modes would otherwise introduce systematic measurement error.

The amplified analog signal is digitized by a 16-bit successive approximation register (SAR) ADC configured to perform hardware oversampling across multiple acquisition cycles per LED activation event. The oversampled measurements are averaged to improve measurement stability and produce a six-band spectral feature vector corresponding to (660 nm, 670 nm, 700 nm,750 nm, 780 nm, 850 nm). The resulting vector is transmitted to the main microcontroller for display, storage, and subsequent analysis. The embedded hardware architecture was designed with sufficient processing and external-memory resources to support a future inference model; however, the present thesis evaluates sensor functionality using the recorded spectral measurements and offline statistical analysis rather than claiming a validated real-time nitrogen-prediction model.

During the present prototype evaluation, the LCD provides the operator interface for initiating measurements and displaying sensor status and acquired measurement information, while the six-band measurements and associated metadata are logged to the onboard SD card in structured files for post-processing. The display-processing unit and external memory were intentionally selected to permit future deployment of a validated embedded prediction model once sufficient ground-truth data are available. Accordingly, the current system is a portable multispectral acquisition platform with an inference-ready embedded architecture. An overview of the sensor's operation is summarized in Figure 1.

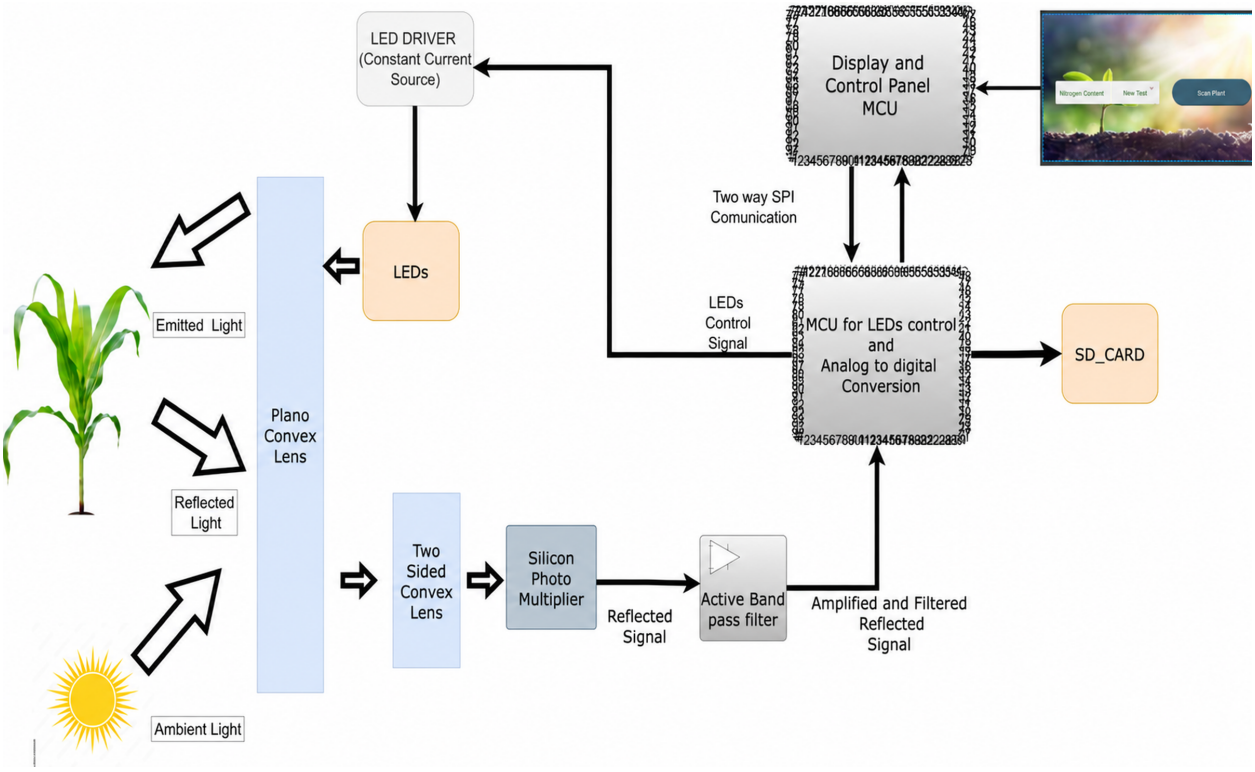


Figure 1. Sensor Operational Chart.

2.2 Power Supply Circuit

In this project the power supply subsystem was designed to deliver stable, well-regulated voltages to all critical loads within the sensor platform, including the dual-microcontroller units, the LCD display module, the LED array driver, and the high-voltage reverse-bias circuit for the Silicon Photomultiplier (SiPM). Given the diverse voltage and current requirements of these subsystems and the need to operate from a portable battery source, a power supply architecture capable of simultaneously generating multiple regulated output rails from a single input was required. A 12 V, 6000 mAh lithium-ion battery pack was selected as the primary energy source, offering a practical balance between energy density, discharge characteristics, and compatibility with the boost converter topology required to generate the SiPM reverse bias voltage (25.5V). The input voltage range was specified as 12 V to 16 V to accommodate variation in battery terminal voltage across the discharge cycle, ensuring that all downstream regulators remain within their specified operating conditions throughout the device's operational lifetime. The power supply board was custom-designed and assembled for this project rather than purchased as a commercial module; its design is therefore documented in this section.

To meet these requirements, the TPS65320DQPWPRQ1 DC-DC voltage converter from Texas Instruments was selected as the primary regulation device. This component was chosen for its dual-output architecture, integrating a high-efficiency synchronous buck (step-down) converter and a low-dropout (LDO) linear regulator within a single package. This level of integration reduces passive component count, minimizes PCB footprint, and simplifies the power distribution network. These are all critical considerations in a compact, handheld instrument. The buck converter topology was specifically favored over a purely linear approach due to its significantly higher conversion efficiency under moderate-to-heavy load conditions, reducing heat dissipation and extending battery life. The integrated LDO regulator provides a low-noise, well-regulated secondary rail suitable for powering noise-sensitive digital logic and communication peripherals, where the ripple voltage rejection and transient response characteristics of a linear regulator are preferable to those of a switching voltage convertor.

The buck regulator was configured to supply a regulated output voltage of 5.5 V at a maximum continuous output current of 2.1 A. This rail provides power to subsystems with higher voltage requirements, specifically the LED constant-current driver circuitry and the SiPM bias boost converter. The integrated LDO regulator was configured to output 3.3 V, also rated at up to 2.1 A, to supply logic-level components, including both STM32 microcontrollers, the SPI and SDMMC communication interfaces, and associated digital peripherals. This two-rail power architecture ensures that switching noise from the buck converter does not propagate to the sensitive digital subsystems, providing inherent noise isolation between power domains.

The output voltages of both the buck regulator and the LDO regulator are programmed using external resistor voltage dividers connected to their respective feedback (FB) pins. The TPS65320 regulates its outputs by maintaining the feedback pin voltage at an internal reference of 0.8 V, adjusting the duty cycle of the switching waveform for the buck stage, or the pass transistor gate drive for the LDO regulator, accordingly. For the buck regulator, resistors R6 and R7 form the feedback divider, with R6 fixed at 13 kΩ and R7 selected as the design variable. For the LDO regulator, resistors R9 and R11 form the equivalent network as shown in Figure 2, with R9 fixed at 10 kΩ and R11 as the design variable. The governing relationships for computing R7 and R11 are derived from the standard feedback regulation equations provided in the device datasheet and are expressed in Equation 1 and Equation 2 respectively.

$$R7 = R6 \times \frac{V_o - 0.8}{0.8} \tag{1}$$

$$R11 = R9 \times \frac{V_o - 0.8}{0.8} \tag{2}$$

where:

V_o is the desired output voltage (5.5 V for the buck stage; 3.3 V for the LDO),

R_6 is the 13 kΩ (fixed upper feedback resistor, buck stage),

R_9 is 10 kΩ (fixed upper feedback resistor, LDO stage), and

0.8 V is the internal feedback reference voltage of the TPS65320.

Applying Equation 1 with $V_o = 5.5$ V and $R_6 = 13$ kΩ yields $R_7 = 77.625$ kΩ, for which the nearest standard value of 78.7 kΩ was selected. Applying Equation 2 with $V_o = 3.3$ V and $R_9 = 10$ kΩ yields $R_{11} = 31.25$ kΩ, for which a standard value of 31.6 kΩ was used. Both selections result in output voltages of 5.6 V for the buck converter and 3.3 V for the low dropout converter. This is within 1% of their respective target values, well within the tolerance requirements of the downstream subsystems.

Under nominal operating conditions, the sensor draws approximately 0.5 A from the 12 V supply, yielding a total system power dissipation of approximately 6 W, as expressed below.

$$P = V_{in} \times I = 12\ V\ \times 0.5\ A\ = 6\ W \tag{3}$$

The switching efficiency of the TPS65320 buck stage, rated at up to 92% under representative load conditions, ensures that the majority of input power is delivered to the load rather than dissipated as heat within the regulation circuitry. The power supply schematic and assembled board are presented in Figure 2 and Figure 3 respectively.

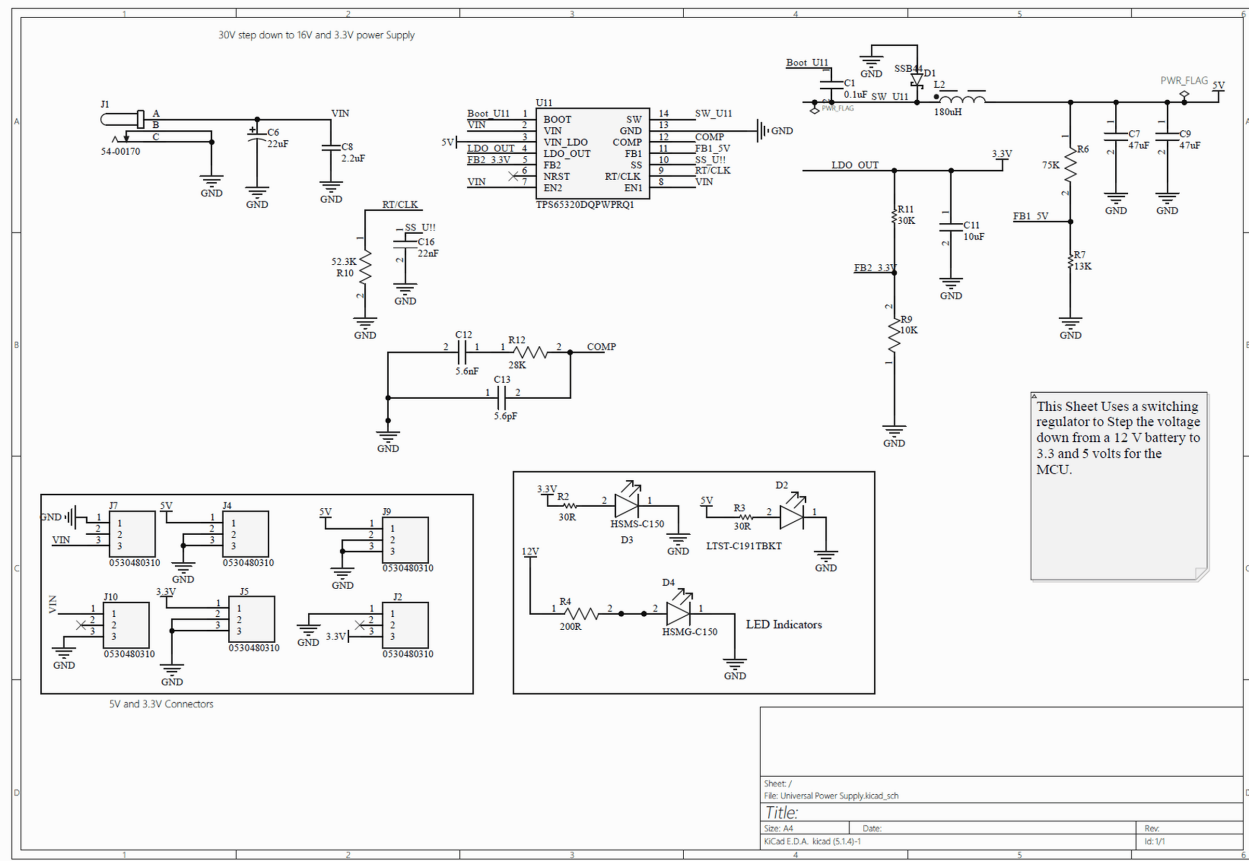


Figure 2. Main Power Supply Schematic (12V is stepped down to 5 V and 3.3 V).

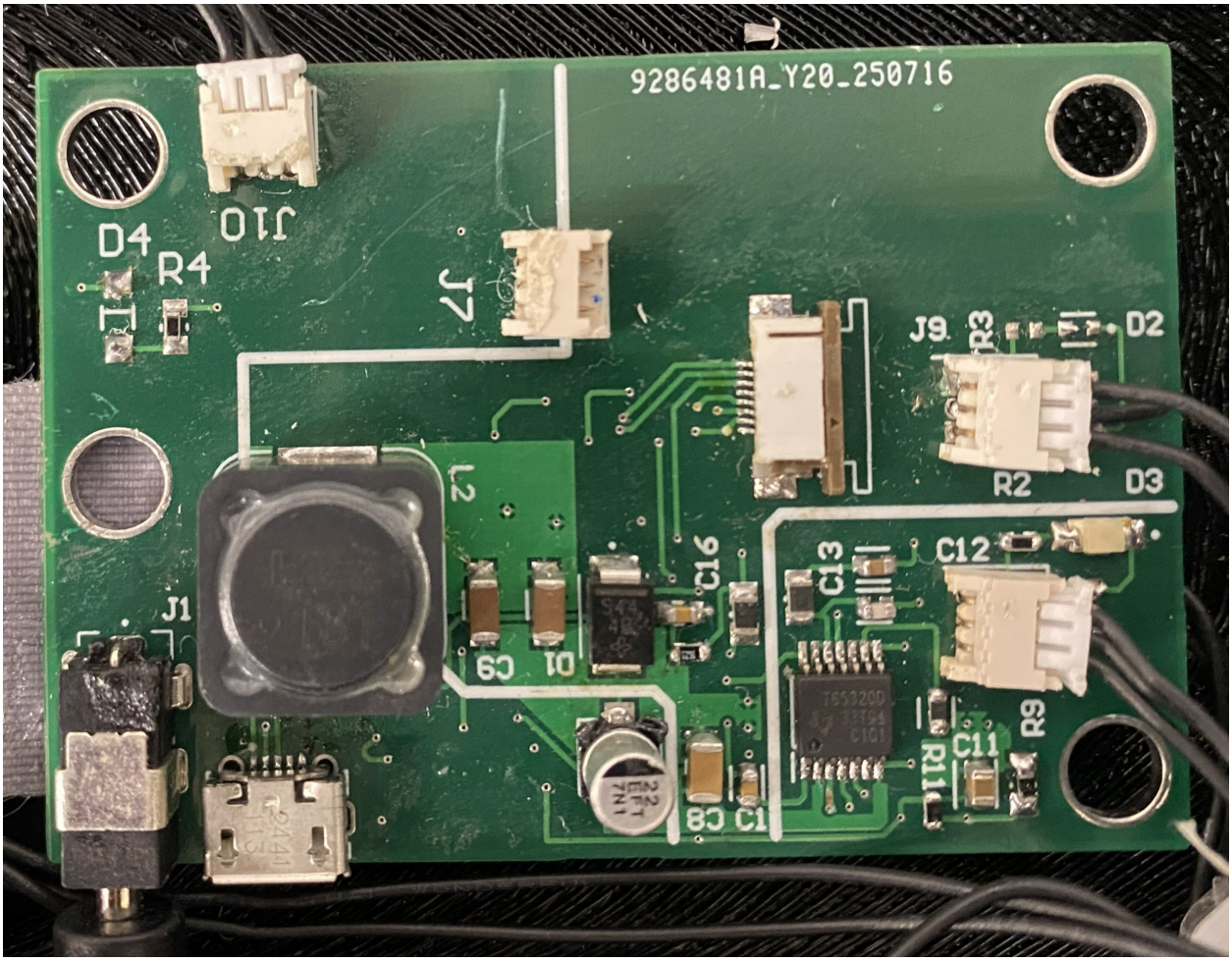


Figure 3. Main Power Supply Board.

2.3 Light Detection and Amplification

Semiconductor photodetectors can be operated in two distinct biasing configurations, photoconductive mode and photovoltaic mode. In the photoconductive mode, a reverse bias voltage is applied across the junction, expanding the depletion region and establishing a strong internal electric field that accelerates the separation of photogenerated electron-hole pairs. This results in reduced carrier transit time, improved linearity, and enhanced sensitivity to low-intensity optical signals, at the cost of increased dark current and shot noise associated with the reverse-biased junction. In the photovoltaic mode, the device operates under zero external bias, relying entirely on the built-in junction potential to drive carrier separation. While this configuration minimizes dark current and junction capacitance, the reduced electric field strength limits sensitivity and dynamic range. Given that the present application requires detection of weak, narrowband reflected signals in the presence of ambient optical background, photoconductive operation was adopted as the preferred detection modality, with the reverse bias voltage optimized to maximize carrier sweep-out efficiency, and minimize noise (Gundacker & Heering, 2020).

To further enhance detection sensitivity beyond that achievable with conventional reverse-biased photodiodes, a Silicon Photomultiplier (SiPM) was employed as the primary optical transducer, as shown in Figure 4. The SiPM operates in Geiger-mode avalanche multiplication, wherein the applied reverse bias voltage exceeds the junction breakdown voltage, enabling each absorbed photon to trigger a self-sustaining avalanche of charge carriers. This internal multiplication mechanism provides single-photon sensitivity and significantly amplifies the primary photocurrent prior to any external signal conditioning. The MICROFC-30035-SMT-TR1 SiPM (onsemi, n.d.) was selected for this application, with a rated breakdown voltage of 24.7 V. (The device was reverse-biased at 25.5 V, establishing an overvoltage of 0.8 V above breakdown. This overvoltage was chosen as a design compromise. Insufficient overvoltage reduces avalanche probability and gain, while excessive overvoltage increases dark current count rate and after-pulsing probability. Both effects degrade the signal-to-noise ratio.

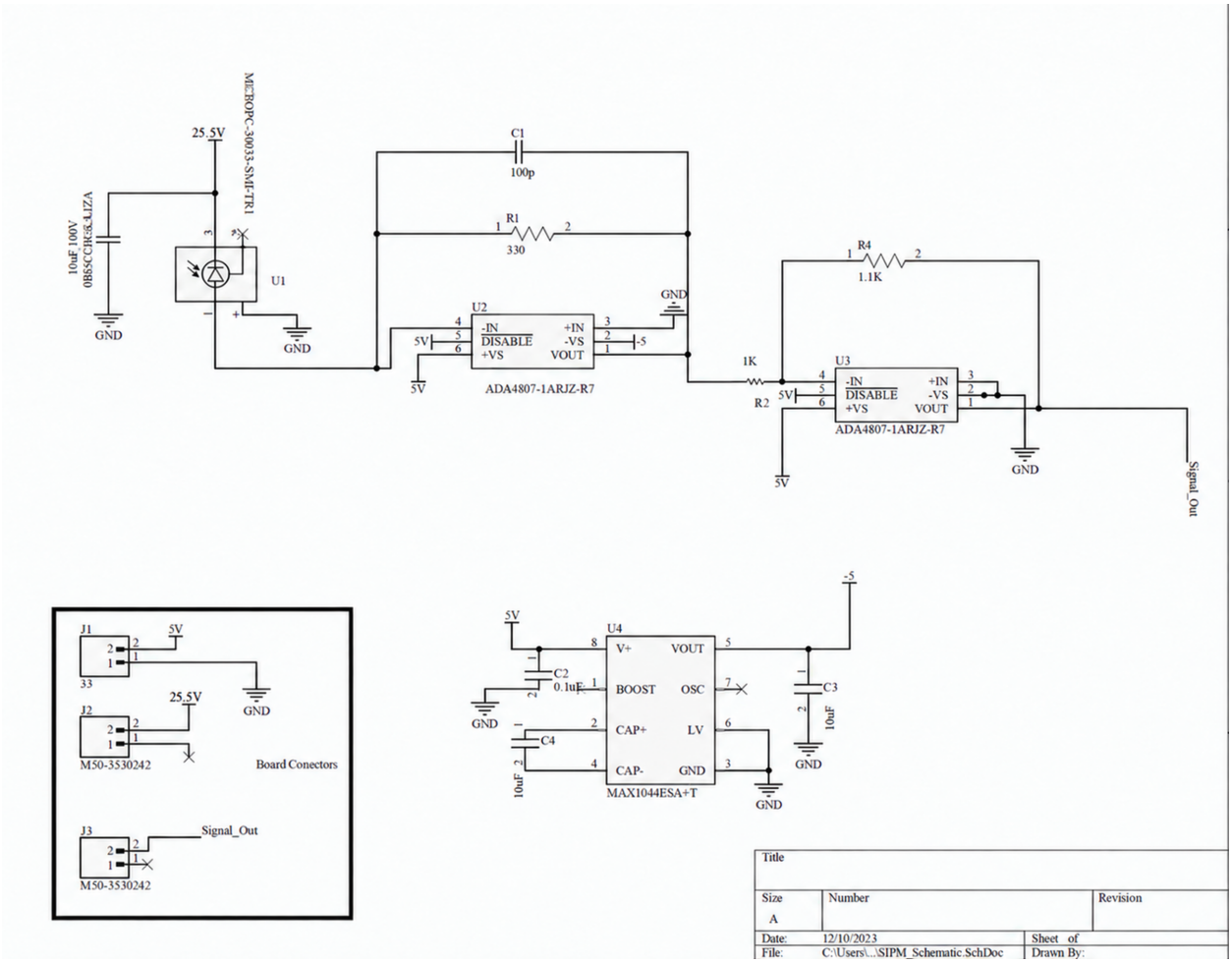


Figure 4. Design schematic of the light detection and amplification stage. The figure shows the first stage transimpedance amplification stage and the second stage voltage amplification.

The effective spectral responsivity of the SiPM, R_{eff}, was estimated from first principles to characterize its photoelectric conversion capability across the sensor's operational bands. Unlike conventional photodiodes where responsivity is a direct material property, the effective responsivity of a SiPM incorporates both the photodetection efficiency and the internal avalanche gain, and is given by Equation 4 (Gundacker & Heering, 2020).

$$R_{eff} = \frac{(PDE)\,(G)\,(q)}{E_{photon}} \tag{4}$$

where:

PDE is the photodetection efficiency, shown in Figure 5,

G is the avalanche multiplication gain (1×10^5 at 25.5 V obtained from the datasheet),

q is the elementary charge (1.6×10^{-19} C), and

$E_{photon} = hc/\lambda$ is the photon energy at wavelength λ.

$$E_{photon} = \frac{hc}{\lambda} \tag{5}$$

where:

h is the Planck's constant (6.626×10^{-34} J·s),

c is the speed of light (3×10^8 m/s), and

λ is the wavelength of light.

Applying Equation 4 at the extreme wavelength of the sensor's operational range yields the effective responsivity estimates as shown in Equation 6 for 850 nm.

$$\begin{aligned} E_{photon} &= \frac{6.626 \times 10^{-34} \times 3 \times 10^8}{850 \times 10^{-9}} = 2.34 \times 10^{-19}\ J \\ R_{eff} &= \frac{0.01 \times 1 \times 10^{5} \times 1.6 \times 10^{-19}}{2.34 \times 10^{-19}} = 683.76\ \frac{A}{W} \\ R_{eff} &= 0.68\ \frac{A}{mW} \end{aligned} \tag{6}$$

The value reflects the combined effect of quantum detection probability and internal gain and represents the effective current output per unit of incident optical power at each wavelength. While the estimated effective responsivity R_{eff} appears high in $\frac{A}{mW}$, the optical power incident on the SiPM active area under typical reflectance measurement conditions is extremely low, on the order of microwatts, resulting in photocurrents that remain small in absolute terms. Consequently, external signal conditioning was still required to raise the output to levels compatible with the downstream analog-to-digital converter.

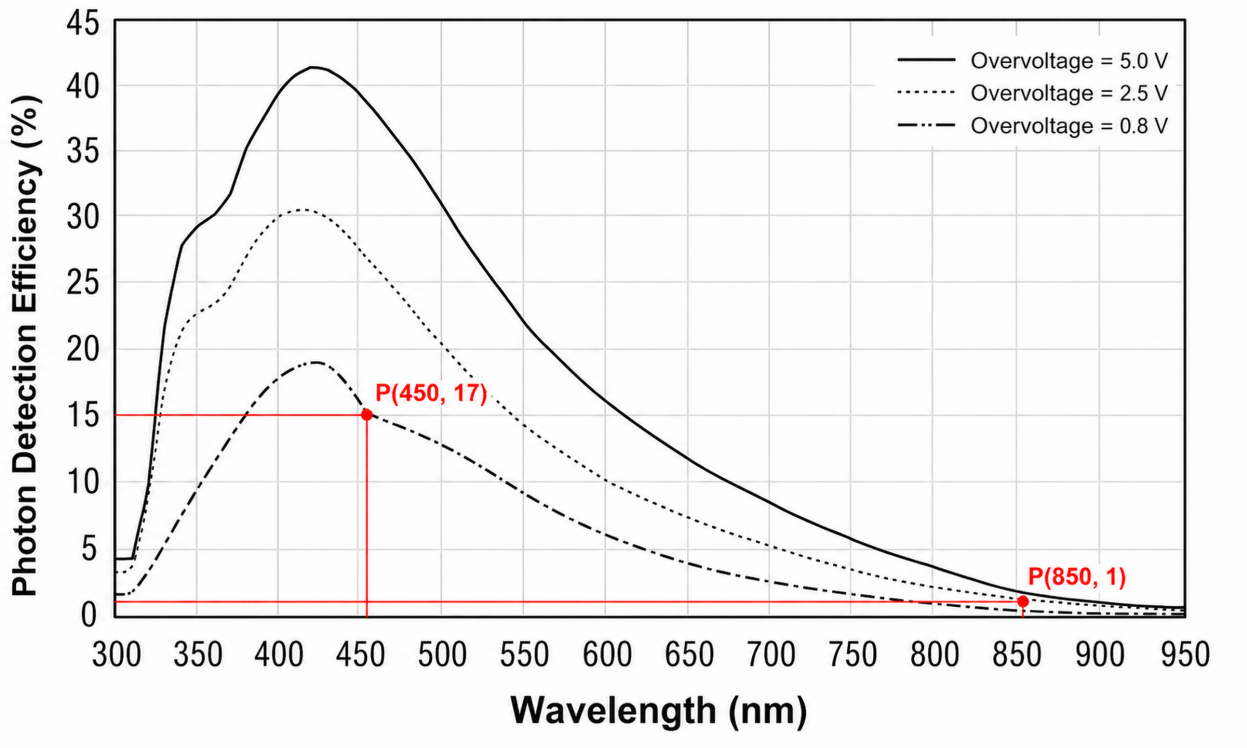


Figure 5. Photo Detection Efficiency of the silicon photomultiplier (SIPM) (Retrieved from the onsemi SiPM datasheet).

A two-stage signal conditioning chain was therefore implemented downstream of the SiPM. The first stage was configured as a transimpedance amplifier (TIA) using the ADA4807-1ARJZ precision operational amplifier. The TIA topology was selected for its ability to convert the SiPM photocurrent directly to a proportional output voltage while maintaining a low input-referred current noise, which is critical for preserving signal fidelity at low photocurrent levels. The transimpedance gain was set by feedback resistor R1 = 330 Ω, yielding the voltage output governed by Equation 7.

$$V_{out_TIA} = I_{photo}\ (R_{TIA}) = I_{photo}\ (330\ \Omega) \tag{7}$$

where:

V_{out_TIA} is the output voltage after the transimpedance amplification stage,

R_{TIA} is the transimpedance gain, which was designed and fixed at 330 Ω, and

I_{photo} is the photocurrent from the silicon photomultiplier.

A 100-pF feedback capacitor (C1) was placed in parallel with R1 to form a first-order low-pass filter within the feedback path as shown in Figure 4. This compensation network limits the closed-loop bandwidth of the TIA, suppressing high-frequency noise components that would otherwise be amplified alongside the signal. The inclusion of C1 extended the dynamic range of the signal conditioning circuit to approximately 71 dB and established a noise floor of 1.37 mV at the TIA output. The maximum photocurrent sourced by the MICROFC-30035-SMT-TR1 under full saturating illumination is approximately 15 mA, which at the specified transimpedance gain produces the maximum TIA output voltage shown in Equation 8.

$$V_{out_max} = 15 \ \times 10^{-3} \ \times 330 \ = 4.95 \ V \tag{8}$$

This value remains within the linear operating range of the ADA4807-1 under its specified maximum power supply rails of $\pm 5V$, hence ensuring that the TIA does not saturate under the maximum expected optical input.

The second stage of the signal conditioning chain was implemented as a non-inverting voltage amplifier, also using an ADA4807-1ARJZ operational amplifier. This stage serves to restore the signal polarity to a positive domain following the inverting configuration of the TIA. The voltage gain of the non-inverting stage is given below.

$$\begin{gathered} A_2 = 1 + \frac{R_4}{R_2} = \left(\frac{1.1}{1}\right) \ \approx 1.1 \\ V_2 = A_2(V_{out_TIA}) \end{gathered} \tag{9}$$

where:

A_2 is the second-stage amplification gain, and

V_2 is the output voltage after the transimpedance amplification stage.

With resistor values R2 = 1 kΩ and R4 = 1.1 kΩ, the second stage applies a gain factor

of approximately 1.1, scaling the maximum TIA output of 4.95 V to approximately 5.44. Beyond this level, the amplifier approaches its output swing limit, and the system loses linearity. The combined two-stage architecture therefore spans the full dynamic range from the 1.37 mV noise floor to the 5.44 V saturation ceiling, yielding an effective system dynamic range consistent with the 71 dB.

The inverting configuration of the TIA stage necessitates a negative supply rail to enable the operational amplifier output to swing below the ground reference, ensuring accurate reproduction of the bipolar photocurrent signal without clipping. A dedicated negative voltage rail was generated using a charge pump circuit based on the MAX1044ESA+T CMOS voltage converter. The MAX1044 was selected over a discrete inverting switching regulator due to its compact form factor, minimal external component requirements, and suitability for low-current conditions. The device operates by alternately charging and discharging external capacitors via an internal oscillator-driven switching network. Capacitor C2 functions as the charge transfer element, while C4 serves as the output reservoir capacitor. During each switching cycle, the charge accumulated on C2 is transferred to C4 with inverted polarity, generating a stable negative output voltage at the VOUT pin (pin 5) equal in magnitude to the positive input supply. This negative rail is applied to the V^- supply pin of the first ADA4807-1 amplification stage, enabling near-rail-to-rail output swing in the negative domain. A bootstrap capacitor (C3) connected between the BOOST and OSC pins of the MAX1044 doubles the internal oscillator frequency, reducing output ripple and improving regulation. A decoupling capacitor (C5) placed at the output node further attenuates residual switching ripple, ensuring that charge pump noise does not couple into the sensitive analog signal path.

Subsequently, the analog voltage from the amplification stage is taken through the analog to digital convertor of the STM32 MCU and converted to a digital equivalent. In this project, the signal is retrieved in two steps; first when the LEDs are off and secondly when the LEDS are turned on. Hence, this produces a finally corrected digital voltage for each wavelength as reported in Equation 10.

$$V_{corrected,\lambda} = V_{LED-ON,\lambda} - V_{DARK,\lambda} \tag{10}$$

where:

$V_{corrected,\lambda}$ is the background-corrected digital voltage at wavelength λ, after LED-OFF subtraction,

$V_{LED\text{-}ON,\lambda}$ is the recorded digital voltage with the LEDs on, for each wavelength (660 nm, 670 nm, 700 nm, 750 nm, 780 nm, and 850 nm), and

$V_{DARK,\lambda}$ is the recorded digital voltage with the LEDs switched off, representing residual ambient light voltage offset for each wavelength (660 nm, 670 nm, 700 nm, 750 nm, 780 nm, and 850 nm).

2.4 Boost Regulator Circuit

To provide the necessary reverse-bias voltage for the SiPM, a boost regulator circuit was implemented using the LT8337JV#PBF, which is a low-quiescent-current, synchronous step-up Silent Switcher converter from Analog Devices (Analog Devices, 2021). Figure 6 shows the schematic implementation of the boost convertor. This device supports output voltages up to 28 V and a peak current output of 5 A, making it suitable for driving high-voltage, low-current loads such as avalanche-mode photodetectors. In this design, the converter was configured to generate a stable output of 25.5 V, slightly above the 24.7 V breakdown voltage of the selected SiPM (MICROFC-30035-SMT-TR1), resulting in an overvoltage of 0.8 V. This ensures the SiPM operates in Geiger (avalanche) mode, where it achieves high photo current gains and photon sensitivity.

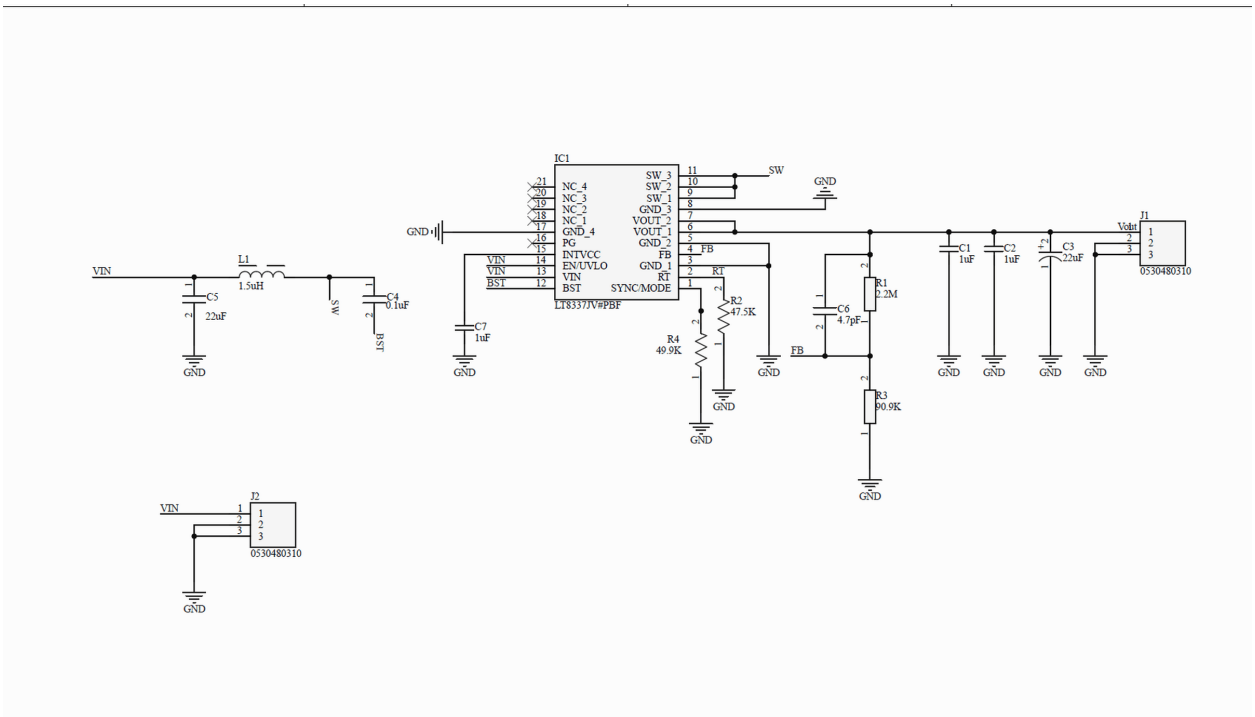


Figure 6. Step-Up Converter Schematic.

The LT8337 integrates an internal low-dropout (LDO) regulator that generates a 3.5 V rail (INTVCC) from the input voltage. This internal supply powers the MOSFET gate drivers and biasing circuitry. As recommended in the datasheet, a 1 µF ceramic capacitor

(C7) was placed between INTVCC and ground to stabilize this internal supply and suppress high-frequency noise.

The output filtering network consists of decoupling capacitors C1 (1 µF), C2 (1 µF), and C3 (22 µF), which work together to reduce output voltage ripple and provide a low-impedance DC output. This clean and stable 25.5 V bias is essential for consistent SiPM performance, as ripple and transient noise could otherwise introduce signal integrity issues during sensitive optical measurements.

The switching frequency of the boost converter was set to approximately 1.9 MHz by selecting R2 = 47.5 kΩ. This frequency was chosen as a compromise between conversion efficiency, output ripple, and the physical size of passive components such as the inductor and capacitors. The converter was also configured to operate in burst mode, a high-efficiency light-load mode enabled by setting R4 = 49.9 kΩ, which helps reduce switching losses when the current output demand is low. To ensure automatic startup upon application of input voltage, the EN (Enable) pin was connected directly to VIN. Additionally, a bootstrap capacitor (C4 = 0.1 µF) was placed between the BST and SW pins, as specified in the datasheet, to support the operation of the high-side gate driver.

The inductor L1 (1.5 µH) is a critical component in the boost topology. During each switching cycle, it stores energy when the internal MOSFET is on and releases it to the output when the switch turns off, thereby increasing the output voltage. The inductor value was selected based on the standard inductor sizing equations for boost converters (Equation 11), considering the target output voltage (25.5 V), minimum input voltage, peak current requirements, switching frequency (1.9 MHz), and ripple current limits.

$$L = \frac{V_{IN(MIN)}}{\Delta I_{SW} \times f_{SW}} \times D_{max} \tag{11}$$

where:

L is the inductance,

$V_{in(min)}$ *is the minimum input voltage which is* 9 V,

ΔI_{SW} is the designed ripple current which is 0.008 mA,

f_{SW} *is the switching frequency* 1.9 MHz, and

D_{max} *is the duty cycle of the power switch.*

$$D_{max} = \frac{V_{out} - V_{IN(MIN)}}{V_{out}} \tag{12}$$

Finally, resistors R1 and R3 in the feedback path of the boost converter are responsible for stepping up the voltage. Hence, to program the output voltage at 25.5 V, R3 was fixed at 90.9 kΩ, and resistor R1 was designed using Equation 13 to obtain a resistance of 2.2 KΩ.

$$R1 = R2 \, . \, \frac{V_{out}}{1} - 1 \tag{13}$$

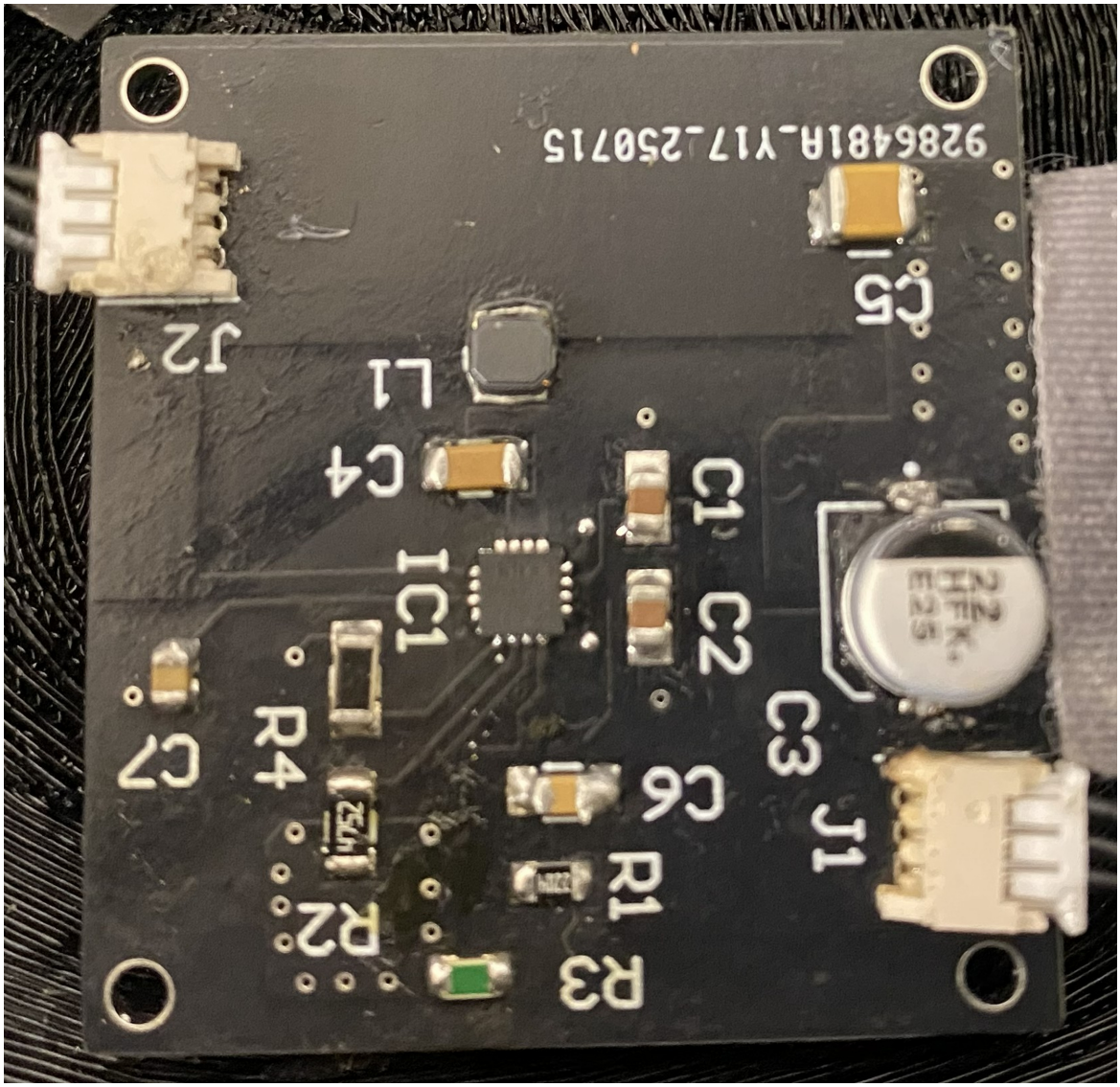


Figure 7. Step-Up Converter Regulator.

2.5 Light-Emitting Diode (LED) Array Module

Narrowband light-emitting diodes (LEDs) were selected as the active illumination source for the multispectral relative reflectance measurement system. LEDs were preferred over broadband alternatives such as halogen or tungsten lamps since their narrow spectral emission bandwidth, deterministic center wavelength, and compact form factor, all of which are advantageous in a portable optical sensing architecture. Six discrete LED types were selected, with center wavelengths of 660 nm, 670 nm, 700 nm, 750 nm, 780 nm, and 850 nm, corresponding to spectral regions of agronomic significance for foliar nitrogen assessment. Which is the chlorophyll-a red absorption maximum (660–670 nm), the red-edge transition (700–750 nm), and the near-infrared reflectance plateau (780–850 nm) used for structural and biomass-related indices. Given that the sensor is designed for operation under ambient outdoor illumination, high-radiance LEDs were required to ensure that the actively emitted signal dominates over background irradiance at the detector. During a measurement cycle, the six channels are activated sequentially in the order 780 nm $\rightarrow$ 670 nm $\rightarrow$ 700 nm $\rightarrow$ 850 nm $\rightarrow$ 660 nm $\rightarrow$ 750 nm, as shown in Table 2. This order can be modified by simply changing the control command logic sequence in the firmware. Each LED string hand a quiescent operating current of $I_{\mathrm{F}} = 700$ mA at 4.5V.

2.5.1 Dual-Ring Geometry for LEDs Array Module

To maximize the optical power delivered to the target within the measurement footprint while maintaining a compact sensor-head geometry, each spectral wavelength is assigned a dedicated LED pair comprising two identical LEDs connected in series. The two LEDs within a pair are positioned diametrically opposite one another on two concentric circular mounting rings, such that their emission axis converges at the center of the sensor's field of view. The inner mounting ring has a diameter of 50 mm, and the outer ring has a diameter of 65 mm. One LED of each pair is mounted on the inner ring, with its complementary LED positioned diametrically opposite on the outer ring as shown in Figure 8. This dual-ring arrangement produces spatial overlap of the two emission beams at the measurement center,

nominally doubling the irradiance delivered to the target. The physical arrangement of the six wavelength pairs on the concentric rings is illustrated in Figure 8.

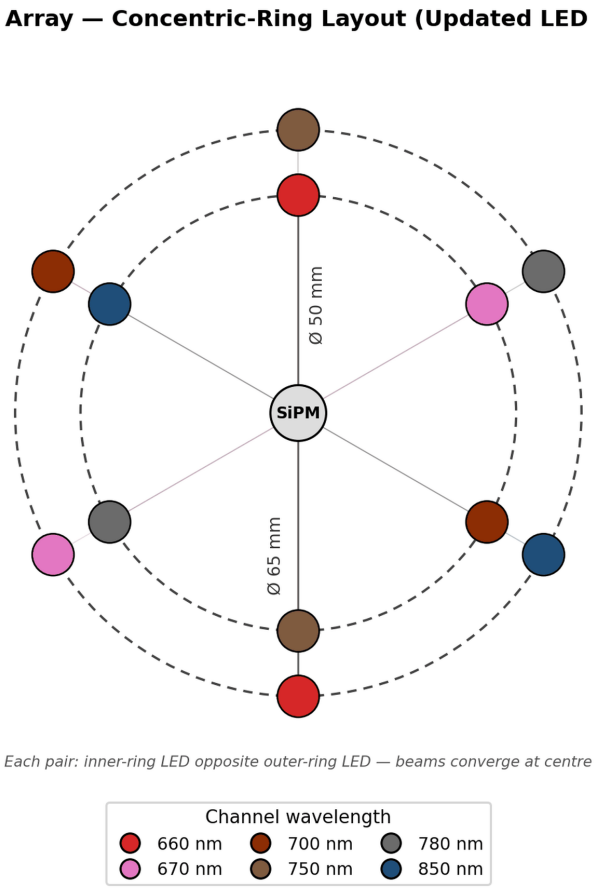


Figure 8. LEDs array concentric-ring layout. Each wavelength pair comprises one LED on the inner ring (Ø 50 mm) and its counterpart diametrically opposite on the outer ring (Ø 65 mm), ensuring beam convergence at the SiPM detector center.

Because the two devices in a pair are the same LED part (same peak wavelength and part number) and series-connected, they carry the same current. The driver that supplies power to the LED modules is designed to budget a forward voltage (V_{F}) of 4.5 V per device, so each two-LED series string is supplied from a 9 V rail. Series connection was chosen deliberately. Because each string is driven by a constant-current source, the same regulated current flows through both LEDs regardless of small device-to-device differences in forward voltage. The main limitation of the series string is that an open-circuit failure of one LED turns off both LEDs of that wavelength; such a failure would appear immediately as a missing channel in

the sensor reading.

Table 1. Parameters of LEDs used in the project (Retrieved from the LED datasheets).

Wavelength (nm)	Device (Manufacturer / P/N)	Forward Voltage (V_F) Pair diode (V)	Supplied string voltage (V), design	Theoretical Optical Power $\Phi_{e,pair}$ (mW)
660	OSRAM GH CSSRM4.24	4.5	9.00	2090
670	EPIGAP OCI-490-20 IE670-XE	4.5	9.00	816
700	EPIGAP OCI-460 IE700-XS	4.5	9.00	1400
750	OSRAM GF PUSRA1.25	4.5	9.00	1802
780	EPIGAP OCI-460 IE780-XS	4.5	9.00	1346
850	Luminus SST-10-IRD-B90H	4.5	9.00	2446

2.5.2 Spectral Bandwidth of Each LED Wavelength

Figure 9 shows the idealized relative spectral emission of the six wavelengths, reconstructed as Gaussian profiles from each device's datasheet peak wavelength and spectral bandwidth (FWHM). The six bands provide near-uniform coverage from the chlorophyll-a absorption region (660–670 nm) through the red-edge transition (700–750 nm) into the near-infrared plateau (780–850 nm), with minimal spectral overlap between adjacent wavelengths.

During a measurement cycle, only one LED pair corresponding to a single wavelength is activated at any given time. This sequential, single-wavelength illumination strategy ensures that the reflectance signal acquired by the SiPM at each time step is attributable exclusively to the active wavelength, eliminating inter-wavelength optical crosstalk and enabling accurate wavelength-resolved reflectance measurement following correction for ambient background irradiance. Sequential activation also ensures that the constant-current LED driver always operates within its rated output capacity, since simultaneous activation of multiple high-

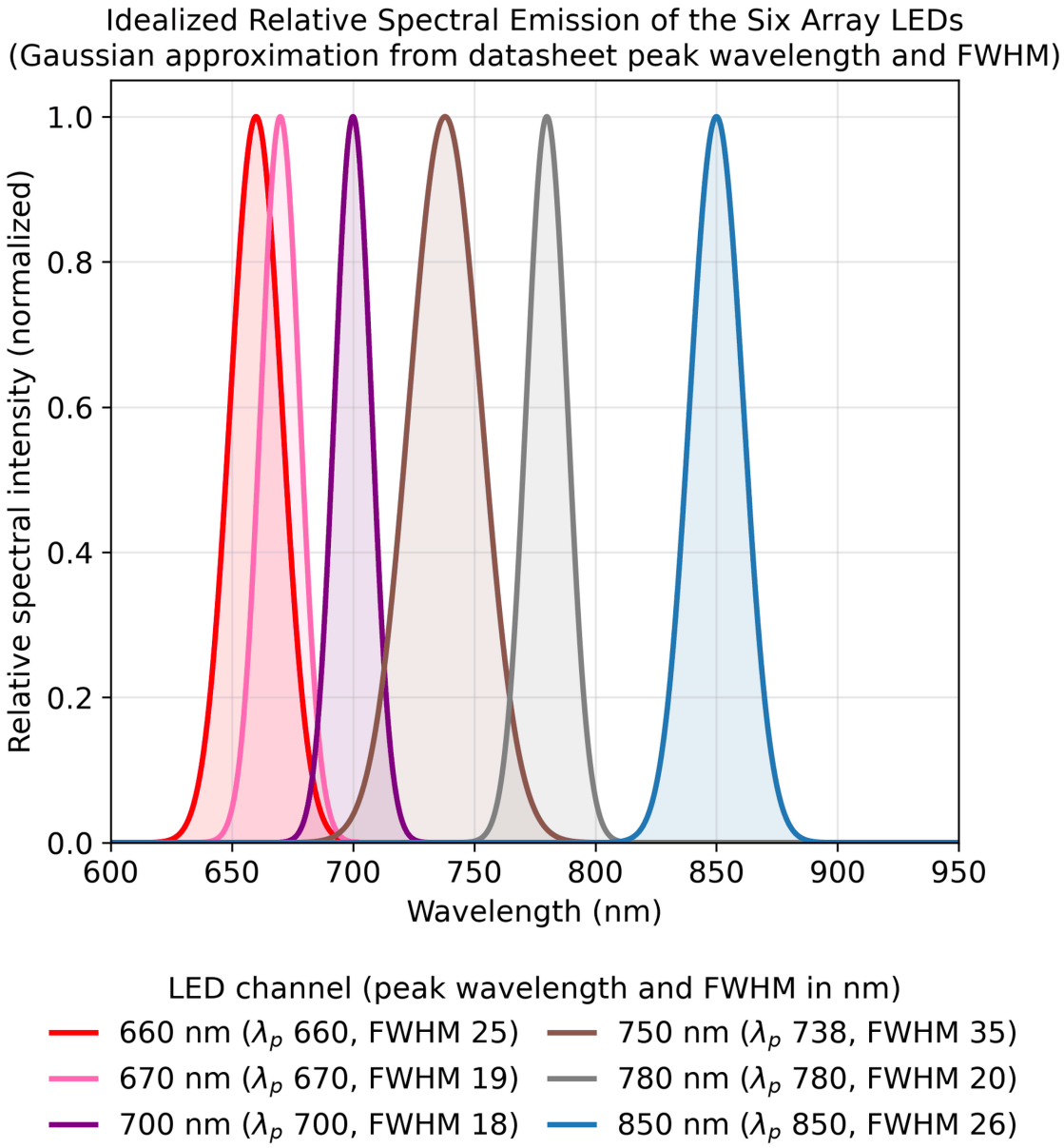


Figure 9. Idealized relative spectral emission of the six array LEDs, reconstructed as Gaussian profiles from the manufacturer-specified peak wavelength. Curves are illustrative of spectral separation and bandwidth, not measured values.

power LED pairs would exceed the available drive-current budget.

2.5.3 Sequential Activation Control

Precise digital control of the LED activation sequence was implemented using the SN74HC164PWR 8-bit serial-in, parallel-out shift register (Texas Instruments). The SN74HC164 accepts serial data through two AND-gated input pins (A and B), clocked into the internal shift register on the rising edge of the clock signal. The parallel output state of the register directly controls the switching of each LED string, with each output bit mapped to a corresponding LED string. A serial-input shift register was chosen specifically to minimize the consumption of general-purpose input/output (GPIO) pins on the STM32 microcontroller, requiring only three signal lines, which are serial data and clock, plus an active-LOW master reset line, rather than six dedicated GPIO outputs; the two unused output bits (Q6 and Q7) provide spare capacity for future expansion. The SN74HC164PWR operates from a supply voltage of 2 V to 6 V, compatible with the 3.3 V logic level of the STM32 and is available in a compact TSSOP-14 package. The sequential activation logic for each measurement stage, following the 780 $\rightarrow$ 670 $\rightarrow$ 700 $\rightarrow$ 850 $\rightarrow$ 660 $\rightarrow$ 750 nm switching order, is summarized in Table 2.

Table 2. LED String Activation Sequence.

Stage	Serial input (Q0…Q7)	LED on (nm)	LEDs off (nm)
1	1000 0000	780	670, 700, 850, 660, 750
2	0100 0000	670	780, 700, 850, 660, 750
3	0010 0000	700	780, 670, 850, 660, 750
4	0001 0000	850	780, 670, 700, 660, 750
5	0000 1000	660	780, 670, 700, 850, 750
6	0000 0100	750	780, 670, 700, 850, 660

2.6 LED Strings Constant-Current Driver Circuit Design

2.6.1 Solid-State Relay Switching Architecture

Each LED series was switched by a dedicated solid-state optical relay (CPC1017NTR, IXYS). The use of optical relays, rather than conventional electromechanical relays or direct MOSFET switching, was motivated by three design considerations: (i) the need for electrical isolation between the low-voltage control logic and the high-current LED power circuit; (ii) the requirement for a compact switching element capable of supporting six independent channels on a single circuit board; and (iii) the elimination of mechanical wear concerns due to repeated switching on and off. The CPC1017NTR solid state relay employs an internal infrared LED optically coupled to a MOSFET output stage, achieving fully isolated switching within a compact 4.089 mm $\times$ 6.096 mm package footprint. This form factor permits all six relay channels to be arranged on the PCB without incurring excessive board-area penalty.

This relay-based switching architecture was necessitated by the voltage and current requirements of the LED series loads. Each LED series, comprising two devices connected in series, requires a combined forward voltage of approximately 9 V at a drive current of 700 mA. This level is incompatible with both the 3.3 V output swing of the STM32 GPIO pins and the 5 V logic outputs of the shift register. The optical relay resolves this incompatibility by interposing an isolation boundary between the low-power control path and the high-power LED loads as shown in Figure 10. The microcontroller (MCU) transmits serial activation data to the shift register, which in turn drives the input LED of the corresponding relay. The relay's MOSFET output stage then connects the selected LED series directly to the 9 V supply rail, providing the necessary forward-voltage required to drive an LED string.

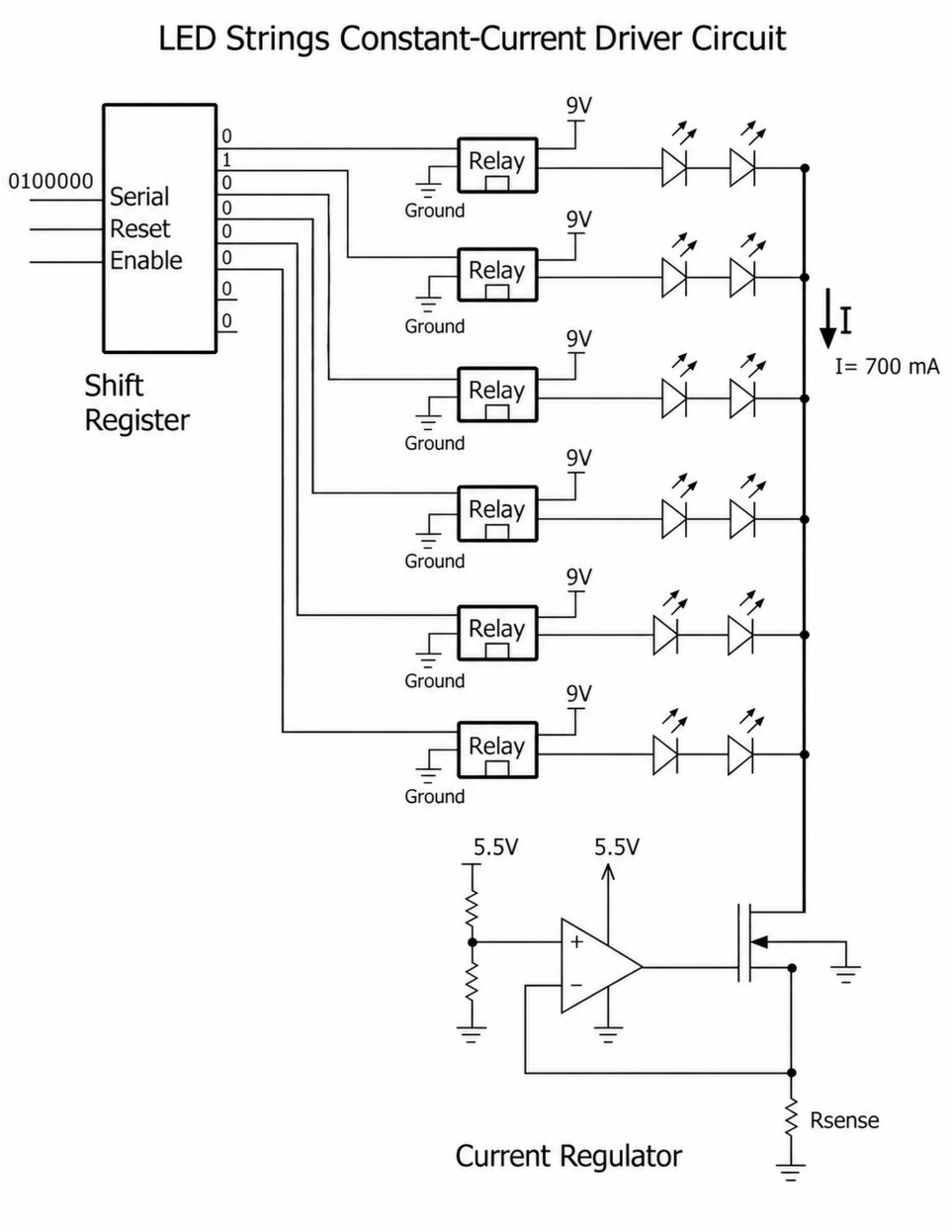


Figure 10. Schematic of the LED strings constant-current driver circuit, showing shift-register-controlled relays for individually switching LED strings and a closed-loop regulator maintaining a constant 700 mA supply current.

2.6.2 Constant-Current Driver

High-power LEDs are inherently current-driven devices. Their luminous flux and spectral output are functions of drive current and applied voltage (Schubert, 2006). Operating an LED from a fixed voltage source without current regulation is impractical, since small variations in supply voltage, LED junction temperature, or device-to-device forward-voltage spread produce large and uncontrolled changes in drive current, leading to optical output instability, premature device degradation, or thermal runaway. For a multispectral reflectance measurement system, in which the accuracy of reflectance values depends directly on the consistency of illumination intensity across measurement cycles, stable and reproducible LED drive current is a fundamental requirement. A dedicated constant-current LED driver circuit was therefore designed to supply a regulated drive current of 700 mA to each active LED series, independent of forward-voltage variation or power supply fluctuation.

2.6.3 Constant-Current Driver Circuit Implementation

The driver circuit is implemented using three principal elements; an ADA4807-1ARJZ-R2 precision operational amplifier, an IRLZ44NPbF logic-level N-channel power MOSFET, and a 1 Ω current-sense resistor (R5). The MOSFET operates as a series current-regulating element in the LED drive path, with its drain connected to the LED string cathode and its source connected to ground through the sense resistor. The operational amplifier closes a negative feedback loop around the MOSFET, continuously comparing the voltage drop across the sense resistor against a fixed reference voltage and adjusting the MOSFET gate drive accordingly to maintain the desired current. An LM317T adjustable linear voltage regulator supplies a stable, regulated anode voltage (9V) to the LED strings, ensuring that the drain-to-source voltage across the MOSFET remains within its safe operating area. Two independent linear voltage regulators are employed, one supplies +9 V for the LED strings, and the other supplies +5.5V for the operational amplifier.

2.6.4 Reference Voltage Derivation

The regulated drive current is set by the reference voltage applied to the non-inverting input (+IN) of the ADA4807-1 operational amplifier, generated by a resistive voltage divider comprising R1 = 1 kΩ and R4 = 162 Ω between the +5.5 V supply rail and ground. The reference voltage is given by Equation 14, shown below.

$$V_{ref} \;=\; V_{\text{sup}}\,(\frac{R4}{R1 \;+\; R4}) \;=\; 5\;V\;(\frac{162}{1000 \;+\; 162}) \;\approx\; 0.697\;V \tag{14}$$

where:

V_{ref} is the reference voltage applied to the non-inverting input of the ADA4807-1 operational amplifier,

V_sup is the supply voltage feeding the divider network, equal to +5 V,

R_1 is the upper voltage divider resistor, 1 kΩ, and

R_4 is the lower voltage divider resistor, 162 Ω.

2.6.5 Regulated LED Current

In closed-loop operation, the ADA4807-1 operational amplifier drives the MOSFET gate so that the voltage at the sense-resistor node equals the 0.697 V reference established above. Because this sense voltage is the product of the LED drive current and the sense resistance, the regulated drive current follows directly from Ohm's law, as given in Equation 15, shown below.

$$I_{LED} \;=\; \frac{V_{ref}}{R_{sense}} \;=\; \frac{0.697\;V}{1\;\Omega} \;\approx\; 700\;mA \tag{15}$$

where:

I_{LED} is the regulated drive current delivered to the active LED series,

V_{ref} is the reference voltage set by the R1–R4 divider, ≈ 0.697 V, and

R_{sense} is the current-sense resistor R5, 1 Ω.

The IRLZ44NPbF power MOSFET was selected because it has a high gate threshold voltage of 2.0 V, high 47 A continuous drain-current rating, and 55 V breakdown voltage, providing substantial margin above both the 700 mA drive current and the maximum LED string anode voltage.

2.7 Signal Acquisition, ADC Configuration, and Timing

2.7.1 Oversampling Ratio Selection and Resolution-Throughput Trade-off

The number of ADC acquisitions within each measurement window is governed by the hardware oversampling ratio (OSR) configured in the STM32H725 ADC. In hardware oversampling mode, the ADC performs OSR successive conversions, accumulates them in hardware, and right-shifts the accumulated result to produce one oversampled output sample. The duration of one oversampled acquisition therefore scales linearly with OSR. Equation 16 shows how the total acquisition time with oversampling is calculated.

For a 16-bit ADC conversion at a 96 MHz ADC kernel clock with 8.5 sampling cycles and 16 conversion cycles, a single-conversion time is as follows.

$$t_{conv} = \frac{8.5 + 16}{96 \times 10^6} \approx 255\ ns$$

Hence, the total acquisition time is;

$$\begin{aligned} t_{acq} &= OSR \times t_{conv} \\ t_{acq} &= 256 \times 255\ ns \approx 65.3\ \mu s \end{aligned} \tag{16}$$

The theoretical improvement in effective number of bits (ENOB) obtained from oversampling an ADC is given by the oversampling theorem (Candy & Temes, 1992; STMicroelectronics, AN2668, 2008). This relationship assumes that the ADC input is accompanied by spectrally white, uncorrelated noise sufficient to dither the quantizer across adjacent levels.

$$\Delta ENOB = \frac{1}{2} \times log2(OSR) \tag{17}$$

The effective resolution improvement from 256× hardware oversampling is as follows.

$$\Delta ENOB \ = \frac{1}{2} \ \times \log_2 (256) \ = \frac{1}{2} \ \times \ 8 \ = \ 4 \ bits$$

The effective ADC resolution per oversampled acquisition is 16 + 4 = 20 bits.

Equation 17, formalizes the central resolution-throughput trade-off. Doubling the OSR adds 0.5 bits of effective resolution but reduces the number of acquisitions collectible within a fixed window to half. Quadrupling OSR adds 1 bit but quarters the count. The net signal-to-noise improvement therefore exhibits diminishing returns as OSR increases, because fewer independent readings are available for post-acquisition averaging.

The STM32H725VGT6 ADC supports OSR values of 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, and 1024. The selection of OSR as 256 was made on the following basis. Applying Equation 17, an OSR of 256 yields ΔENOB = 4 bits, raising the effective per-acquisition resolution from 16-bit to 20-bit. The corresponding acquisition duration of approximately 65.3 µs permits 766 LED-ON and 153 LED-OFF acquisitions to be collected within the 50 ms and 10 ms windows respectively. At OSR of 512 and above, the per-reading resolution gain of 0.5 additional bits is insufficient to justify the halving of the acquisition count, because the noise floor of the SiPM detector and transimpedance amplifier, rather than ADC quantization noise, becomes the dominant limitation on measurement accuracy at high oversampling ratios. OSR of 256, therefore, represents the practical boundary beyond which further oversampling yields diminishing returns, and it is consistent with the ADC performance specifications of the STM32H725VGT6 (STMicroelectronics, 2022). Table 3 quantifies this trade-off across the full range of supported OSR values, and Figure 11 illustrates it graphically.

Table 3. Oversampling ratio trade-off for a 50 ms acquisition window (Retrieved from the STM32 ADC datasheet).

OSR	Time per acquisition	Readings per 50 ms	Effective resolution
16×	≈4 µs	≈12,500	18-bit
32×	≈8 µs	≈6,250	18.5-bit
64×	≈16 µs	≈3,125	19-bit
128×	≈33 µs	≈1,515	19.5-bit
256× (selected)	≈65 µs	≈766	20-bit
512×	≈130 µs	≈384	20.5-bit
1024×	≈261 µs	≈192	21-bit

2.7.2 Total Signal Acquisitions

The number of LED-ON acquisitions collected during the 50 ms illumination interval is as follows.

$$N_{LED} = \frac{50,000\ \mu s}{65.3\ \mu s} \approx 766\ acquisitions$$

The number of dark acquisitions collected during the 10 ms led off interval is as follows.

$$N_{DARK} = \frac{10,000\ \mu s}{65.3\ \mu s} \approx 153\ acquisitions$$

The total number of oversampled acquisitions collected during one 60 ms band cycle is as follows.

$$N_{band} = 766 + 153 = 919\ acquisitions$$

For six sequential wavelength bands, the total number of acquisitions in one full spectral scan is as follows.

$$N_{total} = 6\ \times 919 = 5,514\ acquisitions\ per\ scan$$

where 6 represents the number of wavelength bands.

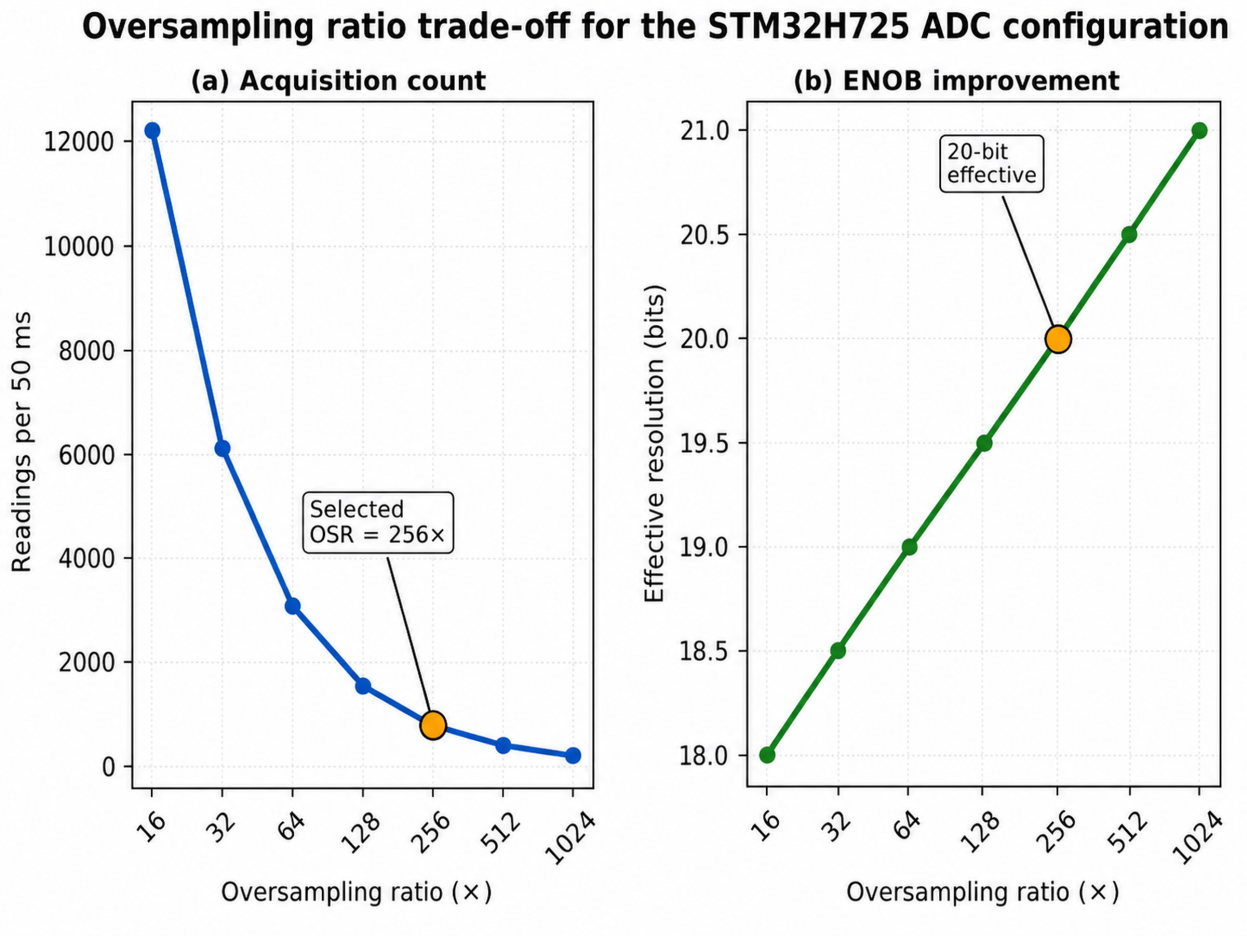


Figure 11. Oversampling ratio trade-off analysis (Retrieved from the STM32 ADC datasheet).

2.7.3 Timing Summary

The complete timing parameters are summarized in Table 4. The full measurement cycle consists of six wavelength bands, each measured using a 50 ms LED-ON window followed by a 10 ms dark window (LED-OFF), resulting in a total spectral scan duration of 360 ms.

Table 4. ADC acquisition timing parameters for direct DC detection with background subtraction.

Parameter	**Value**
LED wavelength sequence	780, 670, 700, 850, 660, 750 nm
LED drive current	700 mA DC, constant
LED-ON window per band	50 ms
Dark window per band	10 ms
Total window per band	60 ms
Signal detection mode	Direct DC detection
Ambient rejection method	Background subtraction: V_corrected= $\bar{X}$_LED − $\bar{X}$_DARK
ADC resolution	16-bit
ADC kernel clock	96 MHz
Hardware oversampling ratio	256×
Effective resolution	20-bit
Duration per oversampled acquisition	≈65.3 µs
LED-ON acquisitions per band	≈766
Dark acquisitions per band	≈153
Total acquisitions per band	919
Number of wavelength bands	6
Total spectral scan duration	6 × 60 ms = 360 ms
Total acquisitions per full scan	6 × 919 = 5,514

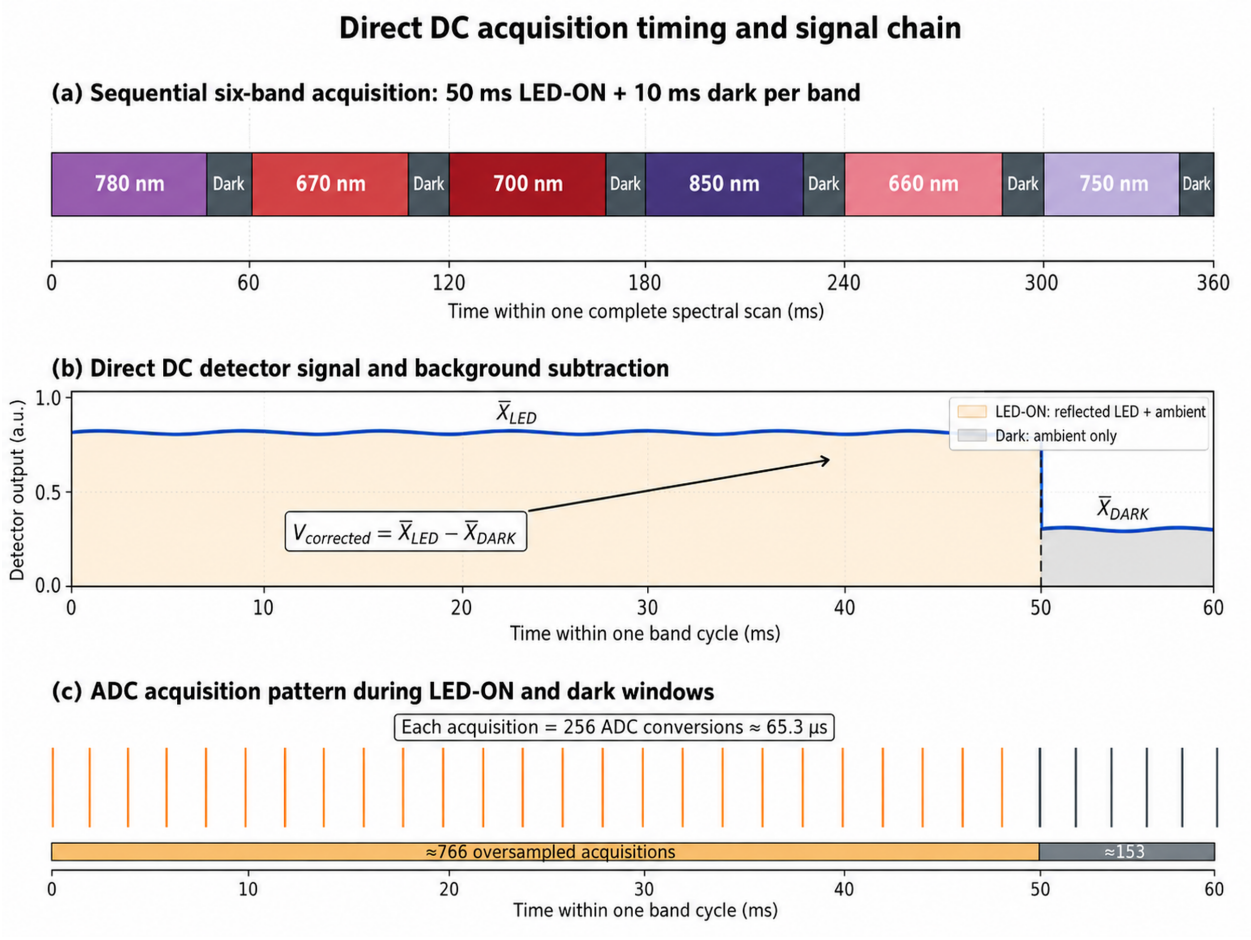


Figure 12. Acquisition timing diagram.

2.8 Micro-Controller Units

The embedded processing architecture of the sensor system was partitioned across two dedicated microcontroller units (MCUs), each selected to meet the specific computational, peripheral, and timing requirements of its assigned subsystem. This dual-MCU architecture was adopted in preference to a single-MCU solution for two principal reasons. First, the LED array control and signal acquisition subsystem demands deterministic, real-time operation at high switching frequencies. These are requirements that are best served by a high-performance MCU with dedicated real-time peripherals and low-latency memory access. Second, the display and inference subsystem requires hardware-accelerated graphics rendering, support for external memory expansion to accommodate neural network model parameters and display frame buffers, and sufficient computational headroom for embedded machine learning inference. These capabilities are better suited to a display-oriented MCU with integrated graphics hardware. Distributing these responsibilities across two specialized processors avoids the architectural compromises that would arise from attempting to fulfill both roles with a single device and enables concurrent operation of the acquisition and display pipelines without scheduling conflicts.

STM32H725VGT6 (STMicroelectronics) was selected as the LED array control and signal acquisition MCU. This device is based on the 32-bit Arm® Cortex®-M7 core, operating at clock frequencies up to 550 MHz, and provides the processing throughput necessary for high-frequency signal generation and real-time data acquisition. Zero-wait-state execution is supported through integrated 32 KB instruction and 32 KB data caches, minimizing memory access latency during time-critical operations. The device incorporates 564 KB of SRAM, including dedicated Tightly Coupled Memory (TCM) RAM blocks that provide deterministic, single-cycle access latency.

Several onboard peripherals of the STM32H725VGT6 are utilized within the system. Two independent SPI peripherals are employed. The first interfaces with the SN74HC164PWR shift register to generate the sequential LED activation control signals, and the second provides the inter-MCU communication link to the display processing unit. The SDMMC1

SD card interface connects to an onboard SD card for persistent logging of raw spectral reflectance vectors, acquisition metadata, and inference outputs. The device operates from a 3.3 V supply and is clocked by an external 25 MHz crystal oscillator, with decoupling capacitors distributed across the power rails to suppress high-frequency switching noise during signal generation and ADC acquisition.

The STM32F746ZET6 (STMicroelectronics) was selected as the display processing and inference MCU. This device is also based on the Arm® Cortex®-M7 core, operating at up to 216 MHz, and was specifically chosen for its integrated LCD-TFT display controller and hardware graphics acceleration capabilities. The built-in Chrom-ART Accelerator (DMA2D) offloads pixel data transfers and framebuffer fill operations from the CPU, enabling smooth display rendering at the required frame rate without monopolising the processor for graphics tasks. The LCD-TFT controller supports resolutions up to XGA and generates the pixel clock, synchronization, and data signals required to drive the capacitive touchscreen LCD directly. Critically, the STM32F746ZET6(Display MCU) was designated as the SPI master in the inter-MCU communication link, with the STM32H725VGT6 (LED control MCU) operating as the SPI slave. This assignment reflects the inference-driven architecture of the system. The display MCU initiates acquisition cycles, receives the spectral reflectance data transmitted by the LED control MCU, executes the embedded nitrogen inference model, and manages the presentation of results to the user. Designating the inference-capable processor as SPI master ensures that data transfer is initiated in synchrony with the inference pipeline, avoiding the need for complex handshaking protocols.

The on-chip flash and SRAM of the STM32F746ZET6 (Display MCU), while adequate for general embedded applications, provide limited capacity for a graphics framebuffer together with future model weights and computation buffers. Accordingly, external memory resources are integrated across two separate interfaces. An AS4C32M8SA-7TCN DRAM device (256 Mbit) is connected through the Flexible Memory Controller (FMC) as external SDRAM, providing high-bandwidth temporary storage for the active LCD framebuffer and other working arrays. An MT25QL128ABA1EW9 NOR flash RAM (128 Mbit) is connected over the Quad-SPI (QSPI) interface and provides non-volatile storage for firmware assets,

configuration parameters, and future model or calibration data. The QSPI peripheral can operate in memory-mapped mode, allowing stored parameters to be accessed efficiently when required. The STM32F746ZET6 (Display MCU) is clocked by a 25 MHz high-speed external oscillator for core and peripheral operation, and a 32.768 kHz low-speed oscillator drives the real-time clock for acquisition timestamping.

Inter-MCU communication is implemented over a dedicated full-duplex SPI link. SPI was selected over UART or I2C because of its higher achievable data rate and deterministic transfer timing for multi-band spectral data. Following completion of a spectral acquisition cycle, the STM32H725VGT6 (LED control MCU) packages the digitized measurements from all six bands into a structured data frame and transmits them to the STM32F746ZET6 (Display MCU). The display MCU receives the spectral vector for presentation, storage coordination, and future model processing. The SPI clock rate is configured so that a complete six-band frame can be transferred within the inter-scan interval.

Both MCUs incorporate standard boot configuration and debug infrastructure. The BOOT0 pin of each device is pulled low through a 10 kΩ resistor, configuring both processors to boot exclusively from internal flash memory upon power-up, preventing inadvertent entry into system bootloader mode during normal operation. Serial Wire Debug (SWD) interfaces are exposed on both MCUs for firmware development, in-circuit debugging, and firmware updates. Reset circuitry ensures clean power-on initialization of both devices and allows independent reset of each subsystem without disrupting the other. Together, these features support a robust, modular embedded architecture in which the LED acquisition pipeline and the inference-display pipeline operate concurrently and in a synchronized manner, within the real-time constraints of the in-field nitrogen measurement application. The complete firmware source code for both MCUs is listed in Appendix D.

2.9 Optical System Design

The optical subsystem of the sensor was designed to fulfil two simultaneous objectives. It delivers controlled illumination from the LED array onto the leaf measurement surface with reduced beam divergence, and it efficiently collects the reflected light from the irradiated leaf (which contains the diffuse component together with any specular reflection from the leaf surface that falls within the collection aperture) and focuses it onto the active area of the SiPM detector. These objectives are served by a dual-lens, coaxial architecture in which both lenses share a common optical axis aligned perpendicular to the leaf surface. The complete optical arrangement is illustrated in the cross-sectional schematic presented in Figure 13. The two LEDs drawn in Figure 13 represent one wavelength pair from Figure 8, sectioned through the front plane and viewed from the side. It contains one LED on the inner ring (Ø 50 mm, 25 mm from the optical axis) and its partner lies diametrically opposite on the outer ring (Ø 65 mm, 32.5 mm from the axis), so that their beams converge on the measurement spot. Only the pair that is active at a given time is shown, and the two LEDs are drawn at equal distances from the axis for simplicity.

The outer optical element is a plano-convex lens with a diameter of 75 mm and a focal length of 150 mm, housed in a 3D-printed circular mount at the front aperture of the sensor head. The lens is oriented with its convex surface facing the target leaf and its planar surface facing the interior of the optical barrel. This orientation distributes the refraction across both lens surfaces, minimising spherical aberration in the transmitted beam. The outer lens mount and front aperture assembly are shown in Figure 14.

The outer lens serves a dual optical function. On the outward illumination path, it acts as a beam-shaping element. Each LED in the ring array emits light with a characteristic half-angle divergence, and the lens intercepts this divergent emission and partially collimates it, concentrating the optical power toward the leaf measurement area. The sensor is designed for a maximum standoff distance of 80 mm (8cm). Since this working distance is less than the lens focal length of 150 mm, the leaf lies inside the front focal plane, and the lens operates in a divergence-reduction rather than focusing mode. The output beam is partially collimated

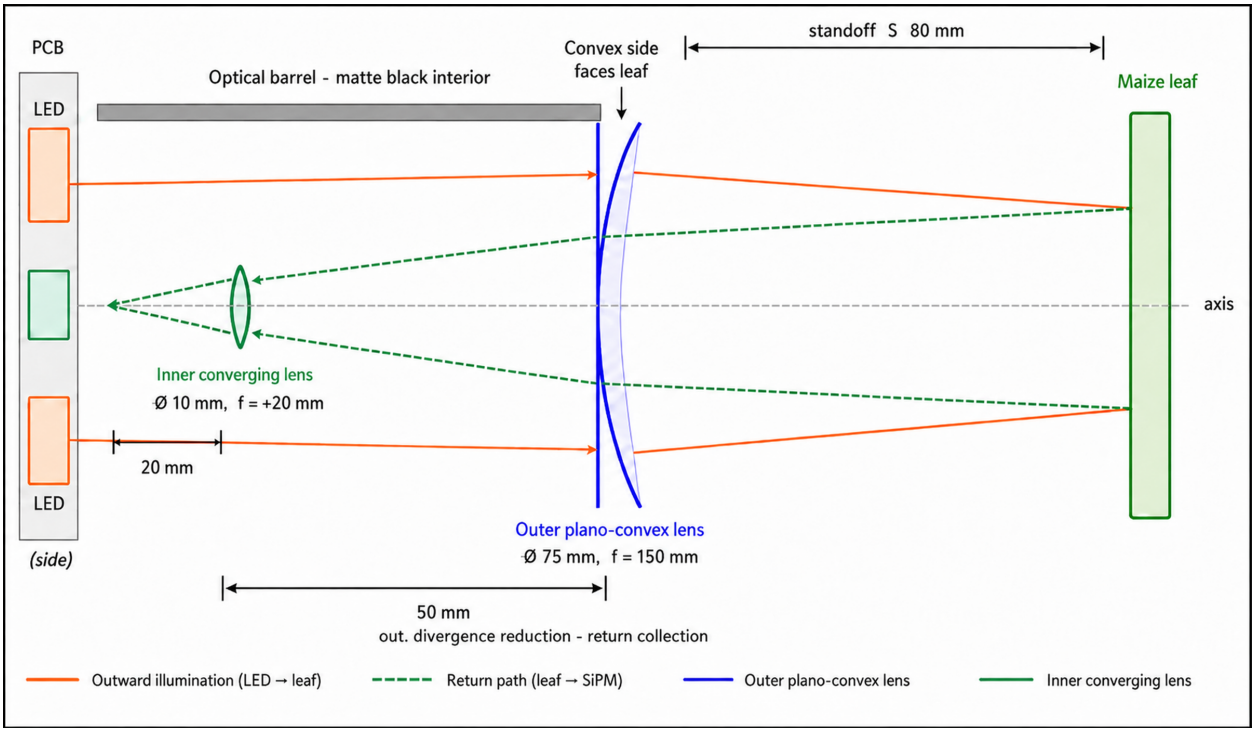


Figure 13. Cross-sectional schematic of the dual-lens optical system. Outer lens: plano-convex, diameter(Ø) 75 mm, focal length (f) = 150 mm. Inner lens: converging, diameter(Ø) 10 mm, focal length (f) = 20 mm. (Sectioned through the front plane).

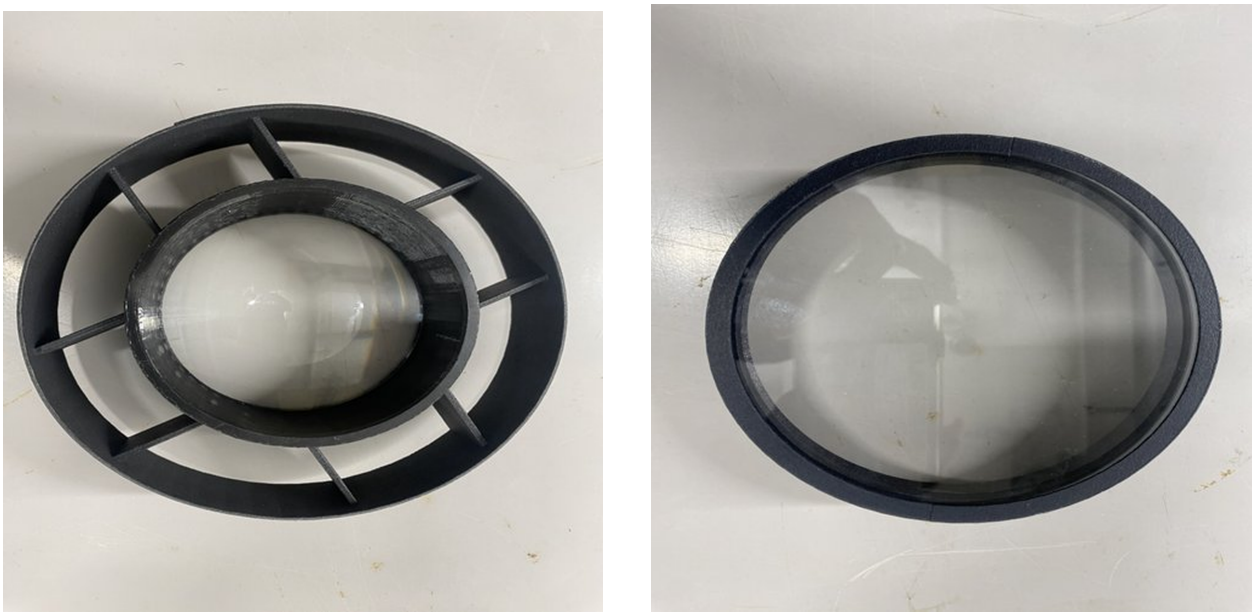


Figure 14. The Left image is the Outer plano-convex lens (Ø 75 mm) in its 3D-printed spoke mount. Right, optical window for protecting inner lens and outer lens from dust and scratch.

but does not converge to a real focal point within the working range. On the return path of the light signal, the outer lens performs the complementary function of collecting a fraction of the diffuse reflected light from the leaf and directs it into the optical barrel toward the detector (SIPM) assembly.

The inner optical element is a convex converging lens with a diameter of 10 mm and a focal length of 20 mm, mounted in a threaded cylindrical barrel that attaches directly to the front face of the detector PCB, positioning the lens approximately 20 mm in front of the SiPM active area. The assembled inner lens barrel and detector PCB are shown in Figure 15. The barrel is separated from the outer lens by approximately 50 mm as shown in Figure 13, forming the interior of the optical enclosure. The converging lens collects the partially collimated backscattered light that has passed through the outer lens and propagated through the barrel, and focuses it onto the SiPM detector. The image distance q for this configuration is derived from the thin lens equation (Hecht, 2017), shown below.

$$\frac{1}{p} + \frac{1}{q} = \frac{1}{f} \tag{18}$$

where:

p is the object distance, measured from the leaf to the inner lens (approximately 50 mm barrel + 80 mm standoff = 130 mm),

q is the image distance, measured from the lens to the SiPM (20 mm), and

f is the focal length of the inner converging lens (20 mm).

Since the object distance p = 130 mm is substantially greater than the focal length f = 20 mm, the approximation $1/p \approx 0$ holds, and Equation 18 reduces to the expression shown below.

$$q \approx f = 20\ mm \tag{19}$$

This confirms that the backscattered light, originating at a large effective object distance relative to the inner lens focal length, is brought to focus at approximately the focal plane of the inner lens. This location coincides with the SiPM active area positioned 20 mm behind it. The 10 mm lens aperture provides a collection solid angle sufficient to intercept a meaningful fraction of the light directed into the barrel by the outer lens, while its compact form factor is compatible with the central mounting geometry within the detector PCB assembly.

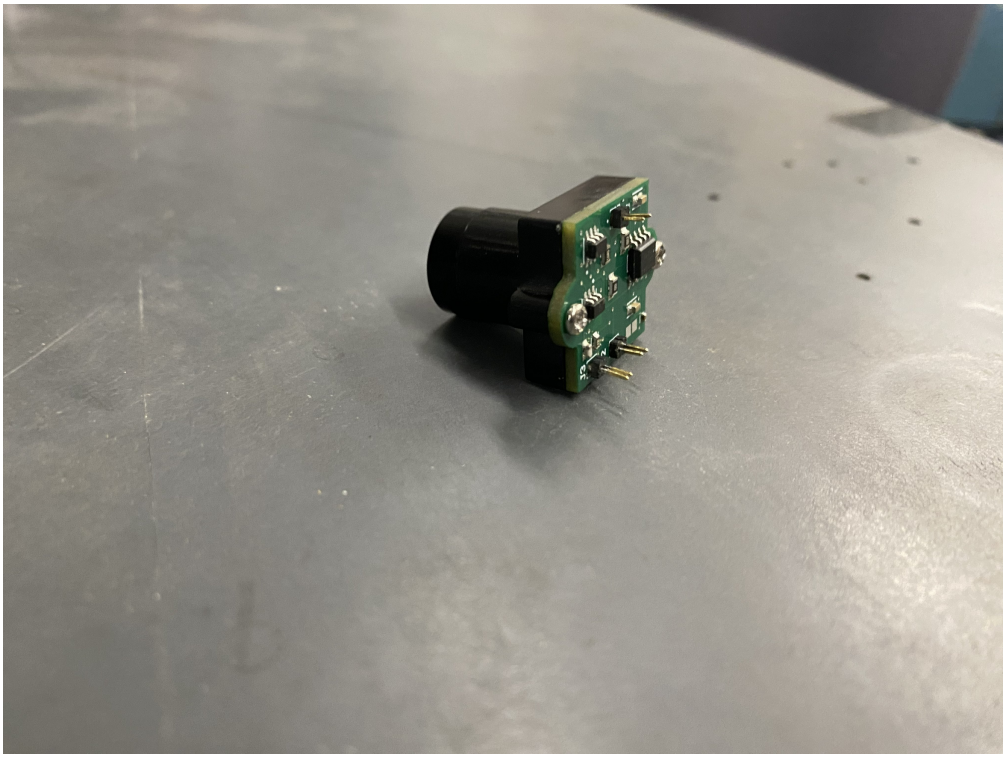


Figure 15. Inner converging lens barrel (Ø 10 mm, f = 20 mm) mounted on the SIPM detector PCB, positioning the lens 20 mm in front of the SiPM.

The interior surfaces of the optical barrel enclosure were coated with matte black paint to suppress internal stray light and minimize multiple-reflection artefacts. This high-absorption surface treatment reduces both specular and diffuse internal reflection, ensuring that only photons propagating along the intended optical paths, either the direct LED illumination beam or the backscattered collection path, reach the SiPM detector. Without this treatment, multiple reflected stray light from the barrel walls would superimpose a spectrally non-selective background signal on the detector output, degrading the signal-to-noise ratio of the reflectance measurement across all spectral bands. The optical barrel assembly em-

ploys threaded connections for both the outer lens mount and the inner lens barrel, enabling precise axial alignment of the optical centers and facilitating disassembly for cleaning, reconfiguration, or lens replacement. An optical window integrated into the front aperture protects the outer plano-convex lens from dust, moisture, and mechanical contact during field operation without introducing significant optical loss or aberration in the illumination or collection paths.

2.10 Device Overall Internal and External Structure

Overall, the device was designed to function as a handheld meter for the initial prototyping stage, comparable in form factor to existing commercial nitrogen meters, such as the handheld GreenSeeker (Trimble; two bands: red at 660 nm and near-infrared at 780 nm), the Crop Circle ACS-470 (Holland Scientific; three bands, commonly configured as red at 670 nm, red-edge at 730 nm, and near-infrared at 760 nm; Cao et al., 2013), and the SPAD-502 chlorophyll meter (Konica Minolta; two wavelengths: red at 650 nm and near-infrared at 940 nm, measured in transmittance rather than reflectance). These instruments use two or three wavelengths, compared with the six wavelengths used in the sensor developed here. To achieve this level of portability and structural integration, a custom enclosure was developed specifically to house the sensor components while maintaining usability and durability. The enclosure was designed with the LCD display positioned at the top as shown in Figure 17, providing users with clear and immediate access to output readings during operation.

This compact integration resulted in a total device weight of 1.2 Kg, making it light enough for extended handheld use without causing fatigue. Additionally, the enclosure was designed with ergonomics in mind, ensuring a form that comfortably fits within the user's hand. The outer casing features contours and surface treatments to reduce slippage and enhance grip, thereby enabling secure and stable operation even under field conditions.

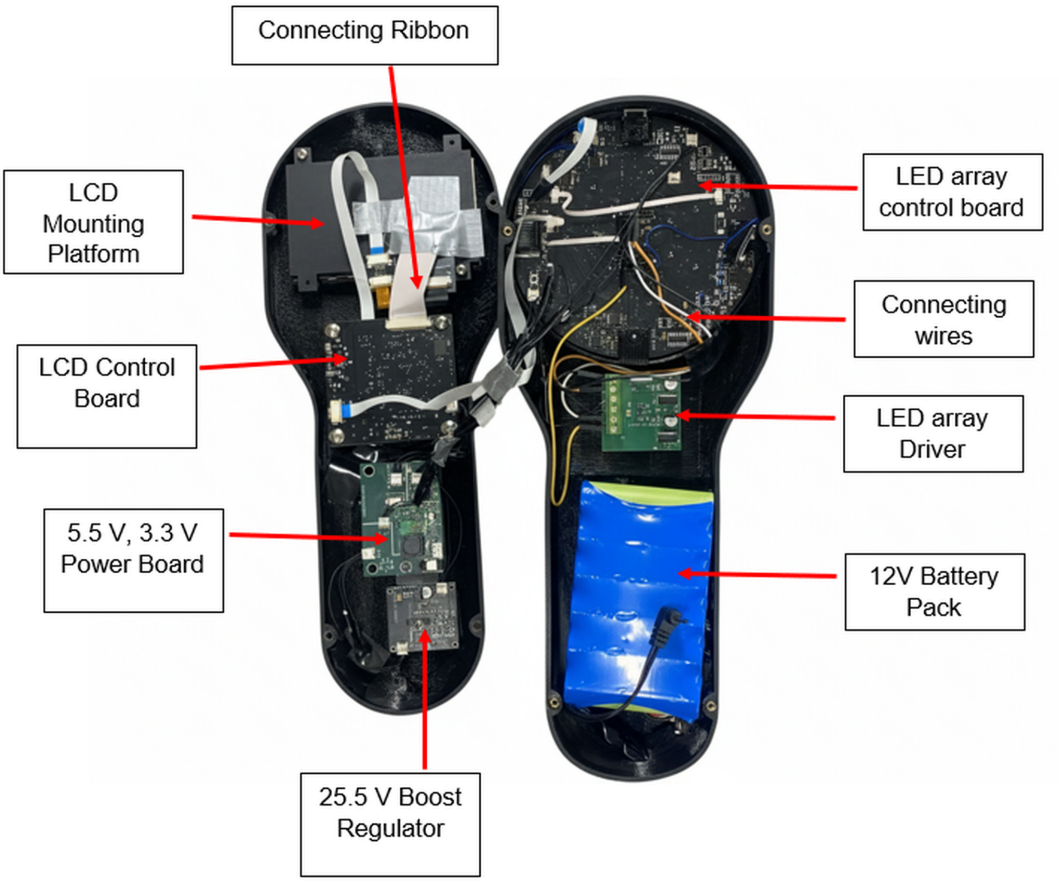


Figure 16. Overall Sensor with Connections and Circuit Board Modules.



Figure 17. Sensor with LCD Display ON.

Chapter 3

Experimental Design - Materials and Method

3.1 Field Experimental Setup

The field experiment was conducted as part of the FogAg Project at the Kansas State University research field (39.211703, −96.595392). The experimental layout comprised 54 maize plots arranged in a three-block, three-planting-date design, with six nitrogen treatments randomized within each block. Each plot measured 25 ft in length by eight rows in width; 5-ft alleys separated adjacent plots, and 10-ft alleys separated the three planting-date sections. The maize hybrid P13050 was planted at a target population of 28,000 plants acre^{-1}. A summary of the experimental factors is provided in Table 5.

Table 5. Summary of field experimental factors.

Factor	Description
Field location	39.211703, −96.595392 (Manhattan, KS). these are the latitude and longitude of the field corresponding to the black mark at the corner in Figure 18
Crop / hybrid	Maize, P13050
Target plant population	28,000 plants acre^{-1}
Total number of plots	54 plots (18 per planting date)
Plot size	25 ft × 8 rows; 5-ft alleys between plots; 10-ft alleys between planting-date sections
Experimental blocks	3 blocks per planting date
Planting date targets	April 20, May 18, and June 15, 2026
Nitrogen treatments	0, 50, 100, 150, 200, and 200 split lb N acre^{-1} (six levels)
Spectral replication	two leaf spectral replicates per plot per growth stage
Ground-truth measurement	Leaf nitrate-nitrogen extraction from laboratory-analyzed samples

Six nitrogen treatments were evaluated: 0, 50, 100, 150, 200, and 200 lb N acre^{-1} split application.‘ Nitrogen treatment was applied as urea (0-0-46; 46% N). The nitrogen treatments and application rates were established by Dr. Rachel Cott (Department of Agronomy, Kansas State University) as part of a nitrogen-rate field trial. Dr. Rachel Cott selected a range of N levels to include 0 Ib N acre^{-1} and beyond sufficient (200 lb N acre^{-1}) to ensure a broad range of nitrogen levels present in the soil. There were no specific calculations included in these levels. The four lower rates and the single 200 lb N acre^{-1} treatment was applied entirely before planting, while the split treatment consisted of 150 lb N acre^{-1} applied before planting and an additional 50 lb N acre^{-1} applied at the V5 growth stage. Planting date targets of April 20, May 18, and June 15, 2026, represented early, mid-season, and late planting windows, respectively; actual planting was targeted within one week of each nominal date. The nitrogen treatment structure is described in Table 6.

Table 6. Nitrogen treatment structure.

Treatment label	**Treatment application rate (lb N acre^{-1})**	**Application timing**
0	0	No fertilizer N applied; relies on base soil N
50	50	Single pre-plant application
100	100	Single pre-plant application
150	150	Single pre-plant application
200	200	Single pre-plant application
200 split	200	150 lb N acre^{-1} pre-plant + 50 lb N acre^{-1} at V5

3.2 Method Used for Data Collection

Spectral data collection was planned at three key maize growth stages: V10, R1, R3. These growth stages were selected because, nitrogen stress first becomes visible at V10, an important management window for farmers (Abendroth et al., 2011), and becomes more pronounced at R1 and R3, which is useful for modelling nitrate-N. Moreover, the leaf area at V10, R1 and R3 is proper to fit into the field of view (FOV) of the sensor. Target sampling dates for each growth stage by planting date are listed in Table 7. Within each plot, two replicate leaf spectral measurements were collected at each sampling event from row 2. To capture within-plot variability, one from each of two different plants growing within that plot was measured. Leaf spectral data were acquired under field conditions using the handheld active multispectral sensing system described in Chapter 2. Corresponding spectral measurements were also collected on the same leaf material under controlled laboratory conditions (dark enclosure) using the same handheld sensor to characterize the relationship between field (ambient light illumination) and laboratory sensor responses (no ambient light illumination).

Table 7. Target growth stages and nominal sampling windows by planting date.

Growth stage	**Description**	**Planting date 1**	**Planting date 2**
V10	Ten collared leaves	June 15	June 29
R1	Flowering / silk stage	July 6	July 20
R3	Grain development / milk stage	July 27	August 10

For ground-truth nitrogen assessment, leaf samples corresponding to the spectral measurement locations were collected and transported to the laboratory for nitrate-nitrogen extraction (NO_3). The extracted nitrate-N values served as reference measurements for evaluating the relationship between multispectral reflectance features and plant nitrogen status. This paired field–laboratory design enabled direct comparison of sensor-derived spectral re-

sponses with chemically measured nitrate-N ground truth. The data streams and their respective collection approaches are summarized in Table 8.

Table 8. Spectral and ground-truth measurement plan.

Data stream	Collection approach	Purpose
Field leaf spectral data (with ambient light illumination)	Two replicate leaf spectral readings per plot per nitrogen treatment at each target growth stage	Captures in-field optical response under actual environmental conditions
Laboratory leaf spectral data (no ambient light illumination)	Spectral readings from corresponding leaf material collected under controlled laboratory conditions	Provides a controlled baseline for comparison against field spectral response
Leaf nitrate-N extraction	Leaf samples collected from spectral measurement locations and analyzed in the laboratory	Serves as the ground-truth reference for evaluating sensor-derived spectral features against chemically measured nitrogen status

Table 9. Rows configuration for each 8-row plot.

Row / location	Assigned use	Notes
Rows 1 and 8	Buffer rows	Maintained untouched for the full growing season to minimize edge effects
Rows 2 and 7	Biomass sampling rows	5-ft biomass sampling sections with 2.5-ft buffer zones. Leaf samples are taken from this section then dried and ground for laboratory N analysis
Rows 3 and 6	Representative buffer rows	Maintained untouched to preserve representative crop conditions for adjacent data collection areas
Rows 4 and 5	Yield sampling rows	Central yield area; untouched during the growing season and machine-harvested with a combine havester following R6 and appropriate dry-down

The complete plot treatment allocation, including planting date, block assignment, and nitrogen treatment for all 54 plots, is provided in Table 9 and Figure 18.

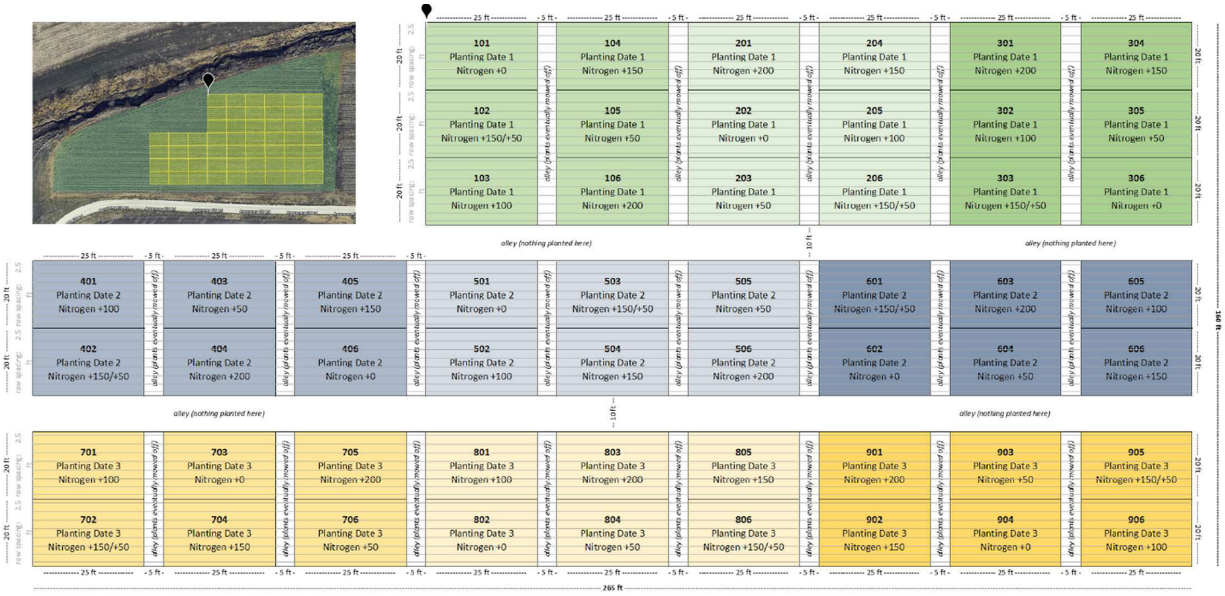


Figure 18. Field map with treatments listed. Plots are colored by planting date. Black pin at the top left corner corresponds to location on map.

3.3 Methods Used for Data Analysis

Every relative-reflectance measurement in this thesis, whether of a maize leaf or of the reference target itself, was computed from the same detected-optical-power conversion (Equation 20). The diffuse white reference used throughout was a 99% Spectralon target, shown in Figure 19, with its manufacturer-specified diffuse reflectance spectral profile shown in Figure 20.

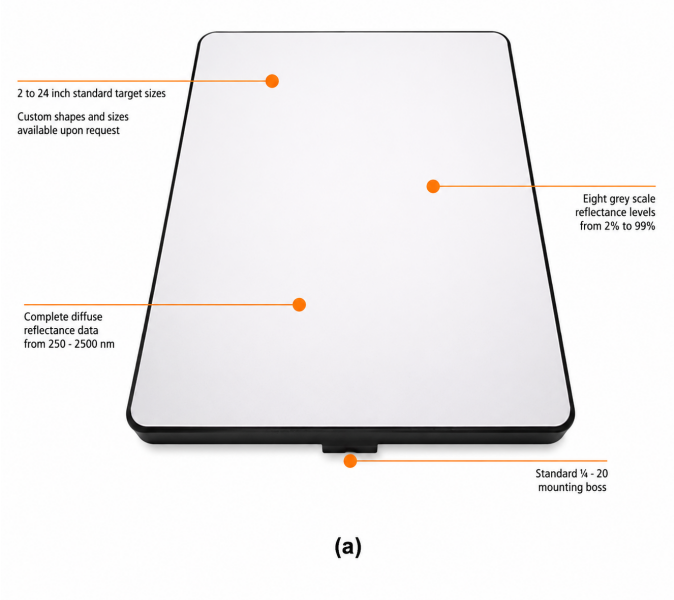


Figure 19. White Spectralon 99 % Reflectance Target (Retrieved from the Labsphere Spectralon target datasheet).

$$P_{det,\lambda} = \frac{V_{corrected,\lambda} \times h \times c}{A_{post} \times R_{TIA} \times PDE_{\lambda} \times G \times q \times \lambda} \tag{20}$$

where:

$P_{det,\lambda}$ is the optical power reaching the SiPM active area after LED illumination of the target, reflection from it, and propagation through the sensor's collection optics,

$V_{corrected,\lambda}$ is the background-corrected digital voltage reported by the firmware at wavelength λ (the LED-on reading with the LED-off reading subtracted),

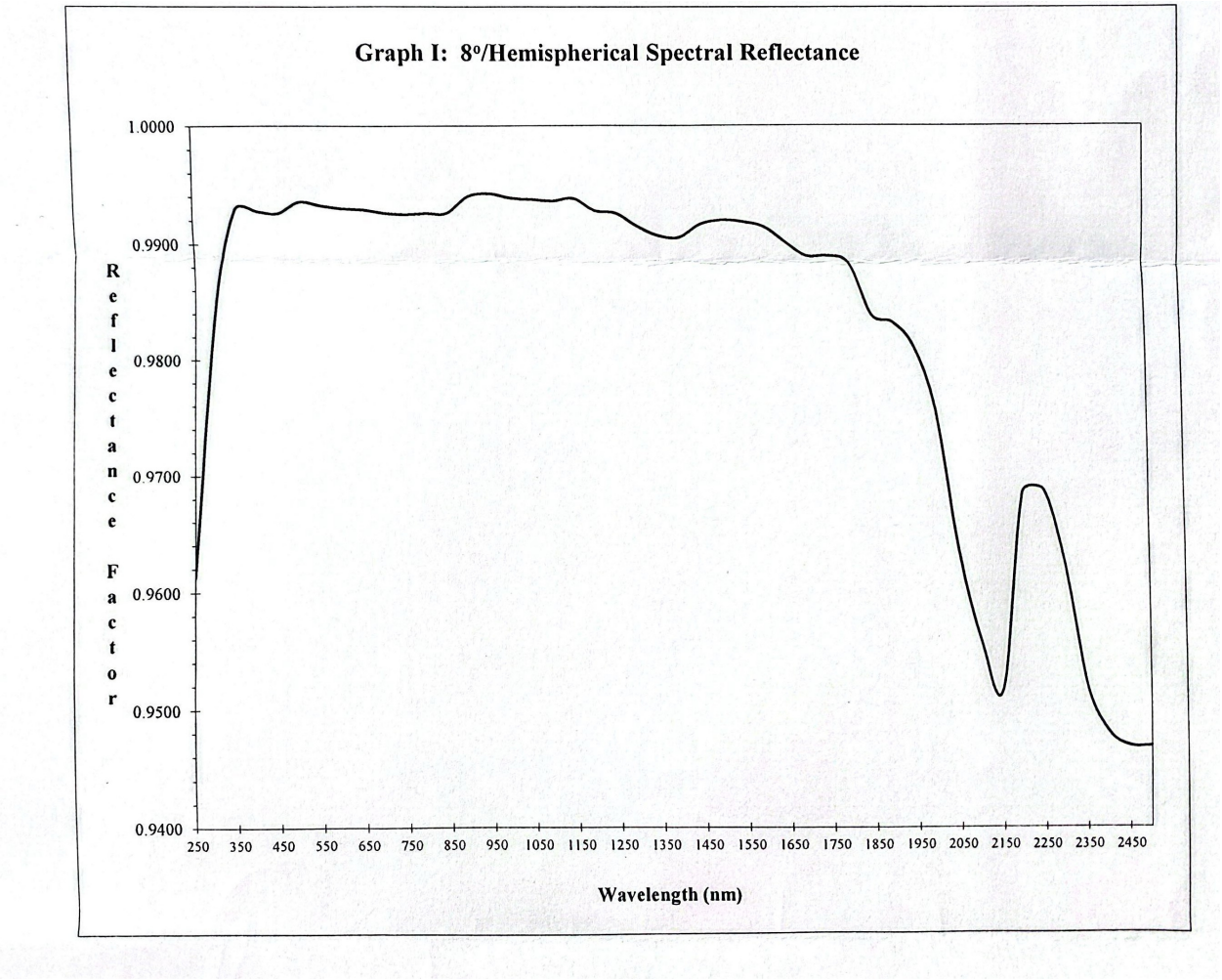


Figure 20. Spectralon 99% diffuse Reflectance Spectral Profile (Retrieved from the Labsphere calibration datasheet supplied with the Spectralon target).

A_{post} is the fixed gain applied downstream of the transimpedance amplifier (1.1 V/V at every wavelength),

R_{TIA} is the fixed transimpedance feedback resistor common to all wavelength measurements (330 Ω),

PDE_λ is the SiPM's photon detection efficiency at wavelength λ, read from the manufacturer's PDE curve,

G is the SiPM's avalanche multiplication gain,

q is the elementary charge (1.6×10^{-19} C),

h is Planck's constant (6.626×10^{-34} J·s),

c is the speed of light (3×10^8 m/s), and

λ is the wavelength; together, h, c, and λ give the photon energy hc/λ.

This expression is obtained by recovering $V_{corrected,\lambda}$ from the transimpedance-amplifier stage, converting it to SiPM photocurrent, and combining it with the SiPM's wavelength-dependent effective responsivity; the full step-by-step derivation is provided in Appendix A.

Prior to statistical analysis, each raw firmware voltage was converted to a detected optical power at the corresponding wavelength using Equation 20, then to relative reflectance using Equation 21, computed as the detected optical power measured on the leaf divided by the detected optical power measured on a 99% Spectralon reference target at the same wavelength. This relative reflectance value, not the raw optical power, was the input for every wavelength-level analysis in this paper. The Python source code used for the data processing and analysis is listed in Appendix E.

$$R_{rel,\lambda} = \frac{P_{det,leaf,\lambda}}{P_{det,Spectralon,\lambda}} \tag{21}$$

where:

$R_{rel,\lambda}$ is the relative reflectance at wavelength λ (dimensionless), the quantity used in every wavelength-level analysis in this thesis, including as R_{850} and R_{670} in the NDVI equation below,

$P_{det,leaf,\lambda}$ is the detected optical power measured with the sensor viewing the maize leaf at wavelength λ, and

$P_{det,Spectralon,\lambda}$ is the detected optical power measured with the sensor viewing the 99% Spectralon reference target at the same wavelength λ, under the same fixed 8 cm sensor-to-target distance.

For each leaf, the seven repeated spectral acquisitions were averaged at each wavelength to give one representative six-band profile. NDVI (Rouse et al., 1973) was then calculated from

the red and near-infrared bands at each measurement location, using Equation 22. Because growth stage strongly affects NDVI, tracking NDVI within plots across growth stage is a simple check of whether the sensor is behaving as expected.

$$NDVI = \frac{R_{rel,NIR} - R_{rel,RED}}{R_{rel,NIR} + R_{rel,RED}} \tag{22}$$

where:

$R_{rel,NIR}$ is the relative reflectance in the near-infrared band, and

$R_{rel,RED}$ is the relative reflectance in the red band.

Plant-level relative reflectance and NDVI values were averaged within each plot, then across the three field blocks, to give one treatment mean per plot. Reported error bars are the standard error across the three block-level means (n = 3 blocks), not across individual plant readings. Subsequently whether nitrogen treatment affected dark-enclosure NDVI at each growth stage was tested with a two-way ANOVA following the trial's randomized-complete-block design (NDVI ~ treatment + block, Type II sums of squares), with the plot as the unit of replication (18 plots per round: six treatments × three blocks). Residual normality and homoscedasticity were checked with the Shapiro–Wilk (Shapiro & Wilk, 1965) and Breusch–Pagan (Breusch & Pagan, 1979) tests for every fit. Treatment p-values were corrected for multiplicity using the Benjamini–Hochberg procedure at $\alpha = 0.05$. The randomized-complete-block ANOVA first tested whether nitrogen treatment affected plot-level nitrate-N. A significant effect would confirm the fertilizer program worked, but the analysis modeled nitrate-N directly rather than treatment, because nitrate-N is the continuous biological quantity of interest and treatment is only a coarse proxy for it. Without a treatment effect, only nitrate-N itself, could contain a signal for the sensor to detect. Either result therefore leads to plant-level regression of the six spectral bands against each plant's own measured nitrate-N, independent of its assigned treatment.

Additionally, an exploratory senesced-leaf re-sampling was analyzed separately, as a

sensor-responsiveness check rather than part of the randomized treatment experiment. Plot 306 from the first planting date was re-sampled after visible senescence, and the three senesced measurements were compared with the same plot's own mean response across its three normal-season growth-stage samplings (V10, R1, R3), rather than a different plot's same-day reading, so the comparison reflects a true before/after change within one plot rather than a plot-to-plot difference. Subsequently average NDVI for plot 306 during senesces was compared to its average NDVI for V10, R1 and R3, for both dark enclosure and ambient light condition. This was done to see whether the sensor registered a clear shift in leaf optical condition.

The laboratory-measured nitrate-N concentration of each plant served as the ground-truth response. To prevent data leakage, the dataset was partitioned at the plot level rather than the individual-plant level, ensuring that all plants from the same plot were assigned to only one subset. The resulting 75% 25% split, stratified by planting date, included 174 plants from 27 of the 36 plots in the training set and 59 plants from the remaining 9 plots in the held-out test set.

Five regression models were evaluated: a baseline model that always predicted the training-set mean, ordinary least squares, ridge regression, kernel ridge regression with a radial basis function (RBF) kernel, and a small feed-forward neural network. Together, these approaches represented a deliberate progression in model flexibility. The mean-prediction baseline provided a null reference that used no spectral information; ordinary least squares and ridge regression assessed linear relationships; and kernel ridge regression and the neural network allowed for nonlinear relationships and interactions among wavelengths. Each model was fitted once using the complete training set (n=174). Performance was then evaluated on the held-out test set. Comparing models of increasing complexity allowed the analysis to determine whether the available sample size supported nonlinear modeling and whether such models provided meaningful improvement over simpler alternatives.

The six relative-reflectance predictors, measured from 660 to 850 nm, were standardized to zero mean and unit variance using means and standard deviations calculated exclusively from the training data. These training-derived parameters were then applied unchanged to

the held-out test set, preventing information leakage during preprocessing. Standardization was particularly important for ridge regression, kernel ridge regression, and the neural network. In ridge regression, predictor scale influences the magnitude of the penalty applied to each coefficient; in the RBF kernel, it affects the contribution of each wavelength to the calculated distances between observations; and in the neural network, large differences in input scale can hinder gradient-based optimization. Although ordinary least squares does not require standardized predictors to generate equivalent predictions, it was trained using the same standardized inputs to maintain a consistent preprocessing pipeline across the regression models.

Predictors were limited to the six relative-reflectance bands and the categorical growth-stage variable. Growth stage was represented by two indicator variables, resulting in eight model inputs.

The same modeling framework was applied to the paired outdoor dataset. This dataset contained the outdoor, ambient-light reflectance measurements for the same plants and retained their corresponding identities, growth stages, and laboratory nitrate-N concentrations. Outdoor measurements were less complete than those collected inside the dark enclosure, because the signal was recorded below the background noise. Of the 233 plants, 90 (39%) were missing at least one of the six spectral bands values, and 23 had no usable outdoor measurement for any band. Signal missing was concentrated in the May R1 measurement round, which was consistent with the greater variability observed under ambient conditions.

Because the regression models required complete predictor vectors, missing outdoor values were imputed so that the affected plants would not have to be excluded for analysis. For plants with at least one usable band, missing bands data were estimated by iterative multivariate imputation using Bayesian ridge regression across the available spectral bands. The imputation model was fitted using the training data only and then applied unchanged to the held-out test data. For the 23 plants with no usable spectral bands, multivariate imputation was not possible; their band values were therefore replaced with training-set band means. Apart from the required imputation step, the outdoor and dark-enclosure analyses used the same growth-stage representation, standardization procedure, fixed model hyperparameters,

and plot-grouped training and test partition.

Tissue nitrate-N concentrations ranged from 10 to 13,937 ppm, spanning more than three orders of magnitude and producing a strongly right-skewed distribution as shown in Figure 21 is physiologically plausible because, when nitrogen availability is limited, absorbed nitrate is assimilated into organic compounds, leaving relatively small amount of nitrate stored in the tissue. Subsequently uptake exceeds the plant's immediate assimilatory demand, nitrate then accumulates in the leaf. This is a response commonly described as luxury consumption. This may leave most observations near the lower end of the range, with a smaller number of observations extends to much higher concentrations. This behavior supports the use of a logarithmic transformation.

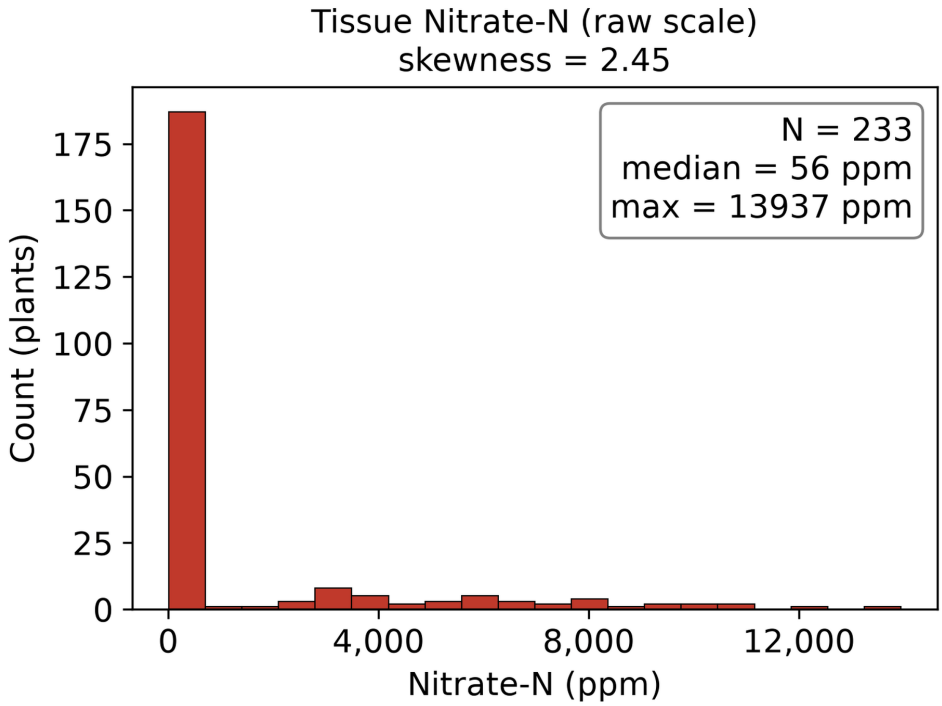


Figure 21. Distribution of tissue nitrate-N on the raw ppm scale, with skewness annotated.

Modeling nitrate-N on its original concentration scale would cause squared-error methods to be strongly influenced by the relatively small number of observations in the upper tail. Under these conditions, model fitting may prioritize errors among the highest concentrations while representing the more densely populated lower portion of the distribution poorly.

Therefore, the response variable was transformed using the natural-logarithm transformation in Equation 23 before model fitting. In this thesis, this transformation is referred to as log1p.

$$y = \ln(1 + C_{NO_3-N}) \tag{23}$$

where:

y is the log1p-transformed response variable used in all regression models,

$C_{\text{NO}_3\text{-N}}$ is the tissue nitrate-N concentration measured in the laboratory, in ppm, and

ln is the natural logarithm.

3.4 Specular-Reflection Angle Experiment

A bench experiment examined how leaf orientation affects the sensor's response and whether part of that effect can be attributed to specular reflection from the leaf cuticle. Six maize leaves were tilted through a controlled range of angles relative to the sensor, and the response at each of the six wavelengths was tested for a systematic dependence on angle.

The existing six-band active sensor and its power supply were used unmodified. The leaf-mounting platform was tilted by a servo motor. A tilt sensor mounted rigidly to the platform was used to establish the true physical angle relative to the sensor's optical axis at each servo position, since the commanded servo position did not correspond linearly to the true mechanical tilt. This calibration mapped the twelve servo positions used for data collection to physical angles ranging from $-25°$ to $+25°$ to the normal of the sensor. Sensor-to-leaf distance, LED settings, detector settings, and acquisition timing were held constant. Measurements were collected inside a dark enclosure to exclude ambient light.

Six fresh maize leaves, each treated as an independent replicate, were mounted in turn on the tilt platform, with the same leaf region and surface held under the sensor throughout each leaf's measurement sequence. At each of the twelve calibrated angles, the sensor acquired repeated spectral readings, between 6 and 24 per angle (median 8), which were averaged to give one six-band reading per leaf per angle. Each leaf was measured in two rounds. Before the second round, the leaf was rotated $90°$ about the axis normal to the mounting plate, remounted, and the full angle sequence was repeated. In the first round, the 850 nm channel read 10 to 14 times lower for leaves 801–803 than for leaves 804–806 at every angle, consistent with an LED or gain fault during those sessions. These 36 readings were excluded from all angle-effect tests and figures, so the 850 nm angle results rest on directly measured readings only. For the diffuse-plus-specular model described below, which requires complete spectra, the affected values were estimated from each leaf's own 750 nm reading using the 750–850 nm relationship of leaves 804–806.

Each acquisition was converted from raw firmware voltage to detected optical power at the corresponding wavelength using the same general equation (Equation 20; full derivation

in Appendix A), then converted to relative reflectance using the same white Spectralon reference target measurement. This was done so that the angle-experiment readings are on the same physical basis as the rest of this thesis.

The angle effect at each wavelength was tested with a repeated-measures mixed-effects model, treating angle as a fixed effect and leaf identity as a random grouping effect, with p-values corrected for multiplicity across the six wavelengths using the Benjamini–Hochberg false-discovery-rate procedure (Benjamini & Hochberg, 1995). A quadratic angle term was included alongside the linear term to test for the nonlinearity expected of a Fresnel-type specular response. The angle effect was tested on the first measurement round. The effect of leaf placement was assessed by comparing readings at normal incidence (0°) between the two rounds with a paired t-test at each wavelength.

$$R_\lambda(\theta) = \beta_0 + \beta_1\theta + \beta_2\theta^2 + u_{leaf} + \varepsilon \tag{24}$$

where:

$R_\lambda(\theta)$ is the reflectance predicted by the significance-test model at wavelength λ and leaf tilt angle θ (distinct from the physically constrained model Equation 25, which additionally separates diffuse and specular components),

β_0 is the fixed-effect intercept at wavelength λ,

β_1 and β_2 are the fixed-effect linear and quadratic coefficients for tilt angle at wavelength λ; a statistically significant quadratic term indicates the curvature expected of a Fresnel-type specular response, rather than a purely linear trend with angle,

u_{leaf} is a random intercept for leaf identity, drawn from a normal distribution with mean zero, accounting for repeated measurements taken on the same leaf across all twelve angles, and

ε is residual measurement error, assumed independent and normally distributed

with mean zero.

A physically constrained diffuse-plus-specular model was then fitted to both rounds pooled, to estimate the share of the signal attributable to specular reflection, expressing the specular contribution at each wavelength as follows:

$$P_\lambda(\theta) = b_\lambda + k_\lambda \times R_F(\theta) \times G(\theta) \tag{25}$$

where:

$P_\lambda(\theta)$ is the relative reflectance predicted at wavelength λ and leaf tilt angle θ,

b_λ is the fitted diffuse-reflectance baseline at wavelength λ, representing the angle-independent, subsurface (multiply-scattered) component of leaf reflectance,

k_λ is the fitted specular-scale coefficient at wavelength λ, weighting the surface-reflected (cuticle-glare) contribution,

$R_F(\theta)$ is the unpolarized Fresnel reflectance of the leaf cuticle at angle θ, obtained from the s- and p-polarized Fresnel reflection coefficients via Snell's law, as defined below,

$G(\theta)$ is a Gaussian angular-falloff term (Equation 29) modeling the finite angular width of the specular lobe produced by cuticle micro-roughness, and

θ is the leaf tilt angle relative to the sensor's optical axis, measured by the tilt sensor.

The unpolarized Fresnel reflectance $R_F(\theta)$ used above is obtained from Snell's law and the standard s- and p-polarized Fresnel reflection coefficients for light reflecting from a dielectric interface such as the leaf cuticle, as follows:

$$\sin\theta_t = \frac{sin\theta}{n_{eff}} \tag{26}$$

where:

θ is the leaf tilt angle relative to the sensor's optical axis (angle of incidence), as defined above,

θ_t is the corresponding angle of refraction inside the leaf cuticle, and

n_{eff} is a fitted effective refractive index of the cuticle, assumed constant with angle and shared across all six wavelengths.

$$R_s(\theta) = [\frac{cos\theta - n_{eff}\cos\theta_t}{cos\theta + n_{eff}\cos\theta_t}]^2 \qquad R_p(\theta) = [\frac{n_{eff}cos\theta - cos\theta_t}{n_{eff}cos\theta + cos\theta_t}]^2 \tag{27}$$

where:

R_s and R_p are the Fresnel power-reflection coefficients for light polarized, respectively, perpendicular (s) and parallel (p) to the plane of incidence; θ, θ_t, and n_{eff} are as defined in the equation above.

$$R_F(\theta) = \frac{1}{2}[R_s(\theta) + R_p(\theta)] \tag{28}$$

where:

$R_F(\theta)$ is the unpolarized Fresnel reflectance used in the physically constrained specular model above, obtained by averaging the s- and p-polarized coefficients, appropriate for the sensor's unpolarized LED illumination and detection.

$$G(\theta) = exp\frac{-\theta^2}{2{\sigma_g}^2} \tag{29}$$

where:

σ_g is a fitted Gaussian angular-width parameter, shared across all six wavelengths, controlling how quickly the specular lobe falls off away from normal incidence.

The 14 free parameters shared across the two equations above (neff, σg, and one kλ/bλ pair at each of the six wavelengths, i.e. $2 + 2\times6 = 14$ parameters in total) were estimated jointly by bounded nonlinear least squares (trust-region-reflective algorithm), fit once on the pooled data from both measurement rounds.

This functional form reflects the physical origin of leaf-surface glare. light reflecting from the cuticle-air interface follows the Fresnel equations for specular reflection at a dielectric boundary, while a real cuticle's micro-scale surface roughness broadens the otherwise mirror-sharp specular lobe into a finite angular range, approximated here by a Gaussian envelope. Separating an angle-independent diffuse baseline from an angle-dependent, Fresnel-shaped specular term is consistent with prior characterizations of specular and polarized leaf reflectance (Grant, 1987; Grant et al., 1993).



Figure 22. The existing six-band sensor mounted above the bench setup used for the specular-reflection angle experiment.

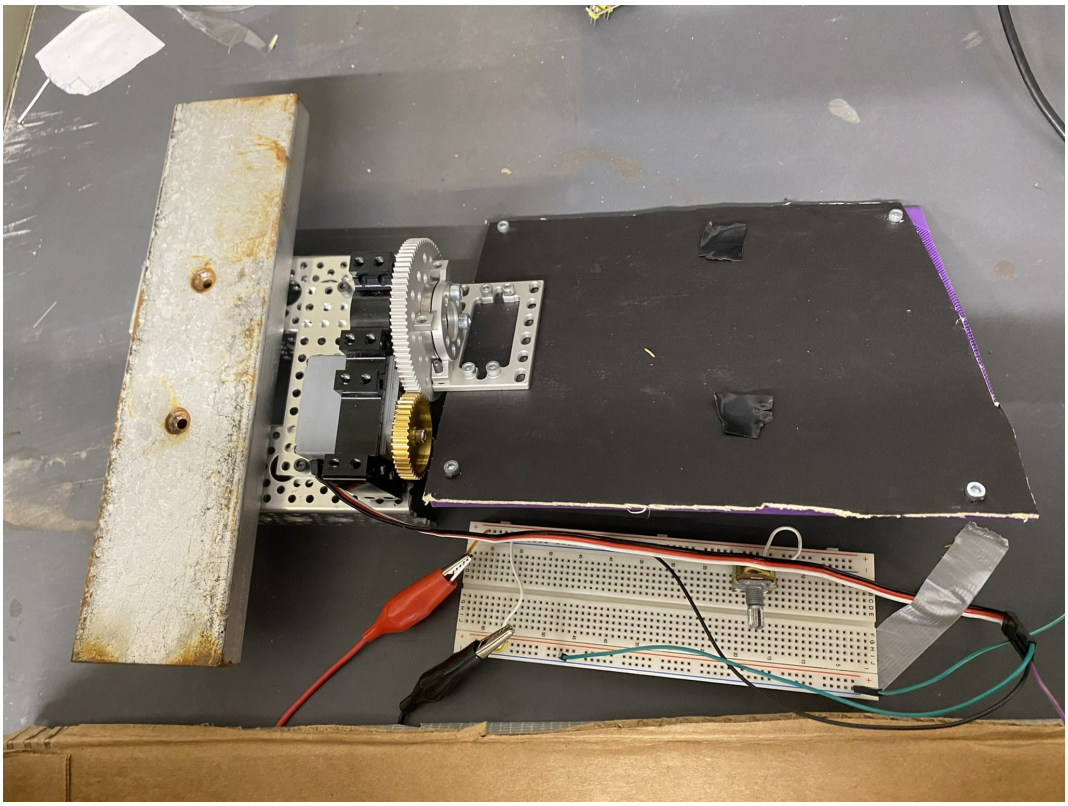


Figure 23. Detail of the servo-driven tilt mechanism used to control the angle for the leaf-mounting platform.

Chapter 4

Results and Discussion

This chapter first establishes how the complete sensing chain responds to a stable 99% Spectralon reference. It then examines whether the sensor recovered a plausible maize-leaf spectrum and whether nitrogen treatments could be distinguished under dark-enclosure and outdoor conditions. The next analyses test how growth stage changes the interpretation of the six-band signal and evaluate plant-level nitrate-N prediction. Model ablation is then used to separate the contributions of growth stage and the spectral bands. The chapter closes with angle experiments, which help explain what the prototype can detect and which optical effects still limit its performance.

4.1 Optical Reflectance Characterization Using a 99% Spectralon Reflectance Target

The Spectralon experiment was used to characterize the response of the complete optical and electronic measurement chain under a stable, highly reflective target. A 99% Spectralon target provides high, nearly wavelength-uniform diffuse reflectance across the sensor's measurement range. Measurements were collected in the dark enclosure with the target held at a fixed orientation normal to the sensor axis. The background-corrected digital output was converted to estimated optical power incident on the SiPM using equation Equation 20. Accordingly, Table 10 reports the estimated wavelength-specific power reaching the detector under this test condition, not the total radiant output of the LEDs or the total power reflected by the target.

Table 10. Optical Power calculated from the retrieved corrected digital voltage, after illuminating the 99% diffuse reflectance target.

Wavelength (nm)	Vcorrected (V)	PDE (%)	Ephoton (J)	ISiPM (mA)	Reff (A/W)	Pdet (W)	Pdet (uW)	Optical Gain (dB)
660	0.767111	3.841	3.01E-19	2.113253	2040.584	1.036E-06	1.0356	-63.05
670	0.761167	3.579	2.97E-19	2.096879	1929.852	1.087E-06	1.0865	-58.75
700	0.776333	2.893	2.84E-19	2.138658	1630.286	1.312E-06	1.3118	-60.28
750	0.793944	2.031	2.65E-19	2.187174	1225.803	1.784E-06	1.7843	-60.04
780	0.823444	1.642	2.55E-19	2.268441	1030.791	2.201E-06	2.2007	-57.86
850	0.883167	1	2.34E-19	2.432967	684.1735	3.556E-06	3.5561	-58.37

Figure 25 presents the wavelength-dependent effective SiPM responsivity while Figure 26 shows the measured electrical (voltage and current) quantities from which the detected optical power was recovered. Together, these results distinguish the nearly uniform electrical output observed for the 99% diffuse reflectance target from the wavelength-dependent optical

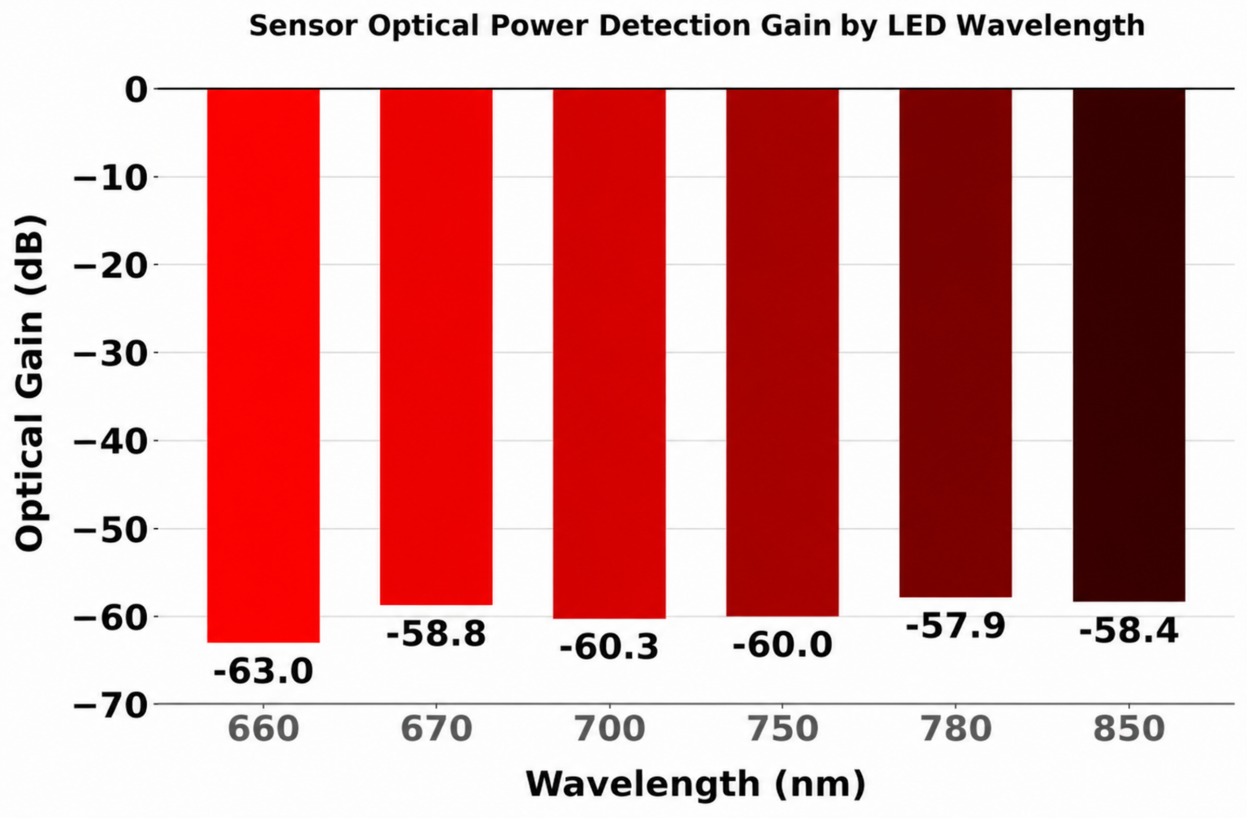


Figure 24. Sensor optical power gain by LED wavelength for the 99% Spectralon reference.

power transmitted from the LEDs.

The background-corrected voltage ranged from approximately 0.76 to 0.88 V, and the recovered SiPM photocurrent ranged from 2.10 to 2.43 mA (Figure 26). This limited variation agrees with the high and nearly uniform reflectance of Spectralon across the six wavelengths. Similar photocurrents, however, do not mean that equal optical power reached the detector. The calculated effective responsivity of the SiPM decreased from approximately 2041 A/W at 660 nm to 684 A/W at 850 nm. Consequently, a greater optical power was required at 850 nm to produce a photocurrent comparable to that measured at 660 nm. After responsivity was considered, estimated detected power increased from 1.0356 µW at 660 nm to 3.5561 µW at 850 nm. Normalizing these values by the manufacturer-rated output of each LED pair produced an estimated optical power detection gain of −63.0 to −58.4 dB (Figure 24). The narrow gain range suggests that the threefold change in estimated detected power did not arise from optical collection efficiency alone. Detector's responsivity and differences in

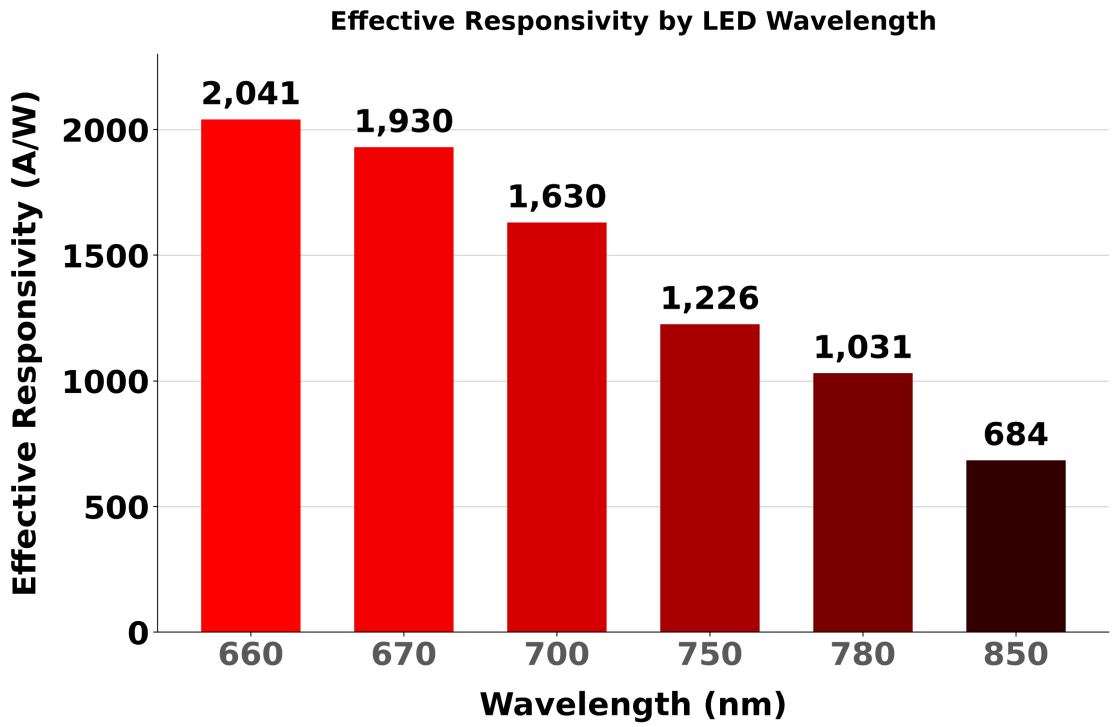


Figure 25. Calculated effective SiPM responsivity across the six measurement wavelengths (660-850 nm).

nominal LED output also contributed. Because installed LED output was not measured directly, these gain values describe relative performance under the test conditions; they are not measurements of absolute optical efficiency. The experiment nevertheless established that all six channels produced measurable signals and that LED emitted power and wavelength depended SIPM responsivity affects the reading. This supports the reason for transforming detected optical power for all leaf spectral readings to relative reflectance, using the 99% diffuse reflectance target in Equation 21.

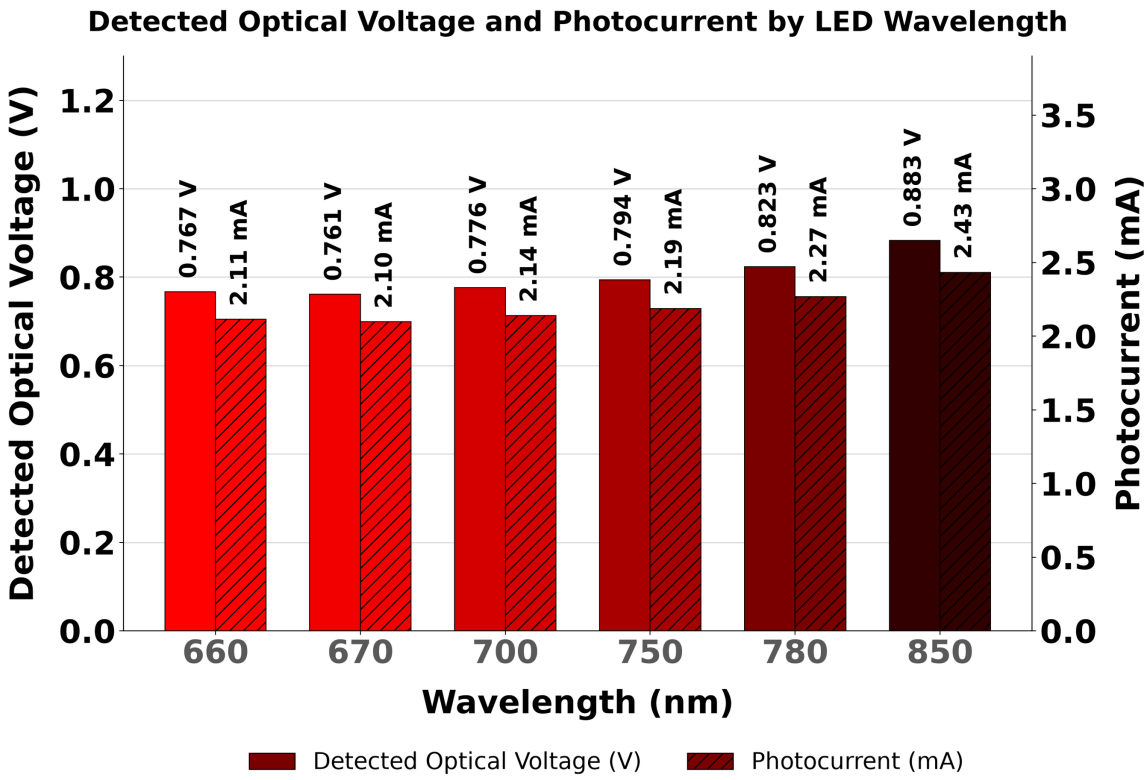


Figure 26. Corrected sensor voltage and recovered SiPM photocurrent by LED wavelength.

4.2 Dark-Enclosure and Outdoor Results by Growth Stage

After confirming a measurable response in all six channels, the analysis turned to maize leaves. The first question was whether the sensor recovered a recognizable leaf spectrum and whether that response remained stable outdoors. Each plant was measured with the active LEDs in the dark enclosure and under outdoor conditions. Within each sampling event, readings were averaged across sampled plants in a plot and then across the three replicate blocks for each nitrogen treatment. NDVI was considered with the six-band response because it summarizes the contrast between red-light absorption by chlorophyll and the stronger near-infrared response associated mainly with internal leaf structure.

The April and May plantings reached equivalent growth stages on different calendar dates because they were established approximately three weeks apart (Table 11). Comparisons were therefore organized by physiological stage (V10, R1, and R3), with the two plantings retained separately within each stage. This organization reduces the risk of interpreting differences in developmental timing as fertilizer-treatment effects.

Table 11. Realized sampling dates by growth stage and planting date, with the approximate calendar gap between the two planting dates at each stage.

Growth stage	April planting	May planting	Approx. gap (days)
V10	16 June 2026	8 July 2026	22
R1	8 July 2026	27 July 2026	19
R3	27 July 2026	17 August 2026	21

Figure 27 and Figure 28 summarize the dark-enclosure and outdoor spectral responses, respectively, for the sampled growth stages and planting dates. Under dark-enclosure conditions, the measurements retained the expected green-leaf pattern of lower red response and higher near-infrared response. However, the treatment profiles overlapped substantially and did not maintain a consistent ordering with applied nitrogen rate. Outdoor measurements were generally less stable, with lower responses and greater variability across wavelengths,

which was due to ambient light noise overshadowing real signal. Thus, the sensor recovered a physiologically plausible spectral shape most clearly under controlled illumination, but visual separation of nitrogen treatments was weak and was further obscured outdoors.

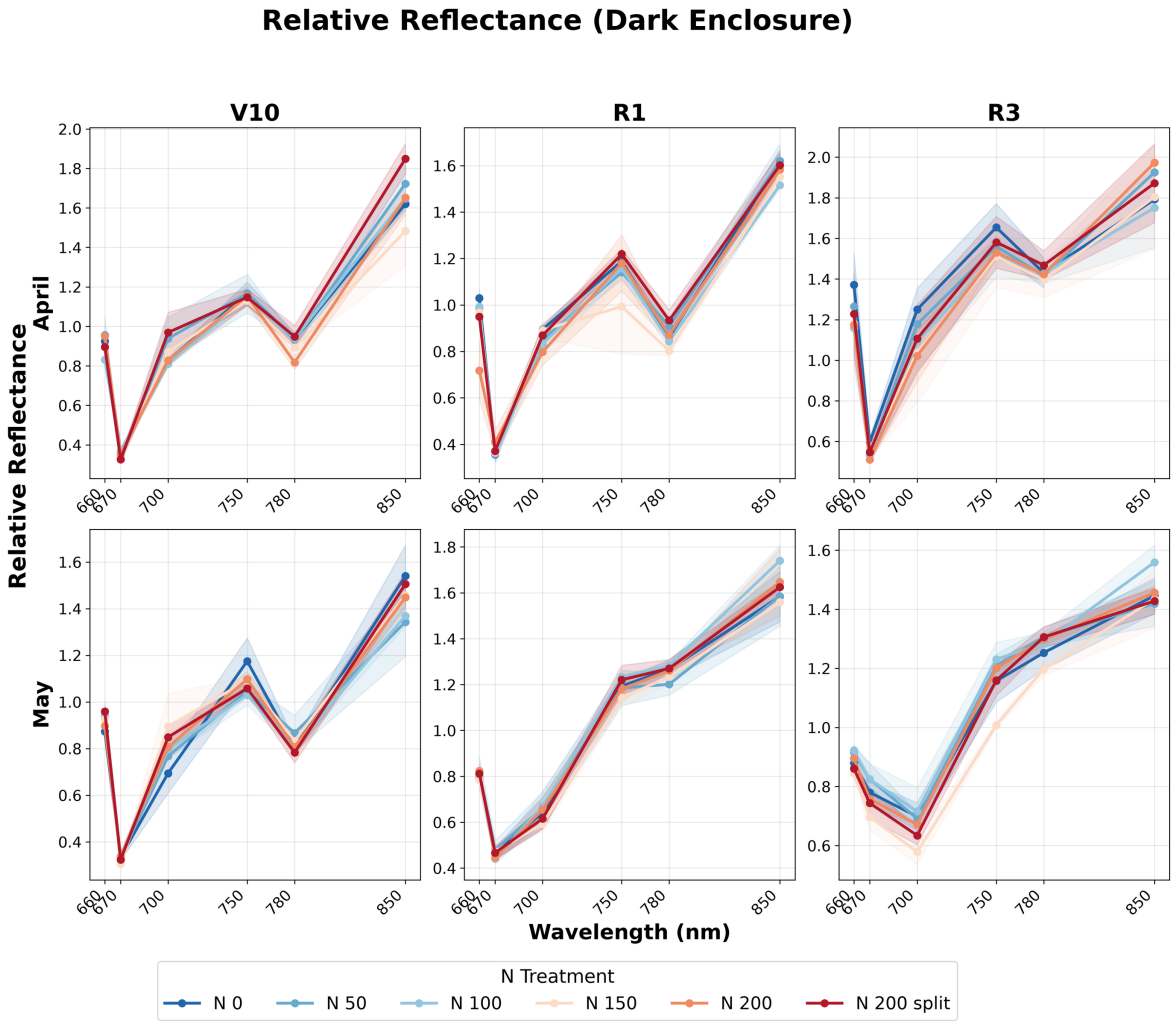


Figure 27. Relative Reflectance for each treatment under dark-enclosure conditions, all growth stages and planting dates.

Laboratory nitrate-N measurements were examined for the 233 plants with available ground truth, spanning all three growth stages (V10, R1, R3) in both the April and May plantings. Concentrations ranged from 10 to 13,937 ppm, with a median of 56 ppm (Figure 29). A randomized-complete-block ANOVA with nitrogen treatment and stage-planting combination as crossed factors confirmed a treatment effect on tissue nitrate-N ($p = 0.013$; Table 12). However, the stage-planting combination explained far more variation than treat-

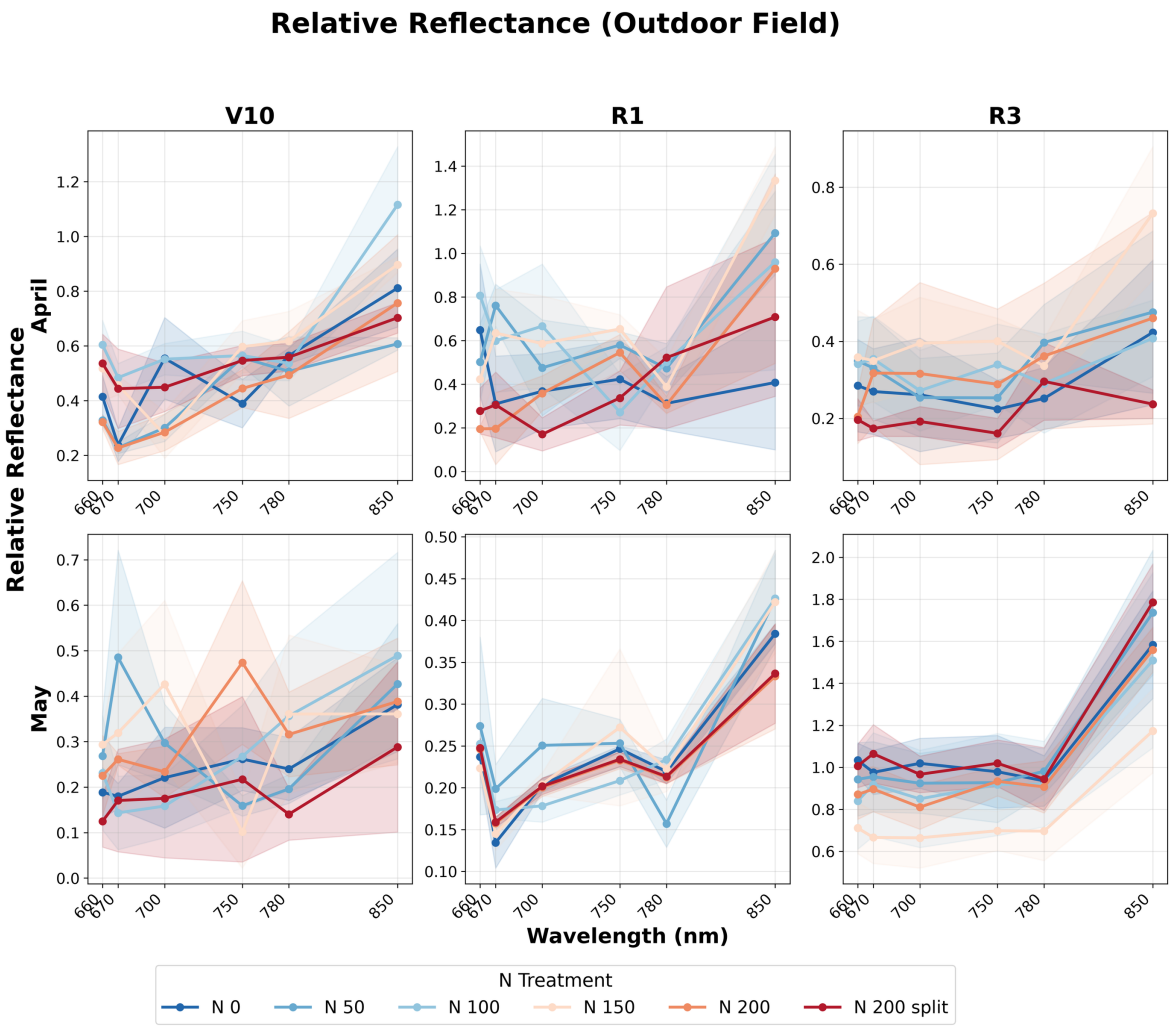


Figure 28. Relative Reflectance for each treatment under outdoor conditions, all growth stages and planting dates.

ment (69.9% vs. 2.6% of total variance), and the significant treatment-by-stage-planting interaction (13.0% of variance) indicated that the treatment response was not constant across sampling events. The nitrate gradient was strongest at V10 in the April planting, where treatment means spanned roughly eightfold, with a smaller but still appreciable four-fold spread at V10 in the May planting; the gradient was comparatively muted in the four R1 and R3 rounds. These results show that fertilizer application altered tissue nitrate-N, while also demonstrating that treatment rate was an incomplete proxy for the nitrate concentration of an individual plant. Subsequent modeling therefore used each plant's laboratory-measured nitrate-N rather than its treatment label.

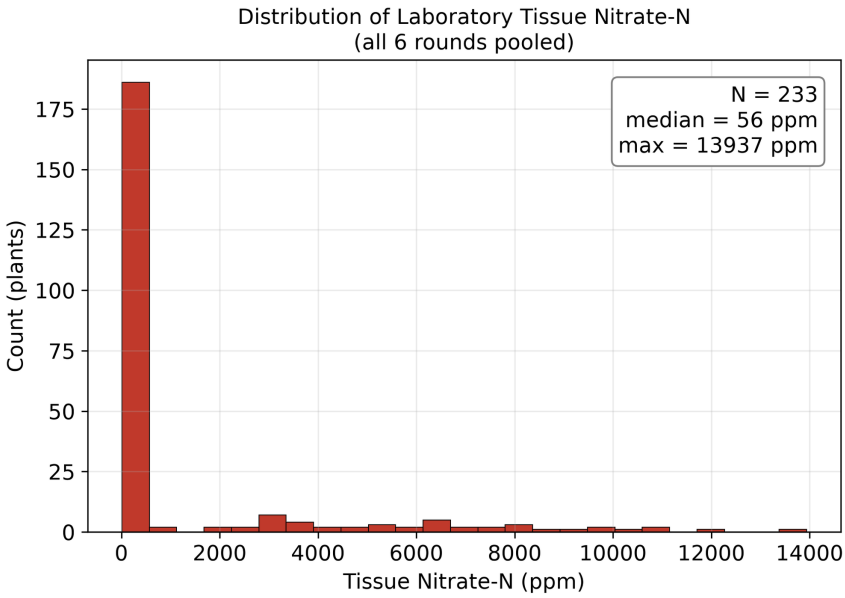


Figure 29. Distribution of laboratory tissue nitrate-N, all available rounds pooled (n = 233 plants).

Table 12. Randomized-complete-block ANOVA of tissue nitrate-N by nitrogen treatment and stage-planting combination.

Source	**df**	**F**	**p**	**% of Variance**
Stage-Planting	5	85.75	<0.001	69.9%
Treatment × Stage-Planting	25	3.18	<0.001	13.0%
Treatment	5	3.18	0.013	2.6%
Block (within Stage-Planting)	12	2.44	0.012	4.8%

4.3 Exploratory Senesced-Leaf Case

The substantial overlap among nitrogen treatments in the spectral data raised an important instrument-level question: was the sensor responsive to biologically meaningful changes in leaf optical properties, even if the differences among treatments were too subtle to separate reliably? An exploratory extreme-condition measurement was used to address this question. Plot 306 (Block 3, April planting, 0 lb N acre^{-1}) was re-sampled on 17 August 2026 after visible leaf senescence, using the same dark-enclosure and outdoor procedures as in the main experiment. Because this comparison involved one plot and was not part of the randomized treatment design, it was interpreted only as a functionality check, not as evidence of a fertilizer effect.

For the dark enclosure, the three senesced leaves had a mean NDVI of 0.006 (range: −0.018 to 0.024), compared with 0.610 for the same plot's own mean across its three normal growth-stage samplings (V10, R1, R3). The decline (~99%) was driven primarily by a large increase in reflectance at 670 nm (+242%), compared to nearly flattened red-band absorption in the same plot's healthy-season leaves. Moreover, 660, 700, and 780 nm also increased (+16 to +29%), 750 nm was essentially unchanged, and 850 nm decreased by 14%. This pattern is consistent with reduced red-light absorption following chlorophyll loss, whereas near-infrared behavior remains strongly influenced by leaf structure (Merzlyak et al. 2002).

Also, for the outdoor illumination condition mean NDVI recorded for the senesced leaves was 0.101 which was lower compared to the same plot's mean NDVI of 0.385 as shown in Figure 31. The direction of the NDVI change was therefore consistent between environments. However, the difference was more pronounced for the dark enclosure than for outdoor illumination conditions, this is because of ambient light contaminating outdoor readings.

Relative to the plot's own healthy-season baseline, senescence reduced NDVI by approximately 99% in the dark enclosure and 75% outdoors. Under controlled illumination, the largest change was concentrated at the chlorophyll-a absorption peak (670 nm). Outdoors, the response shifted across most of the measured spectrum rather than being localized to a single band. The instrument therefore detected the large optical change associated with

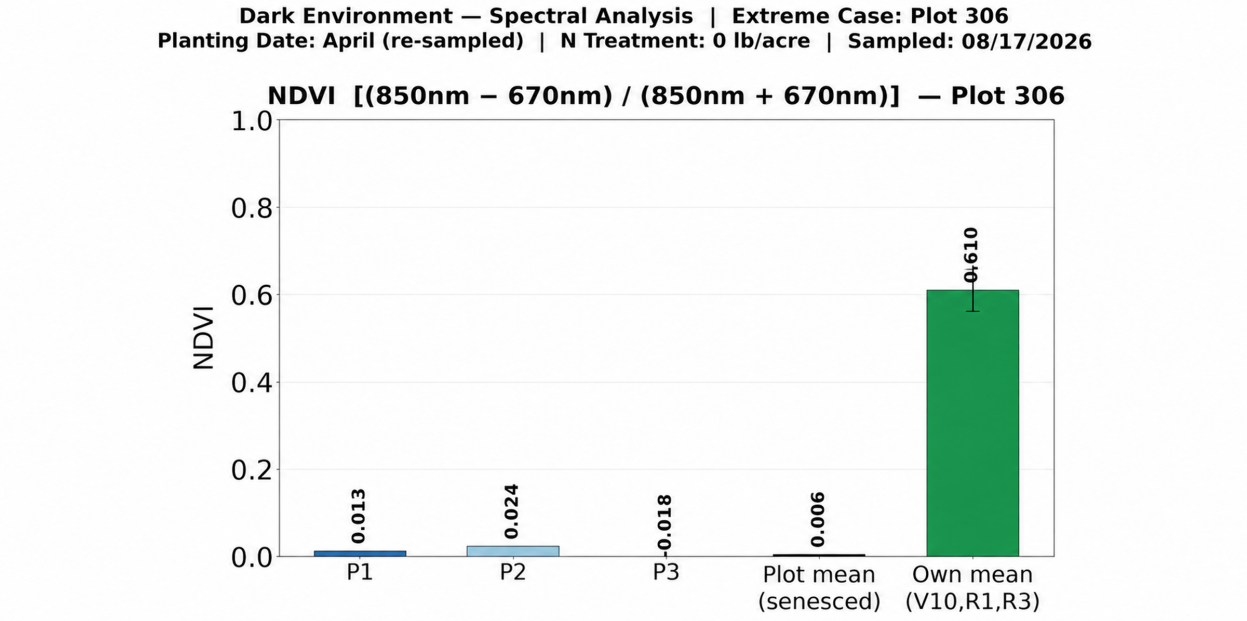


Figure 30. Senesced-leaf NDVI under dark-enclosure conditions (Plot 306, 17 August 2026).

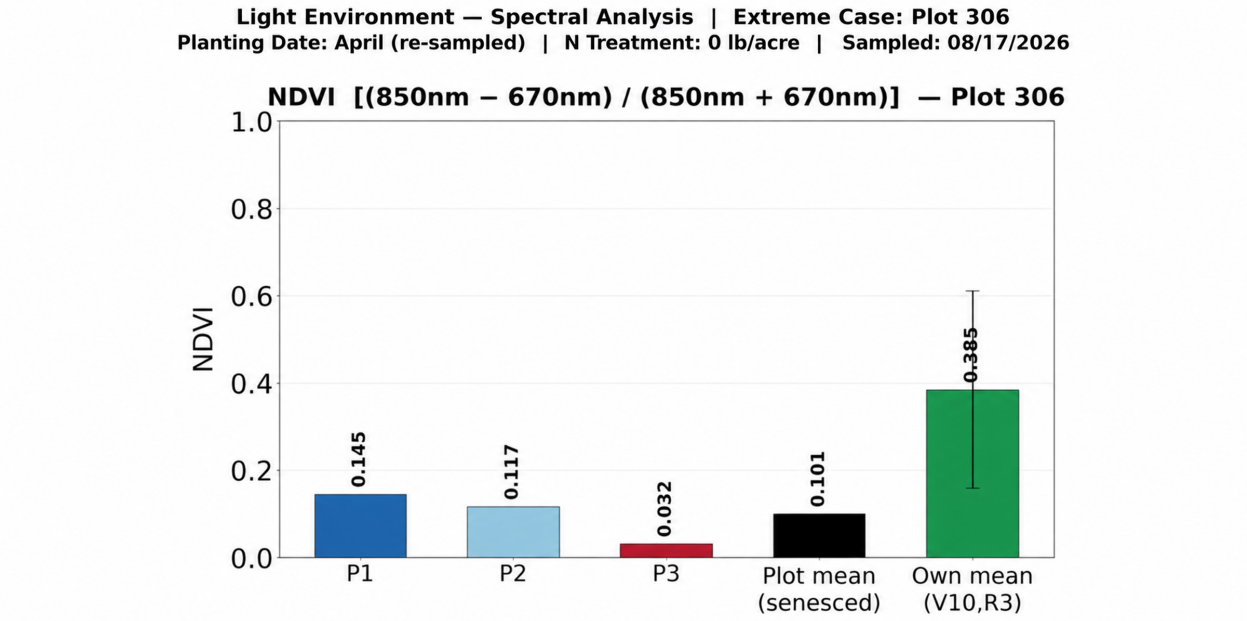


Figure 31. Senesced-leaf spectral profiles and NDVI under outdoor conditions (Plot 306, 17 August 2026).

senescence, but the recorded pattern was less biologically clear when the measurement environment was not well controlled.

4.4 Within-Plot NDVI Variation Across Growth Stage: Rationale for Growth Stage as a Model Feature

The results obtained so far suggested that developmental change could alter the spectral signal independently of fertilizer rate. Dark-enclosure NDVI was therefore examined across V10, R1, and R3 at two levels. First, all sampled plants NDVI were summarized separately for the April and May plantings to characterize the overall stage pattern. Median NDVI declined from 0.66 at V10 to 0.63 at R1 and 0.54 at R3 in the April planting (Figure 32). In the May planting, the corresponding medians were 0.64, 0.56, and 0.29 (Figure 33). Both plantings therefore showed declining NDVI with advancing stage, although the decrease was larger in the May planting.

Second, trajectories were examined within three selected plots from each planting to determine whether the aggregate pattern was also visible at the plot level. NDVI declined from V10 to R3 in all six selected plots. In the April planting, values decreased from 0.65 to 0.59 in Plot 204, from 0.67 to 0.51 in Plot 303, and from 0.69 to 0.57 in Plot 305 (Figure 34). The May planting showed larger decreases: 0.62 to 0.23 in Plot 402, 0.65 to 0.38 in Plot 504, and 0.64 to 0.32 in Plot 505 (Figure 35). This was done for the dark enclosure readings only because the dark enclosure readings were more stable than the field readings. These examples are descriptive rather than a formal longitudinal test, but their agreement with the pooled distributions supports the claim that, the effect of growth stage on spectral readings is also visible at the plot level.

Both the pooled distributions and the selected plot trajectories show that NDVI changed with growth stage. Growth stage was therefore added as a categorical model feature so that each six-band measurement could be interpreted in its developmental context. This approach is related to in-season methods that interpret NDVI relative to crop development, including the INSEY framework used with GreenSeeker (Raun et al. 2001). A categorical stage variable was more appropriate here because measurements did not begin at planting and the

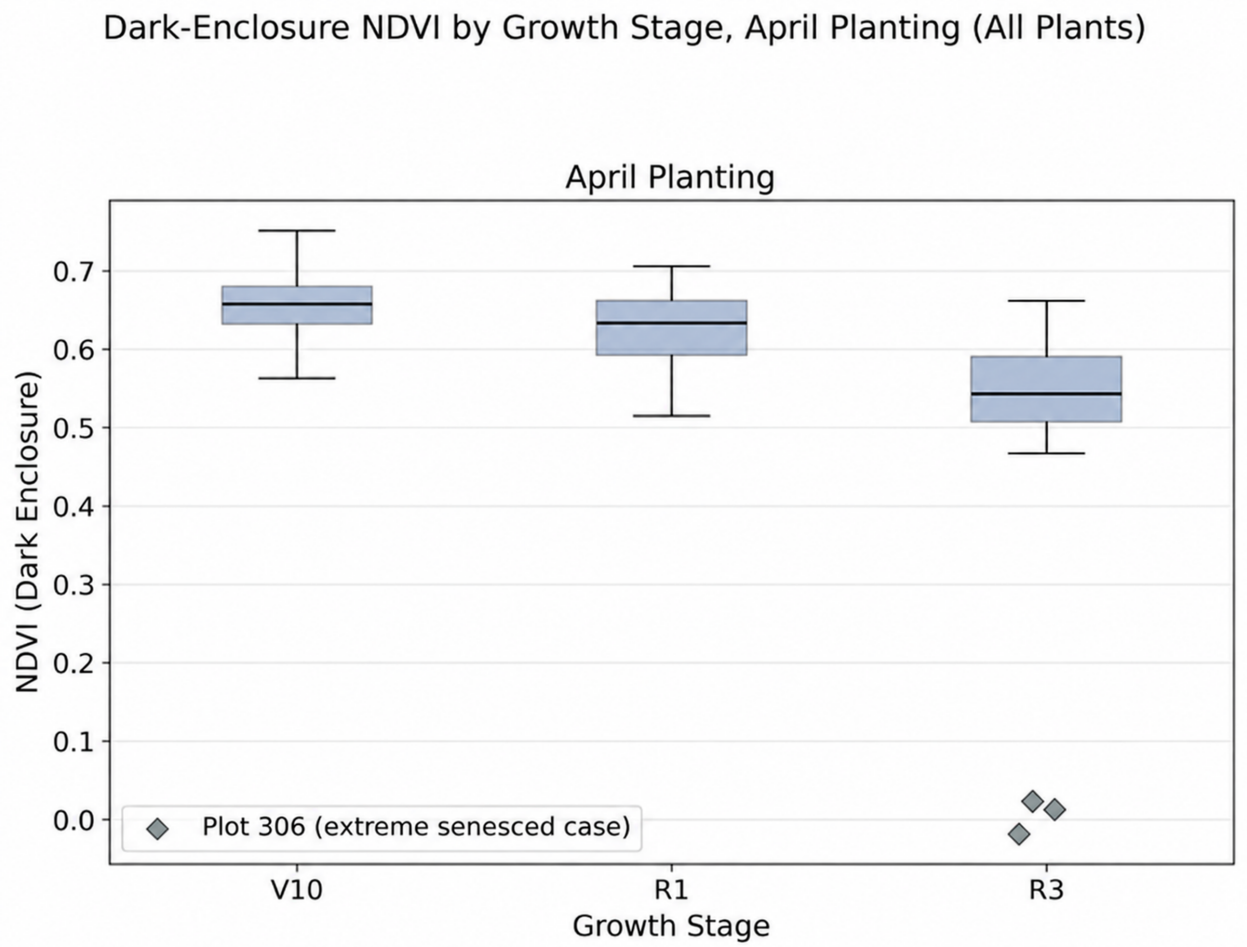


Figure 32. Dark-enclosure NDVI by growth stage (V10, R1, R3), all sampled plants, April planting.

intervals between stages were unequal. Growth stage does not correct specular reflection or other measurement artifacts, it accounts only for systematic phenological differences.

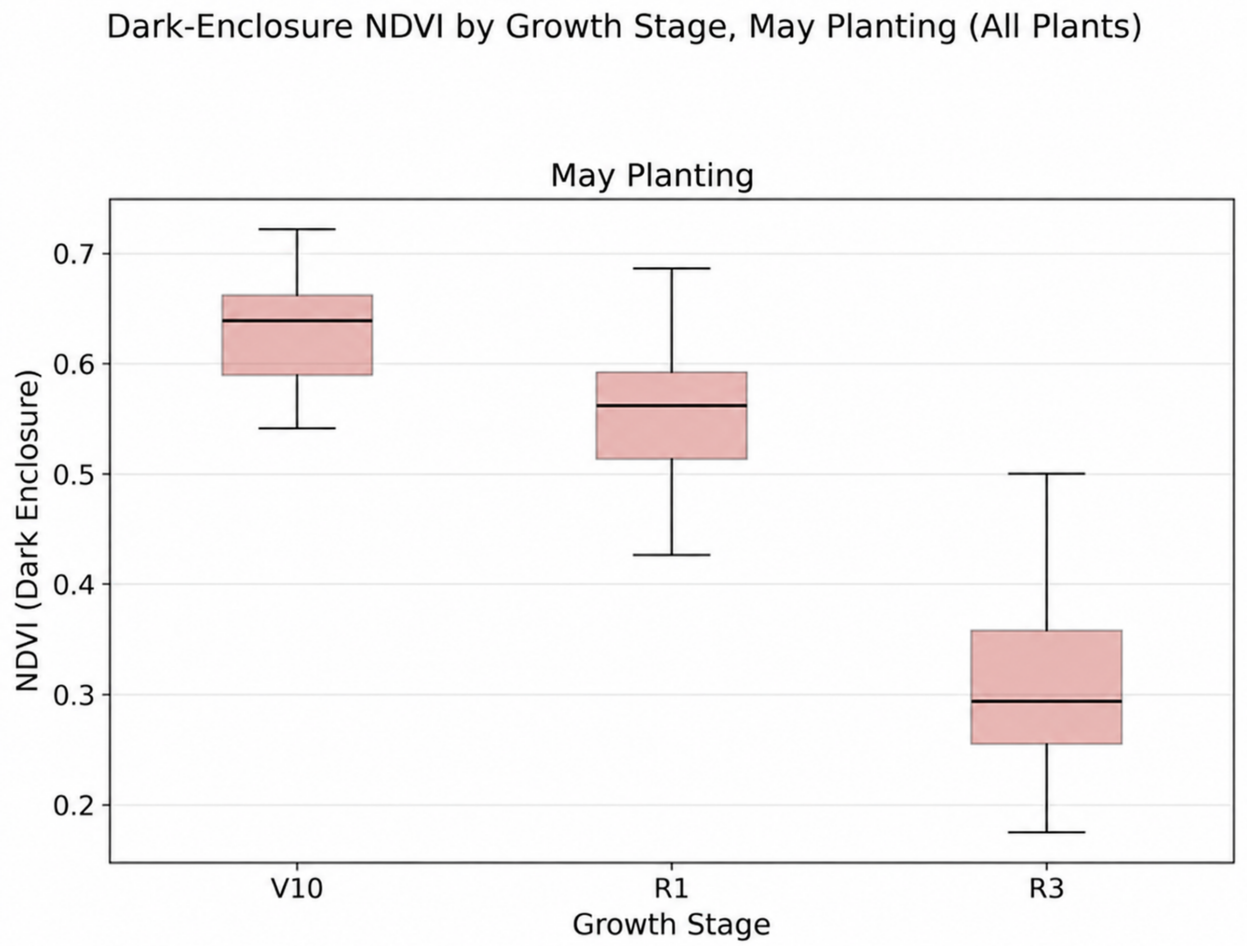


Figure 33. Dark-enclosure NDVI by growth stage (V10, R1, R3), all sampled plants, May planting.

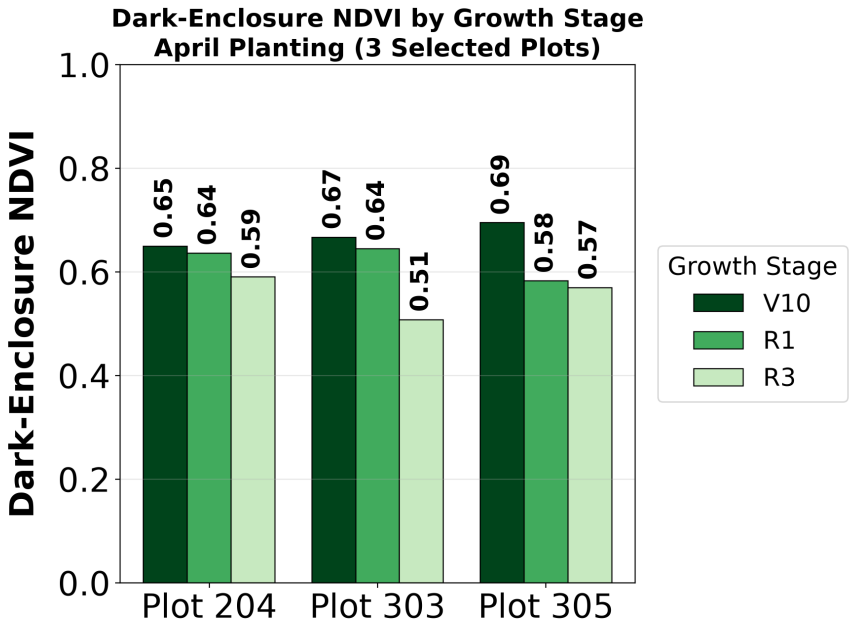


Figure 34. Dark-enclosure NDVI by growth stage (V10, R1, R3) for three randomly selected plots from the April planting.

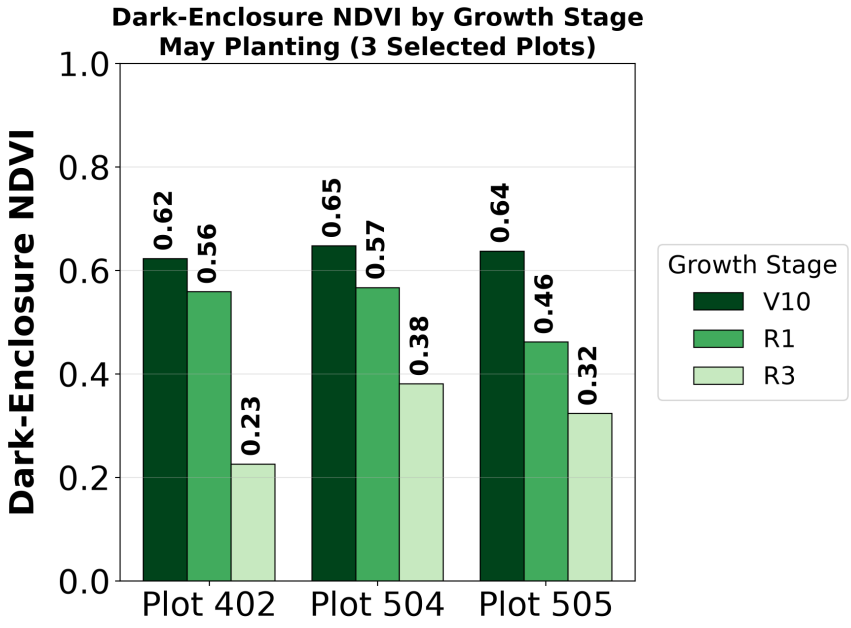


Figure 35. Dark-enclosure NDVI by growth stage (V10, R1, R3) for three randomly selected plots from the May planting.

4.5 Log-Transformed Nitrate-N Regression

Plant-level nitrate-N ranged from 10 to 13,937 ppm and was strongly right-skewed. A small number of high values would therefore dominate a model fitted by minimizing squared error on the raw scale. Such accumulation is biologically plausible when nitrate uptake exceeds its assimilation into organic nitrogen compounds (Crawford, 1995). Nitrate-N was transformed as log(1 + nitrate-N) before modeling. The transformed target gives less influence to the upper tail and makes the analysis more sensitive to proportional, rather than absolute, differences across the concentration range.

The log1p transformation reduced skewness from 2.45 to 1.21 (Figure 36). The transformed values still appeared to form lower- and higher-concentration groups, but the distribution was no longer dominated by a few extreme observations. These groups are treated as a feature of the sample and not as evidence of two distinct physiological populations.

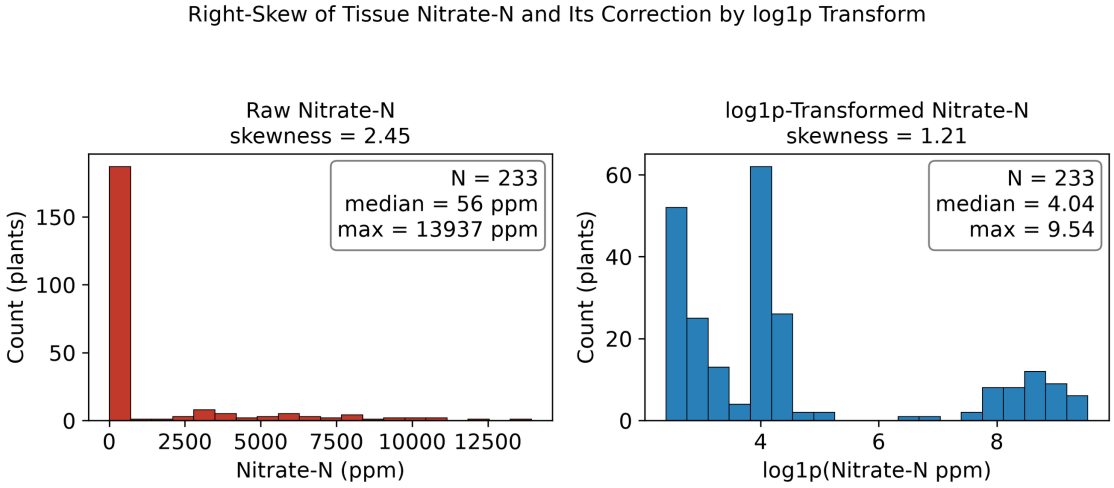


Figure 36. Distribution of tissue nitrate-N on the raw ppm scale(Left) and after a log1p transform (Right), with skewness annotated for each.

Five regression algorithms were evaluated using the same predictor set: the six spectral bands and two categorical growth stage variables (Table 13). Each model was fit once on the training data at a fixed, commonly used hyperparameter setting, and evaluated once on the held-out test set of 59 plants. All R^2 and r values reported in this section, including those in Table 13 to Table 19, were calculated on this held-out test set, not on the training

data. The feed-forward neural network produced the strongest test performance, with R^2 = 0.752 and Pearson r = 0.902 on the log1p scale. Kernel ridge regression was next strongest, at R^2 = 0.673, followed by ordinary least squares (R^2 = 0.569) and ridge regression (R^2 = 0.552). The mean-prediction baseline produced R^2 = -0.026. These results indicate that the combined predictor set captured variation not represented by an overall mean baseline model.

The five models were then fitted to the outdoor measurements using the same plot-grouped train-test split. All models performed worse outdoors. The neural network remained the strongest, but its test R^2 decreased from 0.752 in the dark enclosure to 0.526 outdoors, while r decreased from 0.902 to 0.810. Kernel ridge regression showed the same pattern, with R^2 decreasing from 0.673 to 0.466. Because the decline occurred across model types, it is more consistent with reduced measurement quality outdoors than with the behavior of a single algorithm.

Table 13. Performance of five models compared for predicting log1p(Nitrate_ppm) under both dark-enclosure and outdoor illumination, using relative reflectance from the six raw spectral bands and the growth-stage indicator (8 features total), trained on 174 plants and evaluated once on the same held-out test set of 59 plants for both conditions.

Model/Environment	**Dark**			**Light**		
	R^2	**r**	**RMSE**	**R^2**	**r**	**RMSE**
Mean baseline (no sensor information)	−0.026	undefined (constant prediction)	2.29	−0.026	undefined (constant prediction)	2.29
Ordinary least squares	0.569	0.813	1.48	0.374	0.646	1.79
Ridge regression (α = 1.0)	0.552	0.813	1.51	0.370	0.645	1.79
Kernel ridge regression (RBF, α = 1.0, γ = 0.125)	0.673	0.869	1.29	0.466	0.792	1.65
Feed-forward neural network (3 hidden layers, 16→12→8 units)	**0.752**	**0.902**	**1.13**	**0.526**	**0.810**	**1.56**

Table 13 reports held-out performance for all five models trained on 174 plants. Figure 38 shows observed and predicted values for the selected neural network under both measurement conditions.

The selected neural network used eight standardized inputs: six reflectance bands and two growth-stage indicator variables. These inputs passed through three fully connected layers with 16, 12, and 8 units. Each layer used a ReLU activation and dropout of 0.3, followed by a linear output for log1p(Nitrate_ppm) (Figure 37). The model was trained for 300 epochs with the Adam optimizer, a learning rate of 0.01, and L2 regularization through a weight decay of 0.01.

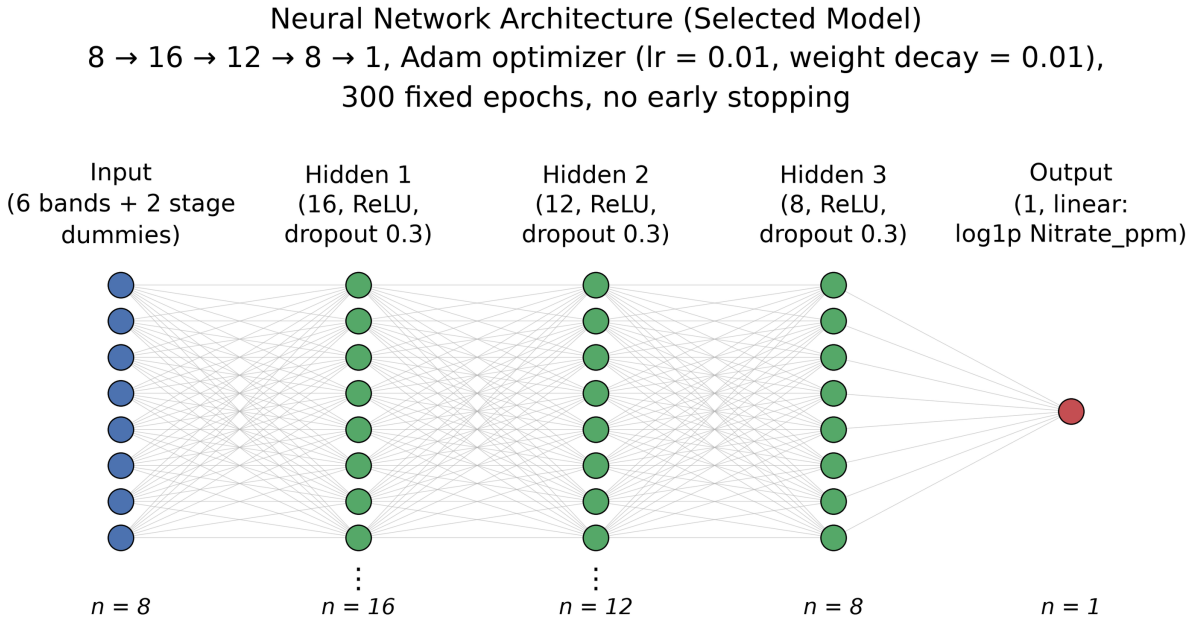


Figure 37. Architecture of the selected feed-forward neural network: eight predictors (six relative reflectance features, two growth-stage features), three hidden layers (16, 12, and 8 units; ReLU activation, dropout 0.3), and a single linear output unit predicting log1p(Nitrate_ppm).

The mean baseline was almost constant across the test set, whereas the neural network recovered a positive association under both measurement conditions (Figure 38). The full predictor set therefore contained information beyond the overall training-set mean. This comparison alone does not show how much of that information came from the six spectral

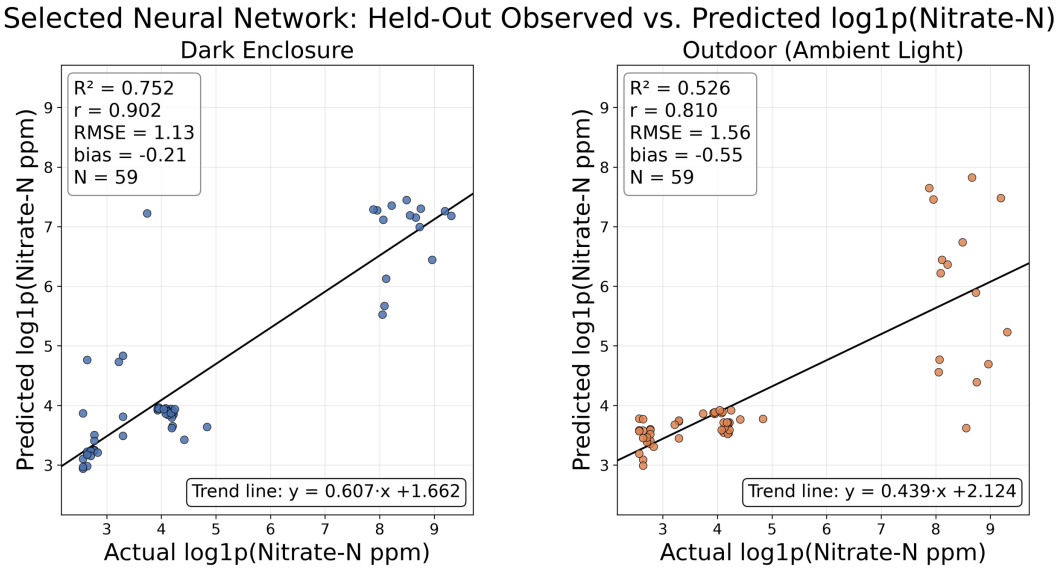


Figure 38. Held-out observed versus predicted log1p(Nitrate_ppm) from the selected neural network model. (left panel) under dark-enclosure conditions and (right panel) outdoor/ambient-light conditions.

bands and how much came from growth stage. However, the ablation analysis carried out subsequently addresses that question.

It was observed that the reported performance is lower than values from several proximal-sensing studies of crop nitrogen status (Cao et al. 2013; Hasituya et al. 2020; Li et al. 2021), but those studies did not predict the same target. They commonly examined chlorophyll-related traits, total leaf nitrogen, or canopy nitrogen status. Tissue nitrate-N has no unique absorption feature within the six red and near-infrared bands used here. Its spectral association is indirect and depends on how nitrate uptake, assimilation, chlorophyll formation, leaf structure, and development vary together. This physiological separation helps explain why growth stage provided important context for the spectral measurements.

4.6 Neural Network Using NDVI Combinations as Inputs

The models above used the six relative-reflectance bands directly as inputs. A further neural network was trained with NDVI values computed from those bands in place of the raw bands. The sensor provides two red bands (660 and 670 nm), one red-edge band (700 nm), and three near-infrared bands (750, 780, and 850 nm). Each red and red-edge band was paired with each near-infrared band in Equation 22, giving nine combinations (Table 14).

Table 14. The nine NDVI combinations used as neural-network inputs, formed from the sensor's red (660, 670 nm), red-edge (700 nm), and near-infrared (750, 780, 850 nm) relative-reflectance bands.

Combination	Red / red-edge band (nm)	NIR band (nm)	Formula
NDVI(660, 750)	660	750	$\frac{R_{rel,750}-R_{rel,660}}{R_{rel,750}+R_{rel,660}}$
NDVI(660, 780)	660	780	$\frac{R_{rel,780}-R_{rel,660}}{R_{rel,780}+R_{rel,660}}$
NDVI(660, 850)	660	850	$\frac{R_{rel,850}-R_{rel,660}}{R_{rel,850}+R_{rel,660}}$
NDVI(670, 750)	670	750	$\frac{R_{rel,750}-R_{rel,670}}{R_{rel,750}+R_{rel,670}}$
NDVI(670, 780)	670	780	$\frac{R_{rel,780}-R_{rel,670}}{R_{rel,780}+R_{rel,670}}$
NDVI(670, 850)	670	850	$\frac{R_{rel,850}-R_{rel,670}}{R_{rel,850}+R_{rel,670}}$
NDRE(700, 750)	700	750	$\frac{R_{rel,750}-R_{rel,700}}{R_{rel,750}+R_{rel,700}}$
NDRE(700, 780)	700	780	$\frac{R_{rel,780}-R_{rel,700}}{R_{rel,780}+R_{rel,700}}$
NDRE(700, 850)	700	850	$\frac{R_{rel,850}-R_{rel,700}}{R_{rel,850}+R_{rel,700}}$

The nine NDVI values were standardized with training-set means and standard deviations and, together with the growth-stage variables, formed eleven inputs to the same neural network used for the six-band model. The architecture, hyperparameters, training procedure, random seed, and plot-grouped training and test partition were unchanged, and the response was again log1p(Nitrate_ppm). Table 15 reports the held-out test performance for the dark-enclosure and outdoor measurements.

Table 15. Held-out test performance of the neural network using the nine NDVI combinations and growth stage as inputs, for dark-enclosure and outdoor measurements.

Model inputs	Test R^2	Test r
Nine NDVI inputs + growth stage (dark enclosure)	0.542	0.771
Nine NDVI inputs + growth stage (outdoors)	0.130	0.470

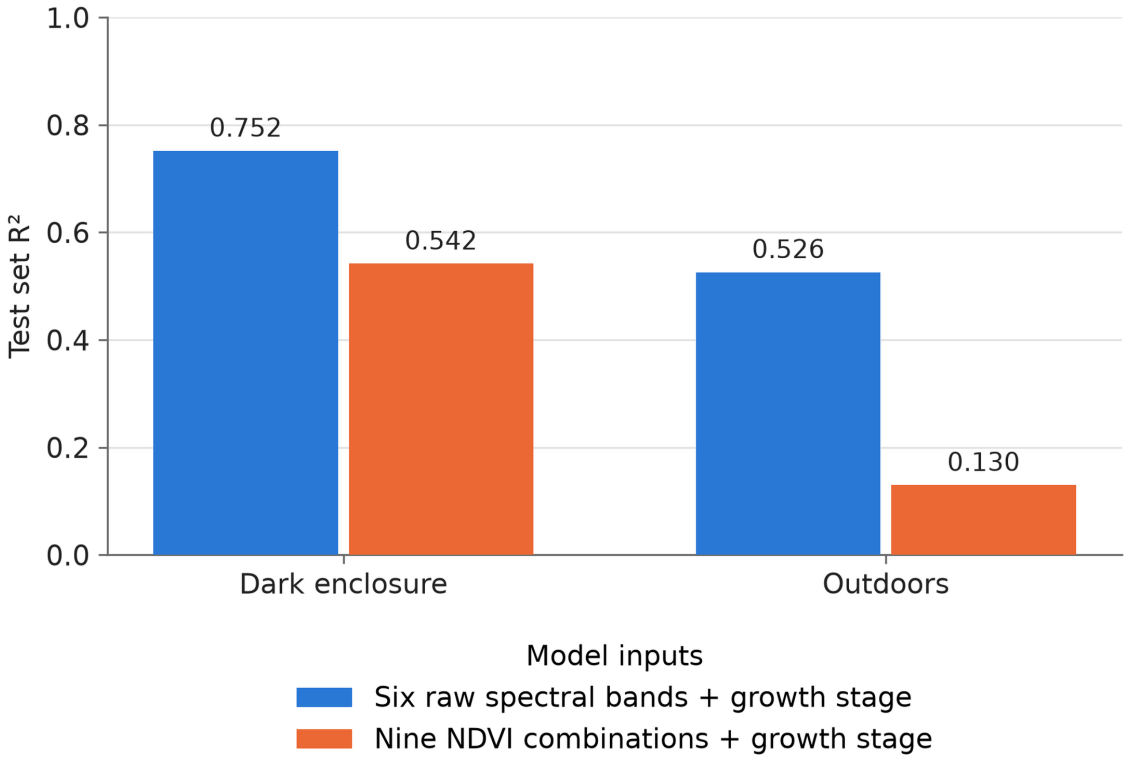


Figure 39. Held-out test R^2 of the neural network comparing the six raw relative-reflectance bands model performance to the nine NDVI combinations inputs model, each with growth stage variable (dark-enclosure and outdoor measurements).

The NDVI-based model performed slight less than the model given the six bands directly: test R^2 fell from 0.752 to 0.542, as shown in Figure 39. This decline is consistent with similar earlier studies that compared vegetation indices with full spectral inputs, and two explanations for it are well documented.

The first is information loss through compression. Each NDVI reduces two bands to a single number. Because NDVI depends only on the ratio of the two reflectances, it keeps their

contrast but removes their overall level, and any information outside that contrast is lost before the model sees it. The network cannot recover what the index has already discarded. Verrelst et al. (2015) reported the same pattern when retrieving leaf area index: a machine-learning model that used all spectral bands ($R^2 = 0.90$) outperformed two- and three-band indices ($R^2 = 0.82$). Li et al. (2014) found a similar result for canopy nitrogen content in winter wheat: partial least squares regression models built on canopy reflectance data increased the average validation R^2 by 75.5% compared with the best-performing spectral indices.

The second is model flexibility. NDVI imposes one fixed relationship between two bands, whereas a neural network given the raw bands can learn which bands, contrasts, and non-linear interactions are related to nitrate-N. Vegetation indices often have only moderate, nonlinear relationships with the plant properties they are used to estimate (Glenn et al., 2008), and reviews of crop nitrogen sensing have favored machine-learning regression for future sensors (Berger et al., 2020).

The larger loss outdoors is consistent with the lower quality of the outdoor readings. Outdoor reflectance values were lower and more variable than those measured in the dark enclosure, some bands were recorded as zero, and 39% of plants required imputation of at least one band. A ratio of two small, noisy values amplifies their error, and a single band recorded as zero forces NDVI to exactly -1 or $+1$, whereas the raw bands keep each error confined to one input. NDVI therefore did not offset the effect of ambient light limitation in this study.

Taken together, these explanations indicate that computing NDVI before modeling removed information that the network could otherwise use. For this sensor, the six relative-reflectance bands are therefore the more informative input.

4.7 Neural Network Using Six Raw Bands and Three NDVI Combinations as Inputs

Three of the nine NDVI Combinations were added to the six relative-reflectance bands and the growth-stage indicator variables, giving eleven inputs, this was done for every possible 84 combination of the three NDVI, six relative-reflectance bands, and growth-stage input. The neural network, training procedure, random seed, and plot-grouped partition were the same, and all 84 possible combinations were tested for both the dark-enclosure and the outdoor dataset. Table 16 summarizes the results, Figure 40 and Figure 41 show the test R^2 of every combination, and the complete results are listed in Table 28 (dark enclosure) and Table 29 (outdoors) in Appendix C.

Table 16. Held-out test performance of the neural network using the six relative-reflectance bands with and without three added NDVI combinations, with growth stage included in every model (n = 59 test plants).

Model inputs	Dark R^2	Dark r	Outdoor R^2	Outdoor r
Six bands + growth stage	0.752	0.902	0.526	0.810
mean of 84 Six bands + three NDVI combinations + growth stage	0.759	0.902	0.337	0.675
best of 84 Six bands + three NDVI combinations + growth stage,	0.817	0.923	0.503	0.773

In the dark enclosure, adding three NDVI combinations to the six bands produced almost the same performance as the six bands alone as shown in Figure 40. Across the 84 combinations, the mean test R^2 was 0.759 and the mean r was 0.902, compared with $R^2 = 0.752$ and $r = 0.902$ for the six bands with growth stage. The best combination reached $R^2 = 0.817$ and $r = 0.923$. For Outdoors, adding NDVI lowered performance as showed Figure 41. The mean R^2 of the 84 combinations was 0.337 ($r = 0.675$), compared with 0.526 ($r = 0.810$) for the six bands, and none of the 84 combinations reached the six-band result. The best outdoor combination reached $R^2 = 0.503$ and $r = 0.773$.

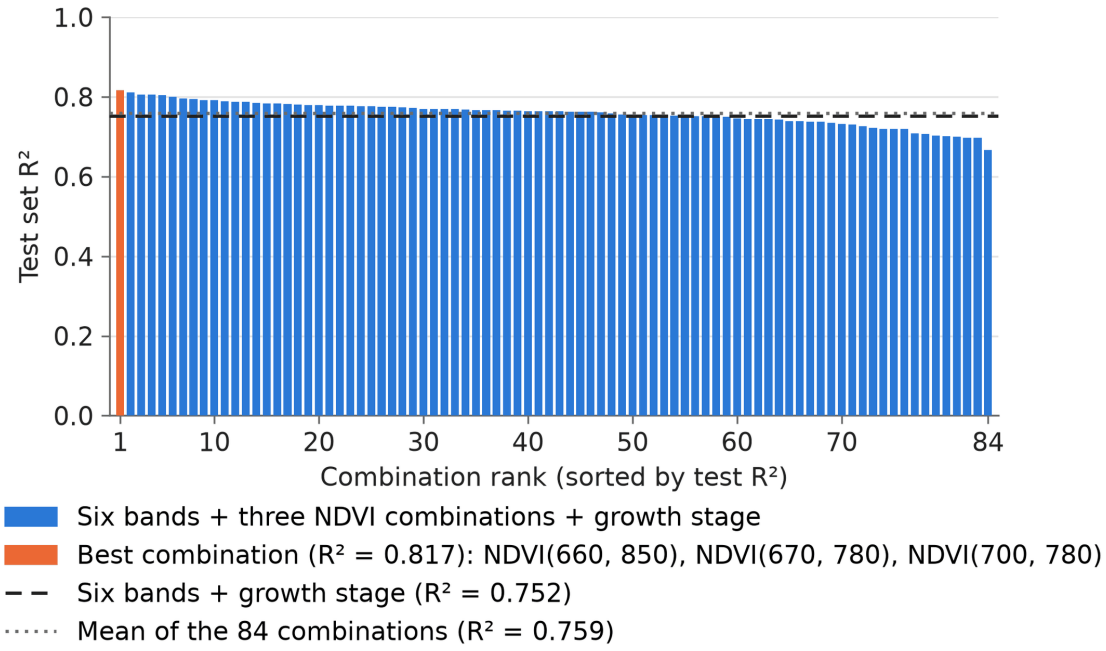


Figure 40. Test-set R^2 of the 84 neural-network models that combined the six relative-reflectance bands, three NDVI combinations, and growth stage, dark-enclosure measurements, ranked by R^2. The dashed line marks the six bands with growth stage, and the dotted line marks the mean of the 84 models.

The small difference in the dark enclosure is expected, because each NDVI is calculated from bands that the network already receives. The added indices therefore repeat information that the six-band model can already learn and do not supply a new measurement.

Outdoors, the added NDVI inputs reduced performance for the same reason that NDVI performed poorly when it replaced the bands: NDVIs of low, partly imputed outdoor reflectance values amplify measurement error. Adding three such inputs gave the network more noise to fit. The difference between training and test R^2 widened from 0.13 for the six bands to 0.37 on average for the 84 combinations, which indicates overfitting. This agrees with the ablation results shown in Table 17, where the outdoor spectral bands alone explained much less of the variation in tissue nitrate-N than the dark-enclosure bands.

Taken together, adding NDVI to the six bands caused the nitrate-N prediction model to overfit in the dark enclosure and reduced it outdoors. The six relative-reflectance bands, together with growth stage, therefore, remain the more informative input set for this sensor.

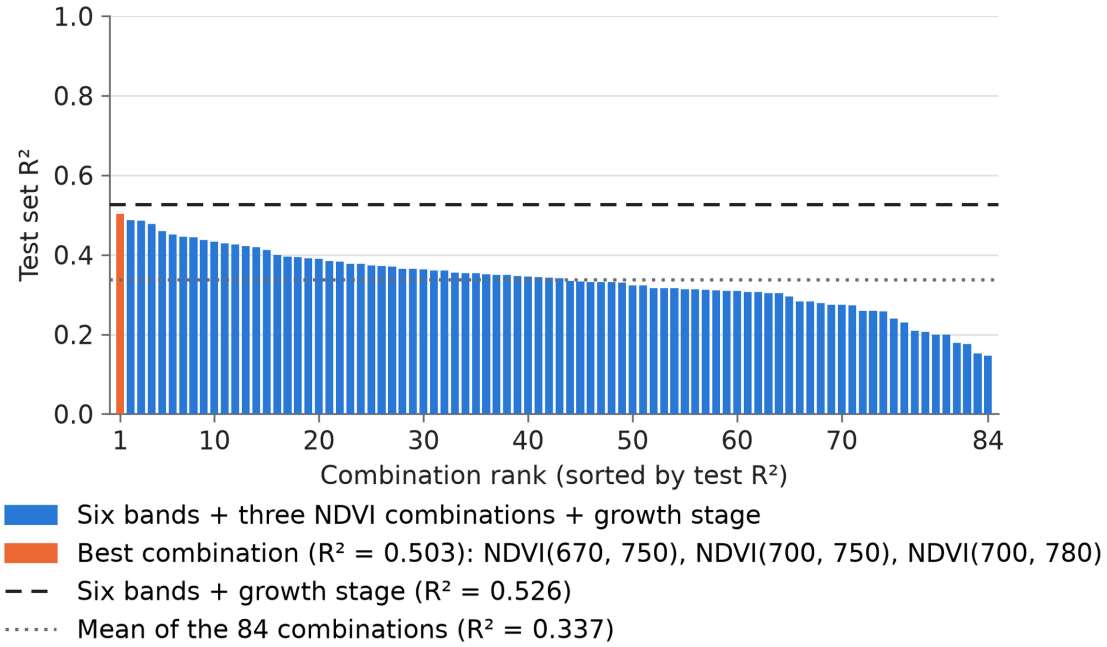


Figure 41. Test-set R^2 of the 84 neural-network models that combined the six relative-reflectance bands, three NDVI combinations, and growth stage, outdoor measurements, ranked by R^2. The dashed line marks the six bands with growth stage, and the dotted line marks the mean of the 84 models.

4.8 Regression Model Ablation Experiment

The models in Table 13 combine the six spectral bands (the six relative-reflectance values entered as separate inputs, not NDVI or other two-band indices) with growth stage. Their performance therefore does not reveal whether the spectral measurements remain informative without developmental context. This distinction matters because the ANOVA showed that stage-planting combination explained much more variation in tissue nitrate-N than fertilizer treatment. Growth stage is also available at the time of scanning without additional hardware. Table 17 addresses the question directly by repeating the model comparison with growth stage withheld, leaving only the six spectral bands as predictors.

Table 17. Performance of the five models using only the six raw spectral bands with growth stage withheld.

Model/Environment	**Dark**			**Light**		
	R^2	**r**	**RMSE**	**R^2**	**r**	**RMSE**
Mean baseline (no sensor information)	−0.026	undefined (constant prediction)	2.29	−0.026	undefined (constant prediction)	2.29
Ordinary least squares	0.099	0.394	2.15	−0.140	0.042	2.41
Ridge regression ($\alpha = 1.0$)	0.098	0.394	2.15	−0.136	0.042	2.41
Kernel ridge regression (RBF, $\alpha = 1.0$, $\gamma = 0.167$)	0.371	0.629	1.79	0.055	0.389	2.20
Feed-forward neural network (3 hidden layers, 16→12→8 units)	**0.471**	**0.700**	**1.64**	**0.096**	**0.377**	**2.15**

Withholding growth stage reduced performance across every non-baseline model under both measurement conditions (Table 17). The effect was largest for the selected neural network: dark-enclosure R^2 fell from 0.752 to 0.471 and outdoor R^2 fell from 0.526 to 0.096, a proportionally much larger loss outdoors than in the dark enclosure. A similar pattern held for kernel ridge regression (dark: 0.673 to 0.371; outdoor: 0.466 to 0.055), while the remaining models (ordinary least squares and ridge) collapsed to near-baseline performance

in the dark enclosure ($R^2 \approx 0.10$) and below-baseline performance outdoors ($R^2 < 0$). Growth stage is therefore not merely a convenient covariate; it is responsible for a substantial share of the model predictive performance, and its contribution is proportionally larger under outdoor illumination, where ambient light adds noise that makes the raw spectral bands alone less reliable. This is consistent with the RCBD ANOVA (Table 12), where stage-planting combination explained roughly 27 times more variance in tissue nitrate-N than treatment. Because growth stage costs nothing extra to observe in the field, its inclusion is a practical asset rather than a limitation, but it does mean that the spectral bands alone, without knowledge of growth stage, are a substantially weaker predictor of tissue nitrate. The share of variance in tissue nitrate-N explained by growth stage alone, by the six spectral bands alone, and by both together is summarized in Table 18.

Table 18. Share of variance in log1p-transformed tissue nitrate-N explained on the held-out test set (n = 59 plants) by growth stage alone, by the six spectral bands alone, and by both combined (held-out R^2 expressed as a percentage).

Predictors	**Dark enclosure**	**Outdoors**
Growth stage only (no spectral data)	42.9%	42.9%
Six spectral bands only (Table 17)	47.1%	9.6%
Spectral bands + growth stage (Table 13)	75.2%	52.6%

A second design question is whether six wavelengths provide more useful information than a smaller subset. Table 19 summarizes the dark-enclosure results from models trained with every possible combination of one to six wavelengths, averaged by the number of wavelengths. The same neural-network architecture and held-out test set were used throughout, and growth stage was excluded so that the comparison reflected the spectral bands alone. Complete results for each combination are reported in Appendix C. The held-out R^2 of each individual model is plotted in Figure 42, so every one of the 63 combinations can be seen alongside the group means summarized in Table 19.

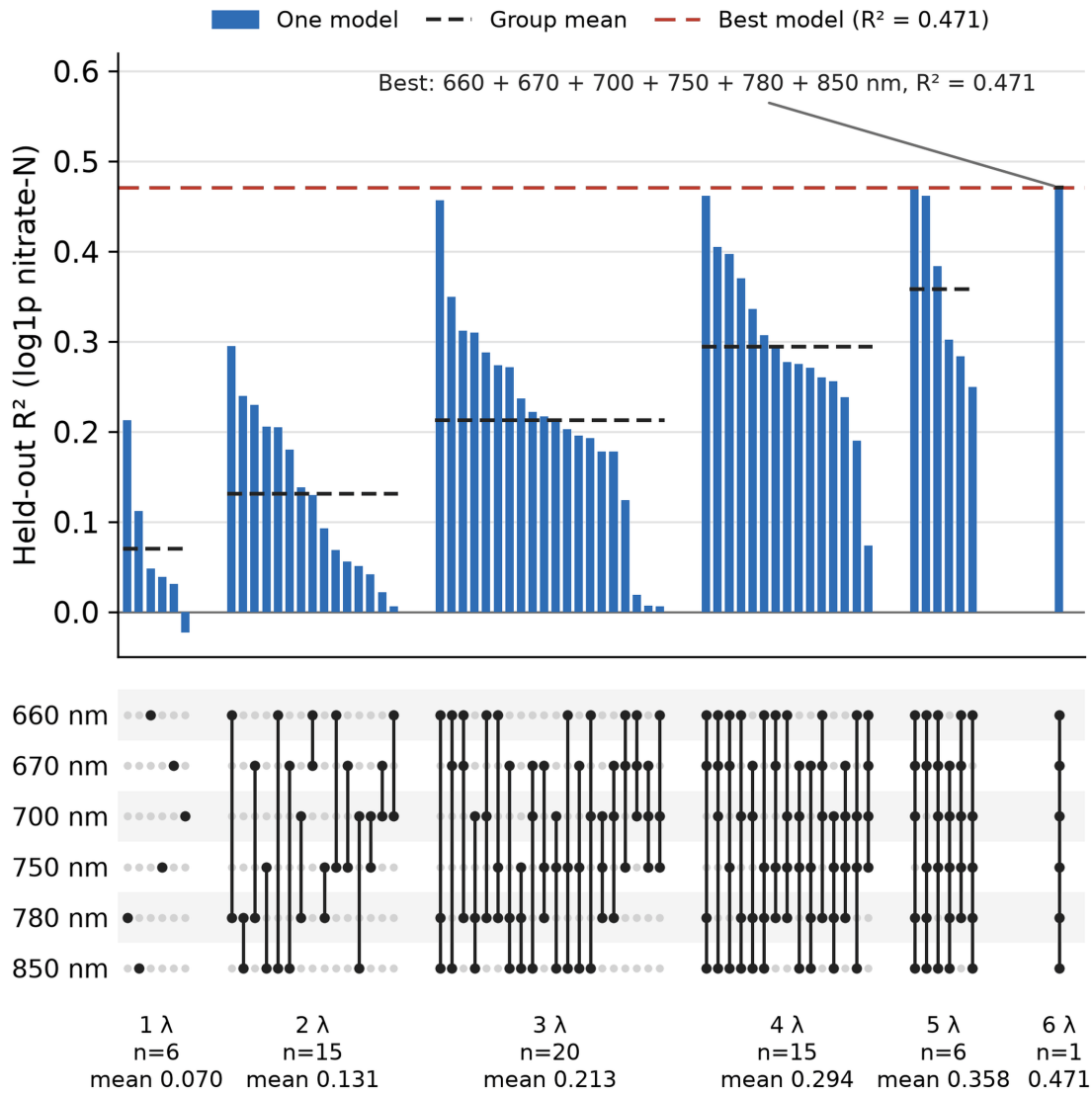


Figure 42. Held-out R^2 of every one of the 63 neural-network models trained on all possible combinations of the six wavelengths (dark-enclosure readings, growth stage withheld). Top: one bar per model, grouped by the number of wavelengths combined; dashed black lines show each group's mean and the dashed red line marks the best model. Bottom: filled dots show which wavelengths each model used. Values are those listed in Tables 22–27 (Appendix C).

Table 19. Mean held-out R^2 and r by number of wavelengths combined using dark room readings, neural network model; each value is the mean over all possible combinations of that number of wavelengths (6, 15, 20, 15, 6, and 1 combinations for one to six wavelengths), not the result of a single selected combination.

Wavelengths combined	**Mean R^2**	**Mean r**
1	0.070	0.321
2	0.131	0.398
3	0.213	0.490
4	0.296	0.566
5	0.361	0.630
6	0.471	0.70

Both R^2 and r increased progressively as additional spectral bands were combined (Table 19). In this analysis, "combined" means that the selected reflectance values were entered together as separate input features to the neural network; they were not merged into NDVI or other ratio-type indices. Some gain is expected when a flexible model is given more inputs, but every value in Table 19 was calculated on held-out plants that were not used for training, so the improvement reflects information that generalized beyond the training data rather than simple overfitting. Across the six single-band models, the mean R^2 was 0.070 (r = 0.321), compared with 0.131 ($r = 0.398$) across the two-band combinations and 0.213 (r = 0.490) across the three-band combinations. Performance increased further when all six bands were included achieving an R^2 of 0.471, substantially exceeding the average performance of the three-band models. Restoring growth stage increased the R^2 to 0.752 (Table 13). These results show that individual wavelengths provided limited predictive information in isolation, whereas using several wavelengths together as model inputs allowed the model to capture complementary spectral information. The additional improvement obtained with all six bands suggests that the neural network benefited from multiband relationships that may not be fully captured by conventional two- or three-band vegetation indices. This flexibility is a key advantage of the neural-network approach over traditional vegetation indices. An empirical index such as NDVI combines a fixed pair or triplet of wavelengths through a predefined formula, so it can only express the relationship that the formula was designed to

capture. A neural network, by contrast, learns the weighting of every input wavelength and the nonlinear interactions among them directly from the data, so it can uncover multiband patterns associated with nitrate-N that a predefined empirical index may miss. The steady rise in held-out R^2 as wavelengths were added, visible across all 63 models in Figure 42, is consistent with the network drawing on such multiband information. The large improvement after growth stage was restored further demonstrates that phenological context helped the model interpret these spectral relationships.

The individual results in Figure 42 also show that adding wavelengths improved prediction only when the right wavelengths were included. Every combination that omitted both near-infrared bands (780 and 850 nm) performed poorly regardless of how many wavelengths it contained: none of these 15 models exceeded $R^2 = 0.13$, and the four-band combination of 660, 670, 700, and 750 nm reached only $R^2 = 0.074$, about one-third of the R^2 obtained from the 780 nm band alone (0.213). Averaged across the two- to five-band models, including 850 nm raised R^2 by 0.13 and including 780 nm raised it by 0.12, whereas including 700 or 750 nm lowered it by 0.04 and 0.03, respectively. In total, 22 of the 57 multiband models performed worse than the best single wavelength. The composition of a wavelength combination, and in particular the presence of near-infrared information, therefore had an effect on predictive performance, alongside the number of wavelengths it contained.

4.9 Effect of Leaf Angle and Evidence of Specular Reflection

Leaf orientation changed the sensor's response at every wavelength. Relative reflectance declined steadily as the leaf was tilted from −25° to +25° as shown in Figure 43. The angle effect was significant at all six wavelengths after Benjamini–Hochberg correction as shown in Table 20.

Table 20. Repeated-Measures Mixed-Effects Model Results for the Angle Effect, per Wavelength.

Wavelength (nm)	**Linear slope β1 (per deg.) [95% CI]**	**Quadratic β2**	**Quadratic p**	**LR-test p (FDR)**	**Significant (FDR α=0.05)**
660	−0.00359 [−0.00426, −0.00291]	−0.00006	0.014	<0.001	Yes
670	−0.00527 [−0.00646, −0.00408]	−0.00012	0.002	<0.001	Yes
700	−0.00472 [−0.00560, −0.00384]	−0.00009	0.002	<0.001	Yes
750	−0.00245 [−0.00318, −0.00172]	−0.00007	0.005	<0.001	Yes
780	−0.00064 [−0.00122, −0.00006]	−0.00002	0.330	0.034	Yes
850	−0.00568 [−0.00745, −0.00390]	−0.00011	0.078	<0.001	Yes

The diffuse-plus-specular model in Equation 25 was then used to estimate how much of the signal could be attributed to specular reflection and diffuse reflection. Across wavelength, it was observed that the specular coefficient had a range of about 0.9 which is about 128 times smaller to the range of the diffused component as shown in Table 21. A smaller range for the specular component is due to the fact that specular reflection produced by the waxy cuticle

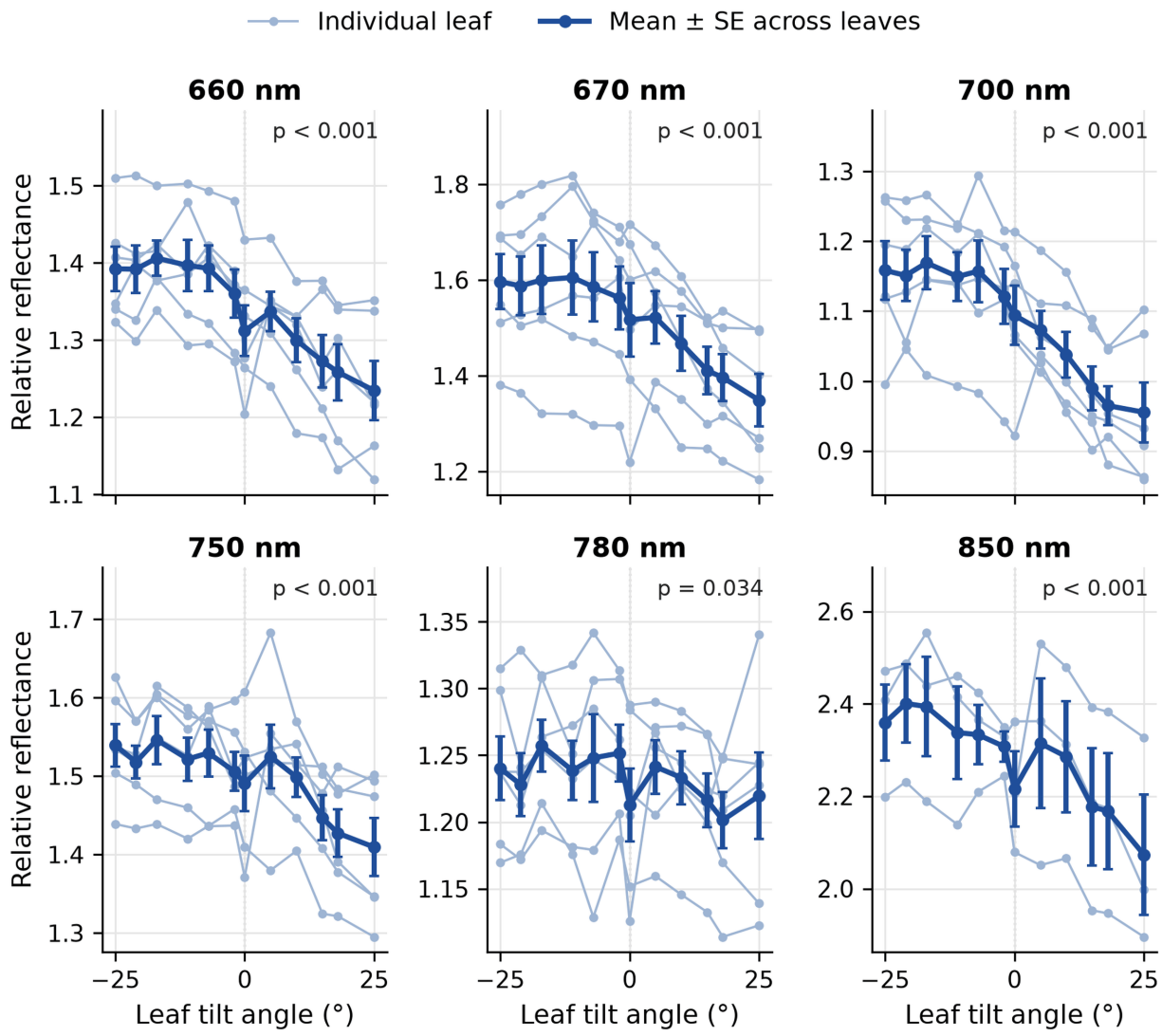


Figure 43. Measured relative reflectance versus leaf tilt angle at each wavelength (in a dark enclosure). Thin lines show individual leaves; thick lines show the mean ± standard error across leaves. The p-value in each panel is the FDR-corrected p-value for the angle effect (Table 20).

is almost constant for all wavelengths. Diffuse reflection, on the other hand, depends on the internal structure and properties of the leaf. Specular fraction is defined as the percentage of specular reflections in the total reflection measured at the sensor's 0° operating angle. This is reported in Table 21.

Table 21. Wavelength-specific diffuse and specular scale coefficients, their sum, and the specular fraction at zero tilt ($\theta = 0°$).

Wavelength (nm)	**bλ (Diffuse scale coefficient)**	**kλ (specular-scale coefficient)**	**Total coefficient**	**Specular fraction at θ=0° (%)**
660	41.26	1.320	42.58	3.1%
670	24.219	1.491	25.71	5.8%
700	21.452	1.058	22.51	4.7%
750	65.741	1.479	67.22	2.2%
780	109.324	1.216	110.54	1.1%
850	156.557	2.223	158.78	1.4%

The fraction of specular contribution was consistently larger in the visible red bands than in the near-infrared. Expressed as the fraction of the total predicted signal attributable to the specular term at the sensor's 0° operating angle, this fraction peaked at 670 nm (5.8%) and declined through the red-edge transition (700, 750, and 780 nm) into the near-infrared plateau, with only a small increase at 850 nm (1.4%).

This pattern aligns with established mechanisms in leaf physiological optics. In the visible spectrum, a leaf's total measured reflectance is a combination of two distinct pathways: surface specular (Fresnel) reflection from the waxy outer cuticle, and internal diffuse scattering from within the mesophyll tissue. Because the cuticular surface lacks photosynthetic pigments, its specular reflectance is driven entirely by index-of-refraction boundaries and remains spectrally flat across these wavelengths.

Conversely, the internal diffuse component is highly dynamic and inversely coupled with pigment absorption. Chlorophyll molecules absorb incident light strongly in the red region, reaching an absorption maximum near 670 nm. This aggressive absorption effectively suppresses the internal diffuse pathway, causing total leaf reflectance to drop to its lowest baseline

value. Because the absolute surface-reflected signal remains constant while the diffuse signal is heavily attenuated, the specular component mathematically expands to constitute a significantly larger fraction of the total optical signal in the red wavelengths than in regions of high reflectance. This is consistent with prior work linking surface and polarization-sensitive leaf reflectance effects to spectral regions of strong pigment absorption (Liu et al., 2022).

The experiment does not show how much specular reflection contributed to nitrate-prediction error. Only six leaves were measured, and the refractive index could not be identified independently from the available observations. Thus, the estimated specular fractions should be regarded as approximate. The result therefore identifies a likely interference mechanism and supports the cross-polarized design recommended in Chapter 5; it is not a validated correction for the field data. No correction for specular reflection was applied to the field measurements analyzed in this thesis, and the size of the specular contribution in those measurements was not adequate for the analysis.

Taken together, the experiment shows that the measurement geometry of a handheld leaf sensor, namely leaf tilt and leaf placement, changed the reading by several percent to more than 20% at individual wavelengths. Variation of this size can obscure the comparatively small spectral differences associated with nitrogen status among healthy green leaves. It therefore offers one explanation, alongside the dominant effect of growth stage and planting date on tissue nitrate-N and the indirect optical link between nitrate-N and leaf reflectance, for the limited separation of nitrogen treatments in the dark-enclosure measurements. Whether specular reflection forms a meaningful part of this geometric variation could not be determined from the tilt data alone. A direct comparison of readings taken with parallel and crossed polarizers, as described in the Recommendations, would measure it without relying on a model.

Chapter 5

Conclusion and Recommendations

5.1 Conclusion

This study developed and evaluated a handheld six-band active optical sensor for detecting nitrogen-related changes in maize leaves. The work progressed from instrument development and reference-target characterization to field measurements, nitrate-N modeling, model ablation, and controlled experiments investigating senescence and leaf-surface reflection. Together, these experiments established the sensor's ability to capture meaningful differences in maize leaf reflectance while also identifying the biological and environmental factors that currently limit interpretation of its measurements.

The first objective was achieved through the development of a portable instrument that sequentially illuminates maize leaves at 660, 670, 700, 750, 780, and 850 nm. The system integrates a silicon photomultiplier, two-stage signal amplification, oversampled analog-to-digital conversion, embedded control, onboard data storage, and a user interface. Testing with a 99% Spectralon reference characterized the wavelength-dependent response of the complete sensing chain. The estimated effective SiPM responsivity decreased from approximately 2041 A/W at 660 nm to 684 A/W at 850 nm, while the source-normalized optical detection gain remained between -63.0 and -58.4 dB.

Measurements collected under dark-enclosure conditions demonstrated that the sensor

could recover the expected spectral contrast between relatively low red reflectance and high near-infrared reflectance in healthy green maize leaves. However, neither the treatment-level spectral profiles nor the calculated vegetation indices followed a consistent order across nitrogen application rates. Although laboratory analysis confirmed that nitrogen treatment affected tissue nitrate-N ($p = 0.013$), the combination of growth stage and planting date explained substantially more variation than treatment alone. NDVI also decreased from V10 to R3 in both plantings. These findings demonstrate that phenological development altered the optical properties of the leaves sufficiently to obscure a direct and consistent relationship between nitrogen treatment and spectral response. Growth stage was therefore necessary context for interpreting the six-band measurements.

This importance of phenological context was reinforced by the nitrate-N modeling results. The selected feed-forward neural network, using all six spectral bands as individual relative reflectance inputs together with growth stage, achieved $R^2 = 0.752$ and $r = 0.902$ for log1p-transformed tissue nitrate-N on the held-out dark-enclosure test set. These results show that the combined inputs contained useful information for estimating relative differences in nitrate-N among plants represented by this dataset. They do not, however, establish an absolute nitrate-N calibration. The fitted slope of 0.607 and the systematic underestimation of high concentrations after back-transformation indicate that the model compressed the observed range, particularly at the upper end of the nitrate-N distribution.

The ablation results clarify how the model obtained its predictive performance. Individual wavelengths provided little information when used alone, but average performance improved as additional bands were combined. The complete six-band model produced the strongest spectral-only result among the wavelength subsets tested, suggesting that the relevant information was distributed across wavelengths rather than concentrated in a single band. When the six bands were replaced by nine NDVI combinations formed from the same wavelengths, with growth stage retained, test R^2 fell from 0.752 to 0.542 and r from 0.902 to 0.771 in the dark enclosure, and from 0.526 to 0.130 and from 0.810 to 0.470 outdoors. Reducing pairs of bands to a single index therefore discarded information that the network could otherwise use, and the six-band model outperformed the vegetation-index inputs. Nev-

ertheless, withholding growth stage reduced the dark-enclosure R^2 from 0.752 to 0.471 and the outdoor R^2 to 0.096. Growth stage therefore contributed substantially to nitrate-N estimation and became especially important under outdoor conditions. This dependence on growth stage is consistent with how established active-sensor nitrogen algorithms are designed. The In-Season Estimated Yield (INSEY) index used with the GreenSeeker sensor, for example, divides NDVI by the number of days from planting to sensing, so that the spectral reading is interpreted relative to crop development rather than in isolation (Raun et al., 2001). The same principle applied here. On the held-out test set, growth stage alone explained 42.9% of the variance in log1p-transformed tissue nitrate-N (Table 18). The six spectral bands alone explained 47.1% under dark-enclosure conditions but only 9.6% outdoors and combining growth stage and spectral bands raised the explained variance to 75.2% and 52.6%, for dark enclosure and outdoor environment respectively. The final model is therefore a growth-stage-informed estimator that interprets multiband spectral measurements within their phenological context, in the same way as INSEY-type algorithms, rather than a reflectance-only nitrate-N meter, and its spectral contribution is greatest under controlled illumination for these devices specifically.

The remaining experiments helped define the boundaries of this interpretation. In the exploratory senesced-leaf experiment, dark-enclosure NDVI was approximately 98% lower than that of a healthy leaf across the growth stage for a single plot. This difference arose primarily from an increase in red-band response, while the near-infrared response changed less. The result confirms that the sensor can detect a large change in leaf optical condition, but it does not demonstrate that it can reliably resolve the smaller physiological differences associated with nitrate-N among healthy green leaves. Similarly, the consistently poorer outdoor results indicate that the existing control of ambient light and measurement geometry was insufficient for stable field measurements. The six-leaf angle experiment further showed that leaf tilt and placement changed the response at every wavelength and suggested a small, wavelength-dependent surface-reflection component, although the extent to which this component contributed to nitrate-N prediction error could not be determined from the available data.

The principal contribution of this thesis is therefore both a functional six-band active optical sensor and an evidence-based definition of what its measurements currently represent. The prototype captured expected maize leaf reflectance patterns, detected large changes in optical condition, and produced useful relative nitrate-N estimates when all six bands were interpreted together with growth stage under controlled conditions. At the same time, the results show that phenology, ambient light, measurement geometry, surface reflection, and the highly variable distribution of tissue nitrate-N affect the relationship between leaf reflectance and nitrate status.

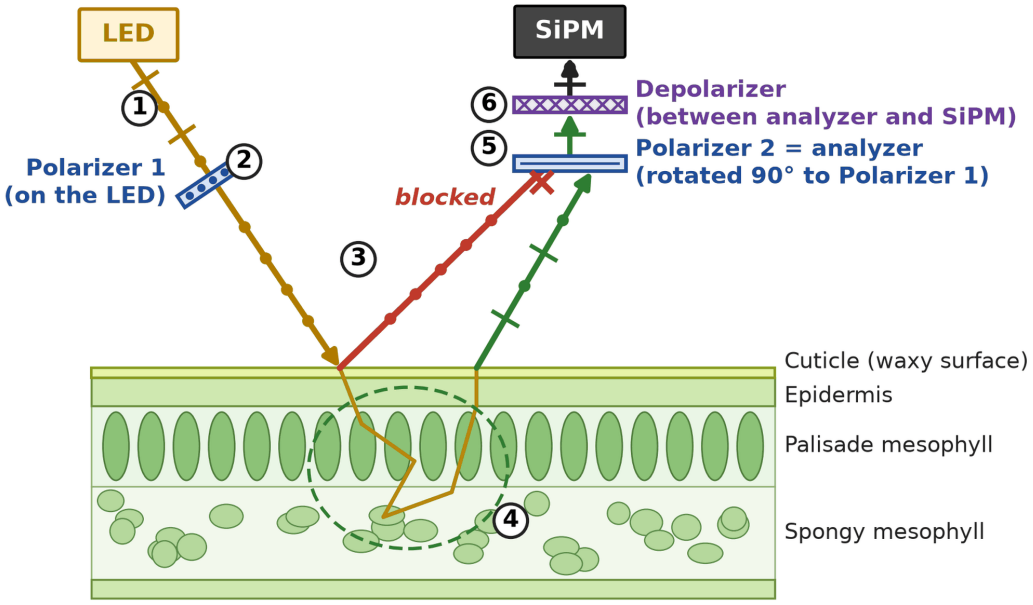
These conclusions remain limited to one growing season and location, a held-out test set containing 59 plants from nine plots, and a surface-reflection experiment involving six leaves. The model's performance must therefore be validated across additional seasons, locations, cultivars, growth stages, and environmental conditions before its generalizability can be established. At its present stage, the prototype is most appropriately used as a research instrument for controlled and relative measurements of maize leaf reflectance. The available evidence does not yet support its use for reporting absolute tissue nitrate-N concentrations or recommending fertilizer application rates.

5.2 Recommendations

The next sensing prototype should consider using cross-polarized illumination and detection. A linear polarizer over the LEDs, a second linear cross polarizer (analyzer) in front of the SiPM, oriented perpendicular to the first, and a depolarizer would suppress much of the surface reflection while retaining more of the diffusely scattered tissue signal as shown in Figure 44. Specular reflection from the cuticle largely keeps the polarization set by the LED polarizer and is therefore blocked by the crossed analyzer, whereas light scattered many times inside the mesophyll re-emerges as diffuse reflection with mixed polarization, so part of it passes the analyzer (Figure 44). A depolarizer placed between the analyzer and the SiPM would then scramble the polarization of the transmitted light, so that the reading does not depend on any polarization sensitivity of the detector. Readings taken with the analyzer parallel and crossed to the first polarizer should be collected from the same leaves; their difference measures the specular component directly. These readings should also be compared with unpolarized readings from the same leaves to test whether this change improves repeatability and prediction. The angle experiment provides a physical basis for this design change.

Ambient light and measurement geometry require tighter control in future studies. An enclosed leaf clip or shrouded sensing head should maintain a fixed sensor-to-leaf distance, viewing angle, and illuminated area while shielding the detector from sunlight. Each measurement session should also include LED-off readings to quantify ambient-light contributions and repeated white-reference measurements to monitor short-term drift in sensor response.

Direct measurements of chlorophyll, such as SPAD readings, and total leaf nitrogen should be collected alongside nitrate-N. Although these variables represent different aspects of plant nitrogen status, chlorophyll and total leaf nitrogen are more directly related to visible and near-infrared reflectance than nitrate-N. Including them would help determine whether the sensor responds primarily to chlorophyll and overall nitrogen status, rather than directly to nitrate accumulation.



① The LED emits unpolarized light.
② Polarizer 1 on the LED passes only one polarization direction (dots).
③ Specular reflection at the waxy cuticle keeps that polarization (dots).
④ Light that enters the leaf is scattered many times in the mesophyll and re-emerges as diffuse reflection with mixed polarization.
⑤ The analyzer, rotated 90° to Polarizer 1, blocks the specular reflection and passes the in-plane part of the diffuse light.
⑥ The depolarizer scrambles the transmitted light so the SiPM reading does not depend on the detector's sensitivity to polarization.

Schematic, not to scale. In the sensor the LEDs surround the central SiPM; source and detector are drawn apart here to separate the reflection paths.

Figure 44. Cross-polarization concept for the next prototype: a linear polarizer on the LED, a crossed analyzer followed by a depolarizer in front of the SiPM, and the specular and diffuse reflection.

References

Abendroth, L.J., Elmore, R.W., Boyer, M.J., Marlay, S.K. (2011). Corn Growth and Development. Iowa State University Extension, PMR 1009.

Analog Devices. (2021). LT8337/LT8337-1: 28V, 5A Low IQ Synchronous Step-Up Silent Switcher with PassThru (Datasheet, Rev. A).

Benjamini, Y., Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. Journal of the Royal Statistical Society: Series B, 57(1), 289-300.

Berger, K., Verrelst, J., Féret, J.-B., Wang, Z., Wocher, M., Strathmann, M., Danner, M., Mauser, W., Hank, T. (2020). Crop nitrogen monitoring: recent progress and principal developments in the context of imaging spectroscopy missions. Remote Sensing of Environment, 242, 111758. https://doi.org/10.1016/j.rse.2020.111758

Breusch, T.S., Pagan, A.R. (1979). A simple test for heteroscedasticity and random coefficient variation. Econometrica, 47(5), 1287-1294.

Candy, J.C., Temes, G.C. (Eds.) (1992). Oversampling Delta-Sigma Data Converters: Theory, Design, and Simulation. IEEE Press.

Cao, Q., Miao, Y., Wang, H., Huang, S., Cheng, S., Khosla, R., Jiang, R. (2013). Non-destructive estimation of rice plant nitrogen status with Crop Circle multispectral active canopy sensor. Field Crops Research, 154, 133-144. https://doi.org/10.1016/j.fcr.2013.08.005

Crawford, N.M. (1995). Nitrate: nutrient and signal for plant growth. The Plant Cell, 7(7), 859-868. https://doi.org/10.1105/tpc.7.7.859

Glenn, E.P., Huete, A.R., Nagler, P.L., Nelson, S.G. (2008). Relationship between remotely-sensed vegetation indices, canopy attributes and plant physiological processes: what vegetation indices can and cannot tell us about the landscape. Sensors, 8(4), 2136-2160. https://doi.org/10.3390/s8042136

Grant, L. (1987). Diffuse and specular characteristics of leaf reflectance. Remote Sensing of Environment, 22(2), 309-322. https://doi.org/10.1016/0034-4257(87)90064-2

Grant, L., Daughtry, C.S.T., Vanderbilt, V.C. (1993). Polarized and specular reflectance variation with leaf surface features. Physiologia Plantarum, 88(1), 1-9. https://doi.org/10.1111/j.1399-3054.1993.tb01753.x

Gundacker, S., Heering, A. (2020). The silicon photomultiplier: fundamentals and applications of a modern solid-state photon detector. Physics in Medicine & Biology, 65(17), 17TR01.

Hasituya, Li, F., Elsayed, S., Hu, Y., Schmidhalter, U. (2020). Passive reflectance sensing using optimized two- and three-band spectral indices for quantifying the total nitrogen yield of maize. Computers and Electronics in Agriculture, 173, 105403. https://doi.org/10.1016/j.compag.2020.105403

He, L., Song, X., Feng, W., Guo, B.-B., Zhang, Y.-S., Wang, Y.-H., Wang, C.-Y., Guo, T.-C. (2016). Improved remote sensing of leaf nitrogen concentration in winter wheat using multi-angular hyperspectral data. Remote Sensing of Environment, 174, 122-133. https://doi.org/10.1016/j.rse.2015.12.007

Hecht, E. (2017). Optics (5th ed.). Pearson.

Li, F., Mistele, B., Hu, Y., Chen, X., Schmidhalter, U. (2014). Reflectance estimation of canopy nitrogen content in winter wheat using optimised hyperspectral spectral indices and partial least squares regression. European Journal of Agronomy, 52, 198-209. https://doi.org/10.1016/j.eja.2013.09.006

Li, H., Zhang, J., Xu, K., Jiang, X., Zhu, Y., Cao, W., Ni, J. (2021). Spectral monitoring of wheat leaf nitrogen content based on canopy structure information compensation. Computers and Electronics in Agriculture, 190, 106434. https://doi.org/10.1016/j.compag.2021.106434

Li, S., Jiao, J., Chen, J., Wang, C. (2022). A new polarization-based vegetation index to improve the accuracy of vegetation health detection by eliminating specular reflection of vegetation. IEEE Transactions on Geoscience and Remote Sensing, 60, 1-18. https://doi.org/10.1109/TGRS.2022.3211503

Liu, M., Sun, Z., Lu, S., Omasa, K. (2022). Combining multiangular, polarimetric, and hyperspectral measurements to estimate leaf nitrogen concentration from different plant species. IEEE Transactions on Geoscience and Remote Sensing, 60, 1-15. https://doi.org/10.1109/TGRS.2021.3106786

Merzlyak, Mark N., Olga B. Chivkunova, T. B. Melø, and K. Razi Naqvi. 2002. "Does a Leaf Absorb Radiation in the near Infrared (780-900 Nm) Region? A New Approach to Quantifying Optical Reflection, Absorption and Transmission of Leaves." *Photosynthesis Research* 72(3):263–70. doi:10.1023/A:1019823303951.

onsemi. (n.d.). MICROFC-30035-SMT-TR1 SiPM Sensor Datasheet.

Padilla, F.M., De Souza, R., Peña-Fleitas, M.T., Grasso, R., Gallardo, M., Thompson, R.B. (2019). Influence of time of day on measurement with chlorophyll meters and canopy reflectance sensors of different crop N status. Precision Agriculture, 20(6), 1087-1106. https://doi.org/10.1007/s11119-019-09641-1

Raun, W.R., Johnson, G.V. (1999). Improving nitrogen use efficiency for cereal production. Agronomy Journal, 91(3), 357-363. https://doi.org/10.2134/agronj1999.00021962009100030001x

Raun, W.R., Solie, J.B., Johnson, G.V., Stone, M.L., Lukina, E.V., Thomason, W.E., Schepers, J.S. (2001). In-season prediction of potential grain yield in winter wheat using canopy reflectance. Agronomy Journal, 93(1), 131-138.

Rouse, J.W., Haas, R.H., Schell, J.A., Deering, D.W. (1973). Monitoring vegetation systems in the Great Plains with ERTS. Third ERTS Symposium, NASA SP-351, 309-317.

Scharf, P.C., Shannon, D.K., Palm, H.L., Sudduth, K.A., Drummond, S.T., Kitchen, N.R., Mueller, L.J., Hubbard, V.C., Oliveira, L.F. (2011). Sensor-based nitrogen applications out-performed producer-chosen rates for corn in on-farm demonstrations. Agronomy Journal, 103(6), 1683-1691. https://doi.org/10.2134/agronj2011.0164

Schlemmer, M., Gitelson, A., Schepers, J., Ferguson, R., Peng, Y., Shanahan, J., Rundquist, D. (2013). Remote estimation of nitrogen and chlorophyll contents in maize at leaf and canopy levels. International Journal of Applied Earth Observation and Geoinformation, 25, 47-54.

Schubert, E.F. (2006). Light-Emitting Diodes (2nd ed.). Cambridge University Press.

Shanahan, J.F., Kitchen, N.R., Raun, W.R., Schepers, J.S. (2008). Responsive in-season nitrogen management for cereals. Computers and Electronics in Agriculture, 61(1), 51-62. https://doi.org/10.1016/j.compag.2007.06.006

Shapiro, S.S., Wilk, M.B. (1965). An analysis of variance test for normality (complete samples). Biometrika, 52(3/4), 591-611.

STMicroelectronics. (2008). AN2668: Improving STM32F101xx and STM32F103xx ADC resolution by oversampling (Application Note, DocID14183 Rev 1).

STMicroelectronics. (2022). STM32H725xE/G: Arm Cortex-M7 32-bit MCU datasheet (DS13311/DS13195).

Ustin, S.L., Jacquemoud, S. (2020). How the optical properties of leaves modify the absorption and scattering of energy and enhance leaf functionality. In J. Cavender-Bares, J.A. Gamon, & P.A. Townsend (Eds.), Remote Sensing of Plant

Biodiversity (pp. 349-384). Springer International Publishing. https://doi.org/10.1007/978-3-030-33157-3_14

Verrelst, J., Rivera, J.P., Veroustraete, F., Muñoz-Marí, J., Clevers, J.G.P.W., Camps-Valls, G., Moreno, J. (2015). Experimental Sentinel-2 LAI estimation using parametric, non-parametric and physical retrieval methods – a comparison. ISPRS Journal of Photogrammetry and Remote Sensing, 108, 260-272. https://doi.org/10.1016/j.isprsjprs.2015.04.013

Winterhalter, L., Mistele, B., Schmidhalter, U. (2012). Assessing the vertical footprint of reflectance measurements to characterize nitrogen uptake and biomass distribution in maize canopies. Field Crops Research, 129, 14-20. https://doi.org/10.1016/j.fcr.2012.01.007

Woolley, J.T. (1971). Reflectance and transmittance of light by leaves. Plant Physiology, 47(5), 656-662. https://doi.org/10.1104/pp.47.5.656

Appendix A

Derivation of the General Expression for Detected Optical Power

Having established the Silicon Photomultiplier (SiPM) responsivity equations and the two-stage amplification chain in the preceding section, this appendix presents the procedure for recovering the optical power incident on the SiPM active area from a measured, background-corrected digital voltage produced in the firmware. The reason behind this method is to determine the fraction of optical power intercepted by the photodetector relative to the theoretical LED output and to convert the corrected digital voltage to estimated incident optical power.

Find Total Gain After Transimpedance Amplification (TIA) Stage

Between the output of the TIA and the firmware-reported value, the signal passes through one additional, wavelength-independent gain stage: the fixed second-stage voltage amplifier gain, $A_2 = 1.1$ V/V. Because this gain is applied identically at every wavelength, it is defined as the total gain after TIA amplification, A_{post}, given in Equation 30:

$$A_{post} = A_2 = 1.1 \tag{30}$$

where:

A_{post} is the total gain applied downstream of the TIA, equal to 1.1 at every wavelength, and

A_2 is the fixed second-stage voltage-amplifier gain (1.1 V/V).

Recover the TIA Output Voltage

Subsequently, the voltage after the TIA stage was obtained using Equation 31.

$$V_{out_TIA,\lambda} = \frac{V_{corrected,\lambda}}{A_{post}} \tag{31}$$

where:

V_{out_TIA} is the voltage at the output of the first-stage transimpedance amplifier, prior to any subsequent amplification, for each wavelength,

$V_{corrected,\lambda}$ is the digital voltage after subtracting the voltage recorded when the LEDs where turned off, for each wavelength, and

A_{post} is the total gain after TIA stage amplification.

Recovering the SiPM Output Current

The transimpedance amplifier converts the SiPM photocurrent to a proportional output voltage through the fixed feedback resistor $R_{TIA} = 330\ \Omega$, shown in Figure 4. Inverting this relationship recovers the SiPM output current, $I_{SiPM,\lambda}$, defined in Equation 32.

$$I_{SiPM,\lambda} = \frac{V_{out_TIA,\lambda}}{R_{TIA}} \tag{32}$$

where:

R_{TIA} is the fixed transimpedance gain resistor common to all wavelength measurements, and

$V_{out_TIA,\lambda}$ is the output voltage after the transimpedance amplification stage.

Estimate Photon Energy

Because the SiPM converts incident photons to charge carriers, the optical power incident on its active area was related to the photocurrent through the energy carried by a single photon at the wavelength of interest. The photon energy was obtained using Equation 33.

$$E_{photon} = \frac{hc}{\lambda} \tag{33}$$

where:

h is the Planck's constant (6.626×10^{-34} J·s),

c is the speed of light (3×10^{8} m/s), and

λ is the wavelength of light.

Estimate Effective Responsivity for Each Wavelength

The effective responsivity was estimated with the photo current obtain in Equation 32 using Equation 34. For each wavelength Photo Detection Efficiency PDE values from the datasheet. Each point was found by tracing a vertical line from the desired wavelength on the x-axis up to where it meets the 0.8 V curve, then a horizontal line from that intersection across to the y-axis to read off the PDE.

$$R_{eff} = \frac{(PDE)\,(G)\,(q)}{E_{photon}} \tag{34}$$

where:

PDE is the photodetection efficiency,

G is the avalanche multiplication gain (1×10^5 at 25.5 V obtained from the datasheet),

q is the elementary charge (1.6×10^{-19} C), and

$E_{photon} = hc/\lambda$ is the photon energy at wavelength λ.

Estimate Detected Optical Power

The detected optical power incident on the SiPM active area, $P_{det,\lambda}$, was obtained by dividing the recovered SiPM current by the effective responsivity at that wavelength, as defined in Equation 35:

$$P_{det,\lambda} = \frac{I_{SiPM,\lambda}}{R_{eff,\lambda}} \tag{35}$$

where:

$P_{det,\lambda}$ is the power detected by the Silicon Photo Multiplier (SiPM) per wavelength,

$I_{SiPM,\lambda}$ is the photo-current from the SiPM per wavelength, and

$R_{eff,\lambda}$ is the effective responsivity per wavelength.

General Expression

Finally, Equation 30, Equation 31, Equation 32Equation 33, Equation 34 and Equation 35 are combined algebraically into a single general expression for estimating detected optical power at any wavelength in Equation 36:

$$P_{det,\lambda} = \frac{V_{corrected,\lambda} \times h \times c}{A_{post} \times R_{TIA} \times PDE_{\lambda} \times G \times q \times \lambda} \tag{36}$$

where:

$P_{det,\lambda}$ the optical power that reaches the active area of the SiPM after the following:

Illumination of the target by the LED source,

reflection from the target,

Propagation through the sensor's collection optics and any associated optical losses along that path, and

PDE_λ is the photo detection efficiency per wavelength.

It is therefore neither the total optical power emitted by the source LED nor the total optical power reflected by the target as a whole; it is the fraction of the reflected radiant flux that is ultimately collected and converted by the SiPM. This procedure is repeated independently for each LED wavelength by substituting the corresponding wavelength-specific parameters.

This working-backward procedure rests on the following assumptions:

- Linear operation of the SiPM and of both amplifier stages across the measured signal range;
- The absence of ADC or amplifier saturation at any measured wavelength;
- A stable SiPM overvoltage of 0.8 V during all measurements;
- PDE values, taken from (Figure 5), that are representative of the specific SiPM device used.

Appendix B

Complete Plant-Level Spectral Dataset

The table below reproduces the complete plant-level spectral dataset underlying this thesis, 234 individual-plant records spanning both plantings (April and May), all three growth stages (V10, R1, R3), and all six nitrogen-treatment rates. This is broader than the 233-plant subset used in the regression analysis in the main text, which further excludes one sample flagged as moldy by the KSU Soil Testing Lab (Plot 302, second leaf) and is restricted to plants with a matched laboratory nitrate-N result. Blank 850 nm, NDVI, and NIR:Red cells for some rounds reflect a dead detector channel on those sampling dates, not a processing omission. Columns identifying the individual plant and sampling event and redundant with those shown (Plant_Number, Planting_Date, Block) and internal data-provenance fields (Source_Pipeline, Notes) are omitted here for width; the complete file, including those fields, is archived with the project data.

Appendix Table B.1. Complete plant-level spectral dataset (n = 234), all growth stages, plantings, and nitrogen-treatment rates, dark-enclosure measurements.

Plant ID	Plot	Round	Stage	N Treatment	Date	660 nm	670 nm	700 nm	750 nm	780 nm	850 nm
102_P1	102	V10_April_0616	V10	200 split	06/16/26	0.957	0.338	1.013	1.220	1.065	1.877
102_P2	102	V10_April_0616	V10	200 split	06/16/26	0.981	0.338	0.989	1.193	1.077	1.826
102_P3	102	V10_April_0616	V10	200 split	06/16/26	0.991	0.319	0.855	1.200	1.006	1.843
103_P1	103	V10_April_0616	V10	100	06/16/26	0.851	0.331	0.855	1.364	0.857	1.599
103_P2	103	V10_April_0616	V10	100	06/16/26	0.836	0.354	0.800	1.038	0.892	1.549
103_P3	103	V10_April_0616	V10	100	06/16/26	0.704	0.378	0.735	1.071	0.908	1.688
302_P1	302	V10_April_0616	V10	100	06/16/26	0.815	0.365	0.805	1.237	0.956	1.647
302_P2	302	V10_April_0616	V10	100	06/16/26	0.873	0.415	0.733	1.111	0.928	1.570
302_P3	302	V10_April_0616	V10	100	06/16/26	0.865	0.430	0.836	1.106	0.937	1.537
205_P1	205	V10_April_0616	V10	100	06/16/26	0.938	0.319	0.848	1.151	1.061	1.718
205_P2	205	V10_April_0616	V10	100	06/16/26	0.843	0.366	0.961	1.132	0.922	1.928
205_P3	205	V10_April_0616	V10	100	06/16/26	0.761	0.363	0.711	1.117	0.990	1.603
106_P1	106	V10_April_0616	V10	200	06/16/26	0.879	0.238	0.830	1.126	0.875	1.541
106_P2	106	V10_April_0616	V10	200	06/16/26	0.917	0.369	0.821	1.410	0.858	1.699
106_P3	106	V10_April_0616	V10	200	06/16/26	1.174	0.685	1.188	1.258	0.321	1.726
303_P1	303	V10_April_0616	V10	200 split	06/16/26	0.773	0.318	0.791	1.187	0.919	1.662
303_P2	303	V10_April_0616	V10	200 split	06/16/26	0.959	0.377	0.888	1.179	0.956	1.849
303_P3	303	V10_April_0616	V10	200 split	06/16/26	0.808	0.358	0.724	1.148	0.948	1.643
203_P1	203	V10_April_0616	V10	50	06/16/26	1.751	0.264	0.816	1.072	0.889	1.706
203_P2	203	V10_April_0616	V10	50	06/16/26	0.800	0.352	1.826	1.110	0.905	1.545
203_P3	203	V10_April_0616	V10	50	06/16/26	0.856	0.322	0.822	0.886	0.971	1.517
105_P1	105	V10_April_0616	V10	50	06/16/26	0.851	0.351	0.773	1.207	0.939	1.723
105_P2	105	V10_April_0616	V10	50	06/16/26	0.881	0.310	0.855	1.062	0.991	1.533
105_P3	105	V10_April_0616	V10	50	06/16/26	0.925	0.327	0.867	1.096	0.949	1.585
306_P1	306	V10_April_0616	V10	0	06/16/26	0.986	0.326	0.854	1.307	0.990	1.499
306_P2	306	V10_April_0616	V10	0	06/16/26	1.057	0.405	0.916	1.231	0.997	1.771
306_P3	306	V10_April_0616	V10	0	06/16/26	0.848	0.345	0.797	1.082	0.951	1.411

Plant ID	Plot	Round	Stage	N Treatment	Date	660 nm	670 nm	700 nm	750 nm	780 nm	850 nm
202_P1	202	V10_April_0616	V10	0	06/16/26	0.860	0.384	0.741	1.315	0.967	1.594
202_P2	202	V10_April_0616	V10	0	06/16/26	0.964	0.423	0.883	1.143	0.924	1.659
202_P3	202	V10_April_0616	V10	0	06/16/26	0.908	0.381	0.801	1.266	0.976	1.622
201_P1	201	V10_April_0616	V10	200	06/16/26	0.778	0.352	0.788	1.267	0.976	1.590
201_P2	201	V10_April_0616	V10	200	06/16/26	1.318	0.264	0.679	1.100	0.808	1.357
201_P3	201	V10_April_0616	V10	200	06/16/26	0.791	0.295	0.825	1.046	0.860	1.480
304_P1	304	V10_April_0616	V10	150	06/16/26	0.948	0.288	0.764	1.119	0.935	0.139
304_P2	304	V10_April_0616	V10	150	06/16/26	0.859	0.408	0.850	1.200	0.843	1.588
304_P3	304	V10_April_0616	V10	150	06/16/26	0.924	0.325	0.985	1.041	0.938	1.692
104_P1	104	V10_April_0616	V10	150	06/16/26	0.935	0.327	0.842	1.093	0.957	1.840
104_P2	104	V10_April_0616	V10	150	06/16/26	0.813	0.338	0.627	1.073	0.880	1.564
104_P3	104	V10_April_0616	V10	150	06/16/26	0.834	0.322	1.314	1.174	0.873	1.703
204_P1	204	V10_April_0616	V10	150	06/16/26	0.976	0.361	0.861	1.104	0.891	1.726
204_P2	204	V10_April_0616	V10	150	06/16/26	0.815	0.326	0.787	1.256	0.859	1.564
204_P3	204	V10_April_0616	V10	150	06/16/26	0.947	0.331	0.810	1.115	0.856	1.532
101_P1	101	V10_April_0616	V10	0	06/16/26	0.822	0.331	0.704	1.179	0.839	1.694
101_P2	101	V10_April_0616	V10	0	06/16/26	0.882	0.321	0.927	0.986	0.818	1.568
101_P3	101	V10_April_0616	V10	0	06/16/26	0.998	0.355	0.826	0.984	0.951	1.760
206_P1	206	V10_April_0616	V10	200 split	06/16/26	0.863	0.314	1.836	1.210	0.892	2.544
206_P2	206	V10_April_0616	V10	200 split	06/16/26	0.893	0.224	0.823	1.057	0.822	1.579
206_P3	206	V10_April_0616	V10	200 split	06/16/26	0.835	0.348	0.801	0.936	0.845	1.822
301_P1	301	V10_April_0616	V10	200	06/16/26	0.974	0.385	0.876	1.183	0.809	1.804
301_P2	301	V10_April_0616	V10	200	06/16/26	0.891	0.343	0.777	1.057	0.915	1.780
301_P3	301	V10_April_0616	V10	200	06/16/26	0.907	0.277	0.745	1.078	0.832	1.631
305_P1	305	V10_April_0616	V10	50	06/16/26	0.885	0.358	0.800	1.087	0.962	1.474
305_P2	305	V10_April_0616	V10	50	06/16/26	0.854	0.335	0.844	1.878	0.885	2.544
305_P3	305	V10_April_0616	V10	50	06/16/26	0.798	0.336	0.842	1.096	0.876	1.879
101_P1	101	R1_April_0708	R1	0	07/08/26	0.989	0.341	0.929	1.270	0.880	1.614
101_P2	101	R1_April_0708	R1	0	07/08/26	0.992	0.388	0.886	1.209	0.916	1.565
102_P1	102	R1_April_0708	R1	200 split	07/08/26	0.914	0.452	0.889	1.254	1.035	1.663

Plant ID	Plot	Round	Stage	N Treatment	Date	660 nm	670 nm	700 nm	750 nm	780 nm	850 nm
102_P2	102	R1_April_0708	R1	200 split	07/08/26	0.944	0.345	0.819	1.265	1.003	1.713
103_P1	103	R1_April_0708	R1	100	07/08/26	1.098	0.423	0.878	1.052	0.818	1.322
103_P2	103	R1_April_0708	R1	100	07/08/26	1.199	0.425	0.920	1.122	0.920	1.658
106_P1	106	R1_April_0708	R1	200	07/08/26	0.979	0.389	0.882	1.194	0.847	1.502
106_P2	106	R1_April_0708	R1	200	07/08/26	1.057	0.328	0.783	1.332	0.946	1.683
105_P1	105	R1_April_0708	R1	50	07/08/26	0.936	0.278	0.806	1.019	0.915	1.503
105_P2	105	R1_April_0708	R1	50	07/08/26	1.032	0.378	0.855	1.164	1.038	2.014
104_P1	104	R1_April_0708	R1	150	07/08/26	0.985	0.448	0.982	1.351	0.642	1.590
104_P2	104	R1_April_0708	R1	150	07/08/26	1.028	0.334	0.898	1.252	0.990	1.581
201_P1	201	R1_April_0708	R1	200	07/08/26	-0.580	0.355	0.848	1.517	0.800	1.647
201_P2	201	R1_April_0708	R1	200	07/08/26	0.918	0.358	0.902	1.164	1.007	1.524
202_P1	202	R1_April_0708	R1	0	07/08/26	0.982	0.403	0.922	1.194	0.789	1.776
202_P2	202	R1_April_0708	R1	0	07/08/26	1.171	0.424	0.927	1.138	0.932	1.658
203_P1	203	R1_April_0708	R1	50	07/08/26	0.959	0.355	0.867	1.217	0.824	1.632
203_P2	203	R1_April_0708	R1	50	07/08/26	1.004	0.357	0.859	1.194	0.944	1.563
206_P1	206	R1_April_0708	R1	200 split	07/08/26	0.958	0.324	0.765	1.299	0.864	1.487
206_P2	206	R1_April_0708	R1	200 split	07/08/26	0.911	0.402	0.853	1.127	0.862	1.510
205_P1	205	R1_April_0708	R1	100	07/08/26	1.032	0.315	0.797	1.103	0.692	1.421
205_P2	205	R1_April_0708	R1	100	07/08/26	0.730	0.332	0.869	1.228	0.860	1.615
204_P1	204	R1_April_0708	R1	150	07/08/26	0.974	0.324	0.789	1.064	0.807	1.667
204_P2	204	R1_April_0708	R1	150	07/08/26	0.971	0.347	0.866	1.077	0.855	1.377
301_P1	301	R1_April_0708	R1	200	07/08/26	0.932	0.316	0.688	0.987	0.818	1.637
301_P2	301	R1_April_0708	R1	200	07/08/26	0.751	0.604	0.793	1.132	0.862	1.514
302_P1	302	R1_April_0708	R1	100	07/08/26	0.918	0.323	0.706	1.312	0.883	1.595
302_P2	302	R1_April_0708	R1	100	07/08/26	0.974	0.345	0.824	1.122	0.885	1.483
303_P1	303	R1_April_0708	R1	200 split	07/08/26	0.926	0.382	0.832	1.171	0.963	1.589
303_P2	303	R1_April_0708	R1	200 split	07/08/26	1.040	0.319	1.055	1.199	0.873	1.650
306_P1	306	R1_April_0708	R1	0	07/08/26	0.973	0.300	0.786	1.260	0.768	1.468
306_P2	306	R1_April_0708	R1	0	07/08/26	1.068	0.283	0.918	1.055	0.880	1.642
305_P1	305	R1_April_0708	R1	50	07/08/26	0.995	0.383	0.800	1.182	0.855	1.505

Plant ID	Plot	Round	Stage	N Treatment	Date	660 nm	670 nm	700 nm	750 nm	780 nm	850 nm
305_P2	305	R1_April_0708	R1	50	07/08/26	0.998	0.407	0.912	1.069	0.870	1.500
304_P1	304	R1_April_0708	R1	150	07/08/26	0.874	0.322	0.942	0.163	0.803	1.582
304_P2	304	R1_April_0708	R1	150	07/08/26	0.976	0.424	0.874	1.049	0.719	1.513
605_P1	605	V10_May_0708	V10	100	07/08/26	0.943	0.399	0.923	1.162	0.854	1.467
605_P2	605	V10_May_0708	V10	100	07/08/26	1.024	0.362	0.727	1.002	0.804	1.341
606_P1	606	V10_May_0708	V10	150	07/08/26	0.988	0.312	0.819	1.119	0.831	1.455
606_P2	606	V10_May_0708	V10	150	07/08/26	0.825	0.301	1.511	1.031	0.843	1.383
604_P1	604	V10_May_0708	V10	50	07/08/26	0.897	0.378	0.801	1.058	0.832	1.487
604_P2	604	V10_May_0708	V10	50	07/08/26	1.187	0.347	0.726	1.094	0.710	1.367
603_P1	603	V10_May_0708	V10	200	07/08/26	0.902	0.361	0.876	1.152	0.839	1.344
603_P2	603	V10_May_0708	V10	200	07/08/26	1.016	0.338	0.896	1.122	0.840	1.561
601_P1	601	V10_May_0708	V10	200 split	07/08/26	0.911	0.334	0.899	0.975	0.840	1.486
601_P2	601	V10_May_0708	V10	200 split	07/08/26	0.973	0.260	0.881	1.092	0.862	1.606
602_P1	602	V10_May_0708	V10	0	07/08/26	0.810	0.375	0.736	1.092	0.872	2.033
602_P2	602	V10_May_0708	V10	0	07/08/26	0.922	0.347	0.820	1.107	0.836	1.563
505_P1	505	V10_May_0708	V10	50	07/08/26	0.921	0.322	0.773	1.036	1.178	1.641
505_P2	505	V10_May_0708	V10	50	07/08/26	0.967	0.378	0.725	1.112	0.860	1.454
506_P1	506	V10_May_0708	V10	200	07/08/26	0.903	0.324	0.825	1.130	0.838	1.551
506_P2	506	V10_May_0708	V10	200	07/08/26	0.806	0.294	0.780	1.119	0.873	1.520
503_P1	503	V10_May_0708	V10	200 split	07/08/26	0.976	0.375	0.886	1.092	0.882	1.444
503_P2	503	V10_May_0708	V10	200 split	07/08/26	0.994	0.327	0.927	1.112	0.521	1.697
504_P1	504	V10_May_0708	V10	150	07/08/26	0.866	0.321	0.814	1.099	0.870	1.483
504_P2	504	V10_May_0708	V10	150	07/08/26	1.070	0.285	0.596	1.038	0.765	1.376
501_P1	501	V10_May_0708	V10	0	07/08/26	0.873	0.300	0.800	1.647	0.718	1.382
501_P2	501	V10_May_0708	V10	0	07/08/26	0.890	0.355	0.247	1.097	0.799	1.342
502_P1	502	V10_May_0708	V10	100	07/08/26	0.841	0.302	0.753	1.002	0.810	1.334
502_P2	502	V10_May_0708	V10	100	07/08/26	0.872	0.292	0.930	0.883	0.810	1.511
405_P1	405	V10_May_0708	V10	150	07/08/26	0.918	0.336	0.868	1.161	0.881	1.532
405_P2	405	V10_May_0708	V10	150	07/08/26	0.884	0.276	0.761	1.102	0.809	1.516
406_P1	406	V10_May_0708	V10	0	07/08/26	0.917	0.329	0.847	1.051	0.820	1.405

Plant ID	Plot	Round	Stage	N Treatment	Date	660 nm	670 nm	700 nm	750 nm	780 nm	850 nm
406_P2	406	V10_May_0708	V10	0	07/08/26	0.827	0.301	0.714	1.053	0.780	1.514
403_P1	403	V10_May_0708	V10	50	07/08/26	0.822	0.296	0.777	1.041	0.760	0.544
403_P2	403	V10_May_0708	V10	50	07/08/26	0.871	0.284	0.801	0.944	0.859	1.564
404_P1	404	V10_May_0708	V10	200	07/08/26	0.950	0.299	0.798	0.982	0.719	1.330
404_P2	404	V10_May_0708	V10	200	07/08/26	0.802	0.412	0.663	1.080	0.729	1.383
401_P1	401	V10_May_0708	V10	100	07/08/26	0.945	0.271	0.723	1.091	0.856	1.224
401_P2	401	V10_May_0708	V10	100	07/08/26	1.020	0.393	0.724	1.043	0.700	1.337
402_P1	402	V10_May_0708	V10	200 split	07/08/26	0.927	0.339	0.724	1.075	0.767	1.315
402_P2	402	V10_May_0708	V10	200 split	07/08/26	0.975	0.304	0.776	1.002	0.825	1.484
101_P1	101	R3_April_0727	R3	0	07/27/26	1.678	0.641	1.447	1.870	1.473	
101_P2	101	R3_April_0727	R3	0	07/27/26	1.613	0.650	1.415	1.870	1.643	
102_P1	102	R3_April_0727	R3	200 split	07/27/26	1.607	0.599	1.410	1.886	1.656	
102_P2	102	R3_April_0727	R3	200 split	07/27/26	1.569	0.663	1.375	1.753	1.565	
103_P1	103	R3_April_0727	R3	100	07/27/26	1.604	0.598	1.410	1.768	1.571	
103_P2	103	R3_April_0727	R3	100	07/27/26	1.425	0.504	1.307	1.654	1.462	
104_P1	104	R3_April_0727	R3	150	07/27/26	1.598	0.675	1.482	1.833	1.611	
104_P2	104	R3_April_0727	R3	150	07/27/26	1.538	0.660	1.419	1.755	1.529	
105_P1	105	R3_April_0727	R3	50	07/27/26	1.353	0.627	1.353	1.635	1.508	
105_P2	105	R3_April_0727	R3	50	07/27/26	1.439	0.474	1.198	1.616	1.455	
106_P1	106	R3_April_0727	R3	200	07/27/26	1.403	0.557	1.205	1.583	1.428	
106_P2	106	R3_April_0727	R3	200	07/27/26	1.274	0.404	1.104	1.529	1.417	
201_P1	201	R3_April_0727	R3	200	07/27/26	1.400	0.560	1.284	1.647	1.399	
201_P2	201	R3_April_0727	R3	200	07/27/26	1.097	0.462	1.003	1.494	1.453	
202_P1	202	R3_April_0727	R3	0	07/27/26	1.389	0.615	1.273	1.621	1.410	
202_P2	202	R3_April_0727	R3	0	07/27/26	1.335	0.533	1.233	1.632	1.452	
203_P1	203	R3_April_0727	R3	50	07/27/26	1.304	0.550	1.264	1.536	1.371	
203_P2	203	R3_April_0727	R3	50	07/27/26	1.300	0.539	1.189	1.554	1.361	
204_P1	204	R3_April_0727	R3	150	07/27/26						
204_P2	204	R3_April_0727	R3	150	07/27/26	1.079	0.531	1.055	1.461	1.314	2.059
205_P1	205	R3_April_0727	R3	100	07/27/26	0.827	0.383	0.695	1.280	1.355	1.485

Plant ID	Plot	Round	Stage	N Treatment	Date	660 nm	670 nm	700 nm	750 nm	780 nm	850 nm
205_P2	205	R3_April_0727	R3	100	07/27/26	1.022	0.558	1.088	1.491	1.361	1.622
206_P1	206	R3_April_0727	R3	200 split	07/27/26	1.175	0.513	1.197	1.538	1.418	2.145
206_P2	206	R3_April_0727	R3	200 split	07/27/26	1.228	0.411	1.025	1.542	1.398	1.987
301_P1	301	R3_April_0727	R3	200	07/27/26	1.158	0.438	1.087	1.524	1.442	2.149
301_P2	301	R3_April_0727	R3	200	07/27/26	0.946	0.611	0.703	1.469	1.402	1.797
302_P1	302	R3_April_0727	R3	100	07/27/26	1.036	0.510	1.039	1.417	1.409	1.971
302_P2	302	R3_April_0727	R3	100	07/27/26	1.038	0.651	1.010	1.463	1.398	1.923
303_P1	303	R3_April_0727	R3	200 split	07/27/26	0.793	0.556	0.733	1.316	1.378	1.700
303_P2	303	R3_April_0727	R3	200 split	07/27/26	0.993	0.539	0.898	1.451	1.392	1.656
304_P1	304	R3_April_0727	R3	150	07/27/26	0.817	0.484	0.521	1.134	1.220	1.334
304_P2	304	R3_April_0727	R3	150	07/27/26	0.968	0.497	0.690	1.419	1.389	1.764
305_P1	305	R3_April_0727	R3	50	07/27/26	1.099	0.481	1.064	1.528	1.410	1.899
305_P2	305	R3_April_0727	R3	50	07/27/26	1.092	0.577	0.997	1.497	1.387	1.951
306_P1	306	R3_April_0727	R3	0	07/27/26	1.159	0.607	1.086	1.515	1.285	1.939
306_P2	306	R3_April_0727	R3	0	07/27/26	1.057	0.525	1.045	1.416	1.318	1.645
606_P1	606	R1_May_0727	R1	150	07/27/26	0.869	0.477	0.649	1.238	1.323	1.591
606_P2	606	R1_May_0727	R1	150	07/27/26	0.885	0.561	0.611	1.185	1.278	1.515
605_P1	605	R1_May_0727	R1	100	07/27/26	0.824	0.394	0.654	1.215	1.296	1.624
605_P2	605	R1_May_0727	R1	100	07/27/26	0.813	0.562	0.676	1.240	1.278	1.617
604_P1	604	R1_May_0727	R1	50	07/27/26	0.780	0.398	0.718	1.199	1.238	1.512
604_P2	604	R1_May_0727	R1	50	07/27/26	0.797	0.394	0.728	1.199	1.197	1.542
603_P1	603	R1_May_0727	R1	200	07/27/26	0.757	0.498	0.567	1.092	1.278	1.437
603_P2	603	R1_May_0727	R1	200	07/27/26	0.772	0.456	0.547	1.087	1.253	1.358
602_P1	602	R1_May_0727	R1	0	07/27/26	0.821	0.399	0.659	1.252	1.352	1.438
602_P2	602	R1_May_0727	R1	0	07/27/26	0.835	0.448	0.616	1.205	1.253	1.406
601_P1	601	R1_May_0727	R1	200 split	07/27/26	0.810	0.478	0.658	1.342	1.308	1.679
601_P2	601	R1_May_0727	R1	200 split	07/27/26	0.805	0.563	0.629	1.258	1.348	1.656
506_P1	506	R1_May_0727	R1	200	07/27/26	0.886	0.452	0.669	1.200	1.277	1.661
506_P2	506	R1_May_0727	R1	200	07/27/26	0.785	0.390	0.676	1.166	1.223	1.610
505_P1	505	R1_May_0727	R1	50	07/27/26	0.718	0.582	0.541	1.056	1.096	1.449

Plant ID	Plot	Round	Stage	N Treatment	Date	660 nm	670 nm	700 nm	750 nm	780 nm	850 nm
505_P2	505	R1_May_0727	R1	50	07/27/26	0.711	0.451	0.499	1.038	1.131	1.340
504_P1	504	R1_May_0727	R1	150	07/27/26	0.758	0.433	0.643	1.166	1.183	1.540
504_P2	504	R1_May_0727	R1	150	07/27/26	0.737	0.429	0.594	1.098	1.178	1.582
503_P1	503	R1_May_0727	R1	200 split	07/27/26	0.793	0.367	0.517	1.098	1.212	1.589
503_P2	503	R1_May_0727	R1	200 split	07/27/26	0.752	0.470	0.537	1.092	1.182	1.555
502_P1	502	R1_May_0727	R1	100	07/27/26	0.752	0.435	0.567	1.241	1.307	1.801
502_P2	502	R1_May_0727	R1	100	07/27/26	0.704	0.356	0.551	1.256	1.311	1.909
501_P1	501	R1_May_0727	R1	0	07/27/26	0.745	0.497	0.531	1.159	1.204	1.569
501_P2	501	R1_May_0727	R1	0	07/27/26	0.788	0.426	0.519	1.138	1.161	1.504
406_P1	406	R1_May_0727	R1	0	07/27/26	0.817	0.557	0.765	1.200	1.309	1.734
406_P2	406	R1_May_0727	R1	0	07/27/26	0.753	0.462	0.759	1.207	1.322	1.851
405_P1	405	R1_May_0727	R1	150	07/27/26	0.780	0.513	0.517	1.087	1.246	1.597
405_P2	405	R1_May_0727	R1	150	07/27/26	0.684	0.427	0.538	1.077	1.198	1.530
404_P1	404	R1_May_0727	R1	200	07/27/26	0.883	0.355	0.742	1.285	1.289	1.908
404_P2	404	R1_May_0727	R1	200	07/27/26	0.860	0.516	0.730	1.255	1.242	1.908
403_P1	403	R1_May_0727	R1	50	07/27/26	0.955	0.466	0.751	1.314	1.264	1.822
403_P2	403	R1_May_0727	R1	50	07/27/26	0.938	0.581	0.734	1.308	1.281	1.849
402_P1	402	R1_May_0727	R1	200 split	07/27/26	0.896	0.501	0.628	1.251	1.293	1.537
402_P2	402	R1_May_0727	R1	200 split	07/27/26	0.815	0.422	0.723	1.283	1.280	1.734
401_P1	401	R1_May_0727	R1	100	07/27/26	0.866	0.455	0.770	1.264	1.257	1.761
401_P2	401	R1_May_0727	R1	100	07/27/26	0.844	0.449	0.801	1.183	1.247	1.728
401_P1	401	R3_May_0817	R3	100	08/17/26	0.890	0.856	0.875	1.332	1.370	1.753
401_P2	401	R3_May_0817	R3	100	08/17/26	1.018	1.006	0.873	1.348	1.335	1.495
402_P1	402	R3_May_0817	R3	200 split	08/17/26	0.898	0.853	0.775	1.269	1.354	1.355
402_P2	402	R3_May_0817	R3	200 split	08/17/26	0.899	0.838	0.618	1.214	1.312	1.323
403_P1	403	R3_May_0817	R3	50	08/17/26	0.886	0.764	0.632	1.146	1.275	1.291
403_P2	403	R3_May_0817	R3	50	08/17/26	0.883	0.885	0.756	1.210	1.280	1.261
404_P1	404	R3_May_0817	R3	200	08/17/26	0.914	0.849	0.634	1.275	1.271	1.414
404_P2	404	R3_May_0817	R3	200	08/17/26	0.872	0.749	0.709	1.182	1.301	1.406
405_P1	405	R3_May_0817	R3	150	08/17/26	0.853	0.706	0.590	0.856	1.065	1.143

Plant ID	Plot	Round	Stage	N Treatment	Date	660 nm	670 nm	700 nm	750 nm	780 nm	850 nm
405_P2	405	R3_May_0817	R3	150	08/17/26	0.879	0.870	0.699	1.194	1.249	1.429
406_P1	406	R3_May_0817	R3	0	08/17/26	0.847	0.709	0.638	1.040	1.120	1.304
406_P2	406	R3_May_0817	R3	0	08/17/26	0.820	0.719	0.573	0.989	1.176	1.366
501_P1	501	R3_May_0817	R3	0	08/17/26	0.869	0.774	0.700	1.237	1.273	1.443
501_P2	501	R3_May_0817	R3	0	08/17/26	0.904	0.865	0.762	1.241	1.264	1.484
502_P1	502	R3_May_0817	R3	100	08/17/26	1.005	0.930	0.774	1.358	1.351	1.381
502_P2	502	R3_May_0817	R3	100	08/17/26	0.810	0.627	0.543	1.051	1.205	1.506
503_P1	503	R3_May_0817	R3	200 split	08/17/26	0.853	0.660	0.584	1.088	1.227	1.526
503_P2	503	R3_May_0817	R3	200 split	08/17/26	0.794	0.681	0.612	1.038	1.245	1.430
504_P1	504	R3_May_0817	R3	150	08/17/26	0.798	0.474	0.567	0.961	1.231	1.424
504_P2	504	R3_May_0817	R3	150	08/17/26	0.811	0.805	0.611	1.054	1.221	1.373
505_P1	505	R3_May_0817	R3	50	08/17/26	0.865	0.737	0.539	1.033	1.158	1.340
505_P2	505	R3_May_0817	R3	50	08/17/26	0.934	0.731	0.763	1.278	1.384	1.543
506_P1	506	R3_May_0817	R3	200	08/17/26	0.869	0.628	0.628	1.129	1.299	1.558
506_P2	506	R3_May_0817	R3	200	08/17/26	0.920	0.800	0.724	1.246	1.365	1.428
601_P1	601	R3_May_0817	R3	200 split	08/17/26	0.862	0.615	0.537	1.110	1.344	1.447
601_P2	601	R3_May_0817	R3	200 split	08/17/26	0.858	0.816	0.677	1.236	1.352	1.486
602_P1	602	R3_May_0817	R3	0	08/17/26	0.921	0.756	0.800	1.237	1.363	1.587
602_P2	602	R3_May_0817	R3	0	08/17/26	0.915	0.857	0.717	1.198	1.320	1.493
603_P1	603	R3_May_0817	R3	200	08/17/26	0.833	0.667	0.567	1.104	1.241	1.416
603_P2	603	R3_May_0817	R3	200	08/17/26	0.965	0.854	0.766	1.269	1.308	1.518
604_P1	604	R3_May_0817	R3	50	08/17/26	0.942	0.969	0.769	1.243	1.408	1.533
604_P2	604	R3_May_0817	R3	50	08/17/26	0.982	0.863	0.688	1.331	1.330	1.546
605_P1	605	R3_May_0817	R3	100	08/17/26	0.836	0.680	0.447	0.958	1.154	1.637
605_P2	605	R3_May_0817	R3	100	08/17/26	0.974	0.847	0.780	1.337	1.291	1.583
606_P1	606	R3_May_0817	R3	150	08/17/26	0.820	0.734	0.553	0.969	1.191	1.573
606_P2	606	R3_May_0817	R3	150	08/17/26	0.907	0.594	0.452	1.014	1.230	1.666

Appendix Table B.1 (continued).

Plant ID	NDVI	NDRE (780,700)	NDRE (750,700)	NIR:Red	RE Ratio	RE Diff	Slope 660-700	Slope 700-750	Slope 750-780	Slope 780-850	Nitrate (ppm)
102_P1	0.695	0.025	0.093	5.56	1.20	0.207	0.0014	0.0041	-0.0051	0.0116	3434
102_P2	0.688	0.042	0.093	5.41	1.21	0.203	0.0002	0.0041	-0.0039	0.0107	2420
102_P3	0.705	0.081	0.168	5.78	1.40	0.345	-0.0034	0.0069	-0.0064	0.0119	3504
103_P1	0.657	0.001	0.229	4.83	1.60	0.509	0.0001	0.0102	-0.0169	0.0106	10802
103_P2	0.628	0.054	0.129	4.37	1.30	0.238	-0.0009	0.0048	-0.0049	0.0094	9351
103_P3	0.634	0.105	0.186	4.46	1.46	0.336	0.0008	0.0067	-0.0054	0.0111	6427
302_P1	0.637	0.086	0.212	4.51	1.54	0.432	-0.0002	0.0086	-0.0094	0.0099	5582
302_P2	0.582	0.117	0.205	3.78	1.51	0.377	-0.0035	0.0075	-0.0061	0.0092	4605
302_P3	0.563	0.057	0.139	3.58	1.32	0.270	-0.0007	0.0054	-0.0056	0.0086	2068
205_P1	0.687	0.111	0.152	5.39	1.36	0.303	-0.0022	0.0061	-0.0030	0.0094	5781
205_P2	0.681	-0.021	0.082	5.26	1.18	0.171	0.0029	0.0034	-0.0070	0.0144	6319
205_P3	0.631	0.164	0.222	4.42	1.57	0.406	-0.0012	0.0081	-0.0042	0.0088	5189
106_P1	0.733	0.026	0.151	6.49	1.36	0.296	-0.0012	0.0059	-0.0084	0.0095	9526
106_P2	0.643	0.022	0.264	4.60	1.72	0.588	-0.0024	0.0118	-0.0184	0.0120	7079
106_P3	0.432	-0.575	0.029	2.52	1.06	0.070	0.0003	0.0014	-0.0312	0.0201	6768
303_P1	0.679	0.075	0.200	5.22	1.50	0.396	0.0004	0.0079	-0.0089	0.0106	4867
303_P2	0.661	0.037	0.141	4.90	1.33	0.291	-0.0018	0.0058	-0.0074	0.0128	3698
303_P3	0.642	0.134	0.226	4.59	1.58	0.424	-0.0021	0.0085	-0.0066	0.0099	9824
203_P1	0.732	0.043	0.136	6.46	1.31	0.256	-0.0234	0.0051	-0.0061	0.0117	6193
203_P2	0.629	-0.337	-0.244	4.38	0.61	-0.716	0.0257	-0.0143	-0.0068	0.0091	10118
203_P3	0.650	0.083	0.037	4.71	1.08	0.063	-0.0009	0.0013	0.0028	0.0078	6243
105_P1	0.662	0.097	0.219	4.91	1.56	0.433	-0.0019	0.0087	-0.0089	0.0112	3188
105_P2	0.664	0.074	0.108	4.95	1.24	0.207	-0.0006	0.0041	-0.0024	0.0078	3341
105_P3	0.658	0.045	0.116	4.85	1.26	0.228	-0.0014	0.0046	-0.0049	0.0091	3138
306_P1	0.643	0.074	0.210	4.60	1.53	0.453	-0.0033	0.0091	-0.0106	0.0073	19
306_P2	0.628	0.042	0.147	4.38	1.34	0.315	-0.0035	0.0063	-0.0078	0.0111	157
306_P3	0.607	0.089	0.152	4.09	1.36	0.285	-0.0013	0.0057	-0.0043	0.0066	27
202_P1	0.612	0.132	0.279	4.15	1.78	0.574	-0.0030	0.0115	-0.0116	0.0090	2846

Plant ID	NDVI	NDRE (780,700)	NDRE (750,700)	NIR:Red	RE Ratio	RE Diff	Slope 660-700	Slope 700-750	Slope 750-780	Slope 780-850	Nitrate (ppm)
202_P2	0.594	0.023	0.128	3.93	1.29	0.260	-0.0020	0.0052	-0.0073	0.0105	3253
202_P3	0.619	0.099	0.225	4.26	1.58	0.465	-0.0027	0.0093	-0.0097	0.0092	2648
201_P1	0.637	0.107	0.233	4.51	1.61	0.479	0.0002	0.0096	-0.0097	0.0088	8543
201_P2	0.674	0.087	0.237	5.14	1.62	0.421	-0.0160	0.0084	-0.0097	0.0078	8198
201_P3	0.667	0.020	0.118	5.01	1.27	0.221	0.0008	0.0044	-0.0062	0.0089	7969
304_P1	-0.348	0.101	0.189	0.48	1.46	0.355	-0.0046	0.0071	-0.0061	-0.0114	7412
304_P2	0.591	-0.004	0.171	3.89	1.41	0.350	-0.0002	0.0070	-0.0119	0.0106	2876
304_P3	0.678	-0.025	0.028	5.20	1.06	0.056	0.0015	0.0011	-0.0035	0.0108	3216
104_P1	0.698	0.064	0.130	5.62	1.30	0.252	-0.0023	0.0050	-0.0045	0.0126	3745
104_P2	0.645	0.168	0.263	4.63	1.71	0.446	-0.0047	0.0089	-0.0064	0.0098	2102
104_P3	0.682	-0.202	-0.056	5.29	0.89	-0.140	0.0120	-0.0028	-0.0100	0.0119	4098
204_P1	0.654	0.017	0.124	4.78	1.28	0.243	-0.0029	0.0049	-0.0071	0.0119	5461
204_P2	0.655	0.043	0.229	4.80	1.60	0.469	-0.0007	0.0094	-0.0132	0.0101	3974
204_P3	0.644	0.028	0.158	4.63	1.38	0.304	-0.0034	0.0061	-0.0086	0.0096	5227
101_P1	0.673	0.088	0.253	5.11	1.68	0.476	-0.0030	0.0095	-0.0113	0.0122	43
101_P2	0.660	-0.062	0.031	4.88	1.06	0.059	0.0011	0.0012	-0.0056	0.0107	64
101_P3	0.665	0.070	0.087	4.96	1.19	0.158	-0.0043	0.0032	-0.0011	0.0116	23
206_P1	0.780	-0.346	-0.205	8.10	0.66	-0.626	0.0243	-0.0125	-0.0106	0.0236	7848
206_P2	0.751	-0.000	0.124	7.04	1.28	0.234	-0.0017	0.0047	-0.0078	0.0108	13937
206_P3	0.679	0.027	0.078	5.23	1.17	0.135	-0.0008	0.0027	-0.0030	0.0140	12110
301_P1	0.648	-0.040	0.149	4.69	1.35	0.306	-0.0024	0.0061	-0.0124	0.0142	6201
301_P2	0.677	0.081	0.152	5.20	1.36	0.279	-0.0028	0.0056	-0.0047	0.0124	11055
301_P3	0.709	0.055	0.183	5.88	1.45	0.333	-0.0041	0.0067	-0.0082	0.0114	7795
305_P1	0.609	0.092	0.152	4.12	1.36	0.287	-0.0021	0.0057	-0.0042	0.0073	631
305_P2	0.767	0.024	0.380	7.59	2.23	1.034	-0.0003	0.0207	-0.0331	0.0237	804
305_P3	0.696	0.019	0.131	5.59	1.30	0.253	0.0011	0.0051	-0.0073	0.0143	76
101_P1	0.651	-0.027	0.155	4.73	1.37	0.341	-0.0015	0.0068	-0.0130	0.0105	26
101_P2	0.603	0.017	0.154	4.03	1.36	0.323	-0.0027	0.0065	-0.0098	0.0093	13
102_P1	0.572	0.076	0.170	3.68	1.41	0.364	-0.0006	0.0073	-0.0073	0.0090	10
102_P2	0.665	0.101	0.214	4.97	1.54	0.446	-0.0031	0.0089	-0.0087	0.0101	12

Plant ID	NDVI	NDRE (780,700)	NDRE (750,700)	NIR:Red	RE Ratio	RE Diff	Slope 660-700	Slope 700-750	Slope 750-780	Slope 780-850	Nitrate (ppm)
103_P1	0.515	-0.035	0.090	3.12	1.20	0.174	-0.0055	0.0035	-0.0078	0.0072	10
103_P2	0.592	-0.000	0.099	3.90	1.22	0.202	-0.0070	0.0040	-0.0067	0.0105	12
106_P1	0.589	-0.020	0.150	3.87	1.35	0.312	-0.0024	0.0062	-0.0115	0.0094	14
106_P2	0.674	0.095	0.260	5.13	1.70	0.549	-0.0068	0.0110	-0.0129	0.0105	16
105_P1	0.688	0.063	0.117	5.41	1.26	0.213	-0.0032	0.0043	-0.0035	0.0084	13
105_P2	0.684	0.097	0.153	5.32	1.36	0.309	-0.0044	0.0062	-0.0042	0.0139	15
104_P1	0.561	-0.209	0.158	3.55	1.38	0.369	-0.0001	0.0074	-0.0236	0.0135	12
104_P2	0.651	0.048	0.165	4.74	1.39	0.354	-0.0032	0.0071	-0.0088	0.0084	14
201_P1	0.645	-0.029	0.283	4.64	1.79	0.668	0.0357	0.0134	-0.0239	0.0121	12
201_P2	0.620	0.055	0.127	4.26	1.29	0.262	-0.0004	0.0052	-0.0052	0.0074	11
202_P1	0.630	-0.078	0.129	4.41	1.30	0.273	-0.0015	0.0055	-0.0135	0.0141	13
202_P2	0.593	0.003	0.102	3.91	1.23	0.211	-0.0061	0.0042	-0.0069	0.0104	12
203_P1	0.643	-0.025	0.168	4.60	1.40	0.350	-0.0023	0.0070	-0.0131	0.0116	16
203_P2	0.628	0.047	0.163	4.37	1.39	0.335	-0.0036	0.0067	-0.0083	0.0088	13
206_P1	0.642	0.061	0.258	4.59	1.70	0.533	-0.0048	0.0107	-0.0145	0.0089	11
206_P2	0.579	0.006	0.138	3.75	1.32	0.274	-0.0014	0.0055	-0.0088	0.0093	13
205_P1	0.637	-0.071	0.161	4.51	1.38	0.306	-0.0059	0.0061	-0.0137	0.0104	12
205_P2	0.659	-0.005	0.171	4.86	1.41	0.359	0.0035	0.0072	-0.0123	0.0108	14
204_P1	0.674	0.011	0.149	5.14	1.35	0.275	-0.0046	0.0055	-0.0086	0.0123	14
204_P2	0.598	-0.006	0.109	3.97	1.24	0.211	-0.0026	0.0042	-0.0074	0.0075	29
301_P1	0.677	0.086	0.179	5.19	1.43	0.299	-0.0061	0.0060	-0.0056	0.0117	14
301_P2	0.430	0.041	0.176	2.51	1.43	0.339	0.0011	0.0068	-0.0090	0.0093	16
302_P1	0.663	0.111	0.300	4.94	1.86	0.606	-0.0053	0.0121	-0.0143	0.0102	13
302_P2	0.622	0.035	0.153	4.29	1.36	0.298	-0.0037	0.0060	-0.0079	0.0085	13
303_P1	0.612	0.073	0.169	4.16	1.41	0.338	-0.0024	0.0068	-0.0069	0.0089	13
303_P2	0.676	-0.095	0.064	5.17	1.14	0.143	0.0004	0.0029	-0.0109	0.0111	12
306_P1	0.661	-0.011	0.232	4.89	1.60	0.474	-0.0047	0.0095	-0.0164	0.0100	11
306_P2	0.706	-0.021	0.069	5.80	1.15	0.137	-0.0037	0.0027	-0.0058	0.0109	12
305_P1	0.594	0.033	0.192	3.93	1.48	0.381	-0.0049	0.0076	-0.0109	0.0093	12
305_P2	0.573	-0.023	0.079	3.68	1.17	0.157	-0.0022	0.0031	-0.0066	0.0090	14

Plant ID	NDVI	NDRE (780,700)	NDRE (750,700)	NIR:Red	RE Ratio	RE Diff	Slope 660-700	Slope 700-750	Slope 750-780	Slope 780-850	Nitrate (ppm)
304_P1	0.662	-0.080	-0.704	4.92	0.17	-0.779	0.0017	-0.0156	0.0213	0.0111	13
304_P2	0.562	-0.097	0.091	3.57	1.20	0.175	-0.0025	0.0035	-0.0110	0.0114	11
605_P1	0.572	-0.039	0.114	3.67	1.26	0.239	-0.0005	0.0048	-0.0103	0.0088	11
605_P2	0.575	0.050	0.159	3.70	1.38	0.274	-0.0074	0.0055	-0.0066	0.0077	18
606_P1	0.647	0.007	0.155	4.66	1.37	0.300	-0.0042	0.0060	-0.0096	0.0089	17
606_P2	0.642	-0.284	-0.189	4.59	0.68	-0.480	0.0172	-0.0096	-0.0063	0.0077	24
604_P1	0.594	0.019	0.138	3.93	1.32	0.257	-0.0024	0.0051	-0.0076	0.0094	13
604_P2	0.595	-0.011	0.202	3.94	1.51	0.368	-0.0115	0.0074	-0.0128	0.0094	12
603_P1	0.576	-0.021	0.136	3.72	1.32	0.276	-0.0007	0.0055	-0.0104	0.0072	14
603_P2	0.644	-0.032	0.112	4.62	1.25	0.226	-0.0030	0.0045	-0.0094	0.0103	16
601_P1	0.633	-0.033	0.041	4.45	1.08	0.076	-0.0003	0.0015	-0.0045	0.0092	31
601_P2	0.722	-0.011	0.107	6.19	1.24	0.211	-0.0023	0.0042	-0.0077	0.0106	72
602_P1	0.688	0.085	0.194	5.42	1.48	0.355	-0.0018	0.0071	-0.0073	0.0166	23
602_P2	0.637	0.010	0.149	4.51	1.35	0.287	-0.0026	0.0057	-0.0091	0.0104	14
505_P1	0.672	0.207	0.145	5.10	1.34	0.262	-0.0037	0.0052	0.0047	0.0066	13
505_P2	0.587	0.085	0.211	3.85	1.53	0.387	-0.0061	0.0077	-0.0084	0.0085	12
506_P1	0.654	0.008	0.156	4.78	1.37	0.305	-0.0020	0.0061	-0.0097	0.0102	13
506_P2	0.676	0.056	0.179	5.18	1.43	0.339	-0.0007	0.0068	-0.0082	0.0092	41
503_P1	0.588	-0.002	0.104	3.85	1.23	0.206	-0.0022	0.0041	-0.0070	0.0080	102
503_P2	0.677	-0.280	0.091	5.18	1.20	0.185	-0.0017	0.0037	-0.0197	0.0168	87
504_P1	0.644	0.033	0.149	4.61	1.35	0.285	-0.0013	0.0057	-0.0076	0.0088	24
504_P2	0.657	0.124	0.271	4.82	1.74	0.442	-0.0118	0.0088	-0.0091	0.0087	26
501_P1	0.644	-0.054	0.346	4.61	2.06	0.846	-0.0018	0.0169	-0.0310	0.0095	21
501_P2	0.582	0.527	0.632	3.78	4.43	0.850	-0.0161	0.0170	-0.0099	0.0077	17
502_P1	0.631	0.036	0.142	4.42	1.33	0.249	-0.0022	0.0050	-0.0064	0.0075	17
502_P2	0.676	-0.069	-0.026	5.17	0.95	-0.047	0.0014	-0.0009	-0.0025	0.0100	18
405_P1	0.640	0.007	0.144	4.56	1.34	0.293	-0.0013	0.0059	-0.0093	0.0093	13
405_P2	0.692	0.030	0.183	5.50	1.45	0.341	-0.0031	0.0068	-0.0098	0.0101	16
406_P1	0.621	-0.016	0.108	4.28	1.24	0.204	-0.0018	0.0041	-0.0077	0.0084	15
406_P2	0.668	0.044	0.192	5.02	1.48	0.339	-0.0028	0.0068	-0.0091	0.0105	19

Plant ID	NDVI	NDRE (780,700)	NDRE (750,700)	NIR:Red	RE Ratio	RE Diff	Slope 660-700	Slope 700-750	Slope 750-780	Slope 780-850	Nitrate (ppm)
403_P1	0.296	-0.011	0.145	1.84	1.34	0.264	-0.0011	0.0053	-0.0094	-0.0031	50
403_P2	0.693	0.035	0.082	5.52	1.18	0.143	-0.0018	0.0029	-0.0028	0.0101	18
404_P1	0.633	-0.052	0.103	4.45	1.23	0.184	-0.0038	0.0037	-0.0088	0.0087	31
404_P2	0.541	0.048	0.239	3.36	1.63	0.417	-0.0035	0.0083	-0.0117	0.0093	23
401_P1	0.637	0.084	0.203	4.51	1.51	0.368	-0.0056	0.0074	-0.0078	0.0053	26
401_P2	0.546	-0.017	0.180	3.41	1.44	0.318	-0.0074	0.0064	-0.0114	0.0091	82
402_P1	0.590	0.029	0.195	3.88	1.48	0.351	-0.0051	0.0070	-0.0103	0.0078	26
402_P2	0.660	0.030	0.127	4.89	1.29	0.226	-0.0050	0.0045	-0.0059	0.0094	125
101_P1		0.009	0.128		1.29	0.423	-0.0058	0.0085	-0.0133		33
101_P2		0.074	0.138		1.32	0.455	-0.0050	0.0091	-0.0076		17
102_P1		0.080	0.145		1.34	0.477	-0.0049	0.0095	-0.0077		17
102_P2		0.065	0.121		1.28	0.378	-0.0048	0.0076	-0.0063		19
103_P1		0.054	0.113		1.25	0.358	-0.0049	0.0072	-0.0065		16
103_P2		0.056	0.117		1.27	0.347	-0.0030	0.0069	-0.0064		14
104_P1		0.042	0.106		1.24	0.351	-0.0029	0.0070	-0.0074		15
104_P2		0.037	0.106		1.24	0.337	-0.0030	0.0067	-0.0076		12
105_P1		0.054	0.094		1.21	0.281	0.0000	0.0056	-0.0042		15
105_P2		0.097	0.149		1.35	0.418	-0.0060	0.0084	-0.0054		15
106_P1		0.085	0.136		1.31	0.378	-0.0050	0.0076	-0.0052		13
106_P2		0.124	0.161		1.38	0.425	-0.0042	0.0085	-0.0037		12
201_P1		0.043	0.124		1.28	0.362	-0.0029	0.0072	-0.0083		11
201_P2		0.183	0.197		1.49	0.491	-0.0024	0.0098	-0.0013		10
202_P1		0.051	0.120		1.27	0.349	-0.0029	0.0070	-0.0070		15
202_P2		0.081	0.139		1.32	0.399	-0.0025	0.0080	-0.0060		13
203_P1		0.041	0.097		1.22	0.272	-0.0010	0.0054	-0.0055		13
203_P2		0.067	0.133		1.31	0.365	-0.0028	0.0073	-0.0065		12
204_P1											12
204_P2	0.590	0.109	0.161	3.88	1.38	0.405	-0.0006	0.0081	-0.0049	0.0106	12
205_P1	0.590	0.322	0.296	3.88	1.84	0.585	-0.0033	0.0117	0.0025	0.0019	12
205_P2	0.488	0.112	0.156	2.91	1.37	0.403	0.0017	0.0081	-0.0043	0.0037	15

Plant ID	NDVI	NDRE (780,700)	NDRE (750,700)	NIR:Red	RE Ratio	RE Diff	Slope 660-700	Slope 700-750	Slope 750-780	Slope 780-850	Nitrate (ppm)
206_P1	0.614	0.085	0.125	4.18	1.28	0.341	0.0005	0.0068	-0.0040	0.0104	13
206_P2	0.657	0.154	0.201	4.83	1.50	0.516	-0.0051	0.0103	-0.0048	0.0084	59
301_P1	0.661	0.141	0.168	4.91	1.40	0.437	-0.0018	0.0087	-0.0027	0.0101	66
301_P2	0.493	0.332	0.353	2.94	2.09	0.767	-0.0061	0.0153	-0.0022	0.0056	60
302_P1	0.589	0.151	0.154	3.87	1.36	0.378	0.0001	0.0076	-0.0003	0.0080	64
302_P2	0.494	0.161	0.183	2.95	1.45	0.453	-0.0007	0.0091	-0.0021	0.0075	
303_P1	0.507	0.306	0.285	3.06	1.80	0.584	-0.0015	0.0117	0.0021	0.0046	64
303_P2	0.509	0.216	0.235	3.07	1.62	0.552	-0.0024	0.0110	-0.0019	0.0038	65
304_P1	0.467	0.401	0.370	2.75	2.18	0.613	-0.0074	0.0123	0.0029	0.0016	60
304_P2	0.560	0.336	0.346	3.55	2.06	0.729	-0.0070	0.0146	-0.0010	0.0054	61
305_P1	0.596	0.140	0.179	3.95	1.44	0.464	-0.0009	0.0093	-0.0039	0.0070	61
305_P2	0.543	0.164	0.201	3.38	1.50	0.500	-0.0024	0.0100	-0.0037	0.0081	60
306_P1	0.523	0.084	0.165	3.19	1.39	0.429	-0.0018	0.0086	-0.0077	0.0093	56
306_P2	0.516	0.116	0.151	3.14	1.35	0.371	-0.0003	0.0074	-0.0033	0.0047	60
606_P1	0.539	0.342	0.312	3.34	1.91	0.590	-0.0055	0.0118	0.0028	0.0038	60
606_P2	0.459	0.353	0.319	2.70	1.94	0.574	-0.0068	0.0115	0.0031	0.0034	56
605_P1	0.610	0.329	0.300	4.12	1.86	0.561	-0.0042	0.0112	0.0027	0.0047	65
605_P2	0.484	0.308	0.294	2.88	1.83	0.564	-0.0034	0.0113	0.0013	0.0048	61
604_P1	0.583	0.266	0.251	3.80	1.67	0.482	-0.0016	0.0096	0.0013	0.0039	66
604_P2	0.593	0.244	0.244	3.91	1.65	0.471	-0.0017	0.0094	-0.0001	0.0049	60
603_P1	0.485	0.386	0.317	2.88	1.93	0.525	-0.0048	0.0105	0.0062	0.0023	55
603_P2	0.497	0.392	0.330	2.98	1.99	0.540	-0.0056	0.0108	0.0055	0.0015	55
602_P1	0.565	0.345	0.310	3.60	1.90	0.593	-0.0040	0.0119	0.0033	0.0012	57
602_P2	0.516	0.341	0.324	3.14	1.96	0.589	-0.0055	0.0118	0.0016	0.0022	62
601_P1	0.557	0.330	0.342	3.51	2.04	0.683	-0.0038	0.0137	-0.0011	0.0053	52
601_P2	0.493	0.364	0.334	2.94	2.00	0.630	-0.0044	0.0126	0.0030	0.0044	51
506_P1	0.572	0.312	0.284	3.67	1.79	0.531	-0.0054	0.0106	0.0026	0.0055	58
506_P2	0.610	0.288	0.266	4.13	1.73	0.490	-0.0027	0.0098	0.0019	0.0055	64
505_P1	0.427	0.339	0.323	2.49	1.95	0.515	-0.0044	0.0103	0.0013	0.0050	55
505_P2	0.496	0.387	0.350	2.97	2.08	0.539	-0.0053	0.0108	0.0031	0.0030	56

Plant ID	NDVI	NDRE (780,700)	NDRE (750,700)	NIR:Red	RE Ratio	RE Diff	Slope 660-700	Slope 700-750	Slope 750-780	Slope 780-850	Nitrate (ppm)
504_P1	0.561	0.296	0.289	3.56	1.81	0.524	-0.0029	0.0105	0.0006	0.0051	61
504_P2	0.573	0.329	0.298	3.69	1.85	0.504	-0.0036	0.0101	0.0026	0.0058	60
503_P1	0.625	0.401	0.360	4.33	2.12	0.581	-0.0069	0.0116	0.0038	0.0054	82
503_P2	0.535	0.375	0.341	3.31	2.03	0.555	-0.0054	0.0111	0.0030	0.0053	71
502_P1	0.611	0.395	0.373	4.14	2.19	0.674	-0.0046	0.0135	0.0022	0.0071	69
502_P2	0.686	0.408	0.390	5.37	2.28	0.705	-0.0038	0.0141	0.0018	0.0085	69
501_P1	0.519	0.388	0.372	3.16	2.18	0.628	-0.0054	0.0126	0.0015	0.0052	70
501_P2	0.558	0.382	0.374	3.53	2.19	0.619	-0.0067	0.0124	0.0008	0.0049	65
406_P1	0.514	0.262	0.221	3.12	1.57	0.435	-0.0013	0.0087	0.0036	0.0061	74
406_P2	0.601	0.270	0.228	4.01	1.59	0.448	0.0002	0.0090	0.0038	0.0076	69
405_P1	0.514	0.414	0.356	3.11	2.10	0.570	-0.0066	0.0114	0.0053	0.0050	64
405_P2	0.563	0.380	0.334	3.58	2.00	0.539	-0.0036	0.0108	0.0040	0.0047	67
404_P1	0.686	0.269	0.268	5.37	1.73	0.543	-0.0035	0.0109	0.0001	0.0088	69
404_P2	0.574	0.259	0.264	3.70	1.72	0.525	-0.0032	0.0105	-0.0005	0.0095	64
403_P1	0.593	0.254	0.272	3.91	1.75	0.562	-0.0051	0.0112	-0.0017	0.0080	61
403_P2	0.522	0.271	0.281	3.18	1.78	0.573	-0.0051	0.0115	-0.0009	0.0081	59
402_P1	0.508	0.346	0.332	3.07	1.99	0.623	-0.0067	0.0125	0.0014	0.0035	65
402_P2	0.609	0.278	0.279	4.11	1.77	0.560	-0.0023	0.0112	-0.0001	0.0065	67
401_P1	0.590	0.240	0.243	3.87	1.64	0.493	-0.0024	0.0099	-0.0002	0.0072	66
401_P2	0.588	0.218	0.193	3.85	1.48	0.382	-0.0011	0.0076	0.0021	0.0069	67
401_P1	0.344	0.221	0.207	2.05	1.52	0.457	-0.0004	0.0091	0.0013	0.0055	51
401_P2	0.195	0.209	0.214	1.49	1.54	0.475	-0.0036	0.0095	-0.0004	0.0023	58
402_P1	0.228	0.272	0.242	1.59	1.64	0.494	-0.0031	0.0099	0.0028	0.0000	50
402_P2	0.224	0.359	0.325	1.58	1.96	0.596	-0.0070	0.0119	0.0033	0.0001	50
403_P1	0.257	0.338	0.289	1.69	1.81	0.514	-0.0064	0.0103	0.0043	0.0002	55
403_P2	0.175	0.257	0.231	1.42	1.60	0.454	-0.0032	0.0091	0.0023	-0.0003	50
404_P1	0.250	0.334	0.336	1.67	2.01	0.641	-0.0070	0.0128	-0.0001	0.0020	59
404_P2	0.305	0.294	0.250	1.88	1.67	0.473	-0.0041	0.0095	0.0040	0.0015	60
405_P1	0.237	0.287	0.183	1.62	1.45	0.265	-0.0066	0.0053	0.0070	0.0011	69
405_P2	0.243	0.282	0.261	1.64	1.71	0.494	-0.0045	0.0099	0.0018	0.0026	56

Plant ID	NDVI	NDRE (780,700)	NDRE (750,700)	NIR:Red	RE Ratio	RE Diff	Slope 660-700	Slope 700-750	Slope 750-780	Slope 780-850	Nitrate (ppm)
406_P1	0.296	0.274	0.240	1.84	1.63	0.402	-0.0052	0.0080	0.0027	0.0026	56
406_P2	0.310	0.345	0.266	1.90	1.73	0.416	-0.0062	0.0083	0.0062	0.0027	47
501_P1	0.302	0.291	0.278	1.87	1.77	0.538	-0.0042	0.0108	0.0012	0.0024	55
501_P2	0.264	0.248	0.239	1.72	1.63	0.479	-0.0036	0.0096	0.0008	0.0031	55
502_P1	0.195	0.272	0.274	1.48	1.76	0.585	-0.0058	0.0117	-0.0002	0.0004	62
502_P2	0.412	0.379	0.319	2.40	1.93	0.508	-0.0067	0.0102	0.0051	0.0043	65
503_P1	0.396	0.355	0.301	2.31	1.86	0.504	-0.0067	0.0101	0.0046	0.0043	57
503_P2	0.355	0.341	0.258	2.10	1.70	0.426	-0.0045	0.0085	0.0069	0.0026	150
504_P1	0.500	0.369	0.257	3.00	1.69	0.393	-0.0058	0.0079	0.0090	0.0028	59
504_P2	0.261	0.333	0.266	1.71	1.72	0.443	-0.0050	0.0089	0.0056	0.0022	51
505_P1	0.290	0.365	0.314	1.82	1.92	0.494	-0.0081	0.0099	0.0042	0.0026	49
505_P2	0.357	0.289	0.252	2.11	1.67	0.515	-0.0043	0.0103	0.0035	0.0023	58
506_P1	0.425	0.348	0.285	2.48	1.80	0.501	-0.0060	0.0100	0.0057	0.0037	56
506_P2	0.282	0.307	0.265	1.79	1.72	0.522	-0.0049	0.0104	0.0040	0.0009	69
601_P1	0.403	0.429	0.348	2.35	2.07	0.573	-0.0081	0.0115	0.0078	0.0015	43
601_P2	0.291	0.333	0.292	1.82	1.82	0.559	-0.0045	0.0112	0.0039	0.0019	54
602_P1	0.355	0.260	0.214	2.10	1.55	0.437	-0.0030	0.0087	0.0042	0.0032	55
602_P2	0.271	0.296	0.251	1.74	1.67	0.481	-0.0050	0.0096	0.0041	0.0025	47
603_P1	0.359	0.373	0.321	2.12	1.95	0.537	-0.0067	0.0107	0.0046	0.0025	61
603_P2	0.280	0.261	0.247	1.78	1.66	0.503	-0.0050	0.0101	0.0013	0.0030	57
604_P1	0.225	0.294	0.235	1.58	1.62	0.474	-0.0043	0.0095	0.0055	0.0018	68
604_P2	0.283	0.318	0.318	1.79	1.93	0.643	-0.0074	0.0129	-0.0000	0.0031	58
605_P1	0.413	0.442	0.364	2.41	2.14	0.511	-0.0097	0.0102	0.0066	0.0069	53
605_P2	0.303	0.247	0.263	1.87	1.71	0.558	-0.0049	0.0112	-0.0015	0.0042	59
606_P1	0.364	0.366	0.274	2.14	1.75	0.417	-0.0067	0.0083	0.0074	0.0055	52
606_P2	0.474	0.463	0.384	2.80	2.24	0.562	-0.0114	0.0112	0.0072	0.0062	50

Appendix C

Exhaustive Wavelength-Combination Diagnostics

The main text (Chapter 5) uses the mean R^2 and r across each combination size to argue for the sensor's six-wavelength design. This appendix reports the complete, unaveraged results behind that comparison: the held-out R^2 and r for every possible single band, every possible two-band combination, and every possible three-band combination, dark-enclosure condition only, using the same fixed neural-network architecture (3 hidden layers, 16-12-8 units) as the main results table, with no growth-stage information included. The final two tables list the 84 models that combined the six bands with three NDVI combinations, for the dark enclosure and outdoors.

Table 22. Held-out test performance of the neural-network architecture, dark room only, using every possible single-band model (n = 6), sorted by R^2.

Bands	**R^2**	**r**
780 nm	0.213	0.541
850 nm	0.112	0.421
660 nm	0.048	0.311
750 nm	0.039	0.279
670 nm	0.031	0.260
700 nm	−0.023	0.114

Table 23. Held-out test performance of the neural-network architecture, dark room only, using every possible two-band combination (n = 15), sorted by R^2.

Bands	$\mathbf{R^2}$	**r**
660 nm + 780 nm	0.295	0.549
780 nm + 850 nm	0.240	0.553
670 nm + 780 nm	0.230	0.524
750 nm + 850 nm	0.206	0.497
660 nm + 850 nm	0.205	0.489
670 nm + 850 nm	0.180	0.511
700 nm + 780 nm	0.138	0.469
660 nm + 670 nm	0.130	0.436
750 nm + 780 nm	0.093	0.396
660 nm + 750 nm	0.069	0.317
670 nm + 750 nm	0.056	0.270
700 nm + 850 nm	0.051	0.272
700 nm + 750 nm	0.042	0.271
670 nm + 700 nm	0.022	0.217
660 nm + 700 nm	0.006	0.200

Table 24. Held-out test performance of the neural-network architecture, dark room only, using every possible three-band combination (n = 20), sorted by R^2.

Bands	$\mathbf{R^2}$	**r**
660 nm + 780 nm + 850 nm	0.457	0.680
660 nm + 670 nm + 850 nm	0.350	0.619
660 nm + 670 nm + 780 nm	0.312	0.583
700 nm + 780 nm + 850 nm	0.310	0.628
660 nm + 700 nm + 780 nm	0.288	0.565
660 nm + 750 nm + 780 nm	0.274	0.559
670 nm + 780 nm + 850 nm	0.272	0.575
750 nm + 780 nm + 850 nm	0.237	0.525
670 nm + 700 nm + 850 nm	0.222	0.581
670 nm + 750 nm + 780 nm	0.217	0.515
700 nm + 750 nm + 850 nm	0.213	0.520
660 nm + 750 nm + 850 nm	0.203	0.472
670 nm + 750 nm + 850 nm	0.196	0.531
660 nm + 700 nm + 850 nm	0.193	0.473
700 nm + 750 nm + 780 nm	0.178	0.466
670 nm + 700 nm + 780 nm	0.178	0.480
660 nm + 670 nm + 750 nm	0.124	0.401
660 nm + 670 nm + 700 nm	0.019	0.256
670 nm + 700 nm + 750 nm	0.007	0.171
660 nm + 700 nm + 750 nm	0.006	0.200

Table 25. Held-out test performance of the neural-network architecture, dark room only, using every possible four-band combination (n = 15), sorted by R^2.

Bands	R^2	r
660 nm + 670 nm + 780 nm + 850 nm	0.462	0.709
660 nm + 670 nm + 700 nm + 850 nm	0.405	0.661
660 nm + 670 nm + 750 nm + 850 nm	0.397	0.644
660 nm + 700 nm + 780 nm + 850 nm	0.370	0.609
670 nm + 700 nm + 780 nm + 850 nm	0.336	0.679
660 nm + 750 nm + 780 nm + 850 nm	0.307	0.560
660 nm + 670 nm + 750 nm + 780 nm	0.296	0.555
660 nm + 700 nm + 750 nm + 780 nm	0.277	0.532
670 nm + 700 nm + 750 nm + 850 nm	0.275	0.579
670 nm + 750 nm + 780 nm + 850 nm	0.271	0.569
660 nm + 670 nm + 700 nm + 780 nm	0.260	0.523
700 nm + 750 nm + 780 nm + 850 nm	0.256	0.583
670 nm + 700 nm + 750 nm + 780 nm	0.238	0.532
660 nm + 700 nm + 750 nm + 850 nm	0.190	0.449
660 nm + 670 nm + 700 nm + 750 nm	0.074	0.309

Table 26. Held-out test performance of the neural-network architecture, dark room only, using every possible five-band combination (n = 6), sorted by R^2.

Bands	R^2	r
660 nm + 670 nm + 700 nm + 780 nm + 850 nm	0.469	0.737
660 nm + 670 nm + 750 nm + 780 nm + 850 nm	0.462	0.714
660 nm + 670 nm + 700 nm + 750 nm + 850 nm	0.384	0.679
670 nm + 700 nm + 750 nm + 780 nm + 850 nm	0.302	0.572
660 nm + 670 nm + 700 nm + 750 nm + 780 nm	0.284	0.558
660 nm + 700 nm + 750 nm + 780 nm + 850 nm	0.250	0.522

Table 27. Held-out test performance of the neural-network architecture, dark room only, using the six-band combination (n = 1).

Bands	R^2	r
660 nm + 670 nm + 700 nm + 750 nm + 780 nm + 850 nm	0.471	0.700

Table 28. Held-out test performance of the neural network using the six relative-reflectance bands, three NDVI combinations, and growth stage as inputs, dark room dataset, for every possible set of three NDVI combinations (n = 84), sorted by R^2. The best combination is shown in bold.

Rank	NDVI combinations added to the six bands	R^2	r
1	**NDVI(660, 850), NDVI(670, 780), NDVI(700, 780)**	**0.817**	**0.923**
2	NDVI(660, 780), NDVI(670, 850), NDVI(700, 780)	0.811	0.922
3	NDVI(660, 850), NDVI(670, 750), NDVI(700, 780)	0.806	0.927
4	NDVI(660, 850), NDVI(670, 780), NDVI(670, 850)	0.806	0.918
5	NDVI(660, 750), NDVI(670, 850), NDVI(700, 780)	0.804	0.914
6	NDVI(660, 750), NDVI(660, 780), NDVI(670, 750)	0.800	0.924
7	NDVI(660, 750), NDVI(670, 750), NDVI(700, 850)	0.796	0.906
8	NDVI(660, 750), NDVI(660, 780), NDVI(700, 780)	0.794	0.913
9	NDVI(660, 850), NDVI(700, 750), NDVI(700, 780)	0.792	0.917
10	NDVI(660, 750), NDVI(670, 780), NDVI(700, 780)	0.792	0.923
11	NDVI(660, 850), NDVI(670, 850), NDVI(700, 780)	0.789	0.913
12	NDVI(670, 750), NDVI(670, 780), NDVI(700, 780)	0.788	0.913
13	NDVI(660, 850), NDVI(670, 750), NDVI(700, 850)	0.788	0.925
14	NDVI(670, 750), NDVI(670, 850), NDVI(700, 750)	0.784	0.916
15	NDVI(660, 750), NDVI(660, 780), NDVI(660, 850)	0.783	0.907
16	NDVI(660, 750), NDVI(670, 750), NDVI(700, 750)	0.783	0.897
17	NDVI(660, 780), NDVI(700, 780), NDVI(700, 850)	0.781	0.905
18	NDVI(660, 850), NDVI(670, 750), NDVI(670, 780)	0.781	0.913
19	NDVI(660, 780), NDVI(660, 850), NDVI(700, 780)	0.779	0.901

Rank	NDVI combinations added to the six bands	R^2	r
20	NDVI(670, 750), NDVI(670, 850), NDVI(700, 850)	0.779	0.907
21	NDVI(660, 780), NDVI(660, 850), NDVI(700, 850)	0.778	0.902
22	NDVI(670, 780), NDVI(670, 850), NDVI(700, 850)	0.777	0.903
23	NDVI(660, 850), NDVI(670, 750), NDVI(700, 750)	0.777	0.899
24	NDVI(660, 750), NDVI(670, 780), NDVI(700, 750)	0.777	0.909
25	NDVI(660, 750), NDVI(660, 850), NDVI(670, 780)	0.777	0.912
26	NDVI(660, 850), NDVI(670, 750), NDVI(670, 850)	0.775	0.916
27	NDVI(660, 750), NDVI(660, 850), NDVI(700, 850)	0.775	0.913
28	NDVI(660, 750), NDVI(700, 780), NDVI(700, 850)	0.774	0.899
29	NDVI(670, 750), NDVI(670, 780), NDVI(700, 850)	0.772	0.908
30	NDVI(700, 750), NDVI(700, 780), NDVI(700, 850)	0.770	0.905
31	NDVI(670, 750), NDVI(700, 750), NDVI(700, 850)	0.770	0.916
32	NDVI(670, 750), NDVI(700, 780), NDVI(700, 850)	0.769	0.913
33	NDVI(670, 780), NDVI(700, 780), NDVI(700, 850)	0.769	0.896
34	NDVI(660, 780), NDVI(670, 750), NDVI(700, 780)	0.768	0.909
35	NDVI(670, 750), NDVI(700, 750), NDVI(700, 780)	0.767	0.900
36	NDVI(660, 780), NDVI(670, 750), NDVI(700, 850)	0.767	0.901
37	NDVI(660, 780), NDVI(670, 750), NDVI(670, 850)	0.766	0.902
38	NDVI(660, 850), NDVI(670, 780), NDVI(700, 850)	0.766	0.910
39	NDVI(660, 750), NDVI(670, 780), NDVI(700, 850)	0.765	0.912
40	NDVI(660, 780), NDVI(670, 780), NDVI(700, 780)	0.765	0.901

Rank	NDVI combinations added to the six bands	R^2	r
41	NDVI(670, 750), NDVI(670, 780), NDVI(670, 850)	0.765	0.897
42	NDVI(660, 750), NDVI(660, 850), NDVI(670, 850)	0.764	0.893
43	NDVI(660, 750), NDVI(660, 780), NDVI(670, 850)	0.763	0.910
44	NDVI(660, 850), NDVI(670, 850), NDVI(700, 750)	0.763	0.899
45	NDVI(660, 750), NDVI(670, 750), NDVI(670, 850)	0.763	0.917
46	NDVI(660, 750), NDVI(700, 750), NDVI(700, 780)	0.763	0.901
47	NDVI(660, 850), NDVI(700, 750), NDVI(700, 850)	0.761	0.908
48	NDVI(660, 750), NDVI(670, 750), NDVI(700, 780)	0.758	0.907
49	NDVI(660, 750), NDVI(670, 850), NDVI(700, 850)	0.756	0.895
50	NDVI(660, 780), NDVI(660, 850), NDVI(670, 750)	0.755	0.905
51	NDVI(660, 780), NDVI(700, 750), NDVI(700, 850)	0.755	0.902
52	NDVI(660, 780), NDVI(670, 850), NDVI(700, 850)	0.754	0.899
53	NDVI(670, 850), NDVI(700, 780), NDVI(700, 850)	0.754	0.908
54	NDVI(660, 780), NDVI(660, 850), NDVI(700, 750)	0.753	0.896
55	NDVI(660, 750), NDVI(660, 850), NDVI(700, 750)	0.753	0.895
56	NDVI(660, 750), NDVI(670, 850), NDVI(700, 750)	0.752	0.891
57	NDVI(670, 750), NDVI(670, 850), NDVI(700, 780)	0.751	0.909
58	NDVI(660, 780), NDVI(670, 850), NDVI(700, 750)	0.750	0.899
59	NDVI(660, 780), NDVI(670, 750), NDVI(670, 780)	0.750	0.894
60	NDVI(660, 850), NDVI(670, 780), NDVI(700, 750)	0.746	0.896
61	NDVI(660, 750), NDVI(660, 850), NDVI(670, 750)	0.745	0.898

Rank	NDVI combinations added to the six bands	R^2	r
62	NDVI(660, 780), NDVI(670, 780), NDVI(700, 850)	0.744	0.889
63	NDVI(660, 850), NDVI(670, 850), NDVI(700, 850)	0.744	0.909
64	NDVI(660, 780), NDVI(700, 750), NDVI(700, 780)	0.743	0.894
65	NDVI(660, 780), NDVI(660, 850), NDVI(670, 780)	0.740	0.889
66	NDVI(670, 780), NDVI(700, 750), NDVI(700, 850)	0.739	0.890
67	NDVI(660, 780), NDVI(660, 850), NDVI(670, 850)	0.738	0.891
68	NDVI(660, 780), NDVI(670, 780), NDVI(670, 850)	0.737	0.881
69	NDVI(660, 750), NDVI(660, 780), NDVI(670, 780)	0.735	0.895
70	NDVI(660, 850), NDVI(700, 780), NDVI(700, 850)	0.731	0.904
71	NDVI(660, 750), NDVI(670, 750), NDVI(670, 780)	0.731	0.898
72	NDVI(660, 750), NDVI(700, 750), NDVI(700, 850)	0.727	0.880
73	NDVI(660, 750), NDVI(660, 850), NDVI(700, 780)	0.722	0.880
74	NDVI(670, 780), NDVI(670, 850), NDVI(700, 750)	0.720	0.887
75	NDVI(660, 750), NDVI(660, 780), NDVI(700, 750)	0.720	0.880
76	NDVI(670, 850), NDVI(700, 750), NDVI(700, 780)	0.719	0.906
77	NDVI(670, 850), NDVI(700, 750), NDVI(700, 850)	0.709	0.903
78	NDVI(670, 750), NDVI(670, 780), NDVI(700, 750)	0.707	0.884
79	NDVI(660, 780), NDVI(670, 750), NDVI(700, 750)	0.703	0.865
80	NDVI(670, 780), NDVI(700, 750), NDVI(700, 780)	0.701	0.863
81	NDVI(660, 750), NDVI(660, 780), NDVI(700, 850)	0.700	0.880
82	NDVI(660, 780), NDVI(670, 780), NDVI(700, 750)	0.698	0.874

Rank	NDVI combinations added to the six bands	R^2	r
83	NDVI(670, 780), NDVI(670, 850), NDVI(700, 780)	0.697	0.888
84	NDVI(660, 750), NDVI(670, 780), NDVI(670, 850)	0.667	0.879

Table 29. Held-out test performance of the neural network using the six relative-reflectance bands, three NDVI combinations, and growth stage as inputs, outdoor dataset, for every possible set of three NDVI combinations (n = 84), sorted by R^2. The best combination is shown in bold.

Rank	NDVI combinations added to the six bands	R^2	r
1	**NDVI(670, 750), NDVI(700, 750), NDVI(700, 780)**	**0.503**	**0.773**
2	NDVI(670, 750), NDVI(670, 850), NDVI(700, 850)	0.488	0.775
3	NDVI(660, 850), NDVI(670, 750), NDVI(670, 780)	0.486	0.786
4	NDVI(660, 850), NDVI(670, 780), NDVI(700, 850)	0.478	0.787
5	NDVI(660, 850), NDVI(670, 850), NDVI(700, 850)	0.460	0.771
6	NDVI(670, 750), NDVI(670, 850), NDVI(700, 750)	0.450	0.757
7	NDVI(670, 750), NDVI(700, 750), NDVI(700, 850)	0.445	0.793
8	NDVI(670, 750), NDVI(670, 850), NDVI(700, 780)	0.444	0.725
9	NDVI(660, 750), NDVI(670, 750), NDVI(700, 750)	0.437	0.728
10	NDVI(660, 850), NDVI(670, 850), NDVI(700, 750)	0.433	0.742
11	NDVI(660, 850), NDVI(670, 750), NDVI(670, 850)	0.429	0.722
12	NDVI(670, 750), NDVI(700, 780), NDVI(700, 850)	0.427	0.759
13	NDVI(660, 850), NDVI(670, 850), NDVI(700, 780)	0.421	0.706
14	NDVI(670, 750), NDVI(670, 780), NDVI(700, 780)	0.420	0.744

Rank	NDVI combinations added to the six bands	R^2	r
15	NDVI(660, 850), NDVI(670, 750), NDVI(700, 780)	0.412	0.714
16	NDVI(660, 780), NDVI(660, 850), NDVI(700, 780)	0.400	0.722
17	NDVI(660, 850), NDVI(700, 750), NDVI(700, 850)	0.396	0.716
18	NDVI(700, 750), NDVI(700, 780), NDVI(700, 850)	0.394	0.710
19	NDVI(660, 850), NDVI(670, 750), NDVI(700, 850)	0.392	0.694
20	NDVI(660, 850), NDVI(700, 750), NDVI(700, 780)	0.390	0.711
21	NDVI(660, 780), NDVI(670, 750), NDVI(670, 780)	0.385	0.726
22	NDVI(660, 850), NDVI(700, 780), NDVI(700, 850)	0.382	0.688
23	NDVI(660, 750), NDVI(660, 850), NDVI(700, 750)	0.378	0.710
24	NDVI(660, 780), NDVI(700, 780), NDVI(700, 850)	0.378	0.730
25	NDVI(660, 750), NDVI(700, 780), NDVI(700, 850)	0.373	0.692
26	NDVI(660, 850), NDVI(670, 780), NDVI(700, 780)	0.372	0.699
27	NDVI(660, 780), NDVI(660, 850), NDVI(700, 850)	0.370	0.715
28	NDVI(660, 750), NDVI(670, 750), NDVI(670, 780)	0.365	0.736
29	NDVI(670, 780), NDVI(670, 850), NDVI(700, 780)	0.365	0.691
30	NDVI(660, 780), NDVI(670, 780), NDVI(700, 780)	0.363	0.677
31	NDVI(660, 780), NDVI(660, 850), NDVI(700, 750)	0.361	0.679
32	NDVI(660, 850), NDVI(670, 780), NDVI(700, 750)	0.360	0.721
33	NDVI(660, 780), NDVI(700, 750), NDVI(700, 780)	0.355	0.719
34	NDVI(660, 850), NDVI(670, 750), NDVI(700, 750)	0.354	0.680
35	NDVI(660, 750), NDVI(660, 850), NDVI(700, 850)	0.353	0.662

Rank	NDVI combinations added to the six bands	R^2	r
36	NDVI(660, 750), NDVI(670, 750), NDVI(670, 850)	0.351	0.669
37	NDVI(670, 750), NDVI(670, 780), NDVI(670, 850)	0.350	0.705
38	NDVI(660, 750), NDVI(660, 780), NDVI(660, 850)	0.349	0.670
39	NDVI(660, 780), NDVI(670, 750), NDVI(700, 780)	0.346	0.662
40	NDVI(660, 780), NDVI(670, 750), NDVI(700, 850)	0.345	0.687
41	NDVI(670, 780), NDVI(670, 850), NDVI(700, 850)	0.344	0.766
42	NDVI(660, 750), NDVI(660, 780), NDVI(670, 750)	0.342	0.686
43	NDVI(660, 780), NDVI(670, 750), NDVI(700, 750)	0.341	0.730
44	NDVI(660, 750), NDVI(660, 850), NDVI(670, 850)	0.334	0.648
45	NDVI(670, 780), NDVI(700, 780), NDVI(700, 850)	0.332	0.673
46	NDVI(660, 750), NDVI(700, 750), NDVI(700, 850)	0.331	0.665
47	NDVI(670, 780), NDVI(700, 750), NDVI(700, 850)	0.331	0.732
48	NDVI(660, 780), NDVI(670, 850), NDVI(700, 850)	0.331	0.634
49	NDVI(660, 780), NDVI(670, 750), NDVI(670, 850)	0.331	0.639
50	NDVI(660, 750), NDVI(670, 750), NDVI(700, 850)	0.324	0.630
51	NDVI(660, 780), NDVI(660, 850), NDVI(670, 850)	0.323	0.639
52	NDVI(660, 750), NDVI(660, 850), NDVI(670, 750)	0.316	0.645
53	NDVI(660, 750), NDVI(660, 780), NDVI(700, 780)	0.316	0.631
54	NDVI(660, 750), NDVI(660, 780), NDVI(700, 750)	0.315	0.654
55	NDVI(660, 750), NDVI(670, 750), NDVI(700, 780)	0.313	0.627
56	NDVI(670, 750), NDVI(670, 780), NDVI(700, 850)	0.313	0.709

Rank	NDVI combinations added to the six bands	R^2	r
57	NDVI(670, 780), NDVI(700, 750), NDVI(700, 780)	0.312	0.757
58	NDVI(660, 780), NDVI(670, 850), NDVI(700, 780)	0.310	0.606
59	NDVI(660, 780), NDVI(660, 850), NDVI(670, 750)	0.309	0.640
60	NDVI(670, 850), NDVI(700, 750), NDVI(700, 780)	0.309	0.650
61	NDVI(670, 750), NDVI(670, 780), NDVI(700, 750)	0.306	0.724
62	NDVI(660, 780), NDVI(700, 750), NDVI(700, 850)	0.306	0.681
63	NDVI(660, 750), NDVI(670, 850), NDVI(700, 850)	0.304	0.600
64	NDVI(670, 850), NDVI(700, 780), NDVI(700, 850)	0.303	0.654
65	NDVI(660, 750), NDVI(660, 850), NDVI(700, 780)	0.296	0.627
66	NDVI(670, 780), NDVI(670, 850), NDVI(700, 750)	0.283	0.672
67	NDVI(660, 780), NDVI(670, 780), NDVI(700, 850)	0.283	0.602
68	NDVI(660, 780), NDVI(670, 780), NDVI(700, 750)	0.279	0.728
69	NDVI(660, 750), NDVI(700, 750), NDVI(700, 780)	0.275	0.630
70	NDVI(660, 750), NDVI(660, 780), NDVI(670, 850)	0.274	0.574
71	NDVI(660, 750), NDVI(670, 780), NDVI(700, 750)	0.274	0.662
72	NDVI(660, 750), NDVI(660, 780), NDVI(700, 850)	0.260	0.576
73	NDVI(660, 750), NDVI(660, 780), NDVI(670, 780)	0.259	0.691
74	NDVI(660, 780), NDVI(670, 780), NDVI(670, 850)	0.258	0.566
75	NDVI(660, 780), NDVI(670, 850), NDVI(700, 750)	0.240	0.614
76	NDVI(660, 750), NDVI(670, 780), NDVI(700, 780)	0.230	0.588
77	NDVI(660, 750), NDVI(670, 850), NDVI(700, 780)	0.210	0.532

Rank	NDVI combinations added to the six bands	R^2	r
78	NDVI(660, 750), NDVI(670, 780), NDVI(670, 850)	0.206	0.553
79	NDVI(660, 780), NDVI(660, 850), NDVI(670, 780)	0.200	0.567
80	NDVI(670, 850), NDVI(700, 750), NDVI(700, 850)	0.199	0.592
81	NDVI(660, 750), NDVI(670, 780), NDVI(700, 850)	0.178	0.550
82	NDVI(660, 850), NDVI(670, 780), NDVI(670, 850)	0.175	0.555
83	NDVI(660, 750), NDVI(660, 850), NDVI(670, 780)	0.152	0.520
84	NDVI(660, 750), NDVI(670, 850), NDVI(700, 750)	0.146	0.515

Appendix D

Firmware Source Code

This appendix lists the firmware source code for the two microcontroller units of the handheld sensor. The LED control unit runs on an STM32H725VGT6 microcontroller. It switches the six LED strings through the shift register, samples the SiPM signal with the on-chip ADC, stores the readings on the SD card, and exchanges commands and results with the display unit over SPI. The display unit runs on an STM32F746ZET6 microcontroller under FreeRTOS with the TouchGFX graphics framework. It provides the touch-screen user interface, sends measurement commands to the LED control unit, and displays the returned results.

Only the source files written for this project are reproduced. Peripheral-initialization files generated by STM32CubeMX (for example adc.c, spi.c, and gpio.c), the STM32 HAL and CMSIS driver libraries, the FatFs and FreeRTOS middleware, the code generated by TouchGFX Designer, and the on-screen keyboard files adapted from the TouchGFX keyboard example are omitted. Each main.c file is reproduced in full, including the CubeMX-generated initialization code that surrounds the source code sections.

LED Control Unit Firmware (STM32H725VGT6)

File: `Core/Inc/main.h`

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.h
```

```
  * @brief          : Header for main.c file.
  *                   This file contains the common defines of the application.
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32h7xx_hal.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
//#include "fatfs.h"

/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */

/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */

typedef struct {
    float gain;
    float phase;
}avg_sensor_read_t;

/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */
void start_detection();
int  modulation_control_sequence();
void generate_sine_wave();
avg_sensor_read_t demodulation_control_sequence(uint8_t spectrumIndex,float initial_reading);
void set_rst_and_enablePin_SPIs();
uint16_t *retrieve_gain_phase_and_Store();
static inline float running_avg(float prev_avg, uint32_t n_prev, float x_new);
float find_reflection_coefficient(float avg_gain,float Led_area,float spot_diameter,float responsivity,float
    ↪ source_radiant_flux);
void reset_recieved_bytes();
static void dcache_clean_by_addr_safe(void *addr, size_t size);
```

```
static inline float apply_gain_normalization(uint8_t spectrum, float raw_gain, const float norm[7]);
int  caliberation_modulation_control_sequence();
int  Test_modulation_control_sequence();
void Test_scan_write_sequence(void);
int  LEDs_control_sequence();
int scan_write_sequence(void);
void reset_recieved_bytes(void);
void transmission_listenner();
void set_opperation_complete_id(uint16_t id);
uint16_t get_opperation_complete_id(void);


/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/
#define Mod_Sig_Pin GPIO_PIN_4
#define Mod_Sig_GPIO_Port GPIOA
#define Gain_MCU_Pin GPIO_PIN_6
#define Gain_MCU_GPIO_Port GPIOA
#define Phase_MCU_Pin GPIO_PIN_4
#define Phase_MCU_GPIO_Port GPIOC
#define SPI2_NSS_Pin GPIO_PIN_12
#define SPI2_NSS_GPIO_Port GPIOB
#define Transmission_Complete_Pin GPIO_PIN_15
#define Transmission_Complete_GPIO_Port GPIOB
#define Start_Mod_MCU_Pin GPIO_PIN_12
#define Start_Mod_MCU_GPIO_Port GPIOD
#define Start_LDD_MCU_Pin GPIO_PIN_13
#define Start_LDD_MCU_GPIO_Port GPIOD
#define SD_Detect_Pin GPIO_PIN_7
#define SD_Detect_GPIO_Port GPIOC
#define LD_Ctrl_CLR_Pin GPIO_PIN_4
#define LD_Ctrl_CLR_GPIO_Port GPIOD
#define LD_Ctrl_Enable_Pin GPIO_PIN_5
#define LD_Ctrl_Enable_GPIO_Port GPIOD

/* USER CODE BEGIN Private defines */


#define PI 3.1415926
#define radius 0.08
#define op_amp_gain 1.1
#define transimpedance_gain 330
//#define responsivity

#define North_Sensor      1U
#define Middle_Sensor     2U
#define East_Sensor       3U
#define West_Sensor       4U

/* control states*/
#define start_detection 0x01
#define stop_detection  0x02


/*silicon photo multiplier structs and data structure*/
typedef struct __attribute__((__packed__)) {
    float gain;
    float phase;
} SIPM_t;

#define NUM_FRAMES   1
#define NUM_SENSORS  4
#define NUM_SPECTRA  7

/*flag events*/
typedef enum {
    SPI_EVENT_START_READ =0x01,         // start reading flag
```

```
    SPI_EVENT_STOP_READ  = 0x00,        // stop reading spi flag
    SPI_EVENT_READ_FINISH = 0x02,       // finish reading spi flag
    SPI_EVENT_SAVE_FRAMES_PLANT_REGION = 0x03,  // save leave frames for a particular plant region spi flag
    SPI_EVENT_SAVE_PLANT = 0x04 // save plant regions into one plant file


} spi_event_flags;
typedef struct {
    uint16_t syncWord;   // e.g., 0xAA55
    uint8_t frameID;
    uint16_t payloadSize;
} __attribute__((packed)) FrameHeader_t;

typedef struct {
    FrameHeader_t header;
    SIPM_t data[NUM_SENSORS][NUM_SPECTRA];
} __attribute__((packed)) FramePacket_t;

typedef enum {
    REGION_LOWER = 0,
    REGION_MIDDLE = 1,
    REGION_UPPER = 2
} RegionID_t;



/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */
```

File: Core/Src/main.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "adc.h"
#include "dma.h"
#include "fatfs.h"
#include "sdmmc.h"
#include "spi.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
```

```
#include "string.h"
#include "stdio.h"
#include "math.h"
#include "stdbool.h"
#include <string.h>
#define MAX_MEMSET_RETRIES  5
//#include "storage.h"

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/*
 * SPI2 Defines
 */
#define XFER_LEN  500

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */

/*
 * Spi2 Variables
 */
//uint8_t txBuf[XFER_LEN] = {0x78};
static volatile uint8_t spi_frame_done = 0;
static volatile uint8_t spi_frame_active = 0;
volatile uint8_t command_received_flag = 0;
volatile uint8_t spi_rx_ready = 0;
HAL_StatusTypeDef st;
uint16_t opperation_complete_id = 0xBB;

//uint8_t rxDummy[XFER_LEN] = {0xAA};
//uint8_t rxDummy2[XFER_LEN] = {0xAA};
uint8_t idx;

__attribute__((aligned(32))) uint8_t rxDummy[XFER_LEN];
__attribute__((aligned(32))) uint8_t txBuf[XFER_LEN];



/*
 * laser reflectance reading storage
 */

FramePacket_t framePacket;
uint8_t currentLeafID = 0;
uint8_t currentRegion = 0;
static uint8_t currentPlantID;
extern int next_plant_index;
bool region_complete;
bool new_plant_flag = false;
FRESULT fr ;

float fixed_electrical_noise = 0.28; // noise when no signal is present.
```

```
int transmission_state = 1;


/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MPU_Config(void);
/* USER CODE BEGIN PFP */

/*
 * SPI2 Functions Prototype
 */

static inline void SPI_PreloadResponse(void);
const char* regionName(uint8_t region_id);



/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MPU Configuration--------------------------------------------------------*/
  MPU_Config();

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_SPI2_Init();
  MX_ADC1_Init();
  MX_SPI1_Init();
  MX_SDMMC1_SD_Init();
  MX_FATFS_Init();
  /* USER CODE BEGIN 2 */

  start_storage(&hsd1);
```

```
  framePacket.header.frameID = 0;
  set_rst_and_enablePin_SPIs();
  HAL_GPIO_WritePin(GPIOD,Start_Mod_MCU_Pin,GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA,Mod_Sig_Pin,GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);

  uint8_t setData_alpha = 0xFF;
  HAL_SPI_Transmit(&hspi1, &setData_alpha, 1, HAL_MAX_DELAY);
  reset_recieved_bytes();  // optional


  //ADC Conviguration
//  ADC_ChannelConfTypeDef ADC_CH_Cfg = {0};  // Declare and initialize
//
//  ADC_CH_Cfg.Channel = ADC_Channels[i];         // Select The ADC Channel [i]
//  HAL_ADC_ConfigChannel(&hadc1, &ADC_CH_Cfg);   // Configure The Selected ADC Channel
    ↪  // Start ADC Conversion @ Selected Channel

//  HAL_Delay(200);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
//    SPI_PreloadResponse();
//    modulation_control_sequence();
//    HAL_GPIO_WritePin(GPIOD,Start_Mod_MCU_Pin,GPIO_PIN_SET);
//    HAL_Delay(5);
//    HAL_GPIO_WritePin(GPIOD,Start_Mod_MCU_Pin,GPIO_PIN_RESET);
//    HAL_Delay(5);

//    Test_scan_write_sequence();


     uint8_t setData = 0xFF;
     HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);




//   st = HAL_SPI_TransmitReceive_DMA(&hspi2, txBuf, rxDummy, XFER_LEN);
//   HAL_Delay(50);
//   scan_write_sequence();

      // Start a DMA transaction only if one is not active
    transmission_state = HAL_GPIO_ReadPin(GPIOB, Transmission_Complete_Pin);


//   if(transmission_state == SET)
//   {
//      HAL_SPI_Abort(&hspi2);
//      HAL_Delay(60);
//
//      st = HAL_SPI_TransmitReceive_DMA(&hspi2, txBuf, rxDummy, XFER_LEN);
//      if (st != HAL_OK)
```

```
//      {
//          HAL_SPI_Abort(&hspi2);
//          spi_frame_active = 0;   // failed to start, allow retry
//          HAL_Delay(60);
//
//      }
//   }


//   HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);
//   HAL_Delay(2000);
//   HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_RESET);
//   HAL_Delay(2000);




    if (!spi_frame_active)
    {
        spi_frame_active = 1;
        spi_rx_ready = 0;
        HAL_SPI_Abort(&hspi2);
//      HAL_Delay(60);
//      HAL_SPI_Init(&hspi2);



//      if(rxDummy[231]==0xAB)
//      {
//          txBuf[231] = 0x00;
////            reset_recieved_bytes();
////            resert_data_storage();
//
//
//
//      }
//
//      else if(rxDummy[231]!=0xAB)
//      {
//          txBuf[231]==0xFF;
//
//            // now rxDummy is a *complete frame*
//      }
        fr = find_last_index_in_dir("field001", "Plant_", &idx);
        txBuf[230] = idx;
        txBuf[231] = 0xBB;
//      HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);

//      HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);


//      switch (rxDummy[1])
//      {
//
//       /* 0xBB = Scan complete
//           * 0xB1 = new plant
//           * 0xB2 = new region
//           * 0xB3 = save*/
//
//          case 0x00:
//
//              set_opperation_complete_id(0xB3);
//              break;
//
//          case 0x01:
//
//              set_opperation_complete_id(0xBB);
```

```
//              break;
//
//
//          case 0x02:
//
//              set_opperation_complete_id(0xB2);
//              break;
//
//
//          case 0x03:
//
//              set_opperation_complete_id(0xB1);
//              break;
//
//      }
//
//      HAL_Delay(2000);


        HAL_Delay(60);
        st = HAL_SPI_TransmitReceive_DMA(&hspi2, txBuf, rxDummy, XFER_LEN);
        if (st != HAL_OK)
        {
            HAL_SPI_Abort(&hspi2);
            spi_frame_active = 0;   // failed to start, allow retry
            HAL_Delay(60);

        }


    }



    // Process ONLY after DMA says it's done
    if (spi_rx_ready)
    {


        st = HAL_SPI_TransmitReceive_DMA(&hspi2, txBuf, rxDummy, XFER_LEN);
        if (st != HAL_OK)
        {
            HAL_SPI_Abort(&hspi2);
            spi_frame_active = 0;   // failed to start, allow retry
            HAL_Delay(60);

        }






//      if(rxDummy[231] != 0xAB)
//      {
//          HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);
//          txBuf[231] = 0xBB;
//          st = HAL_SPI_TransmitReceive_DMA(&hspi2, txBuf, rxDummy, XFER_LEN);
//          reset_recieved_bytes();  // optional
//
//      }
////
//      else
//      {
//          HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);
//          txBuf[231] = 0xBB;
//
//
```

```
////            if(HAL_GPIO_ReadPin(GPIOB,SPI2_NSS_Pin)== 0)
////            {
////                printf("I am here");
////            }
//          st = HAL_SPI_TransmitReceive_DMA(&hspi2, txBuf, rxDummy, XFER_LEN);
//      }
//      HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);

          // now rxDummy is a *complete frame*
//      HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_RESET);
//      reset_recieved_bytes();  // optional
//      spi_rx_ready = 0;
//      spi_frame_active = 0;

////        txBuf[231] = 0xCC;
//      int scan_success = scan_write_sequence();
//
        if(rxDummy[7] != 0x20)
        {

            txBuf[231] = 0xBB;
//          HAL_SPI_Abort(&hspi2);
//          HAL_Delay(60);
//          HAL_SPI_Init(&hspi2);


//          txBuf[231] = get_opperation_complete_id();

//          switch (rxDummy[1])
//          {
//
//           /* 0xBB = Scan complete
//               * 0xB1 = new plant
//               * 0xB2 = new region
//               * 0xB3 = save*/
//
//              case 0x00:
//
//                  set_opperation_complete_id(0xB3);
//                  break;
//
//              case 0x01:
//
//                  set_opperation_complete_id(0xBB);
//                  break;
//
//
//              case 0x02:
//
//                  set_opperation_complete_id(0xB2);
//                  break;
//
//
//              case 0x03:
//
//                  set_opperation_complete_id(0xB1);
//                  break;
//
//          }



//          int scan_success = scan_write_sequence();

            int scan_success = scan_write_sequence();
            st = HAL_SPI_TransmitReceive_DMA(&hspi2, txBuf, rxDummy, XFER_LEN);

            HAL_Delay(50);
```

```
            if (st != HAL_OK)
            {
                HAL_SPI_Abort(&hspi2);
                spi_frame_active = 0;   // failed to start, allow retry
                HAL_Delay(60);

            }



//              allow next DMA start
//          spi_rx_ready = 1;




        }

        /*
         * this if statement checks if the master has given a command and the command is valid. if the master'
    ↪ s command is valid rxDummy[7] ==
         * 0x20 if not it wouldn't be != 0x20
         *
         */
        else if(rxDummy[7] == 0x20)
        {
//          HAL_SPI_Abort(&hspi2);
//          HAL_Delay(60);
//          HAL_SPI_Init(&hspi2);



            if(rxDummy[0] == 0x01 || rxDummy[0] == 0x05 || rxDummy[0] == 0x06)
            {

                spi_rx_ready = 0;
                spi_frame_active = 0;
                txBuf[231] = 0xCB;
                uint8_t tries = 0;

                while(tries != 1)
                {
                    tries++;
                    LEDs_control_sequence();
                }

                HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);
                st = HAL_SPI_TransmitReceive_DMA(&hspi2, txBuf, rxDummy, XFER_LEN);
//
                if (st != HAL_OK)
                {
                    spi_frame_active = 0;   // failed to start, allow retry
                    HAL_SPI_Abort(&hspi2);
                    HAL_Delay(60);

                }


                reset_recieved_bytes();  // optional

            }

            else
```

```
            {

                transmission_listenner();
                reset_recieved_bytes();  // optional
            }



        }

        reset_recieved_bytes();


//      spi_rx_ready = 0;
//      spi_frame_active = 0;
//      reset_recieved_bytes();  // optional







    }

//    HAL_SPI_TransmitReceive_DMA(&hspi2, txBuf, rxDummy, 500);
//
//    HAL_Delay(70);
//
//    scan_write_sequence();

//    HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_RESET);
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Supply configuration update enable
  */
  HAL_PWREx_ConfigSupply(PWR_DIRECT_SMPS_SUPPLY);

  /** Configure the main internal regulator output voltage
  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  while(!__HAL_PWR_GET_FLAG(PWR_FLAG_VOSRDY)) {}

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 24;
  RCC_OscInitStruct.PLL.PLLP = 1;
  RCC_OscInitStruct.PLL.PLLQ = 3;
  RCC_OscInitStruct.PLL.PLLR = 2;
```

```
  RCC_OscInitStruct.PLL.PLLRGE = RCC_PLL1VCIRANGE_3;
  RCC_OscInitStruct.PLL.PLLVCOSEL = RCC_PLL1VCOWIDE;
  RCC_OscInitStruct.PLL.PLLFRACN = 1536;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2
                              |RCC_CLOCKTYPE_D3PCLK1|RCC_CLOCKTYPE_D1PCLK1;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.SYSCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB3CLKDivider = RCC_APB3_DIV2;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_APB1_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_APB2_DIV2;
  RCC_ClkInitStruct.APB4CLKDivider = RCC_APB4_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */

static inline float running_avg(float prev_avg, uint32_t n_prev, float x_new) {
    // returns new average after adding x_new (n_prev is how many samples were already averaged)
    return (prev_avg * n_prev + x_new) / (n_prev + 1);
}

int  caliberation_modulation_control_sequence()
{
//  HAL_GPIO_WritePin(GPIOD, Start_LDD_MCU_Pin, GPIO_PIN_RESET);

uint8_t shiftPattern[] = {0xFE, 0xFD, 0xFB, 0xF7,0xEF, 0xDF, 0xBF,0x7F,0xFF};

    uint32_t samples_per_spectrum[9] = {0};  // how many updates each spectrum has received


    uint32_t hold_duration = 300;
    uint8_t setData = 0xFF;
    uint8_t resetData = 0x00;
//    memset(framePacket.data, 0, sizeof framePacket.data);
    float new_gain  = 0.0;
    float new_phase = 0.0;
    float led_area = 0.0;
    float responsivity = 0.0;
    float spot_diameter = 0.08;
    float source_radiant_flux = 0.0;

    float normFactors[NUM_SPECTRA];


//   uint32_t next_idx= 0;
//   fr = find_last_index_in_dir("field001", "Calibration_", &next_idx);
//   if (fr != FR_OK)
//   {
//       continue;
//
//  }
//
//   else if(fr ==FR_OK)
//   {
//       caliberate_id = next_idx;
```

```
//       load_norm_factors("field001", caliberate_id, normFactors);
//   }




    for (int spectrum = 0; spectrum < 9 ; spectrum++)
    {
        uint32_t start_time = HAL_GetTick();
//            HAL_SPI_Transmit(&hspi1, &resetData, 1, HAL_MAX_DELAY);
        HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);
//            HAL_Delay(50);

        /**
         * below code is responsible for retrieving signal from detector before
         * LEDS are turned on.
         */
//      uint16_t* gain_phase_before;
//      gain_phase_before = retrieve_gain_phase_and_Store();  // returns uint16_t[2]
//      float total_gain_before = 0.0f, total_phase_before = 0.0f;
//      avg_sensor_read_t avg_after;
//
//      HAL_Delay(70);

        uint16_t* gain_phase_before;
        gain_phase_before = retrieve_gain_phase_and_Store();  // returns uint16_t[2]
        float total_gain_before = 0.0f, total_phase_before = 0.0f;
        avg_sensor_read_t avg_after;

        HAL_Delay(50);


        total_gain_before += gain_phase_before[0];
        total_phase_before += gain_phase_before[1];
//      HAL_SPI_Transmit(&hspi1, &shiftPattern[spectrum], 1, HAL_MAX_DELAY);
        HAL_Delay(5);
        avg_after = demodulation_control_sequence(spectrum,total_gain_before);// Process all sensors at this
    ↪ spectrum
        HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);
        HAL_Delay(50);
//
//      avg_after.gain = avg_after.gain - gain_phase_before[0]/65536.0f * 3.3f;

//            float final_avg_gain = avg_after.gain;
//            float final_avg_phase = avg_after.phase;
//
//            framePacket.data[2][spectrum].gain = framePacket.data[2][spectrum].gain +final_avg_gain;
//            framePacket.data[2][spectrum].phase = framePacket.data[2][spectrum].gain + final_avg_phase;
//
        /* Running average per spectrum */
        uint32_t n_prev = samples_per_spectrum[spectrum];       // 0,1,2,... up to 10


        new_gain  = avg_after.gain;
        new_phase = avg_after.phase;
        uint8_t caliberate_id;




        if(spectrum == 0)
        {
```

```
            // later, per spectrum:
//          new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);
            new_phase = avg_after.phase;   // usually leave phase untouched


            new_gain  = avg_after.gain; // 0.25 is a  correction factor

            new_phase = avg_after.phase;
            led_area = 0.002;
            responsivity = 2.8;
            source_radiant_flux = 640E-03;
        }

        else if(spectrum == 1)
        {

//          new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);

            new_gain  = avg_after.gain; // 0.10 is a correction factor
            new_phase = avg_after.phase;
            led_area = 0.00196;
            responsivity = 2.8;
            source_radiant_flux = 500E-03;

        }

        else if(spectrum == 2)
        {

//          new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);
            new_gain  = avg_after.gain;
            led_area = 0.00188;
            responsivity = 3.0;
            source_radiant_flux = 590E-03;
        }

        else if(spectrum == 3)
        {

            new_gain  = avg_after.gain;
            new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);

            new_phase = avg_after.phase;


            led_area = 0.00196;

            responsivity = 2.9;
            source_radiant_flux = 480E-03;
        }

        else if(spectrum == 4)
        {
            new_gain  = avg_after.gain;
//          new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);

            new_phase = avg_after.phase;

            led_area = 0.00196;
            responsivity = 2.8;
            source_radiant_flux = 420E-03;
        }


        else if(spectrum == 5)
        {
```

```
            new_gain  = avg_after.gain;
//          new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);

            new_phase = avg_after.phase;

            led_area = 0.001;
            responsivity = 2.7;
            source_radiant_flux = 753E-03;
        }


        else if(spectrum == 6)
        {

            new_gain  = avg_after.gain;
//          new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);


            new_phase = avg_after.phase;
            led_area = 0.0010;
            responsivity = 2.8;
            source_radiant_flux = 800E-03;
        }



//            float prev_avg_gain  = framePacket.data[2][spectrum].gain;
//            float prev_avg_phase = framePacket.data[2][spectrum].phase;
//
        float cur_avg_gain  = 0.0;//(prev_avg_gain,  n_prev, new_gain);
//            float cur_avg_phase = running_avg(prev_avg_phase, n_prev, new_phase);
//           cur_avg_gain = new_gain;



        cur_avg_gain = find_reflection_coefficient(new_gain,led_area,spot_diameter,responsivity,
    ↪ source_radiant_flux);


//
//            if(spectrum == 1)
//          {
//
//              cur_avg_gain  = cur_avg_gain - 0.15;
//              cur_avg_phase = avg_after.phase;
//          }

//          else{
//              float new_gain  = avg_after.gain;
//              float new_phase = avg_after.phase;
//          }

        framePacket.data[2][spectrum].gain  = cur_avg_gain;
        //framePacket.data[2][spectrum].phase = cur_avg_phase;    // <-- bug fix: update phase with phase

        samples_per_spectrum[spectrum] = n_prev + 1;


//        total_gain_before += gain_phase_before[0];
//        total_phase_before += gain_phase_before[1];
//        HAL_SPI_Transmit(&hspi1, &shiftPattern[spectrum], 1, HAL_MAX_DELAY);
//      HAL_Delay(50);
//      demodulation_control_sequence(spectrum,total_gain_before);// Process all sensors at this spectrum
//      HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);
//      HAL_Delay();
```

```
//        while ((HAL_GetTick() - start_time) <= hold_duration)
//        {
//          HAL_SPI_Transmit(&hspi1, &shiftPattern[spectrum], 1, HAL_MAX_DELAY);
//          HAL_Delay(100);
//            demodulation_control_sequence(spectrum,total_gain_before);// Process all sensors at this
    ↪ spectrum
//            HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);
//            HAL_Delay(100);
//
//        }

        printf("Completed spectra point: %d\n", spectrum);
    }

    return 0;


}


int  LEDs_control_sequence()
{
//  HAL_GPIO_WritePin(GPIOD, Start_LDD_MCU_Pin, GPIO_PIN_RESET);

uint8_t shiftPattern[] = {0xFE, 0xFD, 0xFB, 0xF7,0xEF, 0xDF, 0xBF,0x7F,0xFF};

    uint32_t samples_per_spectrum[9] = {0};  // how many updates each spectrum has received


    uint32_t hold_duration = 300;
    uint8_t setData = 0xFF;
    uint8_t resetData = 0x00;



    for (int s = 0; s< 4; s++)
    {
        uint32_t start_time = HAL_GetTick();
//        HAL_SPI_Transmit(&hspi1, &resetData, 1, HAL_MAX_DELAY);
        HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);


        HAL_Delay(50);

        HAL_SPI_Transmit(&hspi1, &shiftPattern[1], 1, HAL_MAX_DELAY);
        HAL_Delay(5);
        HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);
        HAL_Delay(70);

    }

    return 0;

}


int  modulation_control_sequence()
{
//  HAL_GPIO_WritePin(GPIOD, Start_LDD_MCU_Pin, GPIO_PIN_RESET);

uint8_t shiftPattern[] = {0xFE, 0xFD, 0xFB, 0xF7,0xEF, 0xDF, 0xBF,0x7F,0xFF};

    uint32_t samples_per_spectrum[9] = {0};  // how many updates each spectrum has received


    uint32_t hold_duration = 300;
    uint8_t setData = 0xFF;
    uint8_t resetData = 0x00;
```

```
//    memset(framePacket.data, 0, sizeof framePacket.data);
    float new_gain  = 0.0;
    float new_phase = 0.0;
    float led_area = 0.0;
    float responsivity = 0.0;
    float spot_diameter = 0.08;
    float source_radiant_flux = 0.0;

    float normFactors[NUM_SPECTRA];

     uint32_t next_idx= 0;
     uint8_t caliberate_id;
     fr = find_last_index_in_dir("field001", "Calibration_", &next_idx);
     if (fr != FR_OK)
     {
         return -1;

     }

     else if(fr ==FR_OK)
     {
         if ((idx >= 1) && (idx <= 41))
         {
             caliberate_id = ((idx - 1) / 4) + 1;
         }
         else
         {
             caliberate_id = 3;
         }

//       caliberate_id = (uint8_t)next_idx;
//       load_norm_factors("field001", 3, normFactors);
     }







    for (int spectrum = 0; spectrum < 9 ; spectrum++)
    {
        uint32_t start_time = HAL_GetTick();
        HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);
//            HAL_Delay(50);

        /**
         * below code is responsible for retrieving signal from detector before
         * LEDS are turned on.
         */
//      uint16_t* gain_phase_before;
//      gain_phase_before = retrieve_gain_phase_and_Store();  // returns uint16_t[2]
//      float total_gain_before = 0.0f, total_phase_before = 0.0f;
//      avg_sensor_read_t avg_after;
//
//      HAL_Delay(70);

        uint16_t* gain_phase_before;
        gain_phase_before = retrieve_gain_phase_and_Store();  // returns uint16_t[2]
        float total_gain_before = 0.0f, total_phase_before = 0.0f;
        avg_sensor_read_t avg_after;

        HAL_Delay(50);


        total_gain_before += gain_phase_before[0];
        total_phase_before += gain_phase_before[1];
        HAL_SPI_Transmit(&hspi1, &shiftPattern[spectrum], 1, HAL_MAX_DELAY);
```

```
        HAL_Delay(5);
        avg_after = demodulation_control_sequence(spectrum,total_gain_before);// Process all sensors at this
    ↪ spectrum
        HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);
        HAL_Delay(50);

//      avg_after.gain = avg_after.gain - gain_phase_before[0]/65536.0f * 3.3f;
        avg_after.gain = avg_after.gain;

//            float final_avg_gain = avg_after.gain;
//            float final_avg_phase = avg_after.phase;
//
//            framePacket.data[2][spectrum].gain = framePacket.data[2][spectrum].gain +final_avg_gain;
//            framePacket.data[2][spectrum].phase = framePacket.data[2][spectrum].gain + final_avg_phase;
//
        /* Running average per spectrum */
        uint32_t n_prev = samples_per_spectrum[spectrum];       // 0,1,2,... up to 10


//      new_gain  = avg_after.gain * 3; // 3 is a firmware amplification gain
        new_gain  = avg_after.gain;// 3 is a firmware amplification gain
        new_phase = avg_after.phase;

        load_norm_factors("field001", txBuf[230], normFactors);

        if(spectrum == 0)
        {



            // later, per spectrum:
//          new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);
            new_phase = avg_after.phase;   // usually leave phase untouched


//          new_gain  = avg_after.gain; // 0.25 is a  correction factor

            new_phase = avg_after.phase;
            led_area = 0.002;
            responsivity = 2.8;
            source_radiant_flux = 640E-03;
        }

        else if(spectrum == 1)
        {

//          new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);

//          new_gain  = avg_after.gain; // 0.10 is a correction factor
            new_phase = avg_after.phase;
            led_area = 0.00196;
            responsivity = 2.8;
            source_radiant_flux = 500E-03;

        }

        else if(spectrum == 2)
        {

//          new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);

            led_area = 0.00188;
            responsivity = 3.0;
            source_radiant_flux = 590E-03;
        }

        else if(spectrum == 3)
        {
```

```
            new_gain = new_gain * 2;
//          new_gain  = avg_after.gain;
//          new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);

            new_phase = avg_after.phase;


            led_area = 0.00196;

            responsivity = 2.9;
            source_radiant_flux = 480E-03;
        }

        else if(spectrum == 4)
        {
//          new_gain  = avg_after.gain;
//          new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);

            new_phase = avg_after.phase;

            led_area = 0.00196;
            responsivity = 2.8;
            source_radiant_flux = 420E-03;
        }


        else if(spectrum == 5)
        {
//          new_gain  = avg_after.gain;
//          new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);

            new_phase = avg_after.phase;

            led_area = 0.001;
            responsivity = 2.7;
            source_radiant_flux = 753E-03;
        }


        else if(spectrum == 6)
        {

//          new_gain  = avg_after.gain;
//          new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);


            new_phase = avg_after.phase;
            led_area = 0.0010;
            responsivity = 2.8;
            source_radiant_flux = 800E-03;
        }



//            float prev_avg_gain  = framePacket.data[2][spectrum].gain;
//            float prev_avg_phase = framePacket.data[2][spectrum].phase;
//
        float cur_avg_gain  = 0.0;//(prev_avg_gain,  n_prev, new_gain);
//            float cur_avg_phase = running_avg(prev_avg_phase, n_prev, new_phase);
//           cur_avg_gain = new_gain;



//      cur_avg_gain = find_reflection_coefficient(new_gain,led_area,spot_diameter,responsivity,
    ↪ source_radiant_flux);
```

```
//
//            if(spectrum == 1)
//          {
//
//              cur_avg_gain  = cur_avg_gain - 0.15;
//              cur_avg_phase = avg_after.phase;
//          }

//          else{
//              float new_gain  = avg_after.gain;
//              float new_phase = avg_after.phase;
//          }

//      framePacket.data[2][spectrum].gain  = cur_avg_gain;
        framePacket.data[2][spectrum].gain  = new_gain;
        //framePacket.data[2][spectrum].phase = cur_avg_phase;    // <-- bug fix: update phase with phase

        samples_per_spectrum[spectrum] = n_prev + 1;


//        total_gain_before += gain_phase_before[0];
//        total_phase_before += gain_phase_before[1];
//        HAL_SPI_Transmit(&hspi1, &shiftPattern[spectrum], 1, HAL_MAX_DELAY);
//      HAL_Delay(50);
//      demodulation_control_sequence(spectrum,total_gain_before);// Process all sensors at this spectrum
//      HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);
//      HAL_Delay();



//        while ((HAL_GetTick() - start_time) <= hold_duration)
//        {
//          HAL_SPI_Transmit(&hspi1, &shiftPattern[spectrum], 1, HAL_MAX_DELAY);
//          HAL_Delay(100);
//            demodulation_control_sequence(spectrum,total_gain_before);// Process all sensors at this
    ↪ spectrum
//            HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);
//            HAL_Delay(100);
//
//        }

        printf("Completed spectra point: %d\n", spectrum);
    }

    return 0;
}


int  Test_modulation_control_sequence()
{
//  HAL_GPIO_WritePin(GPIOD, Start_LDD_MCU_Pin, GPIO_PIN_RESET);

uint8_t shiftPattern[] = {0xFE, 0xFD, 0xFB, 0xF7,0xEF, 0xDF, 0xBF,0x7F,0xFF};

    uint32_t samples_per_spectrum[9] = {0};  // how many updates each spectrum has received


    uint32_t hold_duration = 300;
    uint8_t setData = 0xFF;
    uint8_t resetData = 0x00;
//    memset(framePacket.data, 0, sizeof framePacket.data);
    float new_gain  = 0.0;
    float new_phase = 0.0;
    float led_area = 0.0;
    float responsivity = 0.0;
    float spot_diameter = 0.08;
    float source_radiant_flux = 0.0;
```

```
    float normFactors[NUM_SPECTRA];


     uint32_t next_idx= 0;
     uint8_t caliberate_id;
     fr = find_last_index_in_dir("field001", "Calibration_", &next_idx);
     if (fr != FR_OK)
     {
         return -1;

    }

     else if(fr ==FR_OK)
     {
         if ((idx >= 1) && (idx <= 41))
         {
             caliberate_id = ((idx - 1) / 4) + 1;
         }
         else
         {
             caliberate_id = 3;
         }

//       caliberate_id = (uint8_t)next_idx;
         load_norm_factors("field001", 3, normFactors);
     }







    for (int spectrum = 0; spectrum < 9 ; spectrum++)
    {
        uint32_t start_time = HAL_GetTick();
//        HAL_SPI_Transmit(&hspi1, &resetData, 1, HAL_MAX_DELAY);
//      HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);
//            HAL_Delay(50);

        /**
         * below code is responsible for retrijeving signal from detector before
         * LEDS are turned on.u
         */
//      uint16_t* gain_phase_before;
//      gain_phase_before = retrieve_gain_phase_and_Store();  // returns uint16_t[2]
//      float total_gain_before = 0.0f, total_phase_before = 0.0f;
//      avg_sensor_read_t avg_after;
//
//      HAL_Delay(70);

        uint16_t* gain_phase_before;
        gain_phase_before = retrieve_gain_phase_and_Store();  // returns uint16_t[2]
        float total_gain_before = 0.0f, total_phase_before = 0.0f;
        avg_sensor_read_t avg_after;

        HAL_Delay(50);


        total_gain_before += gain_phase_before[0];
        total_phase_before += gain_phase_before[1];
        HAL_SPI_Transmit(&hspi1, &shiftPattern[spectrum], 1, HAL_MAX_DELAY);
        HAL_Delay(5);
        avg_after = demodulation_control_sequence(spectrum,total_gain_before);// Process all sensors at this
    ↪ spectrum
//      HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);
        HAL_Delay(70);
```

```
        avg_after.gain = avg_after.gain - gain_phase_before[0]/65536.0f * 3.3f;

//            float final_avg_gain = avg_after.gain;
//            float final_avg_phase = avg_after.phase;
//
//            framePacket.data[2][spectrum].gain = framePacket.data[2][spectrum].gain +final_avg_gain;
//            framePacket.data[2][spectrum].phase = framePacket.data[2][spectrum].gain + final_avg_phase;
//
        /* Running average per spectrum */
        uint32_t n_prev = samples_per_spectrum[spectrum];       // 0,1,2,... up to 10


        new_gain  = avg_after.gain;
        new_phase = avg_after.phase;
        uint8_t caliberate_id;







        if(spectrum == 0)
        {



            // later, per spectrum:
            new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);
            new_phase = avg_after.phase;   // usually leave phase untouched


//          new_gain  = avg_after.gain; // 0.25 is a  correction factor

            new_phase = avg_after.phase;
            led_area = 0.002;
            responsivity = 2.8;
            source_radiant_flux = 640E-03;
        }

        else if(spectrum == 1)
        {

            new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);

//          new_gain  = avg_after.gain; // 0.10 is a correction factor
            new_phase = avg_after.phase;
            led_area = 0.00196;
            responsivity = 2.8;
            source_radiant_flux = 500E-03;

        }

        else if(spectrum == 2)
        {

            new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);
//          new_gain  = avg_after.gain;
            led_area = 0.00188;
            responsivity = 3.0;
            source_radiant_flux = 590E-03;
        }

        else if(spectrum == 3)
        {

//          new_gain  = avg_after.gain;
            new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);
```

```
            new_phase = avg_after.phase;


            led_area = 0.00196;

            responsivity = 2.9;
            source_radiant_flux = 480E-03;
        }

        else if(spectrum == 4)
        {
//          new_gain  = avg_after.gain;
            new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);

            new_phase = avg_after.phase;

            led_area = 0.00196;
            responsivity = 2.8;
            source_radiant_flux = 420E-03;
        }


        else if(spectrum == 5)
        {
//          new_gain  = avg_after.gain;
            new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);

            new_phase = avg_after.phase;

            led_area = 0.001;
            responsivity = 2.7;
            source_radiant_flux = 753E-03;
        }


        else if(spectrum == 6)
        {

//          new_gain  = avg_after.gain;
            new_gain  = apply_gain_normalization(spectrum, avg_after.gain, normFactors);


            new_phase = avg_after.phase;
            led_area = 0.0010;
            responsivity = 2.8;
            source_radiant_flux = 800E-03;
        }



//            float prev_avg_gain  = framePacket.data[2][spectrum].gain;
//            float prev_avg_phase = framePacket.data[2][spectrum].phase;
//
        float cur_avg_gain  = 0.0;//(prev_avg_gain,  n_prev, new_gain);
//            float cur_avg_phase = running_avg(prev_avg_phase, n_prev, new_phase);
//           cur_avg_gain = new_gain;



        cur_avg_gain = find_reflection_coefficient(new_gain,led_area,spot_diameter,responsivity,
    ↪ source_radiant_flux);


//
//            if(spectrum == 1)
//          {
//
//              cur_avg_gain  = cur_avg_gain - 0.15;
```

```
//              cur_avg_phase = avg_after.phase;
//          }

//          else{
//              float new_gain  = avg_after.gain;
//              float new_phase = avg_after.phase;
//          }

        framePacket.data[2][spectrum].gain  = cur_avg_gain;
        //framePacket.data[2][spectrum].phase = cur_avg_phase;    // <-- bug fix: update phase with phase

        samples_per_spectrum[spectrum] = n_prev + 1;


//        total_gain_before += gain_phase_before[0];
//        total_phase_before += gain_phase_before[1];
//        HAL_SPI_Transmit(&hspi1, &shiftPattern[spectrum], 1, HAL_MAX_DELAY);
//      HAL_Delay(50);
//      demodulation_control_sequence(spectrum,total_gain_before);// Process all sensors at this spectrum
//      HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);
//      HAL_Delay();



//        while ((HAL_GetTick() - start_time) <= hold_duration)
//        {
//          HAL_SPI_Transmit(&hspi1, &shiftPattern[spectrum], 1, HAL_MAX_DELAY);
//          HAL_Delay(100);
//            demodulation_control_sequence(spectrum,total_gain_before);// Process all sensors at this
    ↪ spectrum
//            HAL_SPI_Transmit(&hspi1, &setData, 1, HAL_MAX_DELAY);
//            HAL_Delay(100);
//
//        }

        printf("Completed spectra point: %d\n", spectrum);
    }


}


float find_reflection_coefficient(float avg_gain,float LED_area,float spot_diameter,float responsivity,float
    ↪ source_radiant_flux)
{
    float first_convertion_voltage = avg_gain/op_amp_gain;
    float photo_current = first_convertion_voltage/transimpedance_gain;
    float reflected_power = photo_current/responsivity;
    float spot_area = PI * powf(spot_diameter * 0.5f, 2.0f);  // link with -lm
    float source_exitance = source_radiant_flux/LED_area;
    float source_brightness = source_exitance/PI;
    float solid_angle_leave = (spot_area/(radius*radius));
    float power_leaves = source_brightness * LED_area * solid_angle_leave;
    float co_reflection = (reflected_power/power_leaves)*100;

    return avg_gain;



}


static inline float apply_gain_normalization(uint8_t spectrum, float raw_gain, const float norm[NUM_SPECTRA])
{
    if (spectrum >= NUM_SPECTRA) return raw_gain;
    return raw_gain * norm[spectrum];
}
```

```
avg_sensor_read_t demodulation_control_sequence(uint8_t spectrumIndex,float initial_reading)
{
    if (spectrumIndex >= NUM_SPECTRA) return;

    avg_sensor_read_t avg;

//  float avg_phase;

    const uint32_t hold_duration = 30; // ms



        uint32_t start_time = HAL_GetTick();
        float total_gain = 0.0f, total_phase = 0.0f;
        uint16_t* gain_phase;
        uint32_t tick_n = 0;

        while ((HAL_GetTick() - start_time) <= hold_duration)
        {
            gain_phase = retrieve_gain_phase_and_Store();  // returns uint16_t[2]

            total_gain += gain_phase[0];
            total_phase += gain_phase[1];
            tick_n++;
            HAL_Delay(10);
        }

        if (tick_n > 0)
        {
//          float avg_gain = (total_gain / tick_n);
//          float avg_phase = (total_phase / tick_n);
            //          f

            avg.gain = ((total_gain / tick_n)) /65536.0f * 3.3f; //average gain
            avg.phase = 0.0;


    }

    return avg;


}


void set_rst_and_enablePin_SPIs()
{
    HAL_GPIO_WritePin(GPIOD, LD_Ctrl_Enable_Pin, GPIO_PIN_SET);
//    HAL_GPIO_WritePin(GPIOD, DEM_Ctrl_EN_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOD, LD_Ctrl_CLR_Pin, GPIO_PIN_SET);
//    HAL_GPIO_WritePin(GPIOD, DEM_Ctrl_CLR_Pin, GPIO_PIN_SET);

}


uint16_t *retrieve_gain_phase_and_Store()
{
    HAL_ADC_Start(&hadc1);
    static uint16_t adc_vals[2];

    const uint32_t poll_duration = 50;
    uint32_t start_time = HAL_GetTick();


    HAL_ADC_PollForConversion(&hadc1, 100);         // Poll The ADC Channel With TimeOut = 1mSec
//      uint16_t x = rand();
//      adc_vals[i] = x;
    adc_vals[0] = HAL_ADC_GetValue(&hadc1);
```

```
    HAL_Delay(1);
    HAL_ADC_Stop(&hadc1);

//  HAL_ADC_Stop(&hadc1);







    return adc_vals;


}



//void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
//{
////    SPI_PreloadResonse();
//   SPI_PreloadResponse();
//
//    if (GPIO_Pin == SPI_NSS_Pin) {
//         GPIO_PinState cs = HAL_GPIO_ReadPin(GPIOB, SPI_NSS_Pin);
//         if (cs == GPIO_PIN_RESET) {
//             // NSS asserted low
//             SPI_Slave_ArmDMA();
//         } else {
//             // NSS deasserted high
//             SPI_Slave_AbortIfNeeded();
//         }
//     }
//}


static inline void SPI_PreloadResponse(void)
{
////     Example: first byte is a status, rest zeros
    memcpy(txBuf, (uint8_t[]){
        // 1 of 5
        0x00, 0x0C, 0x8C, 0x7D, 0x72, 0x47, 0x34, 0x2C, 0xD8, 0x10,
        0x0F, 0x2F, 0x6F, 0x77, 0x0D, 0x65, 0xD6, 0x70, 0xE5, 0x8E,
        0x03, 0x51, 0xD8, 0xAE, 0x8E, 0x4F, 0x6E, 0xAC, 0x34, 0x2F,
        0xC2, 0x31, 0xB7, 0xB0, 0x87, 0x16, 0xEB, 0x3F, 0xC1, 0x28,
        0x96, 0xB9, 0x62, 0x23, 0x17, 0x74, 0x94, 0x28, 0x77, 0x33,
        0xC2, 0x8E, 0xE8, 0xBA, 0x53, 0xBD, 0xB5, 0x6B, 0x88, 0x24,
        0x57, 0x7D, 0x53, 0xEC, 0xC2, 0x8A, 0x70, 0xA6, 0x1C, 0x75,
        0x10, 0xA1, 0xCD, 0x89, 0x21, 0x6C, 0xA1, 0x6C, 0xFF, 0xCA,
        0xEA, 0x49, 0x87, 0x47, 0x7E, 0x86, 0xDB, 0xCC, 0xB9, 0x70,
        0x46, 0xFC, 0x2E, 0x18, 0x38, 0x4E, 0x51, 0xD8, 0x20, 0xC5,

        // 2 of 5
        0x00, 0x0C, 0x8C, 0x7D, 0x72, 0x47, 0x34, 0x2C, 0xD8, 0x10,
        0x0F, 0x2F, 0x6F, 0x77, 0x0D, 0x65, 0xD6, 0x70, 0xE5, 0x8E,
        0x03, 0x51, 0xD8, 0xAE, 0x8E, 0x4F, 0x6E, 0xAC, 0x34, 0x2F,
        0xC2, 0x31, 0xB7, 0xB0, 0x87, 0x16, 0xEB, 0x3F, 0xC1, 0x28,
        0x96, 0xB9, 0x62, 0x23, 0x17, 0x74, 0x94, 0x28, 0x77, 0x33,
        0xC2, 0x8E, 0xE8, 0xBA, 0x53, 0xBD, 0xB5, 0x6B, 0x88, 0x24,
        0x57, 0x7D, 0x53, 0xEC, 0xC2, 0x8A, 0x70, 0xA6, 0x1C, 0x75,
        0x10, 0xA1, 0xCD, 0x89, 0x21, 0x6C, 0xA1, 0x6C, 0xFF, 0xCA,
        0xEA, 0x49, 0x87, 0x47, 0x7E, 0x86, 0xDB, 0xCC, 0xB9, 0x70,
        0x46, 0xFC, 0x2E, 0x18, 0x38, 0x4E, 0x51, 0xD8, 0x20, 0xC5,

        // 3 of 5
        0x00, 0x0C, 0x8C, 0x7D, 0x72, 0x47, 0x34, 0x2C, 0xD8, 0x10,
        0x0F, 0x2F, 0x6F, 0x77, 0x0D, 0x65, 0xD6, 0x70, 0xE5, 0x8E,
```

```
        0x03, 0x51, 0xD8, 0xAE, 0x8E, 0x4F, 0x6E, 0xAC, 0x34, 0x2F,
        0xC2, 0x31, 0xB7, 0xB0, 0x87, 0x16, 0xEB, 0x3F, 0xC1, 0x28,
        0x96, 0xB9, 0x62, 0x23, 0x17, 0x74, 0x94, 0x28, 0x77, 0x33,
        0xC2, 0x8E, 0xE8, 0xBA, 0x53, 0xBD, 0xB5, 0x6B, 0x88, 0x24,
        0x57, 0x7D, 0x53, 0xEC, 0xC2, 0x8A, 0x70, 0xA6, 0x1C, 0x75,
        0x10, 0xA1, 0xCD, 0x89, 0x21, 0x6C, 0xA1, 0x6C, 0xFF, 0xCA,
        0xEA, 0x49, 0x87, 0x47, 0x7E, 0x86, 0xDB, 0xCC, 0xB9, 0x70,
        0x46, 0xFC, 0x2E, 0x18, 0x38, 0x4E, 0x51, 0xD8, 0x20, 0xC5,

        // 4 of 5
        0x00, 0x0C, 0x8C, 0x7D, 0x72, 0x47, 0x34, 0x2C, 0xD8, 0x10,
        0x0F, 0x2F, 0x6F, 0x77, 0x0D, 0x65, 0xD6, 0x70, 0xE5, 0x8E,
        0x03, 0x51, 0xD8, 0xAE, 0x8E, 0x4F, 0x6E, 0xAC, 0x34, 0x2F,
        0xC2, 0x31, 0xB7, 0xB0, 0x87, 0x16, 0xEB, 0x3F, 0xC1, 0x28,
        0x96, 0xB9, 0x62, 0x23, 0x17, 0x74, 0x94, 0x28, 0x77, 0x33,
        0xC2, 0x8E, 0xE8, 0xBA, 0x53, 0xBD, 0xB5, 0x6B, 0x88, 0x24,
        0x57, 0x7D, 0x53, 0xEC, 0xC2, 0x8A, 0x70, 0xA6, 0x1C, 0x75,
        0x10, 0xA1, 0xCD, 0x89, 0x21, 0x6C, 0xA1, 0x6C, 0xFF, 0xCA,
        0xEA, 0x49, 0x87, 0x47, 0x7E, 0x86, 0xDB, 0xCC, 0xB9, 0x70,
        0x46, 0xFC, 0x2E, 0x18, 0x38, 0x4E, 0x51, 0xD8, 0x20, 0xC5,

        // 5 of 5
        0x00, 0x0C, 0x8C, 0x7D, 0x72, 0x47, 0x34, 0x2C, 0xD8, 0x10,
        0x0F, 0x2F, 0x6F, 0x77, 0x0D, 0x65, 0xD6, 0x70, 0xE5, 0x8E,
        0x03, 0x51, 0xD8, 0xAE, 0x8E, 0x4F, 0x6E, 0xAC, 0x34, 0x2F,
        0xC2, 0x31, 0xB7, 0xB0, 0x87, 0x16, 0xEB, 0x3F, 0xC1, 0x28,
        0x96, 0xB9, 0x62, 0x23, 0x17, 0x74, 0x94, 0x28, 0x77, 0x33,
        0xC2, 0x8E, 0xE8, 0xBA, 0x53, 0xBD, 0xB5, 0x6B, 0x88, 0x24,
        0x57, 0x7D, 0x53, 0xEC, 0xC2, 0x8A, 0x70, 0xA6, 0x1C, 0x75,
        0x10, 0xA1, 0xCD, 0x89, 0x21, 0x6C, 0xA1, 0x6C, 0xFF, 0xCA,
        0xEA, 0x49, 0x87, 0x47, 0x7E, 0x86, 0xDB, 0xCC, 0xB9, 0x70,
        0x46, 0xFC, 0x2E, 0x18, 0x38, 0x4E, 0x51, 0xD8, 0x20, 0xC5
    }, 500);
//  txBuf[0] = 0x03;
//  txBuf[1] = 0x48;
//  txBuf[2] = 0x78;
//  txBuf[3] = 0x99;
//    for (uint16_t i = 1; i < XFER_LEN; i++) spi_tx_buf[i] = 0x00;
}


const char* regionName(uint8_t region_id)
{
    switch(region_id) {
        case REGION_LOWER:  return "lower";
        case REGION_MIDDLE: return "middle";
        case REGION_UPPER:  return "upper";
        default:            return "unknown";
    }
}

int scan_write_sequence(void)
{



      framePacket.header.syncWord = 0xAA55;
      framePacket.header.payloadSize = sizeof(SIPM_t) * NUM_SENSORS * NUM_SPECTRA;
      int scan_success = 1;

        if (rxDummy[0] == 0x01)
        {

          for(int i = 0; i<4; i++)
          {
            if (rxDummy[1] == 0x01)
            {
                framePacket.header.frameID++; //increament frame Id by 1;
```

```
            modulation_control_sequence();
            size_t len = sizeof(FrameHeader_t) + framePacket.header.payloadSize;
            memcpy(txBuf, &framePacket, len);


            /*
             * 3rd element of the recieved buffer is used as command to move to a new plant
             * within storage
             */

            if (rxDummy[2] == 0x01)
            {


                currentPlantID = rxDummy[4];
                create_plant_directory("field001", currentPlantID);
                currentLeafID = 0;
                HAL_Delay(60);


            }


            /*
             * third byte is used to make region change.
             */

//              if (rxDummy[3] == 0x01)
//              {
//
//
//
//                if (currentRegion < REGION_UPPER)
//                 {
//                      currentRegion++;
//                      currentLeafID = 0;  // Reset leaf index for new region
//                  }
//                 else
//                  {
//
//                     // Already at last region -- optionally set a flag to block writing
//                   currentRegion = REGION_UPPER;
//
//                  }
//
//                 currentLeafID++;
//                  // ensure buffer persists for DMA
//
//              write_frame_to_csv("field001", currentPlantID, regionName(currentRegion), currentLeafID, &
  ↪ framePacket);
//                 /* 0xBB = Scan complete
//                  * 0xB1 = new plant
//                  * 0xB2 = new region
//                  * 0xB3 = save*/
//              txBuf[231] = set_opperation_complete_id(0xB2);
//
//              }


            else if(rxDummy[3] == 0x00)
            {

//                  currentRegion = rxDummy[9];
                currentLeafID++;
                st = find_last_index_in_dir("field001", "Plant_", &idx);
                currentPlantID = idx;
//                  currentPlantID = rxDummy[4];
                fr = write_frame_to_csv("field001", currentPlantID, regionName(currentRegion),
```

```
  ↪ currentLeafID, &framePacket);
                HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_RESET);


            }




        }

//            else if (rxDummy[1] == 0x02)
//            {
//
//                HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);
//                resert_data_storage();
//
//            }



        /*
         * statement fof creating a new plant object
         *
         *
         */
        else if(rxDummy[1] == 0x02)
        {
            currentRegion = REGION_LOWER;
            currentPlantID = rxDummy[4];
            st = find_last_index_in_dir("field001", "Plant_", &idx);
//              currentPlantID = idx;
            FRESULT fr = 21;
            uint8_t tries = 0;
//              int next_index = currentPlantID + 1;


            while(fr  != 0 || tries != 1)
            {
                fr = create_plant_directory("field001", currentPlantID);
//                  HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);
                tries++;
                LEDs_control_sequence();
            }


//              HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET); // This command ensures
  ↪ the spi dma transmmit's scanning results
             /* 0xBB = Scan complete
              * 0xB1 = new plant
              * 0xB2 = new region
              * 0xB3 = save*/



            HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_RESET);


        }


        /*
         * statement fof creating a new plant replicate
         *
         *
         */

        else if (rxDummy[1] == 0x03)
```

```
            {



              if (currentRegion < REGION_UPPER)
               {
                    currentRegion++;
                    currentLeafID = 0;  // Reset leaf index for new region
                }
               else
                {

                   // Already at last region -- optionally set a flag to block writing
                currentRegion = REGION_UPPER;

                }

             currentLeafID++;
              // ensure buffer persists for DMA

            write_frame_to_csv("field001", currentPlantID, regionName(currentRegion), currentLeafID, &
    ↪ framePacket);
             /* 0xBB = Scan complete
             * 0xB1 = new plant
             * 0xB2 = new region
             * 0xB3 = save*/
  //            txBuf[231] = set_opperation_complete_id(0xB2);

  //            HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET); // This command ensures the
    ↪ spi dma transmmit's scanning results
            HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_RESET);


            }


            /*
             * for save everything
             */

            else if(rxDummy[1] == 0x00)
            {
                /* 0xBB = Scan complete
                 * 0xB1 = new plant
                 * 0xB2 = new region
                 * 0xB3 = save*/
  //               txBuf[231] = set_opperation_complete_id(0xB3);

  //               HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET); // This command ensures
    ↪ the spi dma transmmit's scanning results

            }




            scan_success = 0;
          }

        }

        else if(rxDummy[0] == 0x05)
        {

//          HAL_StatusTypeDef status = HAL_SPI_TransmitReceive_DMA(&hspi2, txBuf, rxDummy, 500);
    //      HAL_StatusTypeDef status = HAL_SPI_Transmit(&hspi2, txBuf, 500,HAL_MAX_DELAY);

    //      rxDummy[0] == 0xAA;
```

```
//          HAL_Delay(50);



//          HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET); // This command ensures the spi
    ↪ dma transmmit's scanning results
//          __HAL_SPI_DISABLE(&hspi2);
//          __HAL_SPI_ENABLE(&hspi2);
//          HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);
//          txBuf[231] = 0xBB;
//
//          st = HAL_SPI_TransmitReceive_DMA(&hspi2, txBuf, rxDummy, XFER_LEN);
//          reset_recieved_bytes();

//          spi_frame_active = 0;


    //      uint8_t recived_x = 0;
            scan_success = 0;

        }

        else if(rxDummy[0] == 0x6)
        {
            printf("I am here");
//          int sucess = caliberate(rxDummy);
            int sucess = caliberate_on_the_go(rxDummy);

            scan_success = 0;

        }




        return scan_success;


}



void Test_scan_write_sequence(void)
{



      framePacket.header.syncWord = 0xAA55;
      framePacket.header.payloadSize = sizeof(SIPM_t) * NUM_SENSORS * NUM_SPECTRA;
//    rxDummy[0] = 0x01;
//    rxDummy[1] = 0x01;

//  if (rxDummy[0] == 0xAA)
//  {
//      printf("I am here");
//  }
        int test = 1;
        if (test)

        {
//          int command_received_flag = 1;
    //      uint32_t next_idx;
    //      FRESULT fr = find_next_index_in_dir("field001", "Plant_", &next_idx);
    //      currentPlantID = next_idx-1;
    //      modulation_control_sequence();
```

```
            if (test)
//          if(command_received_flag == 1)
            {
//              data_parket.header.syncWord = 0xAA55;
                framePacket.header.frameID++; //increament frame Id by 1;
                Test_modulation_control_sequence();
                size_t len = sizeof(FrameHeader_t) + framePacket.header.payloadSize;
                memcpy(txBuf, &framePacket, len);


                /*
                 * 3rd element of the recieved buffer is used as command to move to a new plant or not
                 */
    //
                if (rxDummy[2] == 0x01)
                {
    //              uint32_t next_idx= 0;
    //              FRESULT fr = find_next_index_in_dir("field001", "Plant_", &next_idx);
    //              currentPlantID = next_idx;

                    currentPlantID = rxDummy[4];
                    create_plant_directory("field001", currentPlantID);
                    currentRegion = REGION_LOWER;  // enum or string, depending on your design
                    currentLeafID = 0;
                }

                /*
                 * third byte is used to make region change.
                 */

                if (rxDummy[3] == 0x01)
                {



                  if (currentRegion < REGION_UPPER)
                   {
                        currentRegion++;
                        currentLeafID = 0;  // Reset leaf index for new region
                   }
                   else
                    {
                                    // Already at last region -- optionally set a flag to block writing
                    currentRegion = REGION_UPPER;

                    }

                 currentLeafID++;
                  // ensure buffer persists for DMA

                write_frame_to_csv("field001", currentPlantID, regionName(currentRegion), currentLeafID, &
    ↪ framePacket);

                }


                else if(rxDummy[3] == 0x00)
                {


                    currentLeafID++;
                    write_frame_to_csv("field001", currentPlantID, regionName(currentRegion), currentLeafID, &
    ↪ framePacket);
    //                  memcpy(rxDummy, &framePacket, n);
                }

    //          HAL_DMA_Abort(&hspi2);
```

```
    //
                HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET); // This command ensures the
    ↪ spi dma transmmit's scanning results
    //          Hal_Delay(10);
//              resert_data_storage();
    ////            HAL_DMA_Abort(&hspi2);
                reset_recieved_bytes();



            }

            else if (rxDummy[1] == 0x02)
            {
    //          currentPlantID = currentPlantID + 1;
    //          currentRegion = REGION_LOWER;  // enum or string, depending on your design
    //          currentLeafID = 0;
                HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);
                resert_data_storage();
                reset_recieved_bytes();

            }


            /*
             * statement fof creating a new plant object
             */
            else if(rxDummy[1] == 0x03)
            {
                currentRegion = REGION_LOWER;
    //          uint32_t next_idx= 0;
    //          FRESULT fr = find_next_index_in_dir("field001", "Plant_", &next_idx);
    //          currentPlantID = next_idx;
                currentPlantID = rxDummy[4];
                create_plant_directory("field001", currentPlantID);
                HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);
//              spi_frame_active = 0;

                reset_recieved_bytes();

            }


            // Optionally start new SPI transaction if needed
    //      HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);
    //      HAL_Delay(10);
//          resert_data_storage();
//          reset_recieved_bytes();


        }

        if(rxDummy[0] == 0x05)
        {

//          HAL_StatusTypeDef status = HAL_SPI_TransmitReceive_DMA(&hspi2, txBuf, rxDummy, 500);
    //      HAL_StatusTypeDef status = HAL_SPI_Transmit(&hspi2, txBuf, 500,HAL_MAX_DELAY);

    //      rxDummy[0] == 0xAA;
//          HAL_Delay(50);



//          HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET); // This command ensures the spi
    ↪ dma transmmit's scanning results
//          reset_recieved_bytes();
//          spi_frame_active = 0;
```

```
    //      uint8_t recived_x = 0;

        }

        if(rxDummy[0] == 0x6)
        {
            printf("I am here");
            int sucess = caliberate(rxDummy);

        }

    //  reset_recieved_bytes();







}

//
//void reset_recieved_bytes()
//{
//  memset(rxDummy,0xAA,sizeof(rxDummy));
//
//}



void reset_recieved_bytes(void)
{

    for (int attempt = 0; attempt < MAX_MEMSET_RETRIES; attempt++)
    {
        memset(rxDummy, 0xAA, sizeof(rxDummy));

        bool ok = true;
        for (size_t i = 0; i < sizeof(rxDummy); i++)
        {
            if (rxDummy[i] != 0xAA)
            {
                ok = false;
                break;
            }
        }

        if (ok)
        {
//          dcache_clean_by_addr_safe(rxDummy, sizeof(rxDummy));

            return;  // success
        }
    }

    currentRegion = REGION_LOWER;

    // Optional: error handling if memory could not be set
    // e.g. set error flag, assert, log, or reset system
}



static void dcache_clean_by_addr_safe(void *addr, size_t size)
{
    // Only if D-Cache is enabled
    if ((SCB->CCR & SCB_CCR_DC_Msk) == 0U) return;
```

```
    uintptr_t start = (uintptr_t)addr;
    uintptr_t end   = start + size;

    // Align start down to 32-byte boundary
    start &= ~(uintptr_t)0x1F;

    // Align end up to 32-byte boundary
    end = (end + 0x1F) & ~(uintptr_t)0x1F;

    SCB_CleanDCache_by_Addr((uint32_t *)start, (int32_t)(end - start));
}


void resert_data_storage(){
    framePacket.header.frameID = 0;
    framePacket.header.syncWord = 000000;
    framePacket.header.payloadSize = 0;

    for(int i =0; i<8; i++)
    {
        framePacket.data[2][i].gain = 0.0;
        avg_sensor_read_t dummy_reading = demodulation_control_sequence(i,0);

    }

}

void transmission_listenner()
{
    HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);
    txBuf[231] = 0xBB;

    st = HAL_SPI_TransmitReceive_DMA(&hspi2, txBuf, rxDummy, XFER_LEN);
    if (st != HAL_OK)
    {
        HAL_SPI_Abort(&hspi2);
        spi_frame_active = 0;   // failed to start, allow retry
        HAL_Delay(60);

    }
               // allow next DMA start
//          spi_rx_ready = 1;

}



void HAL_SPI_TxRxCpltCallback(SPI_HandleTypeDef *hspi)
{
    if (hspi->Instance == SPI2)
    {
        spi_rx_ready = 1;
//        spi_rx_ready = 0;

//      scan_write_sequence();   // now rxDummy is a *complete frame*
//      reset_recieved_bytes();  // optional
////        HAL_GPIO_WritePin(GPIOB,Transmission_Complete_Pin,GPIO_PIN_SET);
//
//
//      spi_frame_active = 0;    // allow next DMA start
//      HAL_Delay(60);
    }


}


uint16_t get_opperation_complete_id(void)
```

```
{

    return opperation_complete_id;
}

void set_opperation_complete_id(uint16_t id)
{

    /**
     * 0xBB = Scan complete
     * 0xB1 = new plant
     * 0xB2 = new region
     * 0xB3 = save
     */

    txBuf[231] = id;


}




//void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
//{
//    if (GPIO_Pin == SPI2_NSS_Pin)   // your NSS pin
//    {
//
//      if (HAL_GPIO_ReadPin(SPI2_NSS_GPIO_Port, SPI2_NSS_Pin) == GPIO_PIN_SET)
//        {
//            // NSS rising = master ended the frame (or stopped clocking)
//            if (spi_frame_active)
//            {
////                HAL_SPI_Abort_IT(&hspi2);   // or HAL_SPI_DMAStop(&hspi2) depending on HAL version
////                spi_frame_active = 0;       // allow re-arm
//            }
//        }
//        else
//        {
//            // NSS falling = start of frame (optional)
//        }
//    }
//}


/* USER CODE END 4 */

 /* MPU Configuration */

void MPU_Config(void)
{

  /* Disables the MPU */
  HAL_MPU_Disable();
  /* Enables the MPU */
  HAL_MPU_Enable(MPU_PRIVILEGED_DEFAULT);

}

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
```

```
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

File: Core/Inc/caliberation.h

```
/*
 * caliberation.h
 *
 *  Created on: Jan 8, 2026
 *      Author: Albert
 */

#ifndef INC_CALIBERATION_H_
#define INC_CALIBERATION_H_


int caliberate(const uint8_t* rxdummy);
int caliberate_on_the_go(const uint8_t *rxdummy);


#endif /* INC_CALIBERATION_H_ */
```

File: Core/Src/caliberation.c

```
/*
 * caliberation.c
 *
 *  Created on: Jan 8, 2026
 *      Author: Albert
 */

#include "main.h"
#include "adc.h"
#include "dma.h"
#include "fatfs.h"
#include "sdmmc.h"
#include "spi.h"
#include "gpio.h"
#include <stddef.h>

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

#include "string.h"
```

```
#include "stdio.h"
#include "math.h"
#include "stdbool.h"
#include <string.h>
#include <storage.h>
#define MAX_MEMSET_RETRIES  5

extern FramePacket_t framePacket;
#define EPS_GAIN        (1e-6f)
#define CAL_FRAMES      (10)      // how many frames you want to average
#define CAL_SENSOR_IDX  (2)       // pick which sensor to use for calibration (0..NUM_SENSORS-1)




int caliberate(const uint8_t* rxdummy)
{

    uint8_t nFrames = 4;


    framePacket.header.syncWord = 0xAA55;
    FRESULT fr;
    FRESULT fr2 = -10;
    uint8_t caliberate_id;
    framePacket.header.payloadSize = sizeof(SIPM_t) * NUM_SENSORS * NUM_SPECTRA;

    const char* fieldDir = "field001";

    // 1) Find next calibration directory index: Calibration_###


    if (fr2 == FR_OK)
    {
         uint32_t next_idx= 0;
         fr = find_last_index_in_dir("field001", "Calibration_", &next_idx);
         if (fr != FR_OK) return -2;
         caliberate_id = next_idx;

    }

    else if(fr2 !=FR_OK)
    {
        caliberate_id = rxdummy[4];
    }


    // 2) Create directory 0:/field001/Calibration_###
    fr2 = create_entity_directory(fieldDir, "Calibration", caliberate_id,"",0);
    if (fr != FR_OK && fr != FR_EXIST) return -3;

    // 3) Accumulate sums for avg gain (choose one sensor index)
    float sumGain[NUM_SPECTRA] = {0};

    for (uint32_t k = 0; k < nFrames; k++) {
        framePacket.header.frameID++;
        // 3a) Capture a new frame (must update global framePacket)
        caliberation_modulation_control_sequence();

        // 3b) Save raw frame to CSV (each frame gets its own file index)
        fr = write_caliberation_frame_to_csv(fieldDir, caliberate_id, k, &framePacket);
        if (fr != FR_OK) return -4;

        // 3c) Accumulate gain for averaging (sensor CAL_SENSOR_IDX)
        for (uint32_t s = 0; s < NUM_SPECTRA; s++) {
            float g = framePacket.data[CAL_SENSOR_IDX][s].gain;
            sumGain[s] += g;
```

```
        }
    }

    // 4) Compute average gain
    float avgGain[NUM_SPECTRA];
    uint8_t valid[NUM_SPECTRA];
    for (uint32_t s = 0; s < NUM_SPECTRA; s++) {
        avgGain[s] = sumGain[s] / (float)nFrames;
        valid[s] = (avgGain[s] > EPS_GAIN) ? 1 : 0;
    }

    // 5) Find reference = smallest positive avgGain
    float ref = 0.0f;
    uint8_t found = 0;
    for (uint32_t s = 0; s < NUM_SPECTRA; s++) {
        if (valid[s]) {
            if (!found || avgGain[s] < ref) {
                ref = avgGain[s];
                found = 1;
            }
        }
    }
    if (!found) return -5; // all are <= 0 or too small

    // 6) Compute normalization factors
    float norm[NUM_SPECTRA];
    for (uint32_t s = 0; s < NUM_SPECTRA; s++) {
        norm[s] = valid[s] ? (ref / avgGain[s]) : 1.0f;
    }

    // 7) Save avg + norm to CSV in the same calibration folder
    fr = write_avg_and_norm_csv(fieldDir, caliberate_id, avgGain, norm, ref);
    if (fr != FR_OK) return -6;

    return 0; // success
}

int caliberate_on_the_go(const uint8_t *rxdummy)
{

    uint8_t nFrames = 4;


    framePacket.header.syncWord = 0xAA55;
    FRESULT fr;
    FRESULT fr2 = -10;
    uint8_t caliberate_id;
    framePacket.header.payloadSize = sizeof(SIPM_t) * NUM_SENSORS * NUM_SPECTRA;

    const char* fieldDir = "field001";

    // 1) Find next calibration directory index: Calibration_###


    if (fr2 == FR_OK)
    {
         uint32_t next_idx= rxdummy[4];;
         fr = find_last_index_in_dir("field001", "Calibration_", &next_idx);
         if (fr != FR_OK) return -2;
         caliberate_id = next_idx;

    }

    else if(fr2 !=FR_OK)
    {
        caliberate_id = rxdummy[4];
    }
```

```
    // 2) Create directory 0:/field001/Calibration_###
    fr2 = create_entity_directory(fieldDir, "Calibration", caliberate_id,"",0);
    if (fr != FR_OK && fr != FR_EXIST) return -3;

    // 3) Accumulate sums for avg gain (choose one sensor index)
    float sumGain[NUM_SPECTRA] = {0};

    for (uint32_t k = 0; k < nFrames; k++) {
        framePacket.header.frameID++;
        // 3a) Capture a new frame (must update global framePacket)
        caliberation_modulation_control_sequence();

        // 3b) Save raw frame to CSV (each frame gets its own file index)
        fr = write_caliberation_frame_to_csv(fieldDir, caliberate_id, k, &framePacket);
        if (fr != FR_OK) return -4;

        // 3c) Accumulate gain for averaging (sensor CAL_SENSOR_IDX)
        for (uint32_t s = 0; s < NUM_SPECTRA; s++) {
            float g = framePacket.data[CAL_SENSOR_IDX][s].gain;
            sumGain[s] += g;
        }
    }

    // 4) Compute average gain
    float avgGain[NUM_SPECTRA];
    uint8_t valid[NUM_SPECTRA];
    for (uint32_t s = 0; s < NUM_SPECTRA; s++) {
        avgGain[s] = sumGain[s] / (float)nFrames;
        valid[s] = (avgGain[s] > EPS_GAIN) ? 1 : 0;
    }

    // 5) Find reference = smallest positive avgGain
    float ref = 0.0f;
    uint8_t found = 0;
    for (uint32_t s = 0; s < NUM_SPECTRA; s++) {
        if (valid[s]) {
            if (!found || avgGain[s] < ref) {
                ref = avgGain[s];
                found = 1;
            }
        }
    }
    if (!found) return -5; // all are <= 0 or too small

    // 6) Compute normalization factors
    float norm[NUM_SPECTRA];
    for (uint32_t s = 0; s < NUM_SPECTRA; s++) {
        norm[s] = valid[s] ? (ref / avgGain[s]) : 1.0f;
    }

    // 7) Save avg + norm to CSV in the same calibration folder
    fr = write_avg_and_norm_csv(fieldDir, caliberate_id, avgGain, norm, ref);
    if (fr != FR_OK) return -6;

    return 0; // success
}
```

File: Core/Inc/storage.h

```
/*
 * storage.h
 *
 *  Created on: Aug 12, 2025
 *      Author: Albert Benson
 */
```

```
#ifndef INC_STORAGE_H_
#define INC_STORAGE_H_

#include "storage.h"
#include "stm32h7xx_hal.h"
#include "stm32h7xx_hal_sd.h"
#include "fatfs.h"
#include "main.h"
#include "stdio.h"
#include "string.h"


typedef enum
{
    SD_CARD_FAIL = 1
} status_code;

//extern SD_HandleTypeDef hsd1;

FRESULT create_plant_directory(const char* fieldDir, uint8_t plant_id);
HAL_StatusTypeDef storage_getCapacity(SD_HandleTypeDef *hsd,uint8_t lun,uint32_t block_num,uint16_t block_size
    ↪ );
FRESULT create_plant_directory(const char* fieldDir, uint8_t plant_id);
FRESULT write_frame_to_csv(const char* fieldDir, uint8_t plant_id, const char* region, uint8_t leaf_id,
    ↪ FramePacket_t* framePacket);
FRESULT STM32_PLC_SD_Unmount_Card(void);
FRESULT STM32_PLC_SD_Mount_Card(void);
void start_storage(SD_HandleTypeDef *hsd);
FRESULT mkdir_next_plant(const char* base_dir, unsigned pad_width,
                         char* out_path, size_t out_path_sz,uint32_t next_idx);
FRESULT find_next_index_in_dir(const char* dirpath,
                               const char* prefix,
                               uint32_t* next_idx_out);
FRESULT ensure_dir(const char* dirpath);

FRESULT find_last_index_in_dir(const char* dirpath,
                               const char* prefix,
                               uint32_t* next_idx_out);
FRESULT create_entity_directory(
    const char* fieldDir,        // e.g. "field001"
    const char* entityName,      // e.g. "Plant", "Sample", "Sensor"
    uint16_t entity_id,          // numeric ID
    const char* const* subdirs,  // array of subfolder names
    uint8_t num_subdirs
);
FRESULT write_caliberation_frame_to_csv(const char* fieldDir,uint8_t calibration_id,uint8_t caliberate_id ,
    ↪ FramePacket_t* framePacket);
//FRESULT write_average_caliberation_to_csv(const char* fieldDir,uint8_t calibration_id ,float
    ↪ average_calibrate );

FRESULT write_avg_and_norm_csv(const char* fieldDir, uint32_t cal_id,
                                    const float avgGain[NUM_SPECTRA],
                                    const float norm[NUM_SPECTRA],
                                    float ref);

FRESULT load_norm_factors(const char* fieldDir, uint32_t cal_id, float norm[NUM_SPECTRA]);
static int parse_norm_line(const char* line, int* idx, float* val);




#endif /* INC_STORAGE_H_ */
```

File: Core/Src/storage.c

```
/*
 * storage.c
 *
 *  Created on: Aug 12, 2025
 *      Author: Albert Benson
 */

#include "storage.h"
#include "stm32h7xx_hal.h"
#include "stm32h7xx_hal_sd.h"
#include "fatfs.h"
#include "main.h"
#include "stdio.h"
#include "string.h"

#include "ff.h"
#include <stdio.h>

#define NUM_SPECTRA 7



const char* SENSOR_NAMES[] = {"N", "M", "E", "W"};
int next_plant_index = 0;


void start_storage(SD_HandleTypeDef *hsd)
{

    FRESULT res;
    HAL_StatusTypeDef status;
    status =  HAL_SD_Init(hsd);
    res =STM32_PLC_SD_Mount_Card();

}


HAL_StatusTypeDef storage_getCapacity(SD_HandleTypeDef *hsd,uint8_t lun,uint32_t block_num,uint16_t block_size
    ↪ )
{
    HAL_SD_CardInfoTypeDef *sd_Card_info;

    if(HAL_SD_GetCardInfo(hsd,sd_Card_info) !=HAL_OK)
    {
        return SD_CARD_FAIL;
    }

    block_num = sd_Card_info->BlockNbr;
    block_size = sd_Card_info->BlockSize;

    return HAL_OK;
}

FRESULT create_plant_directory(const char* fieldDir, uint8_t plant_id)
{
    FRESULT res;
    char path[128];

    const char* base = "0:/field001";
    char new_path[128];

    FRESULT fr = mkdir_next_plant(base, 3, new_path, sizeof(new_path),plant_id);


    // 0) Make sure the field directory exists: e.g., "/field_001"
```

```
    // Build absolute path: "0:/field_001"
//    snprintf(path, sizeof(path), "0:/%s", fieldDir);
//    res = f_mkdir(path);
//    if (res != FR_OK && res != FR_EXIST) return res;
//
//    // 1) Create main plant folder: "0:/field_001/plant_001"
//    snprintf(path, sizeof(path), "0:%s/plant_%03d", fieldDir, plant_id);
//    res = f_mkdir(path);
//    if (res != FR_OK && res != FR_EXIST) return res;

    // 2) Create subfolders under the plant folder (all ABSOLUTE paths with "0:")
    const char* regions[] = {"lower", "middle", "upper"};
    for (int i = 0; i < 3; i++) {
        snprintf(path, sizeof(path), "0:%s/Plant_%03d/%s", fieldDir, plant_id, regions[i]);
        res = f_mkdir(path);
        if (res != FR_OK && res != FR_EXIST) return res;
    }

    return FR_OK;
}

FRESULT create_entity_directory(
    const char* fieldDir,        // e.g. "field001"
    const char* entityName,      // e.g. "Plant", "Sample", "Sensor"
    uint16_t entity_id,          // numeric ID
    const char* const* subdirs,  // array of subfolder names
    uint8_t num_subdirs
)
{
    FRESULT res;
    char path[128];

    /* 1) Ensure field directory exists */
    snprintf(path, sizeof(path), "0:/%s", fieldDir);
    res = f_mkdir(path);
    if (res != FR_OK && res != FR_EXIST) return res;

    /* 2) Create main entity directory */
    snprintf(path, sizeof(path), "0:/%s/%s_%03d", fieldDir, entityName, entity_id);
    res = f_mkdir(path);
    if (res != FR_OK && res != FR_EXIST) return res;

    /* 3) Create optional subdirectories */
//    for (uint8_t i = 0; i < num_subdirs; i++)
//    {
//        snprintf(path, sizeof(path),
//                 "0:/%s/%s_%03d/%s",
//                 fieldDir, entityName, entity_id, subdirs[i]);
//
//        res = f_mkdir(path);
//        if (res != FR_OK && res != FR_EXIST) return res;
//    }

    return FR_OK;
}



FRESULT write_frame_to_csv(const char* fieldDir, uint8_t plant_id, const char* region, uint8_t leaf_id,
    ↪ FramePacket_t* framePacket)
{
    FRESULT res;
    FIL file;
    char path[128];
    char line[128];

    // Format: /field_001/plant_001/lower/leaf_001.csv
    snprintf(path, sizeof(path), "0:%s/Plant_%03d/%s/leaf_%03d.csv", fieldDir, plant_id, region, leaf_id);
```

```
    // Open file for append or create if it doesn't exist
    res = f_open(&file, path, FA_CREATE_ALWAYS | FA_READ | FA_WRITE);

//    if(res == FR_NO_PATH){
//      create_plant_directory(fieldDir, plant_id);
//      write_frame_to_csv(fieldDir, plant_id, region, leaf_id, framePacket);
//
//
//    }
    if (res != FR_OK) {
        return res;
    }

    // Optionally write CSV header if file is new
    if (f_size(&file) == 0) {
        const char* header = "FrameID,SpectrumIndex,Sensor,Gain,Phase\n";
        UINT bw;
        f_write(&file, header, strlen(header), &bw);
    }

    // Loop through spectra and sensors
    for (uint8_t spectrum = 0; spectrum < NUM_SPECTRA; spectrum++)
    {

        for (uint8_t sensor = 0; sensor < NUM_SENSORS; sensor++)
        {
            SIPM_t* data = &framePacket->data[sensor][spectrum];

            snprintf(line, sizeof(line), "%lu,%u,%s,%.3f,%.3f\n",
                     (unsigned long)framePacket->header.frameID,
                     spectrum,
                     SENSOR_NAMES[sensor],
                     (double)data->gain,
                     (double)data->phase);

            UINT bw;
            res = f_write(&file, line, strlen(line), &bw);

//            if (res == FR_OK && bw == strlen(line)) {
//              printf("Write Successful");
//                // Write successful
//            } else {
//              res = f_write(&file, line, strlen(line), &bw);
//                // Handle error or partial write
//            }

            if (res != FR_OK) {
                f_close(&file);
                return res;
            }
        }
    }



    f_close(&file);
    return FR_OK;
}


FRESULT write_caliberation_frame_to_csv(const char* fieldDir,uint8_t calibration_id,uint8_t caliberate_id ,
    ↪ FramePacket_t* framePacket)
{
    FRESULT res;
    FIL file;
    char path[128];
    char line[128];
```

```
    snprintf(path, sizeof(path), "0:%s/Calibration_%03d/calibrate_%03d.csv", fieldDir,calibration_id,
     ↪ caliberate_id);

    // Open file for append or create if it doesn't exist
    res = f_open(&file, path, FA_CREATE_ALWAYS | FA_READ | FA_WRITE);
//
//    if(res == FR_NO_PATH){
//      create_plant_directory(fieldDir, plant_id);
//      write_frame_to_csv(fieldDir, plant_id, region, leaf_id, framePacket);
//
//
//    }
    if (res != FR_OK) {
        return res;
    }

    // Optionally write CSV header if file is new
    if (f_size(&file) == 0) {
        const char* header = "FrameID,SpectrumIndex,Sensor,Gain,Phase\n";
        UINT bw;
        f_write(&file, header, strlen(header), &bw);
    }

    // Loop through spectra and sensors
    for (uint8_t spectrum = 0; spectrum < NUM_SPECTRA; spectrum++)
    {

        for (uint8_t sensor = 0; sensor < NUM_SENSORS; sensor++)
        {
            SIPM_t* data = &framePacket->data[sensor][spectrum];

            snprintf(line, sizeof(line), "%lu,%u,%s,%.3f,%.3f\n",
                     (unsigned long)framePacket->header.frameID,
                     spectrum,
                     SENSOR_NAMES[sensor],
                     (double)data->gain,
                     (double)data->phase);

            UINT bw;
            res = f_write(&file, line, strlen(line), &bw);

//            if (res == FR_OK && bw == strlen(line)) {
//              printf("Write Successful");
//                // Write successful
//            } else {
//              res = f_write(&file, line, strlen(line), &bw);
//                // Handle error or partial write
//            }

            if (res != FR_OK) {
                f_close(&file);
                return res;
            }
        }
    }



    f_close(&file);
    return FR_OK;
}


//FRESULT write_average_caliberation_to_csv(const char* fieldDir,uint8_t calibration_id ,float
     ↪ average_calibrate )
//{
//    FRESULT res;
```

```
//    FIL file;
//    char path[128];
//    char line[128];
//
//    // Format: /field_001/plant_001/lower/leaf_001.csv
//    snprintf(path, sizeof(path), "0:%s/Calibration_%03d/avg_calibrate.csv", fieldDir, calibration_id );
//
//    // Open file for append or create if it doesn't exist
//    res = f_open(&file, path, FA_CREATE_ALWAYS | FA_READ | FA_WRITE);
//
////    if(res == FR_NO_PATH){
////        create_plant_directory(fieldDir, plant_id);
////        write_frame_to_csv(fieldDir, plant_id, region, leaf_id, framePacket);
////
////
////    }
//    if (res != FR_OK) {
//        return res;
//    }
//
//    // Optionally write CSV header if file is new
//    if (f_size(&file) == 0) {
//        const char* header = "average_caliberation\n";
//        UINT bw;
//        f_write(&file, header, strlen(header), &bw);
//    }
//
//    snprintf(line, sizeof(line), "%.3f\n",average_calibrate);
//    UINT bw;
//    res = f_write(&file, line, strlen(line), &bw);
//
//    // Loop through spectra and sensors
//
//
//
//    f_close(&file);
//    return FR_OK;
//}


FRESULT write_avg_and_norm_csv(const char* fieldDir, uint32_t cal_id,
                                    const float avgGain[NUM_SPECTRA],
                                    const float norm[NUM_SPECTRA],
                                    float ref)
{
    FRESULT res;
    FIL file;
    char path[128];
    char line[128];

    // ---- avg_calibrate.csv ----
    snprintf(path, sizeof(path), "0:/%s/Calibration_%03lu/avg_calibrate.csv",
             fieldDir, (unsigned long)cal_id);

    res = f_open(&file, path, FA_CREATE_ALWAYS | FA_WRITE);
    if (res != FR_OK) return res;

    // header
    UINT bw;
    const char* header1 = "SpectrumIndex,AvgGain\n";
    f_write(&file, header1, strlen(header1), &bw);

    for (uint32_t s = 0; s < NUM_SPECTRA; s++) {
        int n = snprintf(line, sizeof(line), "%lu,%.6f\n",
                         (unsigned long)s, (double)avgGain[s]);
        f_write(&file, line, (UINT)n, &bw);
    }
    f_close(&file);
```

```
    // ---- norm.csv ----
    snprintf(path, sizeof(path), "0:/%s/Calibration_%03lu/norm.csv",
             fieldDir, (unsigned long)cal_id);

    res = f_open(&file, path, FA_CREATE_ALWAYS | FA_WRITE);
    if (res != FR_OK) return res;

    const char* header2 = "SpectrumIndex,NormFactor,RefGain\n";
    f_write(&file, header2, strlen(header2), &bw);

    for (uint32_t s = 0; s < NUM_SPECTRA; s++) {
        int n = snprintf(line, sizeof(line), "%lu,%.6f,%.6f\n",
                         (unsigned long)s, (double)norm[s], (double)ref);
        f_write(&file, line, (UINT)n, &bw);
    }
    f_close(&file);

    return FR_OK;
}




static int parse_norm_line(const char* line, int* idx, float* val)
{
    // expects: "idx,norm" OR "idx,norm,ref"
    // skip header lines
    if (!line || !(line[0] >= '0' && line[0] <= '9')) return 0;

    char* endptr = NULL;
    long i = strtol(line, &endptr, 10);
    if (endptr == line || *endptr != ',') return 0;

    float v = strtof(endptr + 1, &endptr);
    if (endptr == NULL) return 0;

    *idx = (int)i;
    *val = v;
    return 1;
}

FRESULT load_norm_factors(const char* fieldDir, uint32_t cal_id, float norm[NUM_SPECTRA])
{
    FIL file;
    FRESULT fr;
    char path[128];

    // default = 1.0 (no scaling)
    for (int i = 0; i < NUM_SPECTRA; i++) norm[i] = 1.0f;

    snprintf(path, sizeof(path), "0:/%s/Calibration_%03lu/norm.csv",
             fieldDir, (unsigned long)cal_id);

    fr = f_open(&file, path, FA_READ);
    if (fr != FR_OK) return fr;

    char line[128];
    while (f_gets(line, sizeof(line), &file))
    {
        int idx;
        float val;
        if (parse_norm_line(line, &idx, &val))
        {
            if (idx >= 0 && idx < NUM_SPECTRA) {
                norm[idx] = val;
            }
        }
```

```
    }

    f_close(&file);
    return FR_OK;
}



FRESULT STM32_PLC_SD_Mount_Card(void) {

    DSTATUS st = disk_initialize(0);             // your SD_diskio.c init
    FRESULT status;
    uint8_t attempt = 0;
    while(attempt < 255) {
        MX_FATFS_Init();
        status = f_mount(&SDFatFS, SDPath, 1);
        if(status == FR_OK){
            break;
        } else {
            STM32_PLC_SD_Unmount_Card();
            attempt++;
        }
    }
    return status;
}

FRESULT STM32_PLC_SD_Unmount_Card(void){

    FRESULT status = f_mount(NULL, "", 0);

    return status;
}


#include "ff.h"
#include <string.h>
#include <stdio.h>
#include <ctype.h>

/* Ensure base directory exists (create if not) */
FRESULT ensure_dir(const char* dirpath)
{
    DIR dir;
    FRESULT fr = f_opendir(&dir, dirpath);
    if (fr == FR_NO_PATH || fr == FR_NO_FILE) {
        // Base directory missing -> create it
        fr = f_mkdir(dirpath);
        if (fr == FR_EXIST) fr = FR_OK; // already created by another task
    }
    if (fr == FR_OK) f_closedir(&dir);
    return fr;
}

/* Find next numeric index after the highest Plant_### directory */
FRESULT find_next_index_in_dir(const char* dirpath,
                               const char* prefix,
                               uint32_t* next_idx_out)
{
    if (!dirpath || !prefix || !next_idx_out) return FR_INVALID_OBJECT;

    FRESULT fr = ensure_dir(dirpath);
    if (fr != FR_OK) return fr;

    DIR dir;
    FILINFO fno;
#if FF_USE_LFN
    static char lfn[FF_MAX_LFN + 1];
```

```
    fno.lfname = lfn;
    fno.lfsize = sizeof(lfn);
#endif

    *next_idx_out = 0;           // default if none found
    uint32_t max_idx = 0;

    fr = f_opendir(&dir, dirpath);
    if (fr != FR_OK) return fr;

    for (;;) {
        fr = f_readdir(&dir, &fno);
        if (fr != FR_OK) break;
        if (fno.fname[0] == 0)   // end of dir
            break;

        const char* name =
#if FF_USE_LFN
            (fno.lfname && fno.lfname[0]) ? fno.lfname :
#endif
            fno.fname;

        // Directories only
        if ((fno.fattrib & AM_DIR) == 0) continue;
        if (strcmp(name, ".") == 0 || strcmp(name, "..") == 0) continue;

        // Must start with prefix, e.g., "Plant_"
        size_t prelen = strlen(prefix);
        if (strncmp(name, prefix, prelen) != 0) continue;

        // Parse digits after prefix
        const char* p = name + prelen;
        if (*p == '\0') continue;

        uint32_t val = 0;
        int any_digit = 0;
        while (*p) {
            if (!isdigit((unsigned char)*p)) { any_digit = 0; break; }
            any_digit = 1;
            val = val * 10u + (uint32_t)(*p - '0');
            p++;
        }

        if (any_digit && val > max_idx) max_idx = val;
    }

    f_closedir(&dir);

    if (fr == FR_OK) {
        *next_idx_out = max_idx + 1;
//        *next_idx_out = max_idx;

    }

    return fr;
}


FRESULT find_last_index_in_dir(const char* dirpath,
                               const char* prefix,
                               uint32_t* next_idx_out)
{
    if (!dirpath || !prefix || !next_idx_out) return FR_INVALID_OBJECT;

    FRESULT fr = ensure_dir(dirpath);
    if (fr != FR_OK) return fr;

    DIR dir;
```

```
    FILINFO fno;
#if FF_USE_LFN
    static char lfn[FF_MAX_LFN + 1];
    fno.lfname = lfn;
    fno.lfsize = sizeof(lfn);
#endif

    *next_idx_out = 0;           // default if none found
    uint32_t max_idx = 0;

    fr = f_opendir(&dir, dirpath);
    if (fr != FR_OK) return fr;

    for (;;) {
        fr = f_readdir(&dir, &fno);
        if (fr != FR_OK) break;
        if (fno.fname[0] == 0)   // end of dir
            break;

        const char* name =
#if FF_USE_LFN
            (fno.lfname && fno.lfname[0]) ? fno.lfname :
#endif
            fno.fname;

        // Directories only
        if ((fno.fattrib & AM_DIR) == 0) continue;
        if (strcmp(name, ".") == 0 || strcmp(name, "..") == 0) continue;

        // Must start with prefix, e.g., "Plant_"
        size_t prelen = strlen(prefix);
        if (strncmp(name, prefix, prelen) != 0) continue;

        // Parse digits after prefix
        const char* p = name + prelen;
        if (*p == '\0') continue;

        uint32_t val = 0;
        int any_digit = 0;
        while (*p) {
            if (!isdigit((unsigned char)*p)) { any_digit = 0; break; }
            any_digit = 1;
            val = val * 10u + (uint32_t)(*p - '0');
            p++;
        }

        if (any_digit && val > max_idx) max_idx = val;
    }

    f_closedir(&dir);

    if (fr == FR_OK) {
//        *next_idx_out = max_idx + 1;
        *next_idx_out = max_idx;

    }

    return fr;
}


/* Format: "Plant_%03u" into out_name */
void format_indexed_name(char* out_name, size_t out_sz,
                         const char* prefix, uint32_t idx, unsigned pad_width)
{
    snprintf(out_name, out_sz, "%s%0*u", prefix, (int)pad_width, (unsigned)idx);
}
```

```
/* Convenience: create the next Plant_### directory and return the path */
FRESULT mkdir_next_plant(const char* base_dir, unsigned pad_width,
                         char* out_path, size_t out_path_sz,uint32_t next_idx)
{



//    FRESULT fr = find_next_index_in_dir(base_dir, "Plant_", &next_idx);
//    if (fr != FR_OK) return fr;

//    next_plant_index = next_idx;
    FRESULT fr;
    char name[32];
    format_indexed_name(name, sizeof(name), "Plant_", next_idx, pad_width);

    snprintf(out_path, out_path_sz, "%s/%s", base_dir, name);

    fr = f_mkdir(out_path);
    if (fr == FR_EXIST) return FR_OK;
    return fr;
}



//HAL_StatusTypeDef storage_read(SD_HandleTypeDef *hsd,uint8_t lun,uint8_t *buf,uin32_t *blk_addr,uint16_t
    ↪ blk_len)
//{
//  if(HAL_SD_ReadBlocks(hsd,buf,blk_addr,))
//}
```

Display Unit Firmware (STM32F746ZET6, FreeRTOS and TouchGFX)

File: `Core/Src/main.c`

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "cmsis_os.h"
#include "crc.h"
#include "dma.h"
#include "dma2d.h"
#include "i2c.h"
#include "ltdc.h"
#include "quadspi.h"
#include "spi.h"
#include "gpio.h"
#include "fmc.h"
#include "app_touchgfx.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

#include "string.h"
#include "shared.h"
#include "ft5336.h"
#include "stm32746g_discovery_ts.h"
#include "stdbool.h"
#include "semphr.h"

//AI libraries
#include "ai_platform.h"
#include "ai_datatypes_defines.h"
#include "network_data_params.h"
#include "network.h"
#include "network_data.h"


/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
```

```
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */


extern osThreadId_t nextPlant_tskHandle;
extern osSemaphoreId_t SpiTxDoneHandle;;
extern osThreadId_t next_RegionHandle;
extern osThreadId_t start_Disp_LEDSHandle;

/*spi variables*/
//static uint8_t spi_tx_buf[XFER_LEN];
//static uint8_t spi_rx_buf[XFER_LEN];
static volatile uint8_t spi_frame_done = 0;
static volatile uint8_t spi_frame_active = 0;


FramePacket_t framePacket;
uint8_t rxDummy[XFER_LEN] = {0x0A};
extern uint8_t txBuf[XFER_LEN];
extern bool spi_done;
uint8_t data_available_flag = 0;


static const size_t FB_SIZE =
    (size_t)LCD_W * (size_t)LCD_H * (size_t)BYTES_PER_PIXEL; // size of the frame buffer


//AI variables

//AI_ALIGNED(4);
AI_ALIGNED(32)
static ai_u8 activations[AI_NETWORK_DATA_ACTIVATIONS_SIZE];
//AI_ALIGNED(4);


ai_handle network;
ai_buffer *ai_input;
ai_buffer *ai_output;



/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MPU_Config(void);
void MX_FREERTOS_Init(void);
/* USER CODE BEGIN PFP */

void FMC_INIT(void);
void FMC_TEST(void);
void QuadSpiTest(void);
void Scan_I2C_Bus(I2C_HandleTypeDef *hi2c);
void Read_FT5426_ChipID(I2C_HandleTypeDef *hi2c);
bool read_xy(void);
void QuadSpiClean(void);
void debug_verify_weights(void);

/*I2c function prototypes*/
```

```
static HAL_StatusTypeDef I2Cx_ReadMultiple(I2C_HandleTypeDef *i2c_handler, uint8_t Addr, uint16_t Reg,
    ↪ uint16_t MemAddSize, uint8_t *Buffer, uint16_t Length);
static HAL_StatusTypeDef I2Cx_WriteMultiple(I2C_HandleTypeDef *i2c_handler, uint8_t Addr, uint16_t Reg,
    ↪ uint16_t MemAddSize, uint8_t *Buffer, uint16_t Length);
static HAL_StatusTypeDef I2Cx_IsDeviceReady(I2C_HandleTypeDef *i2c_handler, uint16_t DevAddress, uint32_t
    ↪ Trials);
static void              I2Cx_Error(I2C_HandleTypeDef *i2c_handler, uint8_t Addr);
int _write(int file, char *ptr, int len);

//static inline void SPI_PreloadResponse(void);
//static void SPI_Slave_AbortIfNeeded(void);
//static void SPI_Slave_ArmDMA(void);




/* TOUCHSCREEN IO functions */
void            TS_IO_Init(void);
void            TS_IO_Write(uint8_t Addr, uint8_t Reg, uint8_t Value);
uint8_t         TS_IO_Read(uint8_t Addr, uint8_t Reg);
void            TS_IO_Delay(uint32_t Delay);

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */


/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MPU Configuration--------------------------------------------------------*/
  MPU_Config();

  /* Enable the CPU Cache */

  /* Enable I-Cache-----------------------------------------------------------*/
  SCB_EnableICache();

  /* Enable D-Cache-----------------------------------------------------------*/
  SCB_EnableDCache();

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */
   // <== required early
   // <== required early


  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();
```

```
  /* USER CODE BEGIN SysInit */

//  SysTick_Config(SystemCoreClock / 1000);
  NVIC_SetPriority(SysTick_IRQn, 0);
//  NVIC_EnableIRQ(SysTick_IRQn);
//  CSP_QUADSPI_Init();
//


  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_FMC_Init();
  MX_QUADSPI_Init();
  MX_LTDC_Init();
  MX_DMA2D_Init();
  MX_CRC_Init();
  MX_I2C3_Init();
  MX_SPI1_Init();
  MX_TouchGFX_Init();
  /* Call PreOsInit function */
  MX_TouchGFX_PreOSInit();
  /* USER CODE BEGIN 2 */
  CSP_QSPI_EnableMemoryMappedMode();
//  MX_QUADSPI_Init();



//  CSP_QUADSPI_Init(); // <== required early
//  CSP_QSPI_EnableMemoryMappedMode(); // <== required early
//  SCB_DisableDCache(); // Just for testing purposes


//  CSP_QUADSPI_Init();
//  FMC_TEST();
//  QuadSpiClean();
//  Scan_I2C_Bus(&hi2c3);
//  Read_FT5426_ChipID(&hi2c3);
//  TS_IO_Read(FT5336_I2C_SLAVE_ADDRESS  ,FT5336_TD_STAT_REG);
//  read_xy();


/* USER CODE BEGIN 1 */
 debug_verify_weights();

ai_network_params ai_params = {
        AI_NETWORK_DATA_WEIGHTS(ai_network_data_weights_get()),
        AI_NETWORK_DATA_ACTIVATIONS(activations)
};



//  ai_network_get_info(&network_info); // Get info on buffer sizes
//ai_error error = ai_network_create(&network,AI_NETWORK_DATA_CONFIG); // Create instance
//
//ai_bool init_success = ai_network_init(network,&ai_params);
ai_error err;
err = ai_network_create(&network, AI_NETWORK_DATA_CONFIG);

if (err.type != AI_ERROR_NONE) {
    // Handling the error
    // You can check err.code for specific details
    printf("Error Here");
}

if (!ai_network_init(network, &ai_params)) {
```

```
    // Handle Error
}




//  if (spi_done && HAL_SPI_GetState(&hspi1) == HAL_SPI_STATE_READY)
//      {
//          spi_done = false;
//
//      //
//          HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_RESET); // CS low
//
//          HAL_StatusTypeDef st = HAL_SPI_TransmitReceive_DMA(&hspi1, txBuf, spiRxBuffer, 300);
//
//
//      }

//  HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_SET);

//  HAL_SPI_Receive_DMA(&hspi1, spiRxBuffer, 300);
//  HAL_StatusTypeDef s = HAL_SPI_TransmitReceive_DMA(&hspi1, spiRxBuffer,rxDummy, 1);
//  HAL_StatusTypeDef st = HAL_SPI_TransmitReceive_DMA(&hspi1, spi_tx_buf, spi_rx_buf, XFER_LEN);
  HAL_GPIO_WritePin(GPIOB,Display_PWM_Pin,GPIO_PIN_SET);


//  QuadSpiTest();
  memset((void*)FB0_ADDR, 0xAA, FB_SIZE);    // white frame (for RGB888/ARGB8888)

  /* USER CODE END 2 */

  /* Init scheduler */
  osKernelInitialize();

  /* Call init function for freertos objects (in cmsis_os2.c) */
  MX_FREERTOS_Init();

  /* Start scheduler */
  osKernelStart();

  /* We should never get here as control is now taken by the scheduler */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure LSE Drive Capability
  */
  HAL_PWR_EnableBkUpAccess();
```

```
  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 216;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Activate the Over-Drive mode
  */
  if (HAL_PWREx_EnableOverDrive() != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7) != HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */

void FMC_INIT(void)
{
     /* USER CODE BEGIN FMC_Init 2 */
      FMC_SDRAM_CommandTypeDef Command;
      HAL_StatusTypeDef status;
      /* Step 1 and Step 2 already done in HAL_SDRAM_Init() */
      /* Step 3: Configure a clock configuration enable command */
       Command.CommandMode            = FMC_SDRAM_CMD_CLK_ENABLE; /* Set MODE bits to "001" */
       Command.CommandTarget          = FMC_SDRAM_CMD_TARGET_BANK1; /* configure the Target Bank bits */
       Command.AutoRefreshNumber      = 1;
       Command.ModeRegisterDefinition = 0;
       status = HAL_SDRAM_SendCommand(&hsdram2, &Command, 0xfff);
//     HAL_Delay(2); /* Step 4: Insert 100 us minimum delay - Min HAL Delay is 1ms */
       /* Step 5: Configure a PALL (precharge all) command */
       Command.CommandMode            = FMC_SDRAM_CMD_PALL; /* Set MODE bits to "010" */
       status = HAL_SDRAM_SendCommand(&hsdram2, &Command, 0xfff);
       /* Step 6: Configure an Auto Refresh command */
       Command.CommandMode            = FMC_SDRAM_CMD_AUTOREFRESH_MODE; /* Set MODE bits to "011" */
       Command.AutoRefreshNumber      = 2;
       status = HAL_SDRAM_SendCommand(&hsdram2, &Command, 0xfff);
       /* Step 7: Program the external memory mode register */
       Command.CommandMode            = FMC_SDRAM_CMD_LOAD_MODE;/*set the MODE bits to "100" */
```

```
       Command.ModeRegisterDefinition =  (uint32_t)0 | 0<<3 | 2<<4 | 0<<7 | 1<<9;
       status = HAL_SDRAM_SendCommand(&hsdram2, &Command, 0xfff);
       /* Step 8: Set the refresh rate counter - refer to section SDRAM refresh timer register in RM0455 */
       /* Set the device refresh rate
        * COUNT = [(SDRAM self refresh time / number of row) x  SDRAM CLK] - 20
                = [(64ms/4096) * 100MHz] - 20 = 1562.5 - 20 ~ 1542 */
       HAL_SDRAM_ProgramRefreshRate(&hsdram2,1542);
      /* USER CODE END FMC_Init 2 */
}


void FMC_TEST(void)
{

    uint8_t wdata[] = {0x1, 0x2, 0x3, 0x4, 0x10,0x33};
    uint8_t rdata[10];

    for(uint32_t i = 0x0; i < SDRAM_SIZE; i++)
        {
            *(__IO uint8_t *) (SDRAM_ADDRESS_START + i) = (uint8_t)0x00;
        }



//  memcpy((void *) SDRAM_ADDRESS_START, wdata, 6);
//  memset(rdata, 0, sizeof(rdata));
//
//  memcpy(rdata, (uint32_t *) SDRAM_ADDRESS_START, 6);


//
//  uint32_t fmcTest_Start;
//  uint32_t fmcTest_stop;
//
//  fmcTest_Start = HAL_GetTick();
//
//  for(uint32_t i = 0x0; i < SDRAM_SIZE; i++)
//  {
//      *(__IO uint8_t *) (SDRAM_ADDRESS_START + i) = (uint8_t)0x00;
//  }
//
//  fmcTest_stop = HAL_GetTick() - fmcTest_Start;
//
//  ;

//  HAL_Delay(50);

//  fmcTest_Start = HAL_GetTick();
//  volatile uint16_t* sdram = (volatile uint16_t*) 0xC0000000;
//
//  sdram[0] = 0x1234;
//  sdram[1] = 0x5678;
//  sdram[2] = 0x9ABC;
//  sdram[3] = 0xDEF0;
//
//  uint16_t test = sdram[1]; // Should be 0x5678
//  for (uint32_t i = 0x00; i < SDRAM_SIZE; i++)
//  {
//      *(__IO uint8_t *)(SDRAM_ADDRESS_START + i) = (uint8_t)i;
//  }

//  fmcTest_stop = HAL_GetTick() - fmcTest_Start;
//  HAL_Delay(50);



}
```

```
void QuadSpiTest(void)
{
    uint8_t buffer_test[MEMORY_SECTOR_SIZE];
    uint32_t var = 0;

    CSP_QUADSPI_Init();

    for (var = 0; var < MEMORY_SECTOR_SIZE; var++) {
        buffer_test[var] = (var & 0xFF);
    }

    for (var = 0; var < SECTORS_COUNT; var++) {

        if (CSP_QSPI_EraseSector(var * MEMORY_SECTOR_SIZE,
                                 (var + 1) * MEMORY_SECTOR_SIZE - 1) != HAL_OK) {

            while (1)
                ;  //breakpoint - error detected
        }

        if (CSP_QSPI_WriteMemory(buffer_test, var * MEMORY_SECTOR_SIZE, sizeof(buffer_test)) != HAL_OK) {

            while (1)
                ;  //breakpoint - error detected
        }

    }

    if (CSP_QSPI_EnableMemoryMappedMode() != HAL_OK) {

        while (1)
            ; //breakpoint - error detected
    }

    for (var = 0; var < SECTORS_COUNT; var++) {
        if (memcmp(buffer_test,
                   (uint8_t*) (0x90000000 + var * MEMORY_SECTOR_SIZE),
                   MEMORY_SECTOR_SIZE) != HAL_OK) {
            while (1)
                ;  //breakpoint - error detected - otherwise QSPI works properly
        }
    }

}

void QuadSpiClean(void)
{
    uint8_t buffer_test[MEMORY_SECTOR_SIZE];
    uint32_t var = 0;

    CSP_QUADSPI_Init();

    for (var = 0; var < MEMORY_SECTOR_SIZE; var++) {
        buffer_test[var] = (var & 0x00);
    }


}

void Scan_I2C_Bus(I2C_HandleTypeDef *hi2c)
{

    HAL_GPIO_WritePin(GPIOB, CTP_Reset_Pin, GPIO_PIN_RESET);  // Pull RESET low
    HAL_Delay(20);                                           // Wait 20ms
    HAL_GPIO_WritePin(GPIOB, CTP_Reset_Pin, GPIO_PIN_SET);    // Release RESET (pull high)
    HAL_Delay(50);

//  HAL_GPIO_WritePin(GPIOD, CTP_Interupt_Pin, GPIO_PIN_SET);    // Release RESET (pull high)
```

```
    HAL_Delay(50);

    for (uint8_t addr = 0; addr < 127; addr++)
    {
        if (HAL_I2C_IsDeviceReady(hi2c, FT5336_I2C_SLAVE_ADDRESS, 2, 10) == HAL_OK)
        {
            char msg[32];
            sprintf(msg, "Device found at 0x%02X\r\n", addr);
        }
    }
}



void Read_FT5426_ChipID(I2C_HandleTypeDef *hi2c)
{
    uint8_t chip_id = 0;
    HAL_GPIO_WritePin(GPIOB, CTP_Reset_Pin, GPIO_PIN_RESET);  // Pull RESET low
    HAL_Delay(10);
    HAL_GPIO_WritePin(GPIOB, CTP_Reset_Pin, GPIO_PIN_SET);    // Release RESET (pull high)
    HAL_Delay(50);

    if (HAL_I2C_Mem_Read(hi2c,
                         FT5336_I2C_SLAVE_ADDRESS,     // STM32 HAL expects 8-bit address (7-bit << 1)
                         FT5336_ID_VALUE,
                          I2C_MEMADD_SIZE_8BIT,
                          &chip_id,
                          2,
                          HAL_MAX_DELAY) == HAL_OK)
    {
        printf("FT5426 Chip ID: 0x%02X\r\n", chip_id);
    }
    else
    {
        printf("Failed to read FT5426 at I2C address 0x%02X\r\n", FT5336_I2C_SLAVE_ADDRESS);
    }
}

/**
  * @brief  Reads multiple data.
  * @param  i2c_handler : I2C handler
  * @param  Addr: I2C address
  * @param  Reg: Reg address
  * @param  MemAddress: Memory address
  * @param  Buffer: Pointer to data buffer
  * @param  Length: Length of the data
  * @retval Number of read data
  */

static HAL_StatusTypeDef I2Cx_ReadMultiple(I2C_HandleTypeDef *i2c_handler,
                                           uint8_t Addr,
                                           uint16_t Reg,
                                           uint16_t MemAddress,
                                           uint8_t *Buffer,
                                           uint16_t Length)
{
  HAL_StatusTypeDef status = HAL_OK;

  status = HAL_I2C_Mem_Read(i2c_handler, Addr, (uint16_t)Reg, MemAddress, Buffer, Length, 1000);

  /* Check the communication status */
  if(status != HAL_OK)
  {
    /* I2C error occurred */
    I2Cx_Error(i2c_handler, Addr);
  }
  return status;
}
```

```
/**
  * @brief  Writes a value in a register of the device through BUS in using DMA mode.
  * @param  i2c_handler : I2C handler
  * @param  Addr: Device address on BUS Bus.
  * @param  Reg: The target register address to write
  * @param  MemAddress: Memory address
  * @param  Buffer: The target register value to be written
  * @param  Length: buffer size to be written
  * @retval HAL status
  */
static HAL_StatusTypeDef I2Cx_WriteMultiple(I2C_HandleTypeDef *i2c_handler,
                                            uint8_t Addr,
                                            uint16_t Reg,
                                            uint16_t MemAddress,
                                            uint8_t *Buffer,
                                            uint16_t Length)
{
  HAL_StatusTypeDef status = HAL_OK;

  status = HAL_I2C_Mem_Write(i2c_handler, Addr, (uint16_t)Reg, MemAddress, Buffer, Length, 1000);

  /* Check the communication status */
  if(status != HAL_OK)
  {
    /* Re-Initiaize the I2C Bus */
    I2Cx_Error(i2c_handler, Addr);
  }
  return status;
}

/**
  * @brief  Checks if target device is ready for communication.
  * @note   This function is used with Memory devices
  * @param  i2c_handler : I2C handler
  * @param  DevAddress: Target device address
  * @param  Trials: Number of trials
  * @retval HAL status
  */
static HAL_StatusTypeDef I2Cx_IsDeviceReady(I2C_HandleTypeDef *i2c_handler, uint16_t DevAddress, uint32_t
    ↪ Trials)
{
  return (HAL_I2C_IsDeviceReady(i2c_handler, DevAddress, Trials, 1000));
}


/**
  * @brief  Manages error callback by re-initializing I2C.
  * @param  i2c_handler : I2C handler
  * @param  Addr: I2C Address
  * @retval None
  */


static void I2Cx_Error(I2C_HandleTypeDef *i2c_handler, uint8_t Addr)
{
  /* De-initialize the I2C communication bus */
  HAL_I2C_DeInit(i2c_handler);

  /* Re-Initialize the I2C communication bus */
  MX_I2C3_Init();
}


/******************************** LINK TOUCHSCREEN *********************************/

/**
  * @brief  Initializes Touchscreen low level.
```

```
  * @retval None
  */
void TS_IO_Init(void)
{
  MX_I2C3_Init();

  HAL_GPIO_WritePin(GPIOB, CTP_Reset_Pin, GPIO_PIN_RESET);  // Pull RESET low
  HAL_Delay(10);
  HAL_GPIO_WritePin(GPIOB, CTP_Reset_Pin, GPIO_PIN_SET);    // Release RESET (pull high)
  HAL_Delay(50);

}

/**
  * @brief  Writes a single data.
  * @param  Addr: I2C address
  * @param  Reg: Reg address
  * @param  Value: Data to be written
  * @retval None
  */
void TS_IO_Write(uint8_t Addr, uint8_t Reg, uint8_t Value)
{
  I2Cx_WriteMultiple(&hi2c3, Addr, (uint16_t)Reg, I2C_MEMADD_SIZE_8BIT,(uint8_t*)&Value, 1);
}

/**
  * @brief  Reads a single data.
  * @param  Addr: I2C address
  * @param  Reg: Reg address
  * @retval Data to be read
  */
uint8_t TS_IO_Read(uint8_t Addr, uint8_t Reg)
{
  uint8_t read_value = 0;

  I2Cx_ReadMultiple(&hi2c3, Addr, Reg, I2C_MEMADD_SIZE_8BIT, (uint8_t*)&read_value, 1);

  return read_value;
}



/**
  * @brief  TS delay
  * @param  Delay: Delay in ms
  * @retval None
  */
void TS_IO_Delay(uint32_t Delay)
{
  HAL_Delay(Delay);
}


bool read_xy(void)
{

    uint16_t x;
    uint16_t y;

    TS_StateTypeDef state;
    BSP_TS_GetState(&state);
    if (state.touchDetected)
    {
        x = state.touchX[0];
        y = state.touchY[0];
        return true;
    }
```

```
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{

    if (GPIO_Pin == Next_Plant_Pin)
    {
        // <-- replace with your button pin define
        BaseType_t xHigherPriorityTaskWoken = pdFALSE;
        vTaskNotifyGiveFromISR(nextPlant_tskHandle, &xHigherPriorityTaskWoken);
        portYIELD_FROM_ISR(xHigherPriorityTaskWoken);

    }

    if (GPIO_Pin == Next_Region_Pin)
       {
        // <-- replace with your button pin define
        BaseType_t xHigherPriorityTaskWoken = pdFALSE;
        vTaskNotifyGiveFromISR(next_RegionHandle, &xHigherPriorityTaskWoken);
        portYIELD_FROM_ISR(xHigherPriorityTaskWoken);

       }

    if (GPIO_Pin ==CTP_Interrupt_Pin)
         {
            // <-- replace with your button pin define
//          BaseType_t xHigherPriorityTaskWoken = pdFALSE;
//          vTaskNotifyGiveFromISR(next_RegionHandle, &xHigherPriorityTaskWoken);
//          portYIELD_FROM_ISR(xHigherPriorityTaskWoken);

         }


    if (GPIO_Pin == LCD_Boot_Pin)
       {
        // <-- replace with your button pin define
        BaseType_t xHigherPriorityTaskWoken = pdFALSE;
        vTaskNotifyGiveFromISR(start_Disp_LEDSHandle, &xHigherPriorityTaskWoken);
        portYIELD_FROM_ISR(xHigherPriorityTaskWoken);

       }

    if (GPIO_Pin ==Transmission_Complete_Pin)
      {


        if(HAL_GPIO_ReadPin(GPIOB,Transmission_Complete_Pin) == GPIO_PIN_RESET)
        {

             data_available_flag = 0;

        }
    //


     }


//
//    if (GPIO_Pin == SPI_NSS_Pin) {
//         GPIO_PinState cs = HAL_GPIO_ReadPin(GPIOA, SPI_NSS_Pin);
//         if (cs == GPIO_PIN_RESET) {
//             // NSS asserted low
//             SPI_Slave_ArmDMA();
//         } else {
//             // NSS deasserted high
//             SPI_Slave_AbortIfNeeded();
//         }
```

```
//     }
}




// ---- HAL callbacks ----



//void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef *hspi)
//{
//    if (hspi == &hspi1) {
//        BaseType_t w = pdFALSE;
//        xSemaphoreGiveFromISR(SpiTxDoneHandle, &w);
//        portYIELD_FROM_ISR(w);
//        HAL_SPI_Receive_DMA(&hspi1, spiRxBuffer, 300);
//    }
//}

void   HAL_SPI_TxRxCpltCallback(SPI_HandleTypeDef *h)
{

    if (h == &hspi1) {
        BaseType_t w = pdFALSE;
        xSemaphoreGiveFromISR(SpiTxDoneHandle, &w);
        portYIELD_FROM_ISR(w)
        HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_SET); // CS high
        spi_done = true;
//         process rx buffer here or set another flag
    }
}

void   HAL_SPI_RxCpltCallback(SPI_HandleTypeDef *h)
{

    if (h == &hspi1) {

        HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_SET); // CS high
//         process rx buffer here or set another flag
    }
}


void debug_verify_weights(void) {
    // 1. Get the pointer to the weights (usually the address in External Flash)
    ai_handle weights_handle = ai_network_data_weights_get();
    uint8_t* weights_ptr = (uint8_t*)weights_handle;

    printf("--- Weights Debug ---\n");
    printf("Weights Address: %p\n", (void*)weights_ptr);

    // 2. Print the first 16 bytes to check for data
    printf("First 16 bytes: ");
    for (int i = 0; i < 16; i++) {
        printf("%02X ", weights_ptr[i]);
    }
    printf("\n---------------------\n");

    // 3. Simple Check: If all are 0xFF, the flash is empty.
    if (weights_ptr[0] == 0xFF && weights_ptr[1] == 0xFF) {
        printf("WARNING: Weights look empty! (All 0xFF)\n");
    }
}
```

```
//void HAL_SPI_TxRxCpltCallback(SPI_HandleTypeDef *hspi)
//{
//
//    if (hspi == &hspi1) {
//
//    }
//}

int _write(int file, char *ptr, int len)
{
    (void)file;
    for (int i = 0; i < len; i++)
    {
        ITM_SendChar((uint32_t)ptr[i]);  // sends via SWV/ITM
    }
    return len;
}


/* USER CODE END 4 */

 /* MPU Configuration */

void MPU_Config(void)
{
  MPU_Region_InitTypeDef MPU_InitStruct = {0};

  /* Disables the MPU */
  HAL_MPU_Disable();

  /** Initializes and configures the Region and the memory to be protected
  */
  MPU_InitStruct.Enable = MPU_REGION_ENABLE;
  MPU_InitStruct.Number = MPU_REGION_NUMBER0;
  MPU_InitStruct.BaseAddress = 0xC0000000;
  MPU_InitStruct.Size = MPU_REGION_SIZE_32MB;
  MPU_InitStruct.SubRegionDisable = 0x00;
  MPU_InitStruct.TypeExtField = MPU_TEX_LEVEL0;
  MPU_InitStruct.AccessPermission = MPU_REGION_FULL_ACCESS;
  MPU_InitStruct.DisableExec = MPU_INSTRUCTION_ACCESS_DISABLE;
  MPU_InitStruct.IsShareable = MPU_ACCESS_NOT_SHAREABLE;
  MPU_InitStruct.IsCacheable = MPU_ACCESS_CACHEABLE;
  MPU_InitStruct.IsBufferable = MPU_ACCESS_NOT_BUFFERABLE;

  HAL_MPU_ConfigRegion(&MPU_InitStruct);

  /** Initializes and configures the Region and the memory to be protected
  */
  MPU_InitStruct.Number = MPU_REGION_NUMBER1;
  MPU_InitStruct.Size = MPU_REGION_SIZE_1MB;
  MPU_InitStruct.SubRegionDisable = 0x0;
  MPU_InitStruct.IsCacheable = MPU_ACCESS_NOT_CACHEABLE;

  HAL_MPU_ConfigRegion(&MPU_InitStruct);

  /** Initializes and configures the Region and the memory to be protected
  */
  MPU_InitStruct.Number = MPU_REGION_NUMBER3;
  MPU_InitStruct.BaseAddress = 0x90000000;
  MPU_InitStruct.Size = MPU_REGION_SIZE_16MB;
  MPU_InitStruct.TypeExtField = MPU_TEX_LEVEL1;

  HAL_MPU_ConfigRegion(&MPU_InitStruct);
  /* Enables the MPU */
  HAL_MPU_Enable(MPU_HFNMI_PRIVDEF);

}
```

```
/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

File: TouchGFX/gui/include/gui/model/Model.hpp

```
#ifndef MODEL_HPP
#define MODEL_HPP
#define MAX_MEMSET_RETRIES  5

class ModelListener;

#include "stdint.h"

typedef struct {
    uint8_t command;
    uint8_t plant_Id;

}send_data_via_spi_Params;




class Model
{
public:
    Model();

    void bind(ModelListener* listener)
    {
        modelListener = listener;
    }

    void tick();


    int sendDataViaSPI(send_data_via_spi_Params);
```

```
    void reset_recieved_bytes(void);
    void reset_transmit_bytes(void);

    void setBusy(bool busy) { isBusy = busy; }
    bool getBusy() const    { return isBusy; }




private:
    bool isBusy = false;

protected:
    ModelListener* modelListener;
};


#endif // MODEL_HPP
```

File: TouchGFX/gui/src/model/Model.cpp

```
#include <gui/model/Model.hpp>
#include <gui/model/ModelListener.hpp>

#ifndef SIMULATOR
#include "main.h"          // for hspi1 and HAL types



#include <cstdint>        // for uint8_t

#include "FreeRTOS.h"
#include "shared.h"
#include "semphr.h"
#include "stdbool.h"
#include "cmsis_os2.h"
#include "task.h"
#include <string.h>


#define XFER_LEN 500

extern SPI_HandleTypeDef hspi1;
extern uint8_t txBuf[XFER_LEN];
extern uint8_t rxDummy[XFER_LEN];
extern bool spi_done;
extern osSemaphoreId_t SpiTxDoneHandle;
extern  uint8_t data_available_flag;




#endif
```

```
//extern SPI_HandleTypeDef hspi1;



Model::Model() : modelListener(0)
{

}

void Model::tick()
{


//  Model::sendDataViaSPI(1);


    HAL_StatusTypeDef status;


    if(rxDummy[231] == 0xBB)
    {

//      HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_RESET); // CS low
//
//      HAL_SPI_Abort(&hspi1);
//      HAL_Delay(60);
//
//      status = HAL_SPI_TransmitReceive_DMA(&hspi1, txBuf, rxDummy, XFER_LEN);
//
//
//      if (status == HAL_OK)
//      {
//          // Wait for DMA to finish
////            xSemaphoreTake(SpiTxDoneHandle, portMAX_DELAY);
//
//
//      }
//      else
//      {
//          // Handle error
//
//      }
//      Model::reset_tr_bytes();


        send_data_via_spi_Params Params{};

        Params.command = 0x08;
        modelListener->updateFromSpi(rxDummy, 500);



//      data_available_flag = 0;
//      Params.command = 0x05;
//      FramePacket_t data_parket;
////            memcpy(&data_parket, rxDummy, sizeof(data_parket));
//      modelListener->updateFromSpi(rxDummy, 500);
//
//      Model::sendDataViaSPI(Params);


////        modelListener->updateFromSpi(rxDummy, 500);
//      Params.command = 0x08;
//      FramePacket_t data_parket;
////            memcpy(&data_parket, rxDummy, sizeof(data_parket));
//      modelListener->updateFromSpi(rxDummy, 500);
```

```
//      if(rxDummy[231] == 0xBB)
//      {
//
//          /**
//           * 0xBB = Scan complete
//           * 0xB1 = new plant
//           * 0xB2 = new region
//           * 0xB3 = save
//           */
//
//          Model::reset_transmit_bytes();
//          Params.command = 0x08;
////            FramePacket_t data_parket;
////            memcpy(&data_parket, rxDummy, sizeof(data_parket));
//          modelListener->updateFromSpi(rxDummy, 500);
//          Model::reset_recieved_bytes();
//
//
////            __HAL_SPI_DISABLE(&hspi1);
////            __HAL_SPI_ENABLE(&hspi1);
//          Model::sendDataViaSPI(Params);
////            Model::reset_recieved_bytes();
//
//
////            modelListener->setBusy(false);
//      }
//
//
//      else if(rxDummy[231] == 0xB1)
//      {
//
//          /**
//           * 0xBB = Scan complete
//           * 0xB1 = new plant
//           * 0xB2 = new region
//           * 0xB3 = save
//           */
//
//          Model::reset_transmit_bytes();
//          Params.command = 0x08;
////            FramePacket_t data_parket;
////            memcpy(&data_parket, rxDummy, sizeof(data_parket));
//          modelListener->plant_created();
//          Model::reset_recieved_bytes();
//
//
////            __HAL_SPI_DISABLE(&hspi1);
////            __HAL_SPI_ENABLE(&hspi1);
//          Model::sendDataViaSPI(Params);
////            Model::reset_recieved_bytes();
//
//
////            modelListener->setBusy(false);
//      }
//
//
//      else if(rxDummy[231] == 0xB2)
//      {
//
//          /**
//           * 0xBB = Scan complete
//           * 0xB1 = new plant
//           * 0xB2 = new region
//           * 0xB3 = save
//           */
```

```
//
//          Model::reset_transmit_bytes();
//          Params.command = 0x08;
////            FramePacket_t data_parket;
////            memcpy(&data_parket, rxDummy, sizeof(data_parket));
//          modelListener->region_created();
//          Model::reset_recieved_bytes();
//
//
////            __HAL_SPI_DISABLE(&hspi1);
////            __HAL_SPI_ENABLE(&hspi1);
//          Model::sendDataViaSPI(Params);
////            Model::reset_recieved_bytes();
//
//
////            modelListener->setBusy(false);
//      }
//
//
//      else if(rxDummy[231] == 0xB3)
//      {
//
//          /**
//           * 0xBB = Scan complete
//           * 0xB1 = new plant
//           * 0xB2 = new region
//           * 0xB3 = save
//           */
//
//          Model::reset_transmit_bytes();
//          Params.command = 0x08;
////            FramePacket_t data_parket;
////            memcpy(&data_parket, rxDummy, sizeof(data_parket));
//          modelListener->save();
//          Model::reset_recieved_bytes();
//
//
////            __HAL_SPI_DISABLE(&hspi1);
////            __HAL_SPI_ENABLE(&hspi1);
//          Model::sendDataViaSPI(Params);
////            Model::reset_recieved_bytes();
//
//
////            modelListener->setBusy(false);
//      }


    }



//  rxDummy[201] = 0;









}
```

```
int Model::sendDataViaSPI(send_data_via_spi_Params P)
{

#ifndef SIMULATOR
//    uint8_t data[] = {0x00};
    HAL_StatusTypeDef status;

//    FramePacket_t rxBuf[500];

    /*
     * Send command over spi to stop scanning
     *
     *
     *
     */



    if (HAL_SPI_GetState(&hspi1) == HAL_SPI_STATE_READY)
    {

        if(P.command == 0 )
        {

            txBuf[0] = 0x01; // indicate a command is present
            txBuf[1] = 0x00; // what command is present
            txBuf[2] = 0x00; // Should move or not move to a new plant (1 == new plant, 0 == not new plant
            txBuf[3] = 0x00; // should change or not change regioon

            HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_RESET); // CS low

            status = HAL_SPI_TransmitReceive_DMA(&hspi1, txBuf, rxDummy, XFER_LEN);

            if (status == HAL_OK)
            {
                // Wait for DMA to finish
    //          xSemaphoreTake(SpiTxDoneHandle, portMAX_DELAY);

                return 1;
            }
            else
            {
                // Handle error
                return 0;
            }

        }

        /*
         * Send command over spi to start scanning
         */
        else if(P.command == 1)
        {
            txBuf[0] = 0x01; // indicate a command is present
            txBuf[1] = 0x01; // what command is present
            txBuf[2] = 0x01; // Should move or not move to a new plant (1 == new plant, 0 == not new plant
            txBuf[3] = 0x00; // should change or not change region

            HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_RESET); // CS low

            status = HAL_SPI_TransmitReceive_DMA(&hspi1, txBuf, rxDummy, XFER_LEN);

            if (status == HAL_OK)
            {
                // Wait for DMA to finish
    //          xSemaphoreTake(SpiTxDoneHandle, portMAX_DELAY);
```

```
            return 1;
        }
        else
        {
            // Handle error
            return 0;
        }

    }

    /*
     * Send command over spi to move to next plant
     */

    else if(P.command == 2)
    {
        txBuf[0] = 0x01; // indicate a command is present
        txBuf[1] = 0x02; // what command is present
        txBuf[2] = 0x00; // Should move or not move to a new plant (1 == new plant, 0 == not new plant
        txBuf[3] = 0x00; // should change or not change region

        HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_RESET); // CS low

        status = HAL_SPI_TransmitReceive_DMA(&hspi1, txBuf, rxDummy, XFER_LEN);

        if (status == HAL_OK)
        {
            // Wait for DMA to finish
//          xSemaphoreTake(SpiTxDoneHandle, portMAX_DELAY);
            return 1;
        }
        else
        {
            // Handle error
            return 0;
        }

    }

    /*
     * Send command over spi to save and maove to next region
     */


    else if(P.command == 3)
    {
        txBuf[0] = 0x01; // indicate a command is present
        txBuf[1] = 0x03; // what command is present (new replicate command)
        txBuf[2] = 0x00;
        txBuf[3] = 0x00; //

        HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_RESET); // CS low

        status = HAL_SPI_TransmitReceive_DMA(&hspi1, txBuf, rxDummy, XFER_LEN);

        if (status == HAL_OK)
        {
            // Wait for DMA to finish
//          xSemaphoreTake(SpiTxDoneHandle, portMAX_DELAY);
            return 1;
        }
        else
        {
            // Handle error
            return 0;
        }

    }
```

```
        /*
         * Send command over spi to save and mave to next plant
        */


        else if(P.command == 4)
        {
            txBuf[0] = 0x01; // indicate a command is present
            txBuf[1] = 0x01; // what command is present
            txBuf[2] = 0x01; // Should move or not move to a new plant (1 == new plant, 0 == not new plant
            txBuf[3] = 0x00; // should change or not change region

            HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_RESET); // CS low

            status = HAL_SPI_TransmitReceive_DMA(&hspi1, txBuf, rxDummy, XFER_LEN);

            if (status == HAL_OK)
            {
                // Wait for DMA to finish
    //          xSemaphoreTake(SpiTxDoneHandle, portMAX_DELAY);
                return 1;
            }
            else
            {
                // Handle error
                return 0;
            }

        }

        /*
         * Idle message package to receive scanning results
         */
        else if(P.command == 5)
        {
            txBuf[0] = 0x05;
            txBuf[1] = 0x00;
            txBuf[2] = 0x00;
            txBuf[3] = 0x00;

//          txBuf[5] = 0xBB;
            HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_RESET); // CS low

            status = HAL_SPI_TransmitReceive_DMA(&hspi1, txBuf, rxDummy, XFER_LEN);

            if (status == HAL_OK)
            {
                // Wait for DMA to finish
    //          xSemaphoreTake(SpiTxDoneHandle, portMAX_DELAY);
                return 1;
            }
            else
            {
                // Handle error
                return 0;
            }

        }

        /*
         * scan without creating a new plant
         */
        else if(P.command == 6)
        {
            txBuf[0] = 0x01;
            txBuf[1] = 0x01;
            txBuf[2] = 0x00;
```

```
            txBuf[3] = 0x00;
            HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_RESET); // CS low

            status = HAL_SPI_TransmitReceive_DMA(&hspi1, txBuf, rxDummy, XFER_LEN);

            if (status == HAL_OK)
            {
                // Wait for DMA to finish
    //          xSemaphoreTake(SpiTxDoneHandle, portMAX_DELAY);
                return 1;
            }
            else
            {
                // Handle error
                return 0;
            }

            }

        else if(P.command == 7)
        {
            txBuf[0] = 0x01; // indicate a command is present
            txBuf[1] = 0x02; // what command is present
            txBuf[2] = 0x00; // Should move or not move to a new plant (1 == new plant, 0 == not new plant
            txBuf[3] = 0x00; // should change or not change region

            txBuf[4] = P.plant_Id;


            HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_RESET); // CS low

            status = HAL_SPI_TransmitReceive_DMA(&hspi1, txBuf, rxDummy, XFER_LEN);

            if (status == HAL_OK)
            {
                // Wait for DMA to finish
    //          xSemaphoreTake(SpiTxDoneHandle, portMAX_DELAY);
                return 1;
            }
            else
            {
                // Handle error
                return 0;
            }

        }


        else if(P.command == 8)
        {
            txBuf[0] = 0x00; // indicate a command is present
            txBuf[1] = 0x00; // what command is present
            txBuf[2] = 0x00; // Should move or not move to a new plant (1 == new plant, 0 == not new plant
            txBuf[3] = 0x00; // should change or not change region
            txBuf[7] = 0x20;

//          txBuf[4] = P.plant_Id;



            HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_RESET); // CS low

            status = HAL_SPI_TransmitReceive_DMA(&hspi1, txBuf, rxDummy, XFER_LEN);

            if (status == HAL_OK)
            {
                // Wait for DMA to finish
    //          xSemaphoreTake(SpiTxDoneHandle, portMAX_DELAY);
```

```
                data_available_flag = 1;
//              Model::reset_transmit_bytes();
//              Model::reset_recieved_bytes();
                return 1;
            }
            else
            {
                // Handle error
                return 0;
            }

        }


        else if(P.command == 0x09)
        {
            txBuf[0] = 0x06; // indicate a command is present
            txBuf[1] = 0x00; // what command is present
            txBuf[2] = 0x00; // Should move or not move to a new plant (1 == new plant, 0 == not new plant
            txBuf[3] = 0x00; // should change or not change region
            txBuf[7] = 0x30;

            txBuf[4] = P.plant_Id;


            HAL_GPIO_WritePin(GPIOA, SPI_NSS_Pin, GPIO_PIN_RESET); // CS low

            status = HAL_SPI_TransmitReceive_DMA(&hspi1, txBuf, rxDummy, XFER_LEN);

            if (status == HAL_OK)
            {
                // Wait for DMA to finish
    //          xSemaphoreTake(SpiTxDoneHandle, portMAX_DELAY);
                return 1;
            }
            else
            {
                // Handle error
                return 0;
            }

        }




    }
//    HAL_StatusTypeDef status = HAL_SPI_Transmit(&hspi1, data, sizeof(data), HAL_MAX_DELAY);

//    if (status == HAL_OK)
//    {
//        // Transmission successful
//    }
//    else
//    {
//        // Handle error
//    }

#endif

#ifdef SIMULATOR
    return 0;
#endif
}


void Model::reset_transmit_bytes(void)
```

```
{

    for (int attempt = 0; attempt < MAX_MEMSET_RETRIES; attempt++)
    {
        memset(txBuf, 0xAA, sizeof(txBuf));

        bool ok = true;
        for (size_t i = 0; i < sizeof(txBuf); i++)
        {
            if (txBuf[i] != 0xAA)
            {
                ok = false;
                break;
            }
        }

        if (ok)
        {
//          dcache_clean_by_addr_safe(rxDummy, sizeof(rxDummy));

            return;  // success
        }
    }


    // Optional: error handling if memory could not be set
    // e.g. set error flag, assert, log, or reset system
}


void Model::reset_recieved_bytes(void)
{

    for (int attempt = 0; attempt < MAX_MEMSET_RETRIES; attempt++)
    {
        memset(rxDummy, 0xAA, sizeof(rxDummy));

        bool ok = true;
        for (size_t i = 0; i < sizeof(rxDummy); i++)
        {
            if (rxDummy[i] != 0xAA)
            {
                ok = false;
                break;
            }
        }

        if (ok)
        {
//          dcache_clean_by_addr_safe(rxDummy, sizeof(rxDummy));

            return;  // success
        }
    }


    // Optional: error handling if memory could not be set
    // e.g. set error flag, assert, log, or reset system
}

//void Model::reset_recieved_bytes()
//{
//  memset(rxDummy,0xAA,sizeof(rxDummy));
//
//}
```

File: TouchGFX/gui/include/gui/model/ModelListener.hpp

```
#ifndef MODELLISTENER_HPP
#define MODELLISTENER_HPP

#include <gui/model/Model.hpp>

#ifndef SIMULATOR
#include "main.h"          // for hspi1 and HAL types
#endif


class ModelListener
{
public:
    ModelListener() : model(0) {}

    virtual ~ModelListener() {}

    void bind(Model* m)
    {
        model = m;
    }

    virtual void updateFromSpi(uint8_t* data, size_t len) {};
    virtual void plant_created(void) {};
    virtual void region_created(void) {};
    virtual void save(void) {};
    void setBusy(bool busy);

    virtual void operationSuccess(bool ok) {}

protected:
    Model* model;
};

#endif // MODELLISTENER_HPP
```

File: TouchGFX/gui/include/gui/screen_screen/screenView.hpp

```
#ifndef SCREENVIEW_HPP
#define SCREENVIEW_HPP

#include <gui_generated/screen_screen/screenViewBase.hpp>
#include <gui/screen_screen/screenPresenter.hpp>
#include <gui/common/CustomKeyboard.hpp>

class screenView : public screenViewBase
{
public:
    screenView();
    virtual ~screenView() {}
    virtual void setupScreen();
    virtual void tearDownScreen();


    virtual void scanPlant() override;

    virtual void calibration() override;

    void update_View_FromSpi(uint8_t *data,  size_t len);

    virtual void flexButtonExit() override;
    virtual void flexButtonOk() override;
```

```
    void setBusy(bool busy);

    void showProcessingOverlay();

    void hideProcessingOverlay();




    CustomKeyboard Keyboard;
protected:
};

#endif // SCREENVIEW_HPP
```

File: TouchGFX/gui/src/screen_screen/screenView.cpp

```
#include <gui/screen_screen/screenView.hpp>
#include <stdio.h>

#ifndef SIMULATOR

#include "ai_platform.h"
#include "ai_datatypes_defines.h"
#include "network_data_params.h"
#include "network.h"
#include <stdlib.h>




extern uint8_t txBuf[XFER_LEN];
extern uint8_t data_available_flag;

extern ai_handle network;
extern ai_buffer *ai_input;
extern ai_buffer *ai_output;
extern ai_u32 is_create_done;

extern uint8_t plant_Id;



//static ai_float input_data[AI_NETWORK_IN_1_SIZE];
//static ai_float output_data[AI_NETWORK_OUT_1_SIZE];

__attribute__((aligned(32))) static float input_data[AI_NETWORK_IN_1_SIZE];
__attribute__((aligned(32))) static float output_data[AI_NETWORK_OUT_1_SIZE];

#endif

screenView::screenView()
{

    Keyboard.setPosition(80,16,320,240);
    add(Keyboard);
    Keyboard.setVisible(false);
    Keyboard.invalidate();

}

void screenView::setupScreen()
{

    screenViewBase::setupScreen();
```

```
}

void screenView::tearDownScreen()
{
    screenViewBase::tearDownScreen();
}

void screenView::scanPlant(){

#ifndef SIMULATOR

    int state = presenter->onScanPlant();
#endif

}

void screenView::calibration()
{
    presenter->setBusy(true);

    Keyboard.setVisible(true); //show keyboard
    Keyboard.invalidate();
    flexButton1Ok.setVisible(true);
    flexButton1Ok.invalidate();
    flexButton2Exit.setVisible(true);
    flexButton2Exit.invalidate();

    buttonWithLabel1.setVisible(false);
    buttonWithLabel1.invalidate();
    Calibrate.setVisible(false);
    Calibrate.invalidate();



}

void screenView::flexButtonExit()
{

    Keyboard.setVisible(false); //show keyboard
    Keyboard.invalidate();
    flexButton1Ok.setVisible(false);
    flexButton1Ok.invalidate();
    flexButton2Exit.setVisible(false);
    flexButton2Exit.invalidate();

    buttonWithLabel1.setVisible(true);
    buttonWithLabel1.invalidate();
    Calibrate.setVisible(true);
    Calibrate.invalidate();

    presenter->setBusy(false);



}

void screenView::flexButtonOk()
{

    Keyboard.setVisible(false); //show keyboard
    Keyboard.invalidate();
    flexButton1Ok.setVisible(false);
    flexButton1Ok.invalidate();
    flexButton2Exit.setVisible(false);
    flexButton2Exit.invalidate();
```

```
    buttonWithLabel1.setVisible(true);
    buttonWithLabel1.invalidate();
    Calibrate.setVisible(true);
    Calibrate.invalidate();


//  touchgfx::Unicode::UnicodeChar* u16 = Keyboard.getBuffer();

//  Unicode::strncpy(Plant_NoBuffer, Keyboard.getBuffer(), PLANT_NO_SIZE);
//  Plant_No.setWildcard(Plant_NoBuffer);
//  Plant_No.invalidate();

    touchgfx::Unicode::UnicodeChar* u16 = Keyboard.getBuffer();


    char buf[32];
    touchgfx::Unicode::toUTF8(u16, (uint8_t*)buf, sizeof(buf)); // makes a null-terminated UTF-8 string

    char* endptr = nullptr;
    plant_Id = (int)strtol(buf, &endptr, 10);

    if (*endptr != '\0')
    {
        // not a pure number (contains invalid characters)
    }

    presenter->setBusy(false);
//  presenter->newPlant(plant_Id);
    int state = presenter->calibration(plant_Id);

}

void screenView::update_View_FromSpi(uint8_t *data,  size_t len)
{

#ifndef SIMULATOR
    FramePacket_t data_parket;
    memcpy(&data_parket, data, sizeof(data_parket));
    float* results;

    if (data_parket.header.syncWord != 0xAA55) {
        // bad SOF / out of sync -- resync your stream
        return;
    }


    for (int i = 0; i < 7; i++)            // <--- not <=
    {

        input_data[i] = (float)data_parket.data[2][i].gain;

    }

//  input_data[0] = (float) 10.0;
//  input_data[1] = (float)10.0;
//  input_data[2] = (float)10.0;
//  input_data[3] = (float)10.0;
//  input_data[4] = (float)10.0;
//  input_data[5] = (float)10.0;



    ai_input = ai_network_inputs_get(network, NULL);
    ai_output = ai_network_outputs_get(network, NULL);
    ai_input[0].data = AI_HANDLE_PTR(input_data);
    ai_output[0].data = AI_HANDLE_PTR(output_data);
```

```
    // 3. Perform the inference!
    SCB_CleanDCache_by_Addr((uint32_t*)input_data,  sizeof(input_data));

    for (int i = 0; i < 3; i++) output_data[i] = -10.0;

    // Clean input (and optionally output sentinel)
    SCB_CleanDCache_by_Addr((uint32_t*)input_data,  sizeof(input_data));
    SCB_CleanDCache_by_Addr((uint32_t*)output_data, sizeof(output_data));


    ai_i32 n_batch = ai_network_run(network, ai_input, ai_output);
    results = (float*)(ai_output[0].data);

//  float class_1 = (float)results[0];
//  float class_2 = (float)results[1];
//  float class_3 = (float)results[2];

//  SCB_InvalidateDCache_by_Addr((uint32_t*)output_data, sizeof(output_data));
//
//  if (n_batch == 1) {
//      // 2. Access data via the ai_output handle itself
//      // This ensures you are reading from the memory location the network just wrote to
//      results = (float*)(ai_output[0].data);
//
//      float class_1 = (float)results[0];
//      float class_2 = (float)results[1];
//      float class_3 = (float)results[2];
//
//      // Optional: Print to verify
//      printf("Output: %.6f, %.6f, %.6f\n", class_1, class_2, class_3);
//  }

    // Prepare arrays that map each "class" text area to its backing Unicode buffer and buffer size.
    // Then loop through the classes, format each result as a string with 3 decimal places,
    // copy it into the corresponding TouchGFX Unicode buffer, set it as the wildcard text for
    // the text area, optionally resize to fit, and invalidate the widget so it redraws on screen.


    touchgfx::Unicode::UnicodeChar* classBuffers[3] = {
        class1Buffer, class2Buffer, class3Buffer
    };

    static const uint16_t classSizes[3] = {
        CLASS1_SIZE, CLASS2_SIZE, CLASS3_SIZE
    };

    touchgfx::TextAreaWithOneWildcard*   class_text_areas[3] = {
            &class1, &class2, &class3
    };

    for (int i = 0; i < 3; i++)
    {

        // Write formatted float directly into the TouchGFX Unicode buffer
        char buf[5];
        snprintf(buf, sizeof buf, "%.2f", (float)results[i]);
        Unicode::strncpy(classBuffers[i], buf,  classSizes[i]);
        class_text_areas[i]->setWildcard(classBuffers[i]);
        class_text_areas[i]->resizeToCurrentText();   // optional: auto-size to fit text
        class_text_areas[i]->invalidate();            // request redraw
    }



//  char();
    ai_output[0].data;
#endif
```

```
}

void screenView::setBusy(bool busy)
{
    if (busy)
        showProcessingOverlay();
    else
        hideProcessingOverlay();
}

void screenView::showProcessingOverlay()
{
//  Unicode::strncpy(scanCompleteBuffer, "Processing...", SCANCOMPLETE_SIZE);
    processingOverlay1.setVisible(true);
    processingOverlay1.invalidate();
    buttonWithLabel1.setTouchable(false);
    buttonWithLabel1.invalidate();
    Calibrate.setTouchable(false);
    Calibrate.invalidate();
//  new_plant.setTouchable(false);


}

void screenView::hideProcessingOverlay()
{
    processingOverlay1.setVisible(false);
    processingOverlay1.invalidate();
    buttonWithLabel1.setTouchable(true);
    Calibrate.setTouchable(true);
    Calibrate.invalidate();

//  flexButton1Ok.setTouchable(true);
//  flexButton2Exit.setTouchable(true);
}
```

File: TouchGFX/gui/include/gui/screen_screen/screenPresenter.hpp

---

```
#ifndef SCREENPRESENTER_HPP
#define SCREENPRESENTER_HPP

#include <gui/model/ModelListener.hpp>
#include <mvp/Presenter.hpp>

using namespace touchgfx;

class screenView;

class screenPresenter : public touchgfx::Presenter, public ModelListener
{
public:
    screenPresenter(screenView& v);

    /**
     * The activate function is called automatically when this screen is "switched in"
     * (ie. made active). Initialization logic can be placed here.
     */
    virtual void activate();

    /**
     * The deactivate function is called automatically when this screen is "switched out"
     * (ie. made inactive). Teardown functionality can be placed here.
     */
    virtual void deactivate();
```

```
    virtual ~screenPresenter() {}

    int onScanPlant();

    void updateFromSpi(uint8_t* data, size_t len);
    int calibration(uint8_t plant_id);
    int newPlant(uint8_t Plant_No);

    void setBusy(bool busy);



private:
    screenPresenter();

    screenView& view;
};

#endif // SCREENPRESENTER_HPP
```

File: TouchGFX/gui/src/screen_screen/screenPresenter.cpp

```
#include <gui/screen_screen/screenView.hpp>
#include <gui/screen_screen/screenPresenter.hpp>

screenPresenter::screenPresenter(screenView& v)
    : view(v)
{

}

void screenPresenter::activate()
{

}

void screenPresenter::deactivate()
{

}

int screenPresenter::onScanPlant()
{

    send_data_via_spi_Params Params{};

    Params.command = 0x00;

    int state = model->sendDataViaSPI(Params);
//  int state = model->sendDataViaSPI(0x01);
    return state;
//  model->sendDataViaSPI(0x05);

}

int screenPresenter::calibration(uint8_t plant_id)
{
    send_data_via_spi_Params Params{};

    Params.command = 0x09;
    Params.plant_Id = plant_id;

    int state = model->sendDataViaSPI(Params);
//  int state = model->sendDataViaSPI(0x01);
    return state;
}
```

```
int screenPresenter::newPlant(uint8_t Plant_No)
{

    send_data_via_spi_Params Params{};

    Params.command = 0x07;
    Params.plant_Id = Plant_No;

    int state = model->sendDataViaSPI(Params);

//  int state = model->sendDataViaSPI(0x07);
    return state;

//  model->sendDataViaSPI(0x07);


}

void screenPresenter::updateFromSpi(uint8_t* data, size_t len)
{
    view.update_View_FromSpi(data, len); // Pass it to the View
}

void screenPresenter::setBusy(bool busy)
{
    model->setBusy(busy);
    view.setBusy(busy);
}
```

File: TouchGFX/gui/include/gui/screen2_screen/Screen2View.hpp

---

```
#ifndef SCREEN2VIEW_HPP
#define SCREEN2VIEW_HPP

#include <gui_generated/screen2_screen/Screen2ViewBase.hpp>
#include <gui/screen2_screen/Screen2Presenter.hpp>
#include <gui/common/CustomKeyboard.hpp>


class Screen2View : public Screen2ViewBase
{
public:
    Screen2View();
    virtual ~Screen2View() {}
    virtual void setupScreen();
    virtual void tearDownScreen();


    void update_View_FromSpi(uint8_t *data,  size_t len);

    void save();
    void plant_created();
    void region_created();

    virtual void save_Plant() override;

    virtual void Next_Region() override;

    virtual void scan() override;

//    virtual void handleTickEvent();

    virtual void newPlant() override;
```

```
    virtual void plant_name_input() override;

    virtual void flexButtonExit() override;
    virtual void flexButtonOk() override;

    bool pendingNewPlant = false;
    uint8_t pendingNewPlantTicks = 0;


    virtual void handleTickEvent() override;




    uint8_t tickCounter;

    CustomKeyboard Keyboard;

//    virtual void handleTickEvent();

    void showProcessingOverlay();

    void hideProcessingOverlay();

    void setBusy(bool busy);

    void operationSuccess(bool success);


protected:
};

#endif // SCREEN2VIEW_HPP
```

File: TouchGFX/gui/src/screen2_screen/Screen2View.cpp

```
#include <gui/screen2_screen/Screen2View.hpp>
#include <stdio.h>
#include <stdlib.h>


extern uint8_t txBuf[XFER_LEN];
extern uint8_t data_available_flag;
extern uint8_t rxDummy[XFER_LEN];


uint8_t plant_Id;



Screen2View::Screen2View()
{

    Keyboard.setPosition(80,16,320,240);
    add(Keyboard);
    Keyboard.setVisible(false);
    Keyboard.invalidate();

//    if (data_available_flag == 1)
//    {
//
//      Unicode::strncpy(scanCompleteBuffer, "Processing...", SCANCOMPLETE_SIZE);
//      scanComplete.setWildcard(scanCompleteBuffer);
//      scanComplete.resizeToCurrentText();
//      scanComplete.invalidate();
//
```

```
//    }
//
//    else if (data_available_flag == 0)
//    {
//      Unicode::strncpy(scanCompleteBuffer, "Finished...", SCANCOMPLETE_SIZE);
//      scanComplete.setWildcard(scanCompleteBuffer);
//      scanComplete.resizeToCurrentText();
//      scanComplete.invalidate();
//
//    }




}

void Screen2View::setupScreen()
{
    Screen2ViewBase::setupScreen();



}

void Screen2View::tearDownScreen()
{
    Screen2ViewBase::tearDownScreen();
}


void Screen2View::plant_name_input()
{

}

void Screen2View::update_View_FromSpi(uint8_t *data,  size_t len)
{

//  char buf3[5];
//  snprintf(buf3, sizeof buf3, "%f", data[231]);
//  Unicode::strncpy(Plant_NoBuffer, buf3,  PLANT_NO_SIZE);
//  Plant_No.setWildcard(Plant_NoBuffer);
//  Plant_No.invalidate();


#ifndef SIMULATOR
    FramePacket_t data_parket;
    memcpy(&data_parket, data, sizeof(data_parket));
    uint16_t my_var = 6;

    if (data_parket.header.syncWord != 0xAA55) {
        // bad SOF / out of sync -- resync your stream
//      presenter->setBusy(false);
        return;

    }
    // Buffers (pointers to the generated wildcard buffers)
    touchgfx::Unicode::UnicodeChar* G_buffers[7] = {
        G450_MBuffer, G515_MBuffer, G520_MBuffer, G635_MBuffer,
        G660_MBuffer, G780_MBuffer, G840_MBuffer
    };
    static const uint16_t G_Sizes[7] = {
        G450_M_SIZE, G515_M_SIZE, G520_M_SIZE, G635_M_SIZE,
        G660_M_SIZE, G780_M_SIZE, G840_M_SIZE
    };
    // Pointers to the actual widgets on screen
    touchgfx::TextAreaWithOneWildcard* G_text_areas[7] = {
```

```
        &G450_M, &G515_M, &G520_M, &G635_M, &G660_M, &G780_M, &G840_M
    };

    touchgfx::Unicode::UnicodeChar* P_buffers[7] = {
        P450_MBuffer, P515_MBuffer, P520_MBuffer, P635_MBuffer,
        P660_MBuffer, P780_MBuffer, P840_MBuffer
    };
    static const uint16_t P_Sizes[7] = {
        P450_M_SIZE, P515_M_SIZE, P520_M_SIZE, P635_M_SIZE,
        P660_M_SIZE, P780_M_SIZE, P840_M_SIZE
    };
    touchgfx::TextAreaWithOneWildcard* P_text_areas[7] = {
        &P450_M, &P515_M, &P520_M, &P635_M, &P660_M, &P780_M, &P840_M
    };


    for (int i = 0; i < 7; i++)            // <--- not <=
    {
        char buf[5];
        char buf2[5];
        snprintf(buf, sizeof buf, "%.3f", (data_parket.data[2][i].gain));
//      touchgfx::Unicode::snprintf(G_buffers[i], G_Sizes[i], "%s", buf);
        Unicode::strncpy(G_buffers[i], buf,  G_Sizes[i]);
        G_text_areas[i]->setWildcard(G_buffers[i]);
        G_text_areas[i]->resizeToCurrentText(); // only if you want auto-size
        G_text_areas[i]->invalidate();            // force redraw

        snprintf(buf2, sizeof buf2, "%.3f",(data_parket.data[2][i].phase));
//      touchgfx::Unicode::snprintf(P_buffers[i], P_Sizes[i], "%s", buf2);
        Unicode::strncpy(P_buffers[i], buf2, P_Sizes[i]);
        P_text_areas[i]->setWildcard(P_buffers[i]);
        P_text_areas[i]->resizeToCurrentText();
        P_text_areas[i]->invalidate();
    }

    char buf3[5];
    snprintf(buf3, sizeof buf3, "%d", data[230]);
    Unicode::strncpy(Plant_NoBuffer, buf3,  PLANT_NO_SIZE);
    Plant_No.setWildcard(Plant_NoBuffer);
    Plant_No.invalidate();

//  operation_complete_check.setVisible(true);
//  operation_complete_check.invalidate();

//  presenter->setBusy(false);


#endif



}


void Screen2View::save()
{
    operation_complete_check.setVisible(false);
    operation_complete_check.invalidate();

}

void Screen2View::plant_created()
{
    operation_complete_check.setVisible(true);
    operation_complete_check.invalidate();

}
```

```
void Screen2View::region_created()
{
    operation_complete_check.setVisible(true);
    operation_complete_check.invalidate();

}




void Screen2View::Next_Region()
{

#ifndef SIMULATOR
//  presenter->setBusy(true);
    int state = presenter->next_Region();

//  tickCounter = 0;

#endif

//  presenter->onScanPlant();

}

void Screen2View::save_Plant()
{

#ifndef SIMULATOR
//  presenter->setBusy(true);
    int state = presenter->onIdle();
//  operation_complete_check.setVisible(false);
//  operation_complete_check.invalidate();


#endif

}

void Screen2View::newPlant()
{
//  plant_Id = 141;
//  presenter->newPlant(plant_Id);
//  uint8_t plant_id = 00;
//  presenter->newPlant(plant_Id);

    Keyboard.setVisible(true); //show keyboard
    Keyboard.invalidate();
    flexButtonOK.setVisible(true);
    flexButtonOK.invalidate();
    flexButton1Exit.setVisible(true);
    flexButton1Exit.invalidate();

    buttonWithLabel1.setVisible(false);
    buttonWithLabel1.invalidate();
    buttonWithLabel1_1.setVisible(false);
    buttonWithLabel1_1.invalidate();
    buttonWithLabel1_2.setVisible(false);
    buttonWithLabel1_2.invalidate();

#ifndef SIMULATOR
```

```
#endif
}

void Screen2View::flexButtonExit()
{

    Keyboard.setVisible(false); //show keyboard
    Keyboard.invalidate();
    flexButtonOK.setVisible(false);
    flexButtonOK.invalidate();
    flexButton1Exit.setVisible(false);
    flexButton1Exit.invalidate();

    buttonWithLabel1.setVisible(true);
    buttonWithLabel1.invalidate();
    buttonWithLabel1_1.setVisible(true);
    buttonWithLabel1_1.invalidate();
    buttonWithLabel1_2.setVisible(true);
    buttonWithLabel1_2.invalidate();

//  presenter->setBusy(false);


}


void Screen2View::flexButtonOk()
{

    touchgfx::Unicode::UnicodeChar* u16 = Keyboard.getBuffer();




    char buf[32];
    touchgfx::Unicode::toUTF8(u16, (uint8_t*)buf, sizeof(buf)); // makes a null-terminated UTF-8 string

    char* endptr = nullptr;
    plant_Id = (uint8_t)strtol(buf, &endptr, 10);

    if (*endptr != '\0')
    {
        // not a pure number (contains invalid characters)
    }

    pendingNewPlant = true;
    pendingNewPlantTicks = 2;   // delay = 2 refresh cycles



    //  presenter->setBusy(true);


    Unicode::strncpy(Plant_NoBuffer, Keyboard.getBuffer(), PLANT_NO_SIZE);
    Plant_No.setWildcard(Plant_NoBuffer);
    Plant_No.invalidate();

    Keyboard.setVisible(false); //show keyboard
    Keyboard.invalidate();
    flexButtonOK.setVisible(false);
    flexButtonOK.invalidate();
    flexButton1Exit.setVisible(false);
    flexButton1Exit.invalidate();

    buttonWithLabel1.setVisible(true);
    buttonWithLabel1.invalidate();
    buttonWithLabel1_1.setVisible(true);
```

```
    buttonWithLabel1_1.invalidate();
    buttonWithLabel1_2.setVisible(true);
    buttonWithLabel1_2.invalidate();




}

//scanning without creating a new plant object//
void Screen2View::scan()
{
#ifndef SIMULATOR
//  presenter->setBusy(true);
     int state = presenter->onScan();

#endif

}

//void Screen2View::handleTickEvent()
//{
//#ifndef SIMULATOR
//    static uint8_t last = 0xFF; // impossible initial value
//
//    if (data_available_flag != last)
//    {
//        last = data_available_flag;
//
//        if (data_available_flag == 1)
//        {
//
//        }
//        else
//        {
//            Unicode::strncpy(scanCompleteBuffer, "Finished...", SCANCOMPLETE_SIZE);
//            scanComplete.setVisible(true);   // or false if you want it to disappear immediately
//        }
//
//        scanComplete.setWildcard(scanCompleteBuffer);
//        scanComplete.resizeToCurrentText();
//        scanComplete.invalidate();
//    }
//#endif
//}


void Screen2View::handleTickEvent()
{
#ifndef SIMULATOR

    if (pendingNewPlantTicks > 0)
    {
        pendingNewPlantTicks--;

        if (pendingNewPlantTicks == 0)
        {
            presenter->newPlant(plant_Id);
        }
    }



//    if (pendingNewRegionTicks > 0)
//     {
//         pendingNewRegionTicks--;
//
//         if (pendingRegionPlantTicks == 0)
```

```
//         {
//
//         }
//     }

#endif
}

void Screen2View::showProcessingOverlay()
{
//  Unicode::strncpy(scanCompleteBuffer, "Processing...", SCANCOMPLETE_SIZE);
    processingOverlay1.setVisible(true);
    processingOverlay1.invalidate();
    buttonWithLabel1.setTouchable(false);
    buttonWithLabel1_1.setTouchable(false);
    buttonWithLabel1_2.setTouchable(false);
    new_plant.setTouchable(false);

    flexButtonOK.setTouchable(false);
    flexButton1Exit.setTouchable(false);
}

void Screen2View::hideProcessingOverlay()
{
    processingOverlay1.setVisible(false);
    processingOverlay1.invalidate();
    buttonWithLabel1.setTouchable(true);
    buttonWithLabel1_1.setTouchable(true);
    buttonWithLabel1_2.setTouchable(true);
    new_plant.setTouchable(true);

    flexButtonOK.setTouchable(true);
    flexButton1Exit.setTouchable(true);
}


void Screen2View::setBusy(bool busy)
{
    if (busy)
        showProcessingOverlay();
    else
        hideProcessingOverlay();
}

void Screen2View::operationSuccess(bool success)
{

    if(success == true)
    {
        success_check.setVisible(true);
        rxDummy[232] = 0x00;

    }

    else if(success == false)
    {
        success_check.setVisible(false);
//      presenter->setBusy(false);
        presenter->newPlant(plant_Id);


    }
}
```

File: `TouchGFX/gui/include/gui/screen2_screen/Screen2Presenter.hpp`

```
#ifndef SCREEN2PRESENTER_HPP
#define SCREEN2PRESENTER_HPP

#include <gui/model/ModelListener.hpp>
#include <mvp/Presenter.hpp>

using namespace touchgfx;

class Screen2View;

class Screen2Presenter : public touchgfx::Presenter, public ModelListener
{
public:
    Screen2Presenter(Screen2View& v);

    /**
     * The activate function is called automatically when this screen is "switched in"
     * (ie. made active). Initialization logic can be placed here.
     */
    virtual void activate();

    /**
     * The deactivate function is called automatically when this screen is "switched out"
     * (ie. made inactive). Teardown functionality can be placed here.
     */
    virtual void deactivate();

    virtual ~Screen2Presenter() {}

    virtual void updateFromSpi(uint8_t* data, size_t len) override;

    virtual void plant_created(void) override;
    virtual void region_created(void) override;
    virtual void save(void) override;

    void next_Plant(uint8_t plant_id);

    int next_Region();

    int onScanPlant();

    int onScan();

    int newPlant(uint8_t Plant_No);

    int onIdle();

    void setBusy(bool busy);

    virtual void operationSuccess(bool success) override;




private:
    Screen2Presenter();

    Screen2View& view;
};

#endif // SCREEN2PRESENTER_HPP
```

File: TouchGFX/gui/src/screen2_screen/Screen2Presenter.cpp

```
#include <gui/screen2_screen/Screen2View.hpp>
```

```
#include <gui/screen2_screen/Screen2Presenter.hpp>

extern uint8_t rxDummy[500];

Screen2Presenter::Screen2Presenter(Screen2View& v)
    : view(v)
{

}

void Screen2Presenter::activate()
{

}

void Screen2Presenter::deactivate()
{

}

void Screen2Presenter::updateFromSpi(uint8_t* data, size_t len)
{
    view.update_View_FromSpi(data, len); // Pass it to the View
}

void Screen2Presenter::save()
{
    view.save(); // Pass it to the View
}

void Screen2Presenter::plant_created()
{
    view.plant_created(); // Pass it to the View
}

void Screen2Presenter::region_created()
{
    view.region_created(); // Pass it to the View
}







void Screen2Presenter::next_Plant(uint8_t plant_Id){

    send_data_via_spi_Params Params{};

    Params.command = 0x07;
    Params.plant_Id = plant_Id;

    int state = model->sendDataViaSPI(Params);

//  model->sendDataViaSPI(0x07);


}

int Screen2Presenter::newPlant(uint8_t Plant_No){

    send_data_via_spi_Params Params{};

    Params.command = 0x07;
    Params.plant_Id = Plant_No;

    int state = model->sendDataViaSPI(Params);
```

```
//  int state = model->sendDataViaSPI(0x07);
    return state;

//  model->sendDataViaSPI(0x07);


}

int Screen2Presenter::next_Region(){

    send_data_via_spi_Params Params{};

    Params.command = 0x03;

    int state = model->sendDataViaSPI(Params);
//  int state = model->sendDataViaSPI(0x03);
    return state;


    /*
     * This delay is essential to ensure the command to change region has time to be received
     * at the receiver's end
     *
     */

//  HAL_Delay(800);

//  model->sendDataViaSPI(0x01);
}

//scan and create a new plant object//
int Screen2Presenter::onScanPlant(){


    send_data_via_spi_Params Params{};

    Params.command = 0x01;


    int state = model->sendDataViaSPI(Params);
//  int state = model->sendDataViaSPI(0x01);
    return state;
////    model->sendDataViaSPI(0x05);
}


//scan without creating a new plant object//
int Screen2Presenter::onScan()
{

    send_data_via_spi_Params Params{};

    Params.command = 0x06;

    int state = model->sendDataViaSPI(Params);
//  int state = model->sendDataViaSPI(0x06);
    return state;

}


int Screen2Presenter::onIdle()
{

    send_data_via_spi_Params Params{};

    Params.command = 0x00;
```

```
    int state = model->sendDataViaSPI(Params);
//  int state = model->sendDataViaSPI(0x06);
    return state;

}

void Screen2Presenter::setBusy(bool busy)
{
    model->setBusy(busy);
    view.setBusy(busy);
}


void Screen2Presenter::operationSuccess(bool success)
{
    view.operationSuccess(success);
}
```

Appendix E

Data Processing and Analysis Source Code

This appendix lists the Python source code used to process the sensor readings and to carry out the analyses reported in Chapters 3 and 4. The scripts are grouped in the order in which they are applied. The first group converts the raw firmware readings to detected optical power (Equation 20; Appendix A) and to relative reflectance against the 99% Spectralon reference (Equation 21), and assembles the plant-level dark-enclosure and outdoor datasets. The second group contains the randomized-complete-block ANOVA of NDVI and the log1p nitrate-N regression and neural-network models. The third group processes and analyzes the specular-reflection angle experiment.

The scripts were run with Python 3 using NumPy, pandas, SciPy, statsmodels, scikit-learn, PyTorch, openpyxl, and Matplotlib. The file paths shown are relative to the project data folder. Absolute paths written inside the scripts point to the author's working copy of the data and must be changed before the code is run elsewhere.

Data Processing: Raw Readings to Detected Optical Power and Relative Reflectance

File: `Field_experiment/Source_codes/White_reference_correction/power_conversion.py`

Shared conversion of a background-corrected firmware reading to detected optical power (Equation 20).

```
"""
Gain (V_corrected) -> Detected Optical Power, shared physics
============================================================
Same Equations (12)-(20) as compute_detected_optical_power.py, factored out
so leaf_gain_pipeline.py and white_reference_correction.py can convert every
individual reading -- not just the white-reference table -- through the same
responsivity model instead of working in raw sensor Gain units.

gain_to_power_uw() is a straight linear map (P = V * k_lambda, k_lambda > 0)
for a fixed wavelength: same PDE/gain/transimpedance constants apply whether
the V being converted came from a leaf or the white reference, so it preserves
sign (floor_missing logic is unaffected) and commutes with averaging/weighting.
Converting a whole pipeline through this doesn't change *ratios* at a fixed
wavelength (Power_leaf/Power_whiteref == Gain_leaf/Gain_whiteref algebraically)
-- what it changes is that absolute readings become physically meaningful and,
critically, comparable *across* wavelengths, which raw Gain never was (PDE
varies ~4x from 660nm to 850nm, so equal Gain never meant equal light).
"""

# -- Physical constants (identical to compute_detected_optical_power.py) --
H = 6.626e-34          # Planck's constant, J*s
C = 3e8                # speed of light, m/s
Q = 1.6e-19            # elementary charge, C
G = 1e5                # SiPM avalanche multiplication gain
A2 = 1.1               # second-stage voltage amplifier gain, V/V
A_SOFT = 3             # firmware software gain
A_POST = A2 * A_SOFT   # Equation (13)
R_TIA = 330.0          # transimpedance feedback resistor, Ohm

# PDE anchor points read from Figure 5 (Overvoltage = 0.8V curve)
PDE_ANCHOR_LOW_NM, PDE_ANCHOR_LOW_PCT = 450, 17.0
PDE_ANCHOR_HIGH_NM, PDE_ANCHOR_HIGH_PCT = 850, 1.0
_R = (PDE_ANCHOR_HIGH_PCT / PDE_ANCHOR_LOW_PCT) ** (50.0 / (PDE_ANCHOR_HIGH_NM - PDE_ANCHOR_LOW_NM))


def pde_at(wavelength_nm):
    """Geometric interpolation of PDE (%) between the two Figure-5 anchor points."""
    if wavelength_nm == PDE_ANCHOR_HIGH_NM:
        return PDE_ANCHOR_HIGH_PCT
    steps = (wavelength_nm - PDE_ANCHOR_LOW_NM) / 50.0
    return PDE_ANCHOR_LOW_PCT * (_R ** steps)


def wavelength_scale_factor(wavelength_nm):
    """
    k_lambda such that P_det (W) = V_gain * k_lambda. Isolated so callers can
    rescale a constant (e.g. an epsilon tuned in Gain units) into Power units
    by the same factor, rather than re-deriving it.
    """
    pde_frac = pde_at(wavelength_nm) / 100.0
    e_photon = H * C / (wavelength_nm * 1e-9)   # Equation (16)
    r_eff = (pde_frac * G * Q) / e_photon         # Equation (17)
    return 1.0 / (A_POST * R_TIA * r_eff)
```

```
def gain_to_power_uw(v_gain, wavelength_nm):
    """V_corrected (raw sensor Gain) -> detected optical power, in microwatts."""
    return v_gain * wavelength_scale_factor(wavelength_nm) * 1e6
```

File: `Field_experiment/Source_codes/White_reference_correction/compute_detected_optical_power.py`

Step-by-step detected-optical-power calculation for the Spectralon reference readings (Appendix A).

```
"""
Working Backward to Get Optical Power Detected After Reflection
================================================================
Implements Equations (12)-(20) of the thesis methods section: recovers the
optical power incident on the SiPM active area, per wavelength, from the
white-reference-target measurements in White_reference_data.

V_corrected,lambda
------------------
No separate LED-OFF (dark) CSV exists anywhere in the raw data -- every
leaf_*.csv has exactly one Gain value per (SpectrumIndex, Sensor), not a
paired ON/OFF measurement. Per the thesis text, firmware already subtracts
the LED-OFF background before reporting a value, so the mean white-reference
Gain (Sensor E) at each wavelength is used directly as V_corrected,lambda --
Equation (12)'s subtraction is not re-applied here. This mirrors the same
white-reference averaging used in white_reference_correction.py's
parse_white_reference(), using the RAW (uncorrected) mean -- the empirical
780nm x1.44 hardware factor used elsewhere in the pipeline is a separate,
unrelated correction and is not applied to this physics derivation.

PDE_lambda
----------
Read from the Overvoltage = 0.8 V curve in Figure 5. Only two points are
directly labeled on the figure: P(450, 17%) and P(850, 1%). The curve is
smoothly, monotonically decaying between them with no visible inflection,
consistent with a geometric (exponential) decay -- the standard shape for a
SiPM PDE roll-off past its blue/violet peak -- so intermediate wavelengths
(660, 670, 700, 750, 780 nm) are interpolated on that basis:

    PDE(lambda) = PDE(450) * r ** ((lambda - 450) / 50),   r = (1/17)**(1/8)

This is an approximation of a manually-read graph, not a datasheet value --
replace PDE_800V below with exact datasheet figures if/when available.

Usage:
    python compute_detected_optical_power.py
"""

import pandas as pd
from pathlib import Path

# -- Physical constants --
H = 6.626e-34          # Planck's constant, J*s
C = 3e8                # speed of light, m/s
Q = 1.6e-19             # elementary charge, C
G = 1e5                 # SiPM avalanche multiplication gain
A2 = 1.1                # second-stage voltage amplifier gain, V/V
A_SOFT = 3              # firmware software gain
A_POST = A2 * A_SOFT     # Equation (13)
R_TIA = 330.0            # transimpedance feedback resistor, Ohm

# -- Paths --
```

```
WHITE_REF_PATH = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment\White_reference_data")
OUT_PATH = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment\White_reference_data\Detected_Optical_Power_Table.xlsx")

TARGET_SENSOR = "E"
POSITION = "lower"
SPECTRUM_TO_WL = {0: 780, 1: 670, 2: 700, 3: 850, 4: 660, 5: 750}
WAVELENGTHS_NM = sorted(SPECTRUM_TO_WL.values())  # [660, 670, 700, 750, 780, 850]

# -- PDE anchor points read from Figure 5 (Overvoltage = 0.8 V curve) --
PDE_ANCHOR_LOW_NM, PDE_ANCHOR_LOW_PCT = 450, 17.0
PDE_ANCHOR_HIGH_NM, PDE_ANCHOR_HIGH_PCT = 850, 1.0
_R = (PDE_ANCHOR_HIGH_PCT / PDE_ANCHOR_LOW_PCT) ** (50.0 / (PDE_ANCHOR_HIGH_NM - PDE_ANCHOR_LOW_NM))


def pde_at(wavelength_nm):
    """Geometric interpolation of PDE (%) between the two Figure-5 anchor points."""
    if wavelength_nm == PDE_ANCHOR_HIGH_NM:
        return PDE_ANCHOR_HIGH_PCT
    steps = (wavelength_nm - PDE_ANCHOR_LOW_NM) / 50.0
    return PDE_ANCHOR_LOW_PCT * (_R ** steps)


# -- V_corrected,lambda: mean white-reference Gain (Sensor E), raw/uncorrected --

def mean_white_reference_gain():
    plant_dirs = sorted([p for p in WHITE_REF_PATH.iterdir() if p.is_dir()], key=lambda p: p.name)
    plant_means = []
    for plant_dir in plant_dirs:
        lower_dir = plant_dir / POSITION
        csvs = sorted(lower_dir.glob("leaf_*.csv"))
        all_gains = []
        for csv_f in csvs:
            df = pd.read_csv(csv_f)
            se = (
                df[(df["Sensor"] == TARGET_SENSOR) & (df["SpectrumIndex"].isin(SPECTRUM_TO_WL))]
                .set_index("SpectrumIndex")["Gain"]
            )
            all_gains.append(se)
        mean_gains = pd.concat(all_gains, axis=1).mean(axis=1).to_dict()
        plant_means.append({SPECTRUM_TO_WL[k]: v for k, v in mean_gains.items()})
    return pd.DataFrame(plant_means)[WAVELENGTHS_NM].mean()  # indexed by wavelength (nm)


def main():
    v_corrected = mean_white_reference_gain()

    rows = []
    for wl_nm in WAVELENGTHS_NM:
        vc = v_corrected[wl_nm]
        pde_pct = pde_at(wl_nm)
        pde_frac = pde_pct / 100.0

        wl_m = wl_nm * 1e-9
        e_photon = H * C / wl_m                      # Equation (16)
        v_tia = vc / A_POST                            # Equation (14)
        i_sipm = v_tia / R_TIA                         # Equation (15)
        r_eff = (pde_frac * G * Q) / e_photon          # Equation (17)
        p_det_w = i_sipm / r_eff                       # Equation (18)
        p_det_uw = p_det_w * 1e6

        rows.append({
            "Wavelength (nm)": wl_nm,
            "Vcorrected (V)": round(vc, 6),
            "PDE (%)": round(pde_pct, 3),
            "Ephoton (J)": e_photon,
            "ISiPM (A)": i_sipm,
```

```
            "Reff (A/W)": r_eff,
            "Pdet (W)": p_det_w,
            "Pdet (uW)": round(p_det_uw, 4),
        })

    table = pd.DataFrame(rows).sort_values("Wavelength (nm)").reset_index(drop=True)

    OUT_PATH.parent.mkdir(parents=True, exist_ok=True)
    table.to_excel(OUT_PATH, index=False)
    table.to_csv(OUT_PATH.with_suffix(".csv"), index=False)

    print(table.to_string(index=False))
    print(f"\nSaved: {OUT_PATH}")
    print(f"Saved: {OUT_PATH.with_suffix('.csv')}")


if __name__ == "__main__":
    main()
```

File: `Field_experiment/Source_codes/White_reference_correction/white_reference_correction.py`

White-reference correction: relative reflectance as leaf power divided by Spectralon power (Equation 21).

```
"""
White Reference Correction
==========================
Normalizes both dark room and light environment spectral data
by dividing each wavelength's voltage by the corresponding
white reference voltage.

Purpose:
    Each LED in the sensor has a different output power. A white
    reference panel (near-100% reflectance) measured under the same
    sensor captures the relative LED power at each wavelength. Dividing
    plant measurements by the white reference removes LED power variation,
    converting raw voltage to reflectance.

White reference:
    Three replicates (Plant_055, Plant_056, Plant_057) measured from the
    lower leaf position. All CSVs within each replicate are averaged, then
    the three replicates are averaged to give one reference vector.

Hardware corrections (670nm, 780nm):
    The same corrections applied to plant data are applied to the white
    reference before dividing. This ensures the correction factors cancel
    in the ratio, which is the scientifically correct behaviour.

Usage:
    python white_reference_correction.py
"""

import pandas as pd
import numpy as np
from pathlib import Path

from power_conversion import gain_to_power_uw

# -- Configuration ------------------------------------------------------------

WHITE_REF_PATH = r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment\White_reference_data"
```

```
PROCESSED_ROOT = r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment\Processed_Biomass_Sampling_Data"

# Every (Dark, Light) output pair to correct against the same white reference.
# Add a new entry here whenever leaf_gain_pipeline.py's BATCHES registry
# (Source_codes/Data_processing/leaf_gain_pipeline.py) grows a new batch.
BATCH_TARGETS = [
    ("06_16_2026_V10", r"Dark_env\06_16_2026_V10\Data\Dark_06_16_2026_V10.xlsx",
                       r"Light_env\06_16_2026_V10\Data\Light_06_16_2026_V10.xlsx"),
    ("07_08_2026_R1", r"Dark_env\07_08_2026_R1\Data\Dark_07_08_2026_R1.xlsx",
                      r"Light_env\07_08_2026_R1\Data\Light_07_08_2026_R1.xlsx"),
    ("07_08_2026_v8", r"Dark_env\07_08_2026_v8\Data\Dark_07_08_2026_v8.xlsx",
                      r"Light_env\07_08_2026_v8\Data\Light_07_08_2026_v8.xlsx"),
    # Dark-room only (no Light side collected) -- light_rel = None.
    ("06_01_2026_V5", r"Dark_env\06_01_2026_V5\Data\Dark_06_01_2026_V5.xlsx", None),
]

TARGET_SENSOR = "E"
VALID_INDICES  = list(range(6))
POSITION = "lower"

SPECTRUM_TO_WL = {
    0: "780 nm", 1: "670 nm", 2: "700 nm",
    3: "850 nm", 4: "660 nm", 5: "750 nm",
}
WL_COLS = ["660 nm", "670 nm", "700 nm", "750 nm", "780 nm", "850 nm"]
WL_NAME_TO_NM = {"660 nm": 660, "670 nm": 670, "700 nm": 700, "750 nm": 750, "780 nm": 780, "850 nm": 850}

# Plant data (leaf_gain_pipeline.py) now reports detected optical power (uW),
# converted through the same Equations (12)-(20) responsivity model at the
# point each leaf CSV is read. The white reference has to go through the
# identical conversion here so both sides of the reflectance ratio are in the
# same physical unit -- a leaf-power-over-Gain or Gain-over-white-power ratio
# would not mean anything.

# -- Parse white reference CSVs ------------------------------------------------

def parse_white_reference(wr_path: str) -> pd.Series:
    """Read all white reference plant folders and return mean detected power (uW) per wavelength."""
    plant_dirs = sorted(
        [p for p in Path(wr_path).iterdir() if p.is_dir()],
        key=lambda p: p.name,
    )

    plant_means = []
    for plant_dir in plant_dirs:
        lower_dir = plant_dir / POSITION
        csvs = sorted(lower_dir.glob("leaf_*.csv"))

        all_gains = []
        for csv_f in csvs:
            df = pd.read_csv(csv_f)
            se = (
                df[(df["Sensor"] == TARGET_SENSOR) & (df["SpectrumIndex"].isin(VALID_INDICES))]
                .set_index("SpectrumIndex")["Gain"]
            )
            power = pd.Series({
                idx: gain_to_power_uw(v, WL_NAME_TO_NM[SPECTRUM_TO_WL[idx]])
                for idx, v in se.items()
            })
            all_gains.append(power)

        mean_gains = pd.concat(all_gains, axis=1).mean(axis=1).to_dict()
        record = {SPECTRUM_TO_WL[k]: v for k, v in mean_gains.items()}
        plant_means.append(record)
        print(f"  {plant_dir.name}: {len(csvs)} leaves -- "
              f"660nm={record['660 nm']:.4f} uW, 850nm={record['850 nm']:.4f} uW")
```

```
    wr_raw = pd.DataFrame(plant_means)[WL_COLS].mean()

    # Use the directly measured power at every wavelength. The historical
    # 780-nm factor was applied identically to plant and reference values and
    # canceled in their ratio; omitting it keeps both layers in true units.
    wr_corr = wr_raw.copy()

    return wr_raw, wr_corr


# -- Apply correction ---------------------------------------------------------

def apply_correction(df: pd.DataFrame, wr_corr: pd.Series, label: str) -> pd.DataFrame:
    df_corr = df.copy()
    for wl in WL_COLS:
        df_corr[wl] = (df[wl] / wr_corr[wl]).round(6)

    print(f"\n{label} reflectance summary (mean +/- std):")
    for wl in WL_COLS:
        m = df_corr[wl].mean()
        s = df_corr[wl].std()
        flag = "  [>1.0 - LED power stronger than reference]" if m > 1.0 else ""
        print(f"  {wl}: {m:.4f} +/- {s:.4f}{flag}")

    return df_corr


# -- Main ---------------------------------------------------------------------

def main():
    print("Parsing white reference data...")
    wr_raw, wr_corr = parse_white_reference(WHITE_REF_PATH)

    print("\nWhite reference (raw):")
    print(wr_raw.round(4).to_string())
    print("\nWhite reference used for normalization (direct detected power):")
    print(wr_corr.round(4).to_string())

    root = Path(PROCESSED_ROOT)
    for label, dark_rel, light_rel in BATCH_TARGETS:
        dark_path = root / dark_rel
        print(f"\n=== Batch: {label} ===")

        print("Loading plant data...")
        dark = pd.read_excel(dark_path)
        print(f"  Dark:  {len(dark)} rows")
        dark_corr = apply_correction(dark, wr_corr, "Dark room")
        print("Saving corrected files...")
        dark_reflectance_dir = dark_path.parent / "Reflectance"
        dark_reflectance_dir.mkdir(parents=True, exist_ok=True)
        dark_corr.to_excel(
            dark_reflectance_dir / f"{dark_path.stem}_reflectance.xlsx", index=False
        )
        dark_corr.to_excel(dark_path, index=False)
        print(f"  Saved: {dark_path}")

        if light_rel is None:
            print("  (no Light side for this batch)")
            continue

        light_path = root / light_rel
        light = pd.read_excel(light_path)
        print(f"  Light: {len(light)} rows")
        light_corr = apply_correction(light, wr_corr, "Light environment")
        light_reflectance_dir = light_path.parent / "Reflectance"
        light_reflectance_dir.mkdir(parents=True, exist_ok=True)
        light_corr.to_excel(
```

```
            light_reflectance_dir / f"{light_path.stem}_reflectance.xlsx", index=False
        )
        light_corr.to_excel(light_path, index=False)
        print(f"  Saved: {light_path}")

    print("\nWhite reference correction complete for all batches.")
    print("Note: values > 1.0 at NIR bands (750, 850nm) in the dark room indicate")
    print("those LEDs are more powerful than the ambient-light reference baseline.")
    print("Relative differences between treatments remain valid.")


if __name__ == "__main__":
    main()
```

File: `Field_experiment/Source_codes/Data_processing/leaf_gain_pipeline.py`

Processes the raw leaf CSV files of each sampling batch into one averaged six-band reading per plant.

```
"""
Leaf Spectral Gain Processing Pipeline
======================================
Processes raw leaf CSV files into one row per plant (ID | Plant Number |
Date | Planting | wavelengths | Nitrogen Content) and writes the result to
an Excel file.

This is the shared logic behind leaf_gain_processing_copy.ipynb. It is
parameterized per "batch" -- one batch is one (date, planting, sensor
version) combination, e.g. "06_16_2026 V10, first planting" or
"07_08_2026 v8, second planting" -- so the same code handles any number of
plantings without change; adding a new planting just means adding a new
batch entry to BATCHES below.

Usage:
    python leaf_gain_pipeline.py                # process every batch in BATCHES
    python leaf_gain_pipeline.py 06_16_2026_V10  # process one named batch
"""

import re
import sys
import warnings
from pathlib import Path

import numpy as np
import pandas as pd
from openpyxl import Workbook, load_workbook

sys.path.insert(0, str(Path(__file__).resolve().parents[1] / "White_reference_correction"))
from power_conversion import gain_to_power_uw  # noqa: E402

warnings.filterwarnings("ignore")

# -- Fixed sensor / spectrum settings ---------------------------------------

TARGET_SENSOR = "E"
VALID_SPECTRUM_INDICES = [0, 1, 2, 3, 4, 5]
SPECTRUM_TO_WAVELENGTH = {
    0: "780 nm",
    1: "670 nm",
    2: "700 nm",
    3: "850 nm",
    4: "660 nm",
    5: "750 nm",
}
# Numeric wavelength per SpectrumIndex, including 6 (840nm, the no-LED noise
```

```
# probe) -- needed to run every channel, not just the six named outputs,
# through the same responsivity-equation power conversion.
SPECTRUM_TO_WAVELENGTH_NM = {0: 780, 1: 670, 2: 700, 3: 850, 4: 660, 5: 750, 6: 840}
POSITION = "lower"
CORRECTION_780NM = 1.44

# Everything downstream (aggregation, weighting, floor_missing, the shape-
# feature/VI work, all the ICC and treatment-response analysis) now runs on
# detected optical POWER (uW) instead of raw sensor Gain -- every value is
# converted via the same Equations (12)-(20) responsivity model used for the
# white reference in compute_detected_optical_power.py, right at the point
# each leaf CSV is read (see _collect_leaf_gains). This is a positive linear
# rescale per wavelength (P = V * k_lambda, k_lambda > 0), so it preserves
# sign (floor_missing is unaffected) and, at a fixed wavelength, leaves every
# leaf/white-reference RATIO unchanged algebraically -- reflectance values and
# every correlation/ANOVA/ICC/FDR result computed on them should be numerically
# equivalent to the Gain-based pipeline. What changes: readings are now in a
# physically real unit, and -- unlike raw Gain, where PDE varies ~4x across
# these six wavelengths -- are directly comparable *across* wavelengths too.

# SpectrumIndex 6 = 840nm. No LED has ever been populated on this channel
# (old module or new), so it carries no real optical signal -- every reading
# on it is pure device noise (electronics-only in the dark; ambient/handling
# instability between the off/on snapshots in the light). Kept out of
# VALID_SPECTRUM_INDICES and never white-reference corrected; reported
# separately as a noise/QC diagnostic. See Source_codes/Device_Noise_Decoding/.
NOISE_SPECTRUM_INDEX = 6
NOISE_RAW_COL = "840nm Noise (raw, uW)"
NOISE_ABS_COL = "840nm Noise (abs, uW, QC flag)"

# Per-leaf inverse-noise weight = 1 / (NOISE_WEIGHT_EPSILON + |840nm gain|), used
# by stat="weighted" to down-weight (not discard) leaves read during a noisy
# acquisition instead of a hard include/exclude cutoff. Chosen empirically: a
# hard-exclude threshold anywhere in 0.05-0.2 throws out 50-65% of light-room
# leaves (most plants only have 1-2 replicates, so exclusion often leaves
# nothing to average) and did not beat the unweighted median on held-out
# validation. Inverse weighting, tested against the one signal we can verify
# independently -- correlation between 750nm and 850nm, adjacent real NIR
# bands that should track each other in genuine leaf reflectance -- raised
# light-room r from 0.128 (n.s., p=0.15) to 0.200 (p=0.024) while leaving the
# dark room unchanged (r=0.257 vs 0.260) and doing no harm to channel pairs
# with no real relationship to begin with. See Source_codes/Device_Noise_
# Decoding/ for the full validation. Originally tuned in raw-Gain units;
# rescaled through the same 840nm responsivity factor so the weighting
# behaves identically in relative terms now that noise840 is in power (uW).
NOISE_WEIGHT_EPSILON = gain_to_power_uw(0.01, SPECTRUM_TO_WAVELENGTH_NM[6])

BASE = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment")

# -- Ground truth (lab-measured leaf NO3, ppm) -------------------------------
# Every KSU Soil Testing Lab report gets dropped in here as its own .xlsx --
# one file per lab order, arriving at whatever sampling date it covers. There
# is no fixed naming requirement beyond ending in .xlsx; each file carries its
# own sample dates in the "Sample Name" column (e.g. "101 P1 6/1/26"), so new
# files just need to be added to this folder, nothing in code changes.
GROUND_TRUTH_DIR = BASE / "Ground_truth"

# A leaf reading and its ground-truth NO3 sample are almost never taken the
# same day (lab turnaround, sampling logistics), so matching is nearest-date
# per (plot, plant) rather than exact-date. Beyond this many days apart the
# ground truth is treated as too stale to trust (N status drifts, especially
# for the "200 split" treatment which gets a second application partway
# through the season) and Nitrogen Content is left blank rather than guessed.
GROUND_TRUTH_MAX_GAP_DAYS = 21


def _parse_ground_truth_file(path):
```

```
    """
    Parse one KSU Soil Testing Lab report into tidy rows: ID (plot), plant_num
    (e.g. "P1"), sample_date, NO3_ppm. The sheet has a variable-height letterhead
    before the real header ("Lab # (s)" | "Sample Name" | "NO3 ppm"), so the
    header row is located by content rather than a fixed row number.
    """
    raw = pd.read_excel(path, header=None)
    header_row = None
    for i, row in raw.iterrows():
        if row.astype(str).str.contains("Sample Name", na=False).any():
            header_row = i
            break
    if header_row is None:
        print(f"    WARNING: {path.name} -- couldn't find the 'Sample Name' header row, skipping.")
        return pd.DataFrame(columns=["ID", "plant_num", "sample_date", "NO3_ppm"])

    name_col = raw.columns[(raw.iloc[header_row] == "Sample Name")][0]
    no3_col = next(
        (c for c in raw.columns if str(raw.iloc[header_row][c]).strip().replace("\n", " ").startswith("NO3")),
        name_col + 1,
    )

    body = raw.iloc[header_row + 1:][[name_col, no3_col]].copy()
    body.columns = ["sample_name", "NO3_ppm"]
    body["NO3_ppm"] = pd.to_numeric(body["NO3_ppm"], errors="coerce")
    body = body.dropna(subset=["sample_name", "NO3_ppm"])

    parts = body["sample_name"].astype(str).str.extract(r"(\d+)\s+(P\d+)\s+([\d/]+)")
    body["ID"] = pd.to_numeric(parts[0], errors="coerce")
    body["plant_num"] = parts[1]
    # Report dates aren't consistently 2- vs 4-digit year across lab reports
    # (e.g. "6/1/26" vs "7/8/2026"), so try the 2-digit format first and only
    # fall back to dateutil inference for the rows it can't parse.
    sample_date = pd.to_datetime(parts[2], format="%m/%d/%y", errors="coerce")
    unparsed = sample_date.isna() & parts[2].notna()
    if unparsed.any():
        sample_date.loc[unparsed] = pd.to_datetime(parts[2][unparsed], errors="coerce")
    body["sample_date"] = sample_date
    body = body.dropna(subset=["ID", "plant_num", "sample_date"])
    body["ID"] = body["ID"].astype(int)

    return body[["ID", "plant_num", "sample_date", "NO3_ppm"]]


def load_ground_truth():
    """Parse and concatenate every ground-truth lab report found in GROUND_TRUTH_DIR."""
    if not GROUND_TRUTH_DIR.exists():
        return pd.DataFrame(columns=["ID", "plant_num", "sample_date", "NO3_ppm"])
    files = sorted(GROUND_TRUTH_DIR.glob("*.xlsx"))
    if not files:
        return pd.DataFrame(columns=["ID", "plant_num", "sample_date", "NO3_ppm"])
    frames = [_parse_ground_truth_file(f) for f in files]
    frames = [f for f in frames if len(f)]
    if not frames:
        return pd.DataFrame(columns=["ID", "plant_num", "sample_date", "NO3_ppm"])
    combined = pd.concat(frames, ignore_index=True)
    print(f"  Loaded {len(combined)} ground-truth NO3 readings from {len(files)} file(s) in {GROUND_TRUTH_DIR.
    ↪ name}/")
    return combined


def nearest_ground_truth(ground_truth, plot_id, plant_num, sampling_date):
    """
    Among ground-truth rows for this (plot, plant), return the one closest in
    time to sampling_date, as (NO3_ppm, gt_date, gap_days) -- or (None, None, None)
    if there's no match within GROUND_TRUTH_MAX_GAP_DAYS.
    """
    if ground_truth.empty:
```

```
        return None, None, None
    rows = ground_truth[(ground_truth["ID"] == plot_id) & (ground_truth["plant_num"] == plant_num)]
    if rows.empty:
        return None, None, None
    target = pd.to_datetime(sampling_date, format="%m/%d/%y")
    gaps = (rows["sample_date"] - target).abs().dt.days
    best = gaps.idxmin()
    if gaps[best] > GROUND_TRUTH_MAX_GAP_DAYS:
        return None, None, None
    row = rows.loc[best]
    return row["NO3_ppm"], row["sample_date"].strftime("%m/%d/%y"), int(gaps[best])


# -- Batch registry ------------------------------------------------------
# Each batch = one raw folder tree (with Light/ and Dark/ subfolders) for a
# single planting on a single sampling date. "planting" is written verbatim
# into the output's Planting column.

BATCHES = {
    "06_16_2026_V10": dict(
        raw_dir=BASE / "Biomass_Sampling" / "06_16_2026" / "Biomass_Sampling_V10_first_planting",
        index_sheet=BASE / "Biomass_Sampling" / "06_16_2026" / "Biomass_Sampling_V10_first_planting" / "
    ↪ samples_index_sheet.xlsx",
        growth_stage_filter="V8",   # index sheet labels the V10 sampling rows "V8"
        row_offset=None,
        row_count=None,
        sampling_date="06/16/26",
        planting="First",
        version_label="V10",
        out_light=BASE / "Processed_Biomass_Sampling_Data" / "Light_env" / "06_16_2026_V10" / "Data" / "
    ↪ Light_06_16_2026_V10.xlsx",
        out_dark=BASE / "Processed_Biomass_Sampling_Data" / "Dark_env" / "06_16_2026_V10" / "Data" / "
    ↪ Dark_06_16_2026_V10.xlsx",
    ),
    "07_08_2026_R1": dict(
        raw_dir=BASE / "Biomass_Sampling" / "07_08_2026" / "R1_first_planting",
        index_sheet=BASE / "Biomass_Sampling" / "07_08_2026" / "Sampling_index_sheet_07_08_26.xlsx",
        growth_stage_filter=None,   # this sheet has no Growth Stage column
        row_offset=0,               # rows 1-36 = first planting (R1)
        row_count=36,
        sampling_date="07/08/26",
        planting="First",
        version_label="R1",
        out_light=BASE / "Processed_Biomass_Sampling_Data" / "Light_env" / "07_08_2026_R1" / "Data" / "
    ↪ Light_07_08_2026_R1.xlsx",
        out_dark=BASE / "Processed_Biomass_Sampling_Data" / "Dark_env" / "07_08_2026_R1" / "Data" / "
    ↪ Dark_07_08_2026_R1.xlsx",
    ),
    "07_08_2026_v8": dict(
        raw_dir=BASE / "Biomass_Sampling" / "07_08_2026" / "v8_second_planting",
        index_sheet=BASE / "Biomass_Sampling" / "07_08_2026" / "Sampling_index_sheet_07_08_26.xlsx",
        growth_stage_filter=None,
        row_offset=36,              # rows 37-72 = second planting (v8)
        row_count=36,
        sampling_date="07/08/26",
        planting="Second",
        version_label="v8",
        out_light=BASE / "Processed_Biomass_Sampling_Data" / "Light_env" / "07_08_2026_v8" / "Data" / "
    ↪ Light_07_08_2026_v8.xlsx",
        out_dark=BASE / "Processed_Biomass_Sampling_Data" / "Dark_env" / "07_08_2026_v8" / "Data" / "
    ↪ Dark_07_08_2026_v8.xlsx",
    ),
}


# -- Plant folder discovery ----------------------------------------------

def get_plant_folders(raw_data_path):
```

```
    """Return plant folders sorted numerically."""
    return sorted(
        [p for p in Path(raw_data_path).iterdir()
         if p.is_dir() and p.name.lower().startswith("plant_")],
        key=lambda p: int(p.name.split("_")[1]),
    )


# -- Index sheet loading ------------------------------------------------

def load_index_rows(index_path, growth_stage_filter=None, row_offset=0, row_count=None):
    """
    Load the samples index sheet and return rows in sequential order,
    matching Plant_001, Plant_002, ... of the batch being processed.

    Two selection modes, chosen by what the sheet provides:
    - growth_stage_filter: keep only rows whose "Growth Stage" column
      matches (used when one sheet covers multiple sensor versions).
    - row_offset/row_count: no Growth Stage column exists, so instead take
      a contiguous slice of the sheet (used when one sheet covers multiple
      plantings back-to-back, e.g. rows 1-36 = planting 1, 37-72 = planting 2).
    """
    wb = load_workbook(index_path, read_only=True)
    ws = wb.active

    header = [cell.value for cell in next(ws.iter_rows(min_row=1, max_row=1))]
    col_map = {name: idx for idx, name in enumerate(header)}

    rows = []
    for row in ws.iter_rows(min_row=2, values_only=True):
        if growth_stage_filter is not None:
            if row[col_map["Growth Stage"]] != growth_stage_filter:
                continue
        rows.append({
            "plot_treatment": row[col_map["Plot Treatment"]],
            "plant_number": row[col_map["Plant #"]],
            "date": row[col_map["Date"]],
        })
    wb.close()

    if growth_stage_filter is None and row_count is not None:
        rows = rows[row_offset:row_offset + row_count]

    return rows


# -- Gain extraction ----------------------------------------------------

def _collect_leaf_gains(plant_folder, position, target_sensor):
    """
    Read every leaf CSV in the position subfolder of a single plant folder and
    return a DataFrame of detected optical power (uW): index = SpectrumIndex
    (0-6, every channel present in the raw file, LED-active or not), columns
    = one per leaf CSV. Returns None if no usable data.

    Converts each raw Gain reading through the responsivity-equation power
    model (gain_to_power_uw) at the wavelength for that SpectrumIndex,
    immediately on read -- every step downstream (weighting, floor_missing,
    aggregation, the noise-probe stats) operates on power, not Gain.
    """
    position_path = plant_folder / position
    if not position_path.exists():
        print(f"    WARNING: {position_path} not found -- skipping.")
        return None

    csv_files = sorted(position_path.glob("leaf_*.csv"))
    if not csv_files:
        print(f"    WARNING: No leaf CSVs in {position_path} -- skipping.")
```

```
        return None

    all_gains = []
    for csv_path in csv_files:
        df = pd.read_csv(csv_path)
        sensor_df = df[df["Sensor"] == target_sensor]
        gains = sensor_df.set_index("SpectrumIndex")["Gain"]
        power = pd.Series(
            {idx: gain_to_power_uw(v, SPECTRUM_TO_WAVELENGTH_NM[idx])
             for idx, v in gains.items() if idx in SPECTRUM_TO_WAVELENGTH_NM}
        )
        all_gains.append(power)

    if not all_gains:
        return None

    return pd.concat(all_gains, axis=1)


def average_gains_for_plant(plant_folder, position, target_sensor, valid_indices,
                            stat="mean", floor_missing=False, combined=None,
                            drop_negative_leaf_frames=False):
    """
    Aggregate the LED-active channels (valid_indices) across leaf frames.
    Returns dict {spectrum_index: gain}, or None if no usable data.

    stat:
      "mean"     -- dark room. 840nm noise is negligible and roughly constant
                    there (see NOISE_WEIGHT_EPSILON note), so plain averaging
                    is fine.
      "median"   -- legacy light-room behaviour, robust to per-frame outliers
                    but blind to *how* outlying a given leaf's acquisition was.
      "weighted" -- light room. Per-leaf weight = inverse of that leaf's own
                    840nm (no-LED) noise-probe reading, so a leaf read during
                    a disturbed acquisition still contributes but counts for
                    less, rather than being median-ignored or hard-dropped.
                    See NOISE_WEIGHT_EPSILON above for validation.

    floor_missing: if True, a resulting value < 0 is treated as below the
    sensor's detectable signal floor and reported as missing (NaN) rather
    than as a literal negative gain (0 itself is kept as a valid reading).
    Used for light-room data only -- ambient light periodically pushes
    readings below the floor, most often
    at 660/670/700 nm.

    drop_negative_leaf_frames: if True, negative per-leaf gains are treated
    as below-floor frames and excluded before aggregation. This prevents
    signed cancellation (real positive signal averaged down by invalid
    negative frames), which is common in ambient-light acquisitions.

    combined: pre-read output of _collect_leaf_gains, to avoid re-reading the
    same CSVs when the noise-channel stat is also being computed for this
    plant. If not given, reads the CSVs itself.
    """
    if combined is None:
        combined = _collect_leaf_gains(plant_folder, position, target_sensor)
    if combined is None:
        return None

    channels = combined.loc[combined.index.isin(valid_indices)]
    if drop_negative_leaf_frames:
        channels = channels.where(channels >= 0)

    if stat == "weighted":
        if NOISE_SPECTRUM_INDEX in combined.index:
            noise = combined.loc[NOISE_SPECTRUM_INDEX].abs()
            weights = 1.0 / (NOISE_WEIGHT_EPSILON + noise)
        else:
```

```
            weights = pd.Series(1.0, index=channels.columns)

        def _weighted_mean_skipna(row):
            valid = row.notna()
            if not valid.any():
                return np.nan
            w = weights.loc[valid]
            return np.average(row.loc[valid], weights=w)

        result = channels.apply(_weighted_mean_skipna, axis=1)
    elif stat == "median":
        result = channels.median(axis=1)
    else:
        result = channels.mean(axis=1)

    if floor_missing:
        result = result.where(result >= 0)
    return result.to_dict()


def noise_channel_stat_for_plant(plant_folder, position, target_sensor,
                                 noise_index, stat="mean", combined=None):
    """
    Aggregate the no-LED noise-probe channel (840nm, SpectrumIndex 6) across
    leaf frames for one plant. Unlike the real channels, negative readings
    are not "below floor" here -- a negative value is just as meaningful a
    noise reading as a positive one, so no floor_missing filtering applies.

    Returns (raw_stat, abs_stat):
      raw_stat = mean/median of the signed readings (diagnostic -- near zero
                 is expected; a persistent nonzero bias would flag a device
                 fault rather than random instability).
      abs_stat = mean/median of |readings| -- the actual noise-magnitude
                 indicator. Validated against the other 6 channels: |840nm|
                 correlates with how much the real channels deviate from
                 their expected levels on the same reading (r=0.38, p<0.0001
                 in the light room; no relationship in the dark room, where
                 840nm reduces to the device's baseline electronic floor).
                 See Source_codes/Device_Noise_Decoding/ for the full
                 characterization.
    """
    if combined is None:
        combined = _collect_leaf_gains(plant_folder, position, target_sensor)
    if combined is None or noise_index not in combined.index:
        return None, None

    readings = combined.loc[noise_index]
    if stat == "median":
        return readings.median(), readings.abs().median()
    return readings.mean(), readings.abs().mean()


def build_output_records(plant_folders, index_rows, position, target_sensor,
                         valid_indices, spectrum_to_wavelength, sampling_date,
                         planting, stat="mean", floor_missing=False,
                         drop_negative_leaf_frames=False, ground_truth=None):
    """
    Process each plant individually and build output records, one row per
    plant, tagged with the batch's Planting label.

    ground_truth: output of load_ground_truth(), or None. When given, each
    record's Nitrogen Content is the lab-measured NO3 (ppm) for that plot+
    plant closest in date to sampling_date (see nearest_ground_truth /
    GROUND_TRUTH_MAX_GAP_DAYS). The matched ground-truth date and the gap in
    days are recorded alongside it, since a leaf reading and its NO3 sample
    are essentially never taken the same day.
    """
    if ground_truth is None:
```

```
        ground_truth = pd.DataFrame(columns=["ID", "plant_num", "sample_date", "NO3_ppm"])

    records = []
    for folder, idx_row in zip(plant_folders, index_rows):
        treatment_id = idx_row["plot_treatment"]
        plant_num = idx_row["plant_number"]

        print(f"  {folder.name} -> Treatment {treatment_id}, {plant_num}, planting={planting}")
        combined = _collect_leaf_gains(folder, position, target_sensor)
        gains = average_gains_for_plant(folder, position, target_sensor, valid_indices,
                                        stat=stat, floor_missing=floor_missing, combined=combined,
                                        drop_negative_leaf_frames=drop_negative_leaf_frames)
        noise_raw, noise_abs = noise_channel_stat_for_plant(
            folder, position, target_sensor, NOISE_SPECTRUM_INDEX, stat=stat, combined=combined
        )

        record = {
            "ID": treatment_id,
            "Plant Number": plant_num,
            "Date": sampling_date,
            "Planting": planting,
        }
        if gains:
            for spec_idx, wl_col in spectrum_to_wavelength.items():
                val = gains.get(spec_idx, None)
                record[wl_col] = None if (val is None or pd.isna(val)) else val
        else:
            for wl_col in spectrum_to_wavelength.values():
                record[wl_col] = None
            print(f"    WARNING: No data for {folder.name}")

        record[NOISE_RAW_COL] = None if (noise_raw is None or pd.isna(noise_raw)) else noise_raw
        record[NOISE_ABS_COL] = None if (noise_abs is None or pd.isna(noise_abs)) else noise_abs

        no3, gt_date, gap_days = nearest_ground_truth(ground_truth, treatment_id, plant_num, sampling_date)
        record["Nitrogen Content"] = no3
        record["N Ground Truth Date"] = gt_date
        record["N Ground Truth Gap (days)"] = gap_days
        records.append(record)

    return records


def apply_780nm_correction(records, factor=CORRECTION_780NM):
    """
    The 780 nm LED emits less than expected -- an LED-side effect, distinct
    from (and not addressed by) the detector-side PDE/responsivity power
    conversion applied earlier. This factor places 780 nm proportionally
    between 750 nm and 850 nm on the NIR curve. Applied identically to the
    white reference downstream (now also in power units), so it cancels out
    in the reflectance ratio and has no net numeric effect.
    """
    for record in records:
        if record.get("780 nm") is not None:
            record["780 nm"] = record["780 nm"] * factor


# -- Output ------------------------------------------------------------

def write_to_excel(processed_path, records, spectrum_to_wavelength):
    """Write detected power and retain a permanent, unit-labelled power copy."""
    processed_path = Path(processed_path)
    processed_path.parent.mkdir(parents=True, exist_ok=True)

    wb = Workbook()
    ws = wb.active
    ws.title = "Sheet1"
```

```
    headers = (["ID", "Plant Number", "Date", "Planting"] + list(spectrum_to_wavelength.values())
               + [NOISE_RAW_COL, NOISE_ABS_COL]
               + ["Nitrogen Content", "N Ground Truth Date", "N Ground Truth Gap (days)"])
    for col_idx, h in enumerate(headers, start=1):
        ws.cell(row=1, column=col_idx, value=h)

    for row_idx, record in enumerate(records, start=2):
        for col_idx, col_name in enumerate(headers, start=1):
            ws.cell(row=row_idx, column=col_idx, value=record.get(col_name))

    wb.save(processed_path)
    print(f"[OK] {processed_path.name} saved with {len(records)} rows.")

    power_dir = processed_path.parent / "Detected_Power"
    power_dir.mkdir(parents=True, exist_ok=True)
    power_path = power_dir / f"{processed_path.stem}_detected_power_uW.xlsx"
    wb.save(power_path)
    print(f"[OK] {power_path.name} saved as the persistent detected-power layer (uW).")


def write_missing_readings_log(csv_path, records, spectrum_to_wavelength):
    """QC log: for each plant, list which wavelength bands came back NaN."""
    wl_cols = list(spectrum_to_wavelength.values())
    log_rows = []
    for record in records:
        missing = [wl for wl in wl_cols if pd.isna(record.get(wl))]
        if missing:
            log_rows.append({
                "ID": record["ID"],
                "Plant Number": record["Plant Number"],
                "Missing Bands": ", ".join(missing),
            })

    csv_path = Path(csv_path)
    csv_path.parent.mkdir(parents=True, exist_ok=True)
    pd.DataFrame(log_rows, columns=["ID", "Plant Number", "Missing Bands"]).to_csv(csv_path, index=False)
    print(f"[OK] {csv_path.name} saved with {len(log_rows)} rows flagged.")


# -- Batch orchestration ------------------------------------------------

def process_env(raw_env_path, out_path, index_sheet, growth_stage_filter,
                row_offset, row_count, sampling_date, planting, version_label, env_label):
    print(f"\n--- {env_label} : {raw_env_path} ---")
    plant_folders = get_plant_folders(raw_env_path)
    index_rows = load_index_rows(index_sheet, growth_stage_filter, row_offset or 0, row_count)

    print(f"Found {len(plant_folders)} plant folders, {len(index_rows)} matching index rows.")
    if len(plant_folders) != len(index_rows):
        print(f"[!]  MISMATCH: {len(plant_folders)} folders vs {len(index_rows)} index rows -- "
              f"rows will be truncated to the shorter of the two.")

    # Light-room readings are noisier under ambient light: weight each leaf
    # by the inverse of its own 840nm (no-LED) noise-probe reading, so a leaf
    # read during a disturbed acquisition still contributes but counts for
    # less (validated improvement over the plain median -- see
    # NOISE_WEIGHT_EPSILON above). Negative per-leaf frames are excluded
    # before aggregation (below-floor under ambient light), then any residual
    # negative plant-level result is still marked missing. Dark-room keeps the plain
    # mean with no floor filter -- 840nm noise there is negligible and doesn't
    # vary enough to weight against. See Methods_Notes_Spectral_Correction_V10.md.
    is_light = env_label == "Light"
    ground_truth = load_ground_truth()
    records = build_output_records(
        plant_folders, index_rows, POSITION, TARGET_SENSOR,
        VALID_SPECTRUM_INDICES, SPECTRUM_TO_WAVELENGTH, sampling_date, planting,
        stat="weighted" if is_light else "mean", floor_missing=is_light,
        drop_negative_leaf_frames=is_light, ground_truth=ground_truth,
```

```
    )
    # Keep detected power as the direct Equation-21 result. The former 1.44
    # 780-nm multiplier was applied to both leaf and white reference and
    # therefore canceled in reflectance, while obscuring the physical unit.
    write_to_excel(out_path, records, SPECTRUM_TO_WAVELENGTH)

    log_path = out_path.parent / f"{env_label}_{version_label}_missing_readings_log.csv"
    write_missing_readings_log(log_path, records, SPECTRUM_TO_WAVELENGTH)
    return records


def process_batch(name):
    cfg = BATCHES[name]
    print(f"\n=== Batch: {name} (planting={cfg['planting']}) ===")
    process_env(
        cfg["raw_dir"] / "Light", cfg["out_light"], cfg["index_sheet"],
        cfg["growth_stage_filter"], cfg["row_offset"], cfg["row_count"],
        cfg["sampling_date"], cfg["planting"], cfg["version_label"], "Light",
    )
    process_env(
        cfg["raw_dir"] / "Dark", cfg["out_dark"], cfg["index_sheet"],
        cfg["growth_stage_filter"], cfg["row_offset"], cfg["row_count"],
        cfg["sampling_date"], cfg["planting"], cfg["version_label"], "Dark",
    )


# -- 06/01/2026 V5 batch: dark-room only, 3 canopy positions per plot --------
# Doesn't fit the standard BATCHES shape: no Light side, no index sheet (the
# 18 Plant_XXX raw folders map to plots in a fixed, non-sequential order --
# verified empirically against the pre-existing Processed_data.xlsx, whose
# values were reproduced exactly from Plant_001/lower/middle/upper against
# Plot 304's P1/P2/P3), and each plot's 3 canopy positions (lower/middle/
# upper) become 3 separate output rows (P1/P2/P3), not a single row per
# folder like every other batch. Its real significance: sampled 06/01/26 --
# the exact same day as the only ground-truth NO3 lab report -- so this is
# the first batch with a genuine same-day (0-day-gap) spectral/ground-truth
# pairing, rather than the 15-37 day gaps every other batch has.
V5_RAW_DIR = BASE / "Biomass_Sampling" / "06_01_2026" / "Biomass_Sampling_V5" / "Dark_room" / "Raw_Data"
V5_OUT_PATH = BASE / "Processed_Biomass_Sampling_Data" / "Dark_env" / "06_01_2026_V5" / "Data" / "
    ↪ Dark_06_01_2026_V5.xlsx"
V5_SAMPLING_DATE = "06/01/26"
V5_PLANTING = "First"
# Plant_001 .. Plant_018 -> Plot ID, in raw-folder order (index 0 = Plant_001).
V5_PLANT_TO_PLOT = [304, 202, 201, 103, 106, 204, 101, 203, 301, 206, 205, 306, 105, 303, 302, 305, 102, 104]
V5_POSITION_TO_PLANT_NUM = {"lower": "P1", "middle": "P2", "upper": "P3"}


def process_06_01_2026_v5():
    print(f"\n=== Batch: 06_01_2026_V5 (planting={V5_PLANTING}, dark room only) ===")
    plant_folders = get_plant_folders(V5_RAW_DIR)
    if len(plant_folders) != len(V5_PLANT_TO_PLOT):
        print(f"[!]  MISMATCH: {len(plant_folders)} raw folders vs "
              f"{len(V5_PLANT_TO_PLOT)} expected plots -- check V5_PLANT_TO_PLOT.")

    ground_truth = load_ground_truth()
    records = []
    for folder, plot_id in zip(plant_folders, V5_PLANT_TO_PLOT):
        print(f"  {folder.name} -> Plot {plot_id}")
        for position, plant_num in V5_POSITION_TO_PLANT_NUM.items():
            combined = _collect_leaf_gains(folder, position, TARGET_SENSOR)
            gains = average_gains_for_plant(folder, position, TARGET_SENSOR,
                                            VALID_SPECTRUM_INDICES, stat="mean",
                                            floor_missing=False, combined=combined)
            noise_raw, noise_abs = noise_channel_stat_for_plant(
                folder, position, TARGET_SENSOR, NOISE_SPECTRUM_INDEX, stat="mean", combined=combined
            )

            record = {"ID": plot_id, "Plant Number": plant_num,
```

```
                    "Date": V5_SAMPLING_DATE, "Planting": V5_PLANTING}
            if gains:
                for spec_idx, wl_col in SPECTRUM_TO_WAVELENGTH.items():
                    val = gains.get(spec_idx, None)
                    record[wl_col] = None if (val is None or pd.isna(val)) else val
            else:
                for wl_col in SPECTRUM_TO_WAVELENGTH.values():
                    record[wl_col] = None

            record[NOISE_RAW_COL] = None if (noise_raw is None or pd.isna(noise_raw)) else noise_raw
            record[NOISE_ABS_COL] = None if (noise_abs is None or pd.isna(noise_abs)) else noise_abs

            no3, gt_date, gap_days = nearest_ground_truth(ground_truth, plot_id, plant_num, V5_SAMPLING_DATE)
            record["Nitrogen Content"] = no3
            record["N Ground Truth Date"] = gt_date
            record["N Ground Truth Gap (days)"] = gap_days
            records.append(record)

        # P4: the lab report has a 4th leaf per plot that was never spectrally
        # scanned (only 3 canopy positions exist in the raw data) -- keep the
        # row so its ground-truth NO3 isn't silently dropped, all spectral
        # columns blank.
        no3, gt_date, gap_days = nearest_ground_truth(ground_truth, plot_id, "P4", V5_SAMPLING_DATE)
        record = {"ID": plot_id, "Plant Number": "P4", "Date": V5_SAMPLING_DATE, "Planting": V5_PLANTING}
        for wl_col in SPECTRUM_TO_WAVELENGTH.values():
            record[wl_col] = None
        record[NOISE_RAW_COL] = None
        record[NOISE_ABS_COL] = None
        record["Nitrogen Content"] = no3
        record["N Ground Truth Date"] = gt_date
        record["N Ground Truth Gap (days)"] = gap_days
        records.append(record)

    # Keep detected power as the direct Equation-21 result (no empirical
    # wavelength multiplier).
    write_to_excel(V5_OUT_PATH, records, SPECTRUM_TO_WAVELENGTH)
    log_path = V5_OUT_PATH.parent / "Dark_V5_missing_readings_log.csv"
    write_missing_readings_log(log_path, records, SPECTRUM_TO_WAVELENGTH)
    return records


if __name__ == "__main__":
    names = sys.argv[1:] or list(BATCHES.keys())
    for n in names:
        if n in BATCHES:
            process_batch(n)
    if not sys.argv[1:] or "06_01_2026_V5" in sys.argv[1:]:
        process_06_01_2026_v5()
```

File: `Field_experiment/Source_codes/Data_processing/shape_feature_noise_pipeline.py`

Plant-level reflectance for the V10 (April and May) and R1 (April) dark-enclosure rounds, with the 840 nm no-LED noise-channel correction.

```
"""
Raw-to-Feature Pipeline with 840nm Noise Regression
==================================================
Builds plant-level reflectance shape features from raw leaf CSV files.

Key idea:
- SpectrumIndex 6 (840nm) has no active LED, so it is used as an in-device
  noise probe. For each batch/environment and each active wavelength, this
```

```
  script estimates a linear coupling coefficient beta between that wavelength's
  raw gain and 840nm noise across all leaf frames, then subtracts beta*noise
  per frame before plant aggregation.

Outputs per batch/environment:
- Plant-level feature table (CSV + XLSX)
- Feature visuals by N treatment

Usage:
    python shape_feature_noise_pipeline.py
    python shape_feature_noise_pipeline.py 06_16_2026_V10
"""

from __future__ import annotations

import sys
from pathlib import Path

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from openpyxl import load_workbook

BASE = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment")

TARGET_SENSOR = "E"
POSITION = "lower"
VALID_INDICES = [0, 1, 2, 3, 4, 5]
NOISE_INDEX = 6
EPS = 1e-9
NOISE_WEIGHT_EPSILON = 0.01
CORRECTION_780NM = 1.44

SPEC_TO_WL = {
    0: "780 nm",
    1: "670 nm",
    2: "700 nm",
    3: "850 nm",
    4: "660 nm",
    5: "750 nm",
}
WL_ORDER = ["660 nm", "670 nm", "700 nm", "750 nm", "780 nm", "850 nm"]

RATE_ORDER = ["0", "50", "100", "150", "200", "200 split"]
RATE_MAP = {"0": 0, "50": 50, "100": 100, "150": 150, "200": 200, "200 split": 200}
RATE_COLORS = {
    "0": "#2166ac",
    "50": "#67a9cf",
    "100": "#d1e5f0",
    "150": "#fddbc7",
    "200": "#ef8a62",
    "200 split": "#b2182b",
}

BATCHES = {
    "06_16_2026_V10": {
        "raw_dir": BASE / "Biomass_Sampling" / "06_16_2026" / "Biomass_Sampling_V10_first_planting",
        "index_sheet": BASE / "Biomass_Sampling" / "06_16_2026" / "Biomass_Sampling_V10_first_planting" / "
    ↪ samples_index_sheet.xlsx",
        "growth_stage_filter": "V8",
        "row_offset": None,
        "row_count": None,
        "sampling_date": "06/16/26",
        "planting": "First",
        "stage": "V10",
    },
    "07_08_2026_R1": {
```

```
        "raw_dir": BASE / "Biomass_Sampling" / "07_08_2026" / "R1_first_planting",
        "index_sheet": BASE / "Biomass_Sampling" / "07_08_2026" / "Sampling_index_sheet_07_08_26.xlsx",
        "growth_stage_filter": None,
        "row_offset": 0,
        "row_count": 36,
        "sampling_date": "07/08/26",
        "planting": "First",
        "stage": "R1",
    },
    "07_08_2026_v8": {
        "raw_dir": BASE / "Biomass_Sampling" / "07_08_2026" / "v8_second_planting",
        "index_sheet": BASE / "Biomass_Sampling" / "07_08_2026" / "Sampling_index_sheet_07_08_26.xlsx",
        "growth_stage_filter": None,
        "row_offset": 36,
        "row_count": 36,
        "sampling_date": "07/08/26",
        "planting": "Second",
        "stage": "V8",
    },
}


def get_plant_folders(raw_data_path: Path) -> list[Path]:
    return sorted(
        [p for p in raw_data_path.iterdir() if p.is_dir() and p.name.lower().startswith("plant_")],
        key=lambda p: int(p.name.split("_")[1]),
    )


def load_index_rows(index_path: Path, growth_stage_filter=None, row_offset=0, row_count=None):
    wb = load_workbook(index_path, read_only=True)
    ws = wb.active
    header = [cell.value for cell in next(ws.iter_rows(min_row=1, max_row=1))]
    col_map = {name: idx for idx, name in enumerate(header)}

    rows = []
    for row in ws.iter_rows(min_row=2, values_only=True):
        if growth_stage_filter is not None and row[col_map["Growth Stage"]] != growth_stage_filter:
            continue
        rows.append(
            {
                "plot_treatment": row[col_map["Plot Treatment"]],
                "plant_number": row[col_map["Plant #"]],
                "date": row[col_map["Date"]],
            }
        )
    wb.close()

    if growth_stage_filter is None and row_count is not None:
        rows = rows[row_offset : row_offset + row_count]

    return rows


def parse_white_reference() -> pd.Series:
    wr_path = BASE / "White_reference_data"
    plant_dirs = sorted([p for p in wr_path.iterdir() if p.is_dir()], key=lambda p: p.name)

    plant_means = []
    for plant_dir in plant_dirs:
        lower_dir = plant_dir / POSITION
        csvs = sorted(lower_dir.glob("leaf_*.csv"))
        if not csvs:
            continue

        leaf_series = []
        for csv_f in csvs:
            df = pd.read_csv(csv_f)
```

```
            s = (
                df[(df["Sensor"] == TARGET_SENSOR) & (df["SpectrumIndex"].isin(VALID_INDICES))]
                .set_index("SpectrumIndex")["Gain"]
            )
            leaf_series.append(s)

        if not leaf_series:
            continue

        mean_gains = pd.concat(leaf_series, axis=1).mean(axis=1)
        rec = {SPEC_TO_WL[i]: mean_gains.get(i, np.nan) for i in VALID_INDICES}
        plant_means.append(rec)

    wr = pd.DataFrame(plant_means)[WL_ORDER].mean()
    wr["780 nm"] = wr["780 nm"] * CORRECTION_780NM
    return wr


def read_plant_leaf_matrix(plant_folder: Path) -> pd.DataFrame:
    pos = plant_folder / POSITION
    csvs = sorted(pos.glob("leaf_*.csv"))
    rows = []

    for csv_f in csvs:
        df = pd.read_csv(csv_f)
        d = df[df["Sensor"] == TARGET_SENSOR].set_index("SpectrumIndex")["Gain"]

        row = {
            "leaf_file": csv_f.name,
            "noise840": d.get(NOISE_INDEX, np.nan),
        }
        for idx in VALID_INDICES:
            row[SPEC_TO_WL[idx]] = d.get(idx, np.nan)
        rows.append(row)

    return pd.DataFrame(rows)


def estimate_noise_betas(all_leaf_df: pd.DataFrame) -> dict[str, float]:
    betas = {}
    noise = all_leaf_df["noise840"].astype(float)
    v = noise.var(ddof=1)
    if pd.isna(v) or v <= EPS:
        return {wl: 0.0 for wl in WL_ORDER}

    for wl in WL_ORDER:
        y = all_leaf_df[wl].astype(float)
        valid = y.notna() & noise.notna()
        if valid.sum() < 4:
            betas[wl] = 0.0
            continue
        yv = y[valid]
        nv = noise[valid]
        cov = ((yv - yv.mean()) * (nv - nv.mean())).sum() / (len(yv) - 1)
        betas[wl] = float(cov / v)
    return betas


def noise_correct_leaf_df(df: pd.DataFrame, betas: dict[str, float], is_light: bool) -> pd.DataFrame:
    out = df.copy()
    for wl in WL_ORDER:
        out[wl] = out[wl] - betas[wl] * out["noise840"]
        if wl == "780 nm":
            out[wl] = out[wl] * CORRECTION_780NM

    if is_light:
        for wl in WL_ORDER:
            out.loc[out[wl] < 0, wl] = np.nan
```

```
    return out


def aggregate_plant_reflectance(leaf_df: pd.DataFrame, wr: pd.Series, is_light: bool) -> pd.Series:
    if leaf_df.empty:
        return pd.Series({wl: np.nan for wl in WL_ORDER})

    weights = 1.0 / (NOISE_WEIGHT_EPSILON + leaf_df["noise840"].abs()) if is_light else pd.Series(1.0, index=
    ↪ leaf_df.index)

    vals = {}
    for wl in WL_ORDER:
        col = leaf_df[wl]
        valid = col.notna() & weights.notna()
        if valid.sum() == 0:
            vals[wl] = np.nan
            continue
        vals[wl] = float(np.average(col[valid], weights=weights[valid]))

    refl = {wl: (vals[wl] / wr[wl]) if (pd.notna(vals[wl]) and pd.notna(wr[wl]) and wr[wl] != 0) else np.nan
    ↪ for wl in WL_ORDER}
    return pd.Series(refl)


def compute_shape_features(row: pd.Series) -> dict[str, float]:
    r660, r670, r700, r750, r780, r850 = (row["660 nm"], row["670 nm"], row["700 nm"], row["750 nm"], row["780
    ↪  nm"], row["850 nm"])

    def safe_div(a, b):
        if pd.isna(a) or pd.isna(b) or abs(b) < EPS:
            return np.nan
        return a / b

    def nd(a, b):
        if pd.isna(a) or pd.isna(b) or abs(a + b) < EPS:
            return np.nan
        return (a - b) / (a + b)

    return {
        "NDVI_850_670": nd(r850, r670),
        "NDRE_780_700": nd(r780, r700),
        "NDRE_750_700": nd(r750, r700),
        "NIR_Red_Ratio_850_670": safe_div(r850, r670),
        "RedEdge_Ratio_750_700": safe_div(r750, r700),
        "RedEdge_Diff_750_700": (r750 - r700) if pd.notna(r750) and pd.notna(r700) else np.nan,
        "Slope_660_700": (r700 - r660) / 40.0 if pd.notna(r700) and pd.notna(r660) else np.nan,
        "Slope_700_750": (r750 - r700) / 50.0 if pd.notna(r750) and pd.notna(r700) else np.nan,
        "Slope_750_780": (r780 - r750) / 30.0 if pd.notna(r780) and pd.notna(r750) else np.nan,
        "Slope_780_850": (r850 - r780) / 70.0 if pd.notna(r850) and pd.notna(r780) else np.nan,
    }


def create_visuals(df: pd.DataFrame, out_png: Path, title: str):
    features_main = ["NDVI_850_670", "NDRE_780_700", "NIR_Red_Ratio_850_670", "RedEdge_Ratio_750_700"]
    slope_features = ["Slope_660_700", "Slope_700_750", "Slope_750_780", "Slope_780_850"]
    heat_features = ["NDVI_850_670", "NDRE_780_700", "NDRE_750_700", "NIR_Red_Ratio_850_670", "
    ↪ RedEdge_Ratio_750_700", "RedEdge_Diff_750_700"]

    fig, axs = plt.subplots(2, 2, figsize=(20, 14))
    fig.suptitle(title, fontsize=20, fontweight="bold", y=0.98)

    # A) Main shape features vs N treatment
    ax = axs[0, 0]
    x = np.arange(len(RATE_ORDER))
    for feat in features_main:
        means = []
        sems = []
```

```
    for n in RATE_ORDER:
        s = df.loc[df["N Treatment"] == n, feat].dropna()
        means.append(s.mean() if len(s) else np.nan)
        sems.append(s.sem() if len(s) > 1 else np.nan)
    ax.errorbar(x, means, yerr=sems, marker="o", linewidth=2, capsize=3, label=feat)
ax.set_xticks(x)
ax.set_xticklabels(["0", "50", "100", "150", "200", "200 split"], rotation=20)
ax.set_title("A) Normalized/Ratio Shape Features by N Treatment")
ax.set_ylabel("Feature value")
ax.grid(alpha=0.3)
ax.legend(fontsize=9)

# B) Red-edge slope features
ax = axs[0, 1]
for feat in slope_features:
    means = []
    sems = []
    for n in RATE_ORDER:
        s = df.loc[df["N Treatment"] == n, feat].dropna()
        means.append(s.mean() if len(s) else np.nan)
        sems.append(s.sem() if len(s) > 1 else np.nan)
    ax.errorbar(x, means, yerr=sems, marker="s", linewidth=2, capsize=3, label=feat)
ax.set_xticks(x)
ax.set_xticklabels(["0", "50", "100", "150", "200", "200 split"], rotation=20)
ax.set_title("B) First-Difference Slope Features")
ax.set_ylabel("Slope per nm")
ax.grid(alpha=0.3)
ax.legend(fontsize=9)

# C) Heatmap of treatment means (z-scored by feature)
ax = axs[1, 0]
mat = []
for n in RATE_ORDER:
    row = []
    for feat in heat_features:
        row.append(df.loc[df["N Treatment"] == n, feat].mean())
    mat.append(row)
mat = np.array(mat, dtype=float)
z = (mat - np.nanmean(mat, axis=0)) / (np.nanstd(mat, axis=0) + EPS)
im = ax.imshow(z, aspect="auto", cmap="coolwarm", vmin=-2, vmax=2)
ax.set_yticks(range(len(RATE_ORDER)))
ax.set_yticklabels(RATE_ORDER)
ax.set_xticks(range(len(heat_features)))
ax.set_xticklabels(heat_features, rotation=35, ha="right")
ax.set_title("C) Treatment Means (Feature-wise z-score)")
plt.colorbar(im, ax=ax, fraction=0.046, pad=0.04)

# D) Plant-level NDRE distribution
ax = axs[1, 1]
rng = np.random.default_rng(42)
for i, n in enumerate(RATE_ORDER):
    s = df.loc[df["N Treatment"] == n, "NDRE_780_700"].dropna()
    if len(s) == 0:
        continue
    jitter = rng.uniform(-0.12, 0.12, size=len(s))
    ax.scatter(np.full(len(s), i) + jitter, s.values, s=45, alpha=0.7, color=RATE_COLORS[n], edgecolors="
↪ black", linewidth=0.4)
    ax.plot([i - 0.2, i + 0.2], [s.mean(), s.mean()], color="black", linewidth=2)
ax.set_xticks(range(len(RATE_ORDER)))
ax.set_xticklabels(["0", "50", "100", "150", "200", "200 split"], rotation=20)
ax.set_ylabel("NDRE_780_700")
ax.set_title("D) Plant-Level NDRE Distribution by Treatment")
ax.grid(alpha=0.3)

out_png.parent.mkdir(parents=True, exist_ok=True)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.savefig(out_png, dpi=160, bbox_inches="tight")
plt.close(fig)
```

```
def process_env_batch(batch_name: str, cfg: dict, env: str, wr: pd.Series, field_map: pd.DataFrame):
    raw_env = cfg["raw_dir"] / env
    plant_folders = get_plant_folders(raw_env)
    idx_rows = load_index_rows(cfg["index_sheet"], cfg["growth_stage_filter"], cfg["row_offset"] or 0, cfg["
    ↪ row_count"])
    is_light = env.lower() == "light"

    n = min(len(plant_folders), len(idx_rows))
    plant_folders = plant_folders[:n]
    idx_rows = idx_rows[:n]

    # Gather all leaf frames first so betas are estimated batch/env-wide.
    leaf_rows = []
    plant_leaf_cache = {}
    for folder, idx in zip(plant_folders, idx_rows):
        leaf_df = read_plant_leaf_matrix(folder)
        if leaf_df.empty:
            continue
        leaf_df["ID"] = idx["plot_treatment"]
        leaf_df["Plant Number"] = idx["plant_number"]
        leaf_df["plant_folder"] = folder.name
        leaf_rows.append(leaf_df)
        plant_leaf_cache[(idx["plot_treatment"], idx["plant_number"])]=leaf_df

    if not leaf_rows:
        print(f"No leaf data found for {batch_name} {env}.")
        return

    all_leaf = pd.concat(leaf_rows, ignore_index=True)
    betas = estimate_noise_betas(all_leaf)
    print(f"{batch_name} {env} noise betas: " + ", ".join([f"{k}={v:.4f}" for k,v in betas.items()]))

    records = []
    for folder, idx in zip(plant_folders, idx_rows):
        key = (idx["plot_treatment"], idx["plant_number"])
        leaf_df = plant_leaf_cache.get(key, pd.DataFrame())
        leaf_corr = noise_correct_leaf_df(leaf_df, betas, is_light=is_light)
        refl = aggregate_plant_reflectance(leaf_corr, wr=wr, is_light=is_light)

        rec = {
            "batch": batch_name,
            "environment": env,
            "growth_stage": cfg["stage"],
            "Planting": cfg["planting"],
            "ID": idx["plot_treatment"],
            "Plant Number": idx["plant_number"],
            "Date": cfg["sampling_date"],
            "n_leaf_frames": len(leaf_df),
            "noise840_mean": float(leaf_df["noise840"].mean()) if not leaf_df.empty else np.nan,
            "noise840_abs_mean": float(leaf_df["noise840"].abs().mean()) if not leaf_df.empty else np.nan,
        }
        for wl in WL_ORDER:
            rec[wl] = refl[wl]
        rec.update(compute_shape_features(pd.Series(rec)))
        records.append(rec)

    out = pd.DataFrame(records)
    out = out.merge(field_map[["Plot Number", "N Treatment", "Block", "Planting Date"]], left_on="ID",
    ↪ right_on="Plot Number", how="left")
    out.drop(columns=["Plot Number"], inplace=True)
    out["N Treatment"] = out["N Treatment"].astype(str)
    out["N_numeric"] = out["N Treatment"].map(RATE_MAP)

    out_dir = BASE / "Processed_Biomass_Sampling_Data" / "Shape_Feature_Analysis" / env / batch_name
    data_dir = out_dir / "Data"
    vis_dir = out_dir / "Visualizations"
```

```
    data_dir.mkdir(parents=True, exist_ok=True)
    vis_dir.mkdir(parents=True, exist_ok=True)

    csv_path = data_dir / f"{env}_{batch_name}_shape_features_noise_corrected.csv"
    xlsx_path = data_dir / f"{env}_{batch_name}_shape_features_noise_corrected.xlsx"
    out.to_csv(csv_path, index=False)
    out.to_excel(xlsx_path, index=False)

    fig_path = vis_dir / f"{env}_{batch_name}_shape_features.png"
    create_visuals(out, fig_path, title=f"{env} {cfg['stage']} -- Shape Features (840nm Noise-Corrected)")

    print(f"Saved: {csv_path}")
    print(f"Saved: {xlsx_path}")
    print(f"Saved: {fig_path}")


def main():
    names = sys.argv[1:] or list(BATCHES.keys())
    wr = parse_white_reference()
    field_map = pd.read_excel(BASE / "2026 Field Maps.xlsx", sheet_name="Plot treatments")

    for name in names:
        if name not in BATCHES:
            print(f"Unknown batch: {name}")
            continue
        cfg = BATCHES[name]
        print(f"\n=== Processing batch: {name} ({cfg['stage']}) ===")
        process_env_batch(name, cfg, "Light", wr, field_map)
        process_env_batch(name, cfg, "Dark", wr, field_map)


if __name__ == "__main__":
    main()
```

File: `Field_experiment/Source_codes/Data_processing/process_07_27_2026.py`

Dark-enclosure processing of the 27 July 2026 rounds (R3 April planting and R1 May planting).

```
"""
07/27/2026 batch processor -- R3 first planting + R1 second planting, dark
room only.
================================================================================
Unlike every other BATCHES entry in leaf_gain_pipeline.py, this raw data has
not gone through the sensor firmware's usual baseline/offset correction (same
situation as Biomass_Sampling/Test_Data, see process_test_plant.py), so each
raw Gain reading is offset-corrected here per leaf frame -- -0.07 V for every
channel except 850 nm (SpectrumIndex 3), which gets -0.15 V -- before the
shared responsivity-power conversion. This was cross-checked against the
index sheet's own handwritten "Before Illumination" baseline table
(Biomass_Sampling/07_27_2026/index_sheet.pdf), which agrees on every channel
except a handwritten 0.08 at 660 nm; confirmed with the user to apply the
flat 0.07 V / 0.15 V rule uniformly instead of that one outlier value.

Only the dark room is processed here -- the light environment for this date
is intentionally left untouched.

White reference correction is intentionally NOT applied to this batch: the
output below is the direct Equations (12)-(20) detected optical power from
gain_to_power_uw (i.e. "reflected power" computed straight from the
responsivity equations), not a white-reference-normalized reflectance. No
entry is added for this date to White_reference_correction/
white_reference_correction.py's BATCH_TARGETS.

Index sheet: Biomass_Sampling/07_27_2026/index_sheet_transcribed.xlsx is a
```

```
plain-Excel transcription of index_sheet.pdf (Index, Plot, Plant # inferred
as P1/P2 from each plot's repeated row pair), in the same
#/Plot Treatment/Plant #/Date shape load_index_rows() expects. Index rows
1-36 (Plot 101...306) = R3_first_planting; rows 37-72 (Plot 606...401) =
R1_Second_planting.

Usage:
    python process_07_27_2026.py                  # both plantings
    python process_07_27_2026.py 07_27_2026_R3     # one named batch
"""

import sys
from pathlib import Path

import pandas as pd

sys.path.insert(0, str(Path(__file__).resolve().parent))
from leaf_gain_pipeline import (  # noqa: E402
    NOISE_ABS_COL,
    NOISE_RAW_COL,
    NOISE_SPECTRUM_INDEX,
    POSITION,
    SPECTRUM_TO_WAVELENGTH,
    SPECTRUM_TO_WAVELENGTH_NM,
    TARGET_SENSOR,
    VALID_SPECTRUM_INDICES,
    average_gains_for_plant,
    get_plant_folders,
    load_ground_truth,
    load_index_rows,
    nearest_ground_truth,
    noise_channel_stat_for_plant,
    write_missing_readings_log,
    write_to_excel,
)
from power_conversion import gain_to_power_uw  # noqa: E402

BASE = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment")

RAW_850NM_INDEX = 3
RAW_VOLTAGE_OFFSET = 0.07
RAW_VOLTAGE_OFFSET_850NM = 0.15

# 850nm (SpectrumIndex 3) had essentially no illuminated signal for the first
# part of the R3 (first planting) dark-room session: raw Gain sits right at
# the assumed dark baseline (~0.15V) for Plant_001 through Plant_018 (Plots
# 101-106, 201-203) -- e.g. leaf_001.csv reads 0.150-0.161V there, vs. a
# normal ~1.5-2.5V from Plant_020 onward (Plant_019 separately has no raw
# leaf files at all -- an unrelated gap). Every other channel (660-780nm) is
# normal-magnitude for these same plants, so this is isolated to the 850nm
# LED/channel specifically -- most likely it was off or disconnected for that
# part of the session and came back partway through. Confirmed NOT present in
# the R1 second-planting session (normal ~1.5-1.7V throughout). Treated as
# missing (NaN) rather than "corrected": there is no real signal in the raw
# CSVs to recover for these plants at this wavelength.
R3_850NM_DEAD_PLANTS = {f"Plant_{i:03d}" for i in range(1, 19)}

# index_sheet_transcribed.xlsx (referenced in earlier notes) no longer exists;
# index_sheet.xlsx has the identical Plot Treatment / Plant # / Date columns
# and row count (72), so it's used directly (see process_07_27_2026_outdoor.py,
# which established this for the same date's outdoor/light processing).
INDEX_SHEET = BASE / "Biomass_Sampling" / "07_27_2026" / "index_sheet.xlsx"

BATCHES = {
    "07_27_2026_R3": dict(
        raw_dir=BASE / "Biomass_Sampling" / "07_27_2026" / "R3_first_planting" / "Dark_env",
        row_offset=0,
```

```
        row_count=36,
        sampling_date="07/27/26",
        planting="First",
        version_label="R3",
        out_dark=BASE / "Processed_Biomass_Sampling_Data" / "Dark_env" / "07_27_2026_R3" / "Data" / "
    ↪ Dark_07_27_2026_R3.xlsx",
    ),
    "07_27_2026_R1": dict(
        raw_dir=BASE / "Biomass_Sampling" / "07_27_2026" / "R1_Second_planting" / "Dark_env",
        row_offset=36,
        row_count=36,
        sampling_date="07/27/26",
        planting="Second",
        version_label="R1",
        out_dark=BASE / "Processed_Biomass_Sampling_Data" / "Dark_env" / "07_27_2026_R1" / "Data" / "
    ↪ Dark_07_27_2026_R1.xlsx",
    ),
}


def _collect_leaf_gains_raw_voltage(plant_folder, position, target_sensor, null_indices=None):
    """Same as leaf_gain_pipeline._collect_leaf_gains, but subtracts the
    raw-room dark voltage offset from each channel's Gain -- per leaf frame,
    before averaging -- before the shared responsivity-power conversion.

    null_indices: SpectrumIndex values to exclude entirely for this plant
    (used for the R3 dead-850nm-channel plants -- see R3_850NM_DEAD_PLANTS).
    """
    null_indices = null_indices or set()
    position_path = plant_folder / position
    if not position_path.exists():
        print(f"    WARNING: {position_path} not found -- skipping.")
        return None

    csv_files = sorted(position_path.glob("leaf_*.csv"))
    if not csv_files:
        print(f"    WARNING: No leaf CSVs in {position_path} -- skipping.")
        return None

    all_gains = []
    for csv_path in csv_files:
        df = pd.read_csv(csv_path)
        sensor_df = df[df["Sensor"] == target_sensor]
        gains = sensor_df.set_index("SpectrumIndex")["Gain"]
        power = pd.Series({
            idx: gain_to_power_uw(
                v - (RAW_VOLTAGE_OFFSET_850NM if idx == RAW_850NM_INDEX else RAW_VOLTAGE_OFFSET),
                SPECTRUM_TO_WAVELENGTH_NM[idx],
            )
            for idx, v in gains.items() if idx in SPECTRUM_TO_WAVELENGTH_NM and idx not in null_indices
        })
        all_gains.append(power)

    return pd.concat(all_gains, axis=1) if all_gains else None


def process_batch(name):
    cfg = BATCHES[name]
    print(f"\n=== Batch: {name} (planting={cfg['planting']}, dark room only) ===")

    plant_folders = get_plant_folders(cfg["raw_dir"])
    index_rows = load_index_rows(INDEX_SHEET, growth_stage_filter=None,
                                 row_offset=cfg["row_offset"], row_count=cfg["row_count"])

    print(f"Found {len(plant_folders)} plant folders, {len(index_rows)} matching index rows.")
    if len(plant_folders) != len(index_rows):
        print(f"[!]  MISMATCH: {len(plant_folders)} folders vs {len(index_rows)} index rows -- "
              f"rows will be truncated to the shorter of the two.")
```

```
    ground_truth = load_ground_truth()
    records = []
    for folder, idx_row in zip(plant_folders, index_rows):
        treatment_id = idx_row["plot_treatment"]
        plant_num = idx_row["plant_number"]
        print(f"  {folder.name} -> Plot {treatment_id}, {plant_num}")

        null_indices = {RAW_850NM_INDEX} if (name == "07_27_2026_R3" and folder.name in R3_850NM_DEAD_PLANTS)
    ↪ else None
        combined = _collect_leaf_gains_raw_voltage(folder, POSITION, TARGET_SENSOR, null_indices=null_indices)
        gains = average_gains_for_plant(folder, POSITION, TARGET_SENSOR, VALID_SPECTRUM_INDICES,
                                        stat="mean", floor_missing=False, combined=combined,
                                        drop_negative_leaf_frames=False)
        noise_raw, noise_abs = noise_channel_stat_for_plant(
            folder, POSITION, TARGET_SENSOR, NOISE_SPECTRUM_INDEX, stat="mean", combined=combined
        )

        record = {
            "ID": treatment_id,
            "Plant Number": plant_num,
            "Date": cfg["sampling_date"],
            "Planting": cfg["planting"],
        }
        if gains:
            for spec_idx, wl_col in SPECTRUM_TO_WAVELENGTH.items():
                val = gains.get(spec_idx, None)
                record[wl_col] = None if (val is None or pd.isna(val)) else val
        else:
            for wl_col in SPECTRUM_TO_WAVELENGTH.values():
                record[wl_col] = None
            print(f"    WARNING: No data for {folder.name}")

        record[NOISE_RAW_COL] = None if (noise_raw is None or pd.isna(noise_raw)) else noise_raw
        record[NOISE_ABS_COL] = None if (noise_abs is None or pd.isna(noise_abs)) else noise_abs

        no3, gt_date, gap_days = nearest_ground_truth(ground_truth, treatment_id, plant_num, cfg["
    ↪ sampling_date"])
        record["Nitrogen Content"] = no3
        record["N Ground Truth Date"] = gt_date
        record["N Ground Truth Gap (days)"] = gap_days
        records.append(record)

    write_to_excel(cfg["out_dark"], records, SPECTRUM_TO_WAVELENGTH)
    log_path = cfg["out_dark"].parent / f"Dark_{cfg['version_label']}_missing_readings_log.csv"
    write_missing_readings_log(log_path, records, SPECTRUM_TO_WAVELENGTH)
    return records


if __name__ == "__main__":
    names = sys.argv[1:] or list(BATCHES.keys())
    for n in names:
        if n in BATCHES:
            process_batch(n)
```

File: `Field_experiment/Source_codes/Data_processing/process_07_27_2026_outdoor.py`

Outdoor processing of the 27 July 2026 rounds.

```
"""
07/27/2026 batch processor -- R3 first planting + R1 second planting,
OUTDOOR (ambient daylight) room.

process_07_27_2026.py already processes the dark room for this date; its
docstring notes the outdoor/light room was "intentionally left untouched."
This script fills that gap, reusing the same raw-voltage offset correction
```

```
(this batch's raw data has not gone through the firmware's usual baseline
correction, same as process_07_27_2026.py: -0.07 V per channel, -0.15 V at
850 nm/SpectrumIndex 3) and the project's standard outdoor/light-room
aggregation settings (inverse-noise-weighted mean, floor_missing, and
drop_negative_leaf_frames -- see process_env() in leaf_gain_pipeline.py).

As with the dark-room batch, white-reference correction is intentionally NOT
applied here, so there is no Reflectance output for this date -- only
detected optical power (uW).

Index sheet: Biomass_Sampling/07_27_2026/index_sheet.xlsx (72 rows: 1-36 =
R3_first_planting, 37-72 = R1_Second_planting), same file process_07_27_2026.py
describes (the "_transcribed" copy it references no longer exists; this plain
index_sheet.xlsx has the identical Plot Treatment / Plant # / Date columns).

Usage:
    python process_07_27_2026_outdoor.py                  # both plantings
    python process_07_27_2026_outdoor.py 07_27_2026_R3     # one named batch
"""

import sys
from pathlib import Path

import numpy as np
import pandas as pd

sys.path.insert(0, str(Path(__file__).resolve().parent))
from leaf_gain_pipeline import (  # noqa: E402
    NOISE_ABS_COL,
    NOISE_RAW_COL,
    NOISE_SPECTRUM_INDEX,
    NOISE_WEIGHT_EPSILON,
    POSITION,
    SPECTRUM_TO_WAVELENGTH,
    SPECTRUM_TO_WAVELENGTH_NM,
    TARGET_SENSOR,
    VALID_SPECTRUM_INDICES,
    get_plant_folders,
    load_ground_truth,
    load_index_rows,
    nearest_ground_truth,
    write_missing_readings_log,
    write_to_excel,
)
from power_conversion import gain_to_power_uw  # noqa: E402

BASE = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment")

RAW_850NM_INDEX = 3
RAW_VOLTAGE_OFFSET = 0.07
RAW_VOLTAGE_OFFSET_850NM = 0.15

INDEX_SHEET = BASE / "Biomass_Sampling" / "07_27_2026" / "index_sheet.xlsx"

BATCHES = {
    "07_27_2026_R3": dict(
        raw_dir=BASE / "Biomass_Sampling" / "07_27_2026" / "R3_first_planting" / "Light_env",
        row_offset=0,
        row_count=36,
        sampling_date="07/27/26",
        planting="First",
        version_label="R3",
        out_light=BASE / "Processed_Biomass_Sampling_Data" / "Light_env" / "07_27_2026_R3" / "Data" / "
    ↪ Light_07_27_2026_R3.xlsx",
    ),
    "07_27_2026_R1": dict(
        raw_dir=BASE / "Biomass_Sampling" / "07_27_2026" / "R1_Second_planting" / "Light_room",
```

```
        row_offset=36,
        row_count=36,
        sampling_date="07/27/26",
        planting="Second",
        version_label="R1",
        out_light=BASE / "Processed_Biomass_Sampling_Data" / "Light_env" / "07_27_2026_R1" / "Data" / "
    ↪ Light_07_27_2026_R1.xlsx",
    ),
}


def _collect_leaf_gains_raw_voltage(plant_folder, position, target_sensor):
    """Same as leaf_gain_pipeline._collect_leaf_gains, but subtracts the
    raw-room dark voltage offset from each channel's Gain -- per leaf frame,
    before averaging -- before the shared responsivity-power conversion.
    Identical to process_07_27_2026.py's helper of the same name."""
    position_path = plant_folder / position
    if not position_path.exists():
        print(f"    WARNING: {position_path} not found -- skipping.")
        return None

    csv_files = sorted(position_path.glob("leaf_*.csv"))
    if not csv_files:
        print(f"    WARNING: No leaf CSVs in {position_path} -- skipping.")
        return None

    all_gains = []
    for csv_path in csv_files:
        df = pd.read_csv(csv_path)
        sensor_df = df[df["Sensor"] == target_sensor]
        gains = sensor_df.set_index("SpectrumIndex")["Gain"]
        power = pd.Series({
            idx: gain_to_power_uw(
                v - (RAW_VOLTAGE_OFFSET_850NM if idx == RAW_850NM_INDEX else RAW_VOLTAGE_OFFSET),
                SPECTRUM_TO_WAVELENGTH_NM[idx],
            )
            for idx, v in gains.items() if idx in SPECTRUM_TO_WAVELENGTH_NM
        })
        all_gains.append(power)

    return pd.concat(all_gains, axis=1) if all_gains else None


def _average_gains_outdoor(combined, valid_indices):
    """Outdoor/light-room aggregation: inverse-840nm-noise-weighted mean,
    negative per-leaf frames dropped before aggregation, residual negative
    plant-level results marked missing. Mirrors process_env()'s is_light
    branch in leaf_gain_pipeline.py."""
    channels = combined.loc[combined.index.isin(valid_indices)]
    channels = channels.where(channels >= 0)  # drop_negative_leaf_frames

    if NOISE_SPECTRUM_INDEX in combined.index:
        noise = combined.loc[NOISE_SPECTRUM_INDEX].abs()
        weights = 1.0 / (NOISE_WEIGHT_EPSILON + noise)
    else:
        weights = pd.Series(1.0, index=channels.columns)

    def _weighted_mean_skipna(row):
        valid = row.notna()
        if not valid.any():
            return np.nan
        w = weights.loc[valid]
        return np.average(row.loc[valid], weights=w)

    result = channels.apply(_weighted_mean_skipna, axis=1)
    result = result.where(result >= 0)  # floor_missing
    return result.to_dict()
```

```
def _noise_stat_outdoor(combined):
    if combined is None or NOISE_SPECTRUM_INDEX not in combined.index:
        return None, None
    readings = combined.loc[NOISE_SPECTRUM_INDEX]
    weights = 1.0 / (NOISE_WEIGHT_EPSILON + readings.abs())
    valid = readings.notna()
    if not valid.any():
        return None, None
    raw_stat = np.average(readings.loc[valid], weights=weights.loc[valid])
    abs_stat = np.average(readings.abs().loc[valid], weights=weights.loc[valid])
    return raw_stat, abs_stat


def process_batch(name):
    cfg = BATCHES[name]
    print(f"\n=== Batch: {name} (planting={cfg['planting']}, outdoor/light room) ===")

    plant_folders = get_plant_folders(cfg["raw_dir"])
    index_rows = load_index_rows(INDEX_SHEET, growth_stage_filter=None,
                                 row_offset=cfg["row_offset"], row_count=cfg["row_count"])

    print(f"Found {len(plant_folders)} plant folders, {len(index_rows)} matching index rows.")
    if len(plant_folders) != len(index_rows):
        print(f"[!]  MISMATCH: {len(plant_folders)} folders vs {len(index_rows)} index rows -- "
              f"rows will be truncated to the shorter of the two.")

    ground_truth = load_ground_truth()
    records = []
    for folder, idx_row in zip(plant_folders, index_rows):
        treatment_id = idx_row["plot_treatment"]
        plant_num = idx_row["plant_number"]
        print(f"  {folder.name} -> Plot {treatment_id}, {plant_num}")

        combined = _collect_leaf_gains_raw_voltage(folder, POSITION, TARGET_SENSOR)

        record = {
            "ID": treatment_id,
            "Plant Number": plant_num,
            "Date": cfg["sampling_date"],
            "Planting": cfg["planting"],
        }
        if combined is not None:
            gains = _average_gains_outdoor(combined, VALID_SPECTRUM_INDICES)
            for spec_idx, wl_col in SPECTRUM_TO_WAVELENGTH.items():
                val = gains.get(spec_idx, None)
                record[wl_col] = None if (val is None or pd.isna(val)) else val
            noise_raw, noise_abs = _noise_stat_outdoor(combined)
        else:
            for wl_col in SPECTRUM_TO_WAVELENGTH.values():
                record[wl_col] = None
            noise_raw, noise_abs = None, None
            print(f"    WARNING: No data for {folder.name}")

        record[NOISE_RAW_COL] = None if (noise_raw is None or pd.isna(noise_raw)) else noise_raw
        record[NOISE_ABS_COL] = None if (noise_abs is None or pd.isna(noise_abs)) else noise_abs

        no3, gt_date, gap_days = nearest_ground_truth(ground_truth, treatment_id, plant_num, cfg["
    ↪ sampling_date"])
        record["Nitrogen Content"] = no3
        record["N Ground Truth Date"] = gt_date
        record["N Ground Truth Gap (days)"] = gap_days
        records.append(record)

    write_to_excel(cfg["out_light"], records, SPECTRUM_TO_WAVELENGTH)
    log_path = cfg["out_light"].parent / f"Light_{cfg['version_label']}_missing_readings_log.csv"
    write_missing_readings_log(log_path, records, SPECTRUM_TO_WAVELENGTH)
    return records
```

```
if __name__ == "__main__":
    names = sys.argv[1:] or list(BATCHES.keys())
    for n in names:
        if n in BATCHES:
            process_batch(n)
```

File: `Field_experiment/Source_codes/Data_processing/process_08_17_2026.py`

Processing of the 17 August 2026 round (R3 May planting) and the Plot 306 senesced-leaf re-sampling.

```
"""
08/17/2026 batch processor -- R3 second planting (May-planted plots
401-606), plus a 3-plant "extreme case" re-sample of Plot 306 (April/first
planting) tacked onto the end of the same raw session and index sheet.
========================================================================
Raw Gain baseline check (SpectrumIndex 6, the no-LED 840nm noise probe):
dark_env readings sit at ~0.07-0.08V and light readings are always positive
(~0.2-1.2V) -- the same signature as 07_27_2026 (not near-zero like the
firmware-corrected BATCHES entries in leaf_gain_pipeline.py), so this batch
gets the same per-frame raw-voltage offset correction as
process_07_27_2026.py / process_07_27_2026_outdoor.py: -0.07V for every
channel except 850nm (SpectrumIndex 3), which gets -0.15V, before the shared
responsivity-power conversion.

Index sheet: Biomass_Sampling/08_17_2026/R3_second_planting/Sample_index_sheet/
Sample_index_08-17-26.xlsx -- a different column shape (Index, Plant, Plot,
Treatment, Dates, ...) than load_index_rows() expects (Plot Treatment,
Plant #, Date), so this file has its own loader instead. Rows 1-36 (Plot
401...606) = the R3 second-planting batch; rows 37-39 (Plot 306, P1/P2/P3)
are a re-sample of an April-planting plot, processed here as a separate
"extreme" batch (single plot, single N=0 treatment, 3 individual plants) and
routed to its own Dark_env/extreem, Light_env/extreem output folders.

Position-folder fallback: light/Plant_011's "lower" subfolder (the position
every other plant/batch reads) has 0 leaf CSVs, but "upper" has 10 -- data
clearly landed in the wrong position folder for this one plant. Confirmed
with the user to use whichever position folder actually has data rather
than treating it as missing.

Usage:
    python process_08_17_2026.py                    # both batches
    python process_08_17_2026.py 08_17_2026_R3       # main batch only
    python process_08_17_2026.py 08_17_2026_extreme  # extreme case only
"""

import sys
from pathlib import Path

import pandas as pd
from openpyxl import load_workbook

sys.path.insert(0, str(Path(__file__).resolve().parent))
from leaf_gain_pipeline import (  # noqa: E402
    NOISE_ABS_COL,
    NOISE_RAW_COL,
    NOISE_SPECTRUM_INDEX,
    POSITION,
    SPECTRUM_TO_WAVELENGTH,
    SPECTRUM_TO_WAVELENGTH_NM,
    TARGET_SENSOR,
    VALID_SPECTRUM_INDICES,
    average_gains_for_plant,
```

```
    get_plant_folders,
    load_ground_truth,
    nearest_ground_truth,
    noise_channel_stat_for_plant,
    write_missing_readings_log,
    write_to_excel,
)
from power_conversion import gain_to_power_uw  # noqa: E402

BASE = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment")

RAW_850NM_INDEX = 3
RAW_VOLTAGE_OFFSET = 0.07
RAW_VOLTAGE_OFFSET_850NM = 0.15

FALLBACK_POSITIONS = ("middle", "upper")

SAMPLE_ROOT = BASE / "Biomass_Sampling" / "08_17_2026" / "R3_second_planting"
INDEX_SHEET = SAMPLE_ROOT / "Sample_index_sheet" / "Sample_index_08-17-26.xlsx"

BATCHES = {
    "08_17_2026_R3": dict(
        raw_dark=SAMPLE_ROOT / "dark_env",
        raw_light=SAMPLE_ROOT / "light",
        row_offset=0,
        row_count=36,
        sampling_date="08/17/26",
        planting="Second",
        version_label="R3",
        out_dark=BASE / "Processed_Biomass_Sampling_Data" / "Dark_env" / "08_17_2026_R3" / "Data" / "
    ↪ Dark_08_17_2026_R3.xlsx",
        out_light=BASE / "Processed_Biomass_Sampling_Data" / "Light_env" / "08_17_2026_R3" / "Data" / "
    ↪ Light_08_17_2026_R3.xlsx",
    ),
    "08_17_2026_extreme": dict(
        raw_dark=SAMPLE_ROOT / "dark_env",
        raw_light=SAMPLE_ROOT / "light",
        row_offset=36,
        row_count=3,
        sampling_date="08/17/26",
        planting="First",  # Plot 306 belongs to the April/first-planting cohort (2026 Field Maps.xlsx)
        version_label="Extreme_Plot306",
        out_dark=BASE / "Processed_Biomass_Sampling_Data" / "Dark_env" / "extreem" / "Data" / "
    ↪ Dark_extreme_Plot306.xlsx",
        out_light=BASE / "Processed_Biomass_Sampling_Data" / "Light_env" / "extreem" / "Data" / "
    ↪ Light_extreme_Plot306.xlsx",
    ),
}


# -- Index sheet loading (schema: Index, Plant, Plot, Treatment, Dates, ...) -

def load_08_17_index_rows(index_path, row_offset, row_count):
    wb = load_workbook(index_path, read_only=True)
    ws = wb.active
    header = [cell.value for cell in next(ws.iter_rows(min_row=1, max_row=1))]
    col_map = {name: idx for idx, name in enumerate(header)}

    rows = []
    for row in ws.iter_rows(min_row=2, values_only=True):
        rows.append({
            "plot_treatment": row[col_map["Plot"]],
            "plant_number": row[col_map["Plant"]],
        })
    wb.close()
    return rows[row_offset:row_offset + row_count]
```

```
# -- Gain extraction with raw-voltage offset + position fallback ------------

def _collect_leaf_gains_raw_voltage(plant_folder, position, target_sensor):
    """Same as leaf_gain_pipeline._collect_leaf_gains, but subtracts the
    raw-room dark voltage offset from each channel's Gain -- per leaf frame,
    before averaging -- before the shared responsivity-power conversion.

    If `position` has no leaf CSVs, falls back to the first of
    FALLBACK_POSITIONS that does (handles light/Plant_011, whose readings
    landed in 'upper' instead of 'lower').
    """
    used_position = position
    position_path = plant_folder / position
    csv_files = sorted(position_path.glob("leaf_*.csv")) if position_path.exists() else []

    if not csv_files:
        # Pick whichever fallback position folder has the most leaf readings,
        # not just the first non-empty one (e.g. light/Plant_011 has 1 leaf
        # in 'middle' but 10 in 'upper' -- 'upper' is clearly where the real
        # session landed).
        candidates = []
        for fb in FALLBACK_POSITIONS:
            fb_path = plant_folder / fb
            fb_files = sorted(fb_path.glob("leaf_*.csv")) if fb_path.exists() else []
            if fb_files:
                candidates.append((fb, fb_files))
        if candidates:
            used_position, csv_files = max(candidates, key=lambda c: len(c[1]))

    if not csv_files:
        print(f"    WARNING: No leaf CSVs in {plant_folder.name} (checked {position}, {FALLBACK_POSITIONS}) --
    ↪  skipping.")
        return None
    if used_position != position:
        print(f"    NOTE: {plant_folder.name}: '{position}' empty, using '{used_position}' instead ({len(
    ↪ csv_files)} leaves).")

    all_gains = []
    for csv_path in csv_files:
        df = pd.read_csv(csv_path)
        sensor_df = df[df["Sensor"] == target_sensor]
        gains = sensor_df.set_index("SpectrumIndex")["Gain"]
        power = pd.Series({
            idx: gain_to_power_uw(
                v - (RAW_VOLTAGE_OFFSET_850NM if idx == RAW_850NM_INDEX else RAW_VOLTAGE_OFFSET),
                SPECTRUM_TO_WAVELENGTH_NM[idx],
            )
            for idx, v in gains.items() if idx in SPECTRUM_TO_WAVELENGTH_NM
        })
        all_gains.append(power)

    return pd.concat(all_gains, axis=1) if all_gains else None


def process_env(raw_env_path, out_path, index_rows, row_offset, row_count, sampling_date, planting,
                version_label, env_label, ground_truth):
    print(f"\n--- {env_label} : {raw_env_path} ---")
    all_plant_folders = get_plant_folders(raw_env_path)
    plant_folders = all_plant_folders[row_offset:row_offset + row_count]

    print(f"Found {len(plant_folders)} plant folders, {len(index_rows)} matching index rows.")
    if len(plant_folders) != len(index_rows):
        print(f"[!]  MISMATCH: {len(plant_folders)} folders vs {len(index_rows)} index rows -- "
              f"rows will be truncated to the shorter of the two.")

    is_light = env_label == "Light"
    records = []
```

```
    for folder, idx_row in zip(plant_folders, index_rows):
        treatment_id = idx_row["plot_treatment"]
        plant_num = idx_row["plant_number"]
        print(f"  {folder.name} -> Plot {treatment_id}, {plant_num}")

        combined = _collect_leaf_gains_raw_voltage(folder, POSITION, TARGET_SENSOR)
        gains = average_gains_for_plant(
            folder, POSITION, TARGET_SENSOR, VALID_SPECTRUM_INDICES,
            stat="weighted" if is_light else "mean",
            floor_missing=is_light, combined=combined,
            drop_negative_leaf_frames=is_light,
        )
        noise_raw, noise_abs = noise_channel_stat_for_plant(
            folder, POSITION, TARGET_SENSOR, NOISE_SPECTRUM_INDEX,
            stat="weighted" if is_light else "mean", combined=combined,
        )

        record = {
            "ID": treatment_id,
            "Plant Number": plant_num,
            "Date": sampling_date,
            "Planting": planting,
        }
        if gains:
            for spec_idx, wl_col in SPECTRUM_TO_WAVELENGTH.items():
                val = gains.get(spec_idx, None)
                record[wl_col] = None if (val is None or pd.isna(val)) else val
        else:
            for wl_col in SPECTRUM_TO_WAVELENGTH.values():
                record[wl_col] = None
            print(f"    WARNING: No data for {folder.name}")

        record[NOISE_RAW_COL] = None if (noise_raw is None or pd.isna(noise_raw)) else noise_raw
        record[NOISE_ABS_COL] = None if (noise_abs is None or pd.isna(noise_abs)) else noise_abs

        no3, gt_date, gap_days = nearest_ground_truth(ground_truth, treatment_id, plant_num, sampling_date)
        record["Nitrogen Content"] = no3
        record["N Ground Truth Date"] = gt_date
        record["N Ground Truth Gap (days)"] = gap_days
        records.append(record)

    write_to_excel(out_path, records, SPECTRUM_TO_WAVELENGTH)
    log_path = out_path.parent / f"{env_label}_{version_label}_missing_readings_log.csv"
    write_missing_readings_log(log_path, records, SPECTRUM_TO_WAVELENGTH)
    return records


def process_batch(name):
    cfg = BATCHES[name]
    print(f"\n=== Batch: {name} (planting={cfg['planting']}) ===")

    index_rows = load_08_17_index_rows(INDEX_SHEET, cfg["row_offset"], cfg["row_count"])
    ground_truth = load_ground_truth()

    process_env(cfg["raw_dark"], cfg["out_dark"], index_rows, cfg["row_offset"], cfg["row_count"],
                cfg["sampling_date"], cfg["planting"], cfg["version_label"], "Dark", ground_truth)
    process_env(cfg["raw_light"], cfg["out_light"], index_rows, cfg["row_offset"], cfg["row_count"],
                cfg["sampling_date"], cfg["planting"], cfg["version_label"], "Light", ground_truth)


if __name__ == "__main__":
    names = sys.argv[1:] or list(BATCHES.keys())
    for n in names:
        if n in BATCHES:
            process_batch(n)
```

File: `Field_experiment/Source_codes/White_reference_correction/apply_white_reference_07_27_2026.py`

White-reference correction of the 27 July 2026 rounds.

```
"""
Applies white-reference correction to 07/27/2026 (R3 first planting, R1 second
planting) -- the one date pair white_reference_correction.py's BATCH_TARGETS
skipped, because that date's raw data needed the special per-frame voltage
offset correction in process_07_27_2026.py / process_07_27_2026_outdoor.py
before it could be converted to detected power (see those scripts' docstrings).

That offset correction was specifically designed and cross-checked against the
index sheet's own baseline table to bring 07/27/2026's power values onto the
same physical footing as every other batch (i.e. equivalent to the standard
firmware-corrected pipeline). Once that's done, dividing by the same white
reference used everywhere else is valid -- there's nothing date-specific left
to reconcile.

Reads from each batch's Detected_Power/ file specifically (not the main Data/
file), since Detected_Power/ is the one copy white_reference_correction.py
never overwrites -- it's the safe, permanent pre-correction source regardless
of whether this script has already been run for a given date. Writes results
to a new Reflectance/ subfolder and also refreshes the main Data/ file, matching
the convention every other batch already follows.

Usage:
    python apply_white_reference_07_27_2026.py
"""

import sys
from pathlib import Path

import pandas as pd

sys.path.insert(0, str(Path(__file__).resolve().parent))
from white_reference_correction import (  # noqa: E402
    WHITE_REF_PATH,
    WL_COLS,
    apply_correction,
    parse_white_reference,
)

PROCESSED_ROOT = Path(
    r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\Field_experiment"
    r"\Processed_Biomass_Sampling_Data"
)

BATCH_TARGETS_072726 = [
    ("07_27_2026_R3",
     PROCESSED_ROOT / "Dark_env" / "07_27_2026_R3" / "Data" / "Dark_07_27_2026_R3.xlsx",
     PROCESSED_ROOT / "Dark_env" / "07_27_2026_R3" / "Data" / "Detected_Power" / "
    ↪ Dark_07_27_2026_R3_detected_power_uW.xlsx",
     PROCESSED_ROOT / "Light_env" / "07_27_2026_R3" / "Data" / "Light_07_27_2026_R3.xlsx",
     PROCESSED_ROOT / "Light_env" / "07_27_2026_R3" / "Data" / "Detected_Power" / "
    ↪ Light_07_27_2026_R3_detected_power_uW.xlsx"),
    ("07_27_2026_R1",
     PROCESSED_ROOT / "Dark_env" / "07_27_2026_R1" / "Data" / "Dark_07_27_2026_R1.xlsx",
     PROCESSED_ROOT / "Dark_env" / "07_27_2026_R1" / "Data" / "Detected_Power" / "
    ↪ Dark_07_27_2026_R1_detected_power_uW.xlsx",
     PROCESSED_ROOT / "Light_env" / "07_27_2026_R1" / "Data" / "Light_07_27_2026_R1.xlsx",
     PROCESSED_ROOT / "Light_env" / "07_27_2026_R1" / "Data" / "Detected_Power" / "
    ↪ Light_07_27_2026_R1_detected_power_uW.xlsx"),
]
```

```
def correct_one(power_path, out_data_path, wr_corr, label):
    power_path = Path(power_path)
    if not power_path.exists():
        print(f"  WARNING: {power_path} not found -- skipping {label}.")
        return
    df = pd.read_excel(power_path)
    print(f"  {label}: {len(df)} rows (source: {power_path.name})")
    corr = apply_correction(df, wr_corr, label)

    out_data_path = Path(out_data_path)
    reflectance_dir = out_data_path.parent / "Reflectance"
    reflectance_dir.mkdir(parents=True, exist_ok=True)
    corr.to_excel(reflectance_dir / f"{out_data_path.stem}_reflectance.xlsx", index=False)
    corr.to_excel(out_data_path, index=False)
    print(f"  Saved: {reflectance_dir / f'{out_data_path.stem}_reflectance.xlsx'}")
    print(f"  Saved: {out_data_path}")


def main():
    print("Parsing white reference data (same reference used for every other batch)...")
    _, wr_corr = parse_white_reference(WHITE_REF_PATH)
    print("\nWhite reference used for normalization (direct detected power):")
    print(wr_corr.round(4).to_string())

    for label, dark_data, dark_power, light_data, light_power in BATCH_TARGETS_072726:
        print(f"\n=== Batch: {label} ===")
        correct_one(dark_power, dark_data, wr_corr, "Dark room")
        correct_one(light_power, light_data, wr_corr, "Outdoor")

    print("\nWhite reference correction complete for 07/27/2026 (R3, R1 second planting).")


if __name__ == "__main__":
    main()
```

File: `Field _ experiment / Source _ codes / White _ reference _ correction / apply _ white _ reference_08_17_2026.py`

White-reference correction of the 17 August 2026 round.

```
"""
Applies white-reference correction to 08/17/2026 (R3 second planting, plus
the Plot 306 "extreme case" re-sample) -- the same situation as
apply_white_reference_07_27_2026.py: this date's raw data needed the special
per-frame voltage offset correction in process_08_17_2026.py before it could
be converted to detected power, so it was kept out of
white_reference_correction.py's BATCH_TARGETS. Once that offset correction
puts the power values on the same physical footing as every other batch,
dividing by the same white reference used everywhere else is valid.

Reads from each batch's Detected_Power/ file specifically (not the main
Data/ file), since Detected_Power/ is the one copy white_reference_correction.py
never overwrites. Writes results to a new Reflectance/ subfolder and also
refreshes the main Data/ file, matching the convention every other batch
follows.

Usage:
    python apply_white_reference_08_17_2026.py
"""

import sys
from pathlib import Path

sys.path.insert(0, str(Path(__file__).resolve().parent))
from white_reference_correction import (  # noqa: E402
```

```
    WHITE_REF_PATH,
    apply_correction,
    parse_white_reference,
)

PROCESSED_ROOT = Path(
    r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\Field_experiment"
    r"\Processed_Biomass_Sampling_Data"
)

BATCH_TARGETS_081726 = [
    ("08_17_2026_R3",
     PROCESSED_ROOT / "Dark_env" / "08_17_2026_R3" / "Data" / "Dark_08_17_2026_R3.xlsx",
     PROCESSED_ROOT / "Dark_env" / "08_17_2026_R3" / "Data" / "Detected_Power" / "
     ↪ Dark_08_17_2026_R3_detected_power_uW.xlsx",
     PROCESSED_ROOT / "Light_env" / "08_17_2026_R3" / "Data" / "Light_08_17_2026_R3.xlsx",
     PROCESSED_ROOT / "Light_env" / "08_17_2026_R3" / "Data" / "Detected_Power" / "
     ↪ Light_08_17_2026_R3_detected_power_uW.xlsx"),
    ("08_17_2026_extreme",
     PROCESSED_ROOT / "Dark_env" / "extreem" / "Data" / "Dark_extreme_Plot306.xlsx",
     PROCESSED_ROOT / "Dark_env" / "extreem" / "Data" / "Detected_Power" / "
     ↪ Dark_extreme_Plot306_detected_power_uW.xlsx",
     PROCESSED_ROOT / "Light_env" / "extreem" / "Data" / "Light_extreme_Plot306.xlsx",
     PROCESSED_ROOT / "Light_env" / "extreem" / "Data" / "Detected_Power" / "
     ↪ Light_extreme_Plot306_detected_power_uW.xlsx"),
]


def correct_one(power_path, out_data_path, wr_corr, label):
    import pandas as pd
    power_path = Path(power_path)
    if not power_path.exists():
        print(f"  WARNING: {power_path} not found -- skipping {label}.")
        return
    df = pd.read_excel(power_path)
    print(f"  {label}: {len(df)} rows (source: {power_path.name})")
    corr = apply_correction(df, wr_corr, label)

    out_data_path = Path(out_data_path)
    reflectance_dir = out_data_path.parent / "Reflectance"
    reflectance_dir.mkdir(parents=True, exist_ok=True)
    corr.to_excel(reflectance_dir / f"{out_data_path.stem}_reflectance.xlsx", index=False)
    corr.to_excel(out_data_path, index=False)
    print(f"  Saved: {reflectance_dir / f'{out_data_path.stem}_reflectance.xlsx'}")
    print(f"  Saved: {out_data_path}")


def main():
    print("Parsing white reference data (same reference used for every other batch)...")
    _, wr_corr = parse_white_reference(WHITE_REF_PATH)
    print("\nWhite reference used for normalization (direct detected power):")
    print(wr_corr.round(4).to_string())

    for label, dark_data, dark_power, light_data, light_power in BATCH_TARGETS_081726:
        print(f"\n=== Batch: {label} ===")
        correct_one(dark_power, dark_data, wr_corr, "Dark room")
        correct_one(light_power, light_data, wr_corr, "Light room")

    print("\nWhite reference correction complete for 08/17/2026 (R3 second planting + extreme).")


if __name__ == "__main__":
    main()
```

File: `Field_experiment/Processed_Biomass_Sampling_Data/Complete_Plant_Level_`

Spectral_Dataset/build_light_dataset.py

Assembles the plant-level outdoor dataset for all six rounds, matched to the dark-enclosure dataset.

```
# -*- coding: utf-8 -*-
"""Build complete_plant_level_spectral_dataset_light.csv, mirroring the
dark version's schema/logic exactly, but sourced from the outdoor/ambient
(Light_env) measurements instead of dark-enclosure.

Source-selection differences from the dark build, established by direct
investigation (not assumed):
  - Shape_Feature_Analysis/Light/* (the "fuller pipeline" used for 3 of the
    dark rounds) exists as folders but is EMPTY for all 4 rounds checked --
    never actually populated. So, unlike dark, ALL 6 light rounds use the
    same source: Light_env/<round>/Data/Reflectance/*.xlsx (plain
    white-reference-corrected reflectance, produced by leaf_gain_pipeline.py
    for the 3 "normal" rounds and by the dedicated
    apply_white_reference_07_27_2026.py / apply_white_reference_08_17_2026.py
    scripts for the other 3, verified by comparing Reflectance/ vs
    Detected_Power/ values directly -- they differ, confirming correction
    was really applied despite one script's own docstring claiming
    otherwise, same situation as the dark build).
  - The R3_April_0727 dead-850nm-channel issue found in the DARK data does
    NOT carry over to light: checked directly, only 4 scattered NaN 850nm
    values in the light R3_April_0727 round (vs. a clean 18-plant block in
    dark), consistent with it being a dark-enclosure-session-specific
    hardware state, not a device-wide failure. So no special-case
    imputation/exclusion is applied here for that round.
  - Metadata (Plot_ID, Plant_Number, Round, Planting_Date, Growth_Stage,
    Block, Nitrogen_Treatment, Nitrate_ppm) is reused directly from the
    already-built, already-verified complete_plant_level_spectral_dataset_dark.csv
    by joining on Plant_ID + Round, rather than re-deriving it -- the plant
    identities, treatments, and lab nitrate results are the same regardless
    of which lighting condition the reflectance was measured under (same
    leaf, same day, same lab sample).
  - The extreme-case Plot 306 senescence re-sample (Light_env/extreem/) is
    excluded here too, for the same reason it's excluded from the dark
    file: kept as a separate, opportunistic, non-RCBD case.

Outdoor measurements are known throughout this project to be noisier / more
often missing than dark-enclosure ones -- NaNs here are left as NaN, not
imputed or dropped, matching the dark file's own convention.
"""
from pathlib import Path

import pandas as pd

ROOT = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment")
PBSD = ROOT / "Processed_Biomass_Sampling_Data"
DARK_CSV = PBSD / "Complete_Plant_Level_Spectral_Dataset" / "complete_plant_level_spectral_dataset_dark.csv"
OUT_CSV = PBSD / "Complete_Plant_Level_Spectral_Dataset" / "complete_plant_level_spectral_dataset_light.csv"

BAND_COLS = ["660 nm", "670 nm", "700 nm", "750 nm", "780 nm", "850 nm"]

ROUNDS = {
    "V10_April_0616": dict(
        path=PBSD / "Light_env/06_16_2026_V10/Data/Reflectance/Light_06_16_2026_V10_reflectance.xlsx",
        pipeline="leaf_gain_pipeline (weighted mean, outdoor) + white_reference_correction",
    ),
    "R1_April_0708": dict(
        path=PBSD / "Light_env/07_08_2026_R1/Data/Reflectance/Light_07_08_2026_R1_reflectance.xlsx",
        pipeline="leaf_gain_pipeline (weighted mean, outdoor) + white_reference_correction",
    ),
    "V10_May_0708": dict(
```

```
        path=PBSD / "Light_env/07_08_2026_v8/Data/Reflectance/Light_07_08_2026_v8_reflectance.xlsx",
        pipeline="leaf_gain_pipeline (weighted mean, outdoor) + white_reference_correction",
    ),
    "R3_April_0727": dict(
        path=PBSD / "Light_env/07_27_2026_R3/Data/Reflectance/Light_07_27_2026_R3_reflectance.xlsx",
        pipeline="process_07_27_2026_outdoor + apply_white_reference_07_27_2026",
    ),
    "R1_May_0727": dict(
        path=PBSD / "Light_env/07_27_2026_R1/Data/Reflectance/Light_07_27_2026_R1_reflectance.xlsx",
        pipeline="process_07_27_2026_outdoor + apply_white_reference_07_27_2026",
    ),
    "R3_May_0817": dict(
        path=PBSD / "Light_env/08_17_2026_R3/Data/Reflectance/Light_08_17_2026_R3_reflectance.xlsx",
        pipeline="apply_white_reference_08_17_2026",
    ),
}


def load_round(round_name, cfg):
    df = pd.read_excel(cfg["path"])
    df["Plant_ID"] = df["ID"].astype(str) + "_" + df["Plant Number"].astype(str)
    df["Round"] = round_name
    df["Source_Pipeline"] = cfg["pipeline"]
    keep = ["Plant_ID", "Round", "Source_Pipeline"] + BAND_COLS
    out = df[keep].copy()
    dup = out["Plant_ID"].duplicated()
    if dup.any():
        print(f"  WARNING [{round_name}]: {dup.sum()} duplicate Plant_ID rows, keeping first")
        out = out[~dup]
    return out


def main():
    dark = pd.read_csv(DARK_CSV)
    meta_cols = ["Plant_ID", "Plot_ID", "Plant_Number", "Round", "Planting_Date",
                 "Growth_Stage", "Block", "Nitrogen_Treatment", "Nitrate_ppm"]
    meta = dark[meta_cols].copy()
    print(f"Loaded dark-dataset metadata: {len(meta)} rows")

    frames = []
    for round_name, cfg in ROUNDS.items():
        r = load_round(round_name, cfg)
        print(f"{round_name}: {len(r)} plants loaded from {cfg['path'].name}")
        frames.append(r)
    light = pd.concat(frames, ignore_index=True)
    print(f"\nTotal light-condition rows loaded: {len(light)}")

    merged = meta.merge(light, on=["Plant_ID", "Round"], how="outer", indicator=True)
    print("\nMerge diagnostic (meta vs light-reflectance coverage):")
    print(merged["_merge"].value_counts())

    only_meta = merged[merged["_merge"] == "left_only"]
    only_light = merged[merged["_merge"] == "right_only"]
    if len(only_meta):
        print(f"\n{len(only_meta)} plants have dark-side metadata but NO light reflectance row "
              f"(will appear with all-NaN bands):")
        print(only_meta[["Plant_ID", "Round"]].to_string(index=False))
    if len(only_light):
        print(f"\n{len(only_light)} plants have light reflectance but NO matching dark-side metadata "
              f"(unexpected -- inspect):")
        print(only_light[["Plant_ID", "Round"]].to_string(index=False))

    merged = merged.drop(columns=["_merge"])

    # ---- Derived indices (same formulas as the dark build) ----
    def safe_ndvi(nir, red):
        return (nir - red) / (nir + red)
```

```
    merged["NDVI_850_670"] = safe_ndvi(merged["850 nm"], merged["670 nm"])
    merged["NDRE_780_700"] = safe_ndvi(merged["780 nm"], merged["700 nm"])
    merged["NDRE_750_700"] = safe_ndvi(merged["750 nm"], merged["700 nm"])
    merged["NIR_Red_Ratio_850_670"] = merged["850 nm"] / merged["670 nm"]
    merged["RedEdge_Ratio_750_700"] = merged["750 nm"] / merged["700 nm"]
    merged["RedEdge_Diff_750_700"] = merged["750 nm"] - merged["700 nm"]
    merged["Slope_660_700"] = (merged["700 nm"] - merged["660 nm"]) / (700 - 660)
    merged["Slope_700_750"] = (merged["750 nm"] - merged["700 nm"]) / (750 - 700)
    merged["Slope_750_780"] = (merged["780 nm"] - merged["750 nm"]) / (780 - 750)
    merged["Slope_780_850"] = (merged["850 nm"] - merged["780 nm"]) / (850 - 780)

    merged["Notes"] = None
    missing_band_mask = merged[BAND_COLS].isna().any(axis=1)
    merged.loc[missing_band_mask, "Notes"] = "One or more bands missing (typical for outdoor/ambient-light
    ↪ measurement)"

    out_cols = ["Plant_ID", "Plot_ID", "Plant_Number", "Round", "Planting_Date", "Growth_Stage",
                "Block", "Nitrogen_Treatment", "660 nm", "670 nm", "700 nm", "750 nm", "780 nm",
                "850 nm", "NDVI_850_670", "NDRE_780_700", "NDRE_750_700", "NIR_Red_Ratio_850_670",
                "RedEdge_Ratio_750_700", "RedEdge_Diff_750_700", "Slope_660_700", "Slope_700_750",
                "Slope_750_780", "Slope_780_850", "Nitrate_ppm", "Source_Pipeline", "Notes"]
    # "Date" column exists in dark schema too; light source files also have a Date column,
    # but we dropped it during load_round for simplicity -- re-add from dark's own Date col.
    if "Date" in dark.columns:
        date_map = dark.set_index(["Plant_ID", "Round"])["Date"]
        merged["Date"] = merged.set_index(["Plant_ID", "Round"]).index.map(date_map)
        out_cols.insert(out_cols.index("Nitrogen_Treatment") + 1, "Date")

    final = merged[out_cols].sort_values(["Round", "Plot_ID", "Plant_Number"]).reset_index(drop=True)
    final.to_csv(OUT_CSV, index=False)
    print(f"\nWrote {OUT_CSV}: {len(final)} rows, {len(final.columns)} columns")

    print("\nPer-round row counts:")
    print(final.groupby("Round").size())
    print("\nBand completeness (non-null count / total):")
    for c in BAND_COLS:
        print(f"  {c}: {final[c].notna().sum()} / {len(final)}")
    print("\nNitrate_ppm non-null:", final["Nitrate_ppm"].notna().sum(), "/", len(final))


if __name__ == "__main__":
    main()
```

File: `Field_experiment/Processed_Biomass_Sampling_Data/Complete_Plant_Level_Spectral_Dataset/impute_light_dataset.py`

Imputation of missing outdoor band values (iterative Bayesian ridge imputation, round-mean fallback).

```
# -*- coding: utf-8 -*-
"""Fill missing band values in complete_plant_level_spectral_dataset_light.csv
via interpolation, per explicit instruction. Two tiers, both transparently
flagged in a new Imputed_bands column (never silently mixed with real
measurements):

  Tier 1 (partial missingness, 1-5 of 6 bands present): multivariate
  round-robin regression imputation (sklearn IterativeImputer -- a
  generalization of the cross-band linear regression already used to
  recover the dark dataset's 850nm gap, extended to handle multiple
  simultaneously-missing bands per row by iterating). Fit across all 234
  rows so it can use whatever cross-band structure exists.

  Tier 2 (total missingness, all 6 bands NaN -- 23 rows, 21 of them in
```

```
  R1_May_0727 alone): no signal exists in that row for a regression to
  condition on. Falls back to that band's mean within the SAME Round
  (growth stage + planting + date all shift typical reflectance, so a
  round-level mean is a much closer prior than a dataset-wide mean).

After imputation, all derived indices (NDVI, NDRE, ratios, slopes) are
recomputed from the now-complete bands.
"""
from pathlib import Path

import numpy as np
import pandas as pd
from sklearn.experimental import enable_iterative_imputer  # noqa: F401
from sklearn.impute import IterativeImputer
from sklearn.linear_model import BayesianRidge

CSV = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment\Processed_Biomass_Sampling_Data\Complete_Plant_Level_Spectral_Dataset\
    ↪ complete_plant_level_spectral_dataset_light.csv")
BACKUP = CSV.with_name("complete_plant_level_spectral_dataset_light.BACKUP_20260915_before_impute.csv")
BAND_COLS = ["660 nm", "670 nm", "700 nm", "750 nm", "780 nm", "850 nm"]

df = pd.read_csv(CSV)
df.to_csv(BACKUP, index=False)
print(f"Backed up {len(df)}-row pre-imputation file to {BACKUP}")

n_missing_per_row = df[BAND_COLS].isna().sum(axis=1)
tier1_mask = (n_missing_per_row > 0) & (n_missing_per_row < 6)
tier2_mask = n_missing_per_row == 6
print(f"Tier 1 (partial, regression-imputed): {tier1_mask.sum()} rows")
print(f"Tier 2 (total, round-mean fallback):   {tier2_mask.sum()} rows")
print(f"Complete already:                      {(n_missing_per_row == 0).sum()} rows")

# ---- Tier 1: iterative multivariate regression imputation, fit globally ----
imputer = IterativeImputer(
    estimator=BayesianRidge(), max_iter=25, random_state=42, sample_posterior=False,
)
imputed_array = imputer.fit_transform(df[BAND_COLS])
imputed_df = pd.DataFrame(imputed_array, columns=BAND_COLS, index=df.index)

df_orig = df[BAND_COLS].copy()
for col in BAND_COLS:
    fill_mask = tier1_mask & df_orig[col].isna()
    df.loc[fill_mask, col] = imputed_df.loc[fill_mask, col]
print(f"\nTier 1 applied: filled {sum((tier1_mask & df_orig[c].isna()).sum() for c in BAND_COLS)} "
      f"individual band values across {tier1_mask.sum()} rows.")

# ---- Tier 2: same-round band mean, computed from ORIGINAL (non-imputed) values ----
round_means = df_orig.assign(Round=df["Round"]).groupby("Round")[BAND_COLS].mean()
print("\nRound-level band means used for Tier 2 fallback:")
print(round_means)

for idx in df.index[tier2_mask]:
    rnd = df.loc[idx, "Round"]
    for col in BAND_COLS:
        df.loc[idx, col] = round_means.loc[rnd, col]
print(f"\nTier 2 applied: filled all 6 bands for {tier2_mask.sum()} rows using their round's mean.")

# ---- Flag column ----
df["Imputed_bands"] = "no"
df.loc[tier1_mask, "Imputed_bands"] = df_orig[BAND_COLS].isna().apply(
    lambda row: "regression-partial(" + ",".join(c for c in BAND_COLS if row[c]) + ")", axis=1
)[tier1_mask]
df.loc[tier2_mask, "Imputed_bands"] = "round-mean-fallback(all 6 bands)"

assert df[BAND_COLS].isna().sum().sum() == 0, "bands still missing after imputation"

# ---- Recompute derived indices from the now-complete bands ----
```

```
def safe_ratio(a, b):
    return a / b

df["NDVI_850_670"] = (df["850 nm"] - df["670 nm"]) / (df["850 nm"] + df["670 nm"])
df["NDRE_780_700"] = (df["780 nm"] - df["700 nm"]) / (df["780 nm"] + df["700 nm"])
df["NDRE_750_700"] = (df["750 nm"] - df["700 nm"]) / (df["750 nm"] + df["700 nm"])
df["NIR_Red_Ratio_850_670"] = safe_ratio(df["850 nm"], df["670 nm"])
df["RedEdge_Ratio_750_700"] = safe_ratio(df["750 nm"], df["700 nm"])
df["RedEdge_Diff_750_700"] = df["750 nm"] - df["700 nm"]
df["Slope_660_700"] = (df["700 nm"] - df["660 nm"]) / (700 - 660)
df["Slope_700_750"] = (df["750 nm"] - df["700 nm"]) / (750 - 700)
df["Slope_750_780"] = (df["780 nm"] - df["750 nm"]) / (780 - 750)
df["Slope_780_850"] = (df["850 nm"] - df["780 nm"]) / (850 - 780)

# Notes: keep existing moldy/etc notes, but the generic "missing band" note
# no longer applies now that every row has complete bands.
df.loc[df["Notes"] == "One or more bands missing (typical for outdoor/ambient-light measurement)", "Notes"] =
    ↪ None

df.to_csv(CSV, index=False)
print(f"\nWrote fully-imputed dataset back to {CSV}: {len(df)} rows, {df[BAND_COLS].isna().sum().sum()}
    ↪ missing bands remain")
print("\nImputed_bands value counts:")
print(df["Imputed_bands"].apply(lambda s: "no" if s == "no" else ("regression-partial" if s.startswith("
    ↪ regression") else "round-mean-fallback")).value_counts())
```

Statistical Analysis and Nitrate-N Regression Models

File: `Field_experiment/Source_codes/Statistical_analysis/growth_stage_ndvi_treatment_tests.py`

Randomized-complete-block ANOVA of dark-enclosure NDVI for each growth-stage round (NDVI ~ treatment + block, Type II).

```
"""
Formal randomized-complete-block-design (RCBD) treatment tests for dark-enclosure
NDVI at each of the six paired growth-stage sampling rounds reported in the
thesis Results and Discussion chapter (V10/R1/R3 x April/May planting).

This mirrors the design already used in complete_nitrogen_analysis.py
(response ~ treatment + block, Type II ANOVA, Benjamini-Hochberg FDR across
the family of tests) but is applied directly to the same white-reference
reflectance files, and therefore the same NDVI values, already reported in
the manuscript prose and figures for all six rounds -- not just the three
rounds covered by the older Shape_Feature_Analysis pipeline.

Outputs go to Processed_Biomass_Sampling_Data/Complete_Statistical_Analysis/
Tables/growth_stage_ndvi_treatment_tests.csv and
.../growth_stage_ndvi_plot_level.csv, plus a printed summary.
"""

from __future__ import annotations

from pathlib import Path

import numpy as np
import pandas as pd
import scipy.stats as st
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.stats.anova import anova_lm
from statsmodels.stats.multitest import multipletests

ROOT = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment")
FIELD_MAP = ROOT / "2026 Field Maps.xlsx"
OUT_DIR = ROOT / "Processed_Biomass_Sampling_Data" / "Complete_Statistical_Analysis" / "Tables"
OUT_DIR.mkdir(parents=True, exist_ok=True)

TREATMENT_ORDER = ["0", "50", "100", "150", "200", "200 split"]

ROUNDS = [
    dict(
        key="V10_April_0616",
        label="V10, April planting, sampled 16 June 2026",
        dark_refl=ROOT / "Processed_Biomass_Sampling_Data/Dark_env/06_16_2026_V10/Data/Reflectance/
    ↪ Dark_06_16_2026_V10_reflectance.xlsx",
        light_refl=ROOT / "Processed_Biomass_Sampling_Data/Light_env/06_16_2026_V10/Data/Reflectance/
    ↪ Light_06_16_2026_V10_reflectance.xlsx",
    ),
    dict(
        key="V10_May_0708",
        label="V10, May planting, sampled 8 July 2026",
        dark_refl=ROOT / "Processed_Biomass_Sampling_Data/Dark_env/07_08_2026_v8/Data/Reflectance/
    ↪ Dark_07_08_2026_v8_reflectance.xlsx",
        light_refl=ROOT / "Processed_Biomass_Sampling_Data/Light_env/07_08_2026_v8/Data/Reflectance/
    ↪ Light_07_08_2026_v8_reflectance.xlsx",
    ),
    dict(
```

```
        key="R1_April_0708",
        label="R1, April planting, sampled 8 July 2026",
        dark_refl=ROOT / "Processed_Biomass_Sampling_Data/Dark_env/07_08_2026_R1/Data/Reflectance/
    ↪ Dark_07_08_2026_R1_reflectance.xlsx",
        light_refl=ROOT / "Processed_Biomass_Sampling_Data/Light_env/07_08_2026_R1/Data/Reflectance/
    ↪ Light_07_08_2026_R1_reflectance.xlsx",
    ),
    dict(
        key="R1_May_0727",
        label="R1, May planting, sampled 27 July 2026",
        dark_refl=ROOT / "Processed_Biomass_Sampling_Data/Dark_env/07_27_2026_R1/Data/Reflectance/
    ↪ Dark_07_27_2026_R1_reflectance.xlsx",
        light_refl=ROOT / "Processed_Biomass_Sampling_Data/Light_env/07_27_2026_R1/Data/Reflectance/
    ↪ Light_07_27_2026_R1_reflectance.xlsx",
    ),
    dict(
        key="R3_April_0727",
        label="R3, April planting, sampled 27 July 2026",
        dark_refl=ROOT / "Processed_Biomass_Sampling_Data/Dark_env/07_27_2026_R3/Data/Reflectance/
    ↪ Dark_07_27_2026_R3_reflectance.xlsx",
        light_refl=ROOT / "Processed_Biomass_Sampling_Data/Light_env/07_27_2026_R3/Data/Reflectance/
    ↪ Light_07_27_2026_R3_reflectance.xlsx",
    ),
    dict(
        key="R3_May_0817",
        label="R3, May planting, sampled 17 August 2026",
        dark_refl=ROOT / "Processed_Biomass_Sampling_Data/Dark_env/08_17_2026_R3/Data/Reflectance/
    ↪ Dark_08_17_2026_R3_reflectance.xlsx",
        light_refl=ROOT / "Processed_Biomass_Sampling_Data/Light_env/08_17_2026_R3/Data/Reflectance/
    ↪ Light_08_17_2026_R3_reflectance.xlsx",
    ),
]


def normalize_treatment(value: object) -> str:
    if pd.isna(value):
        return ""
    text = str(value).strip()
    if text.endswith(".0"):
        text = text[:-2]
    return text


def corrected_block(plot: int) -> int:
    """Block is the hundreds digit within a planting (101-106 => block 1,
    401-406 => block 1, etc.). Matches the correction already documented in
    complete_nitrogen_analysis.py / RESULTS_REPORT.md: the raw field-map
    workbook mislabels plot 106 as Block 2; plot numbering and treatment
    balance show it is Block 1. Applied uniformly here so every round's
    block assignment is internally consistent, not just the ones the older
    pipeline happened to touch."""
    return (int(plot) // 100 - 1) % 3 + 1


def load_field_map() -> pd.DataFrame:
    fm = pd.read_excel(FIELD_MAP, sheet_name="Plot treatments")
    fm["N Treatment"] = fm["N Treatment"].map(normalize_treatment)
    fm["Block"] = fm["Plot Number"].map(corrected_block)
    return fm[["Plot Number", "N Treatment", "Block"]]


def ndvi_plot_level(refl_path: Path, field_map: pd.DataFrame) -> pd.DataFrame | None:
    if not refl_path.exists():
        return None
    df = pd.read_excel(refl_path)
    df["NDVI"] = (df["850 nm"] - df["670 nm"]) / (df["850 nm"] + df["670 nm"])
    df = df.merge(field_map, left_on="ID", right_on="Plot Number", how="left")
    plot = df.groupby(["ID", "N Treatment", "Block"], as_index=False)["NDVI"].mean()
```

```
    return plot


def rcbd_test(plot: pd.DataFrame, response: str = "NDVI") -> dict:
    d = plot[["N Treatment", "Block", response]].dropna().copy()
    counts = d.groupby("N Treatment", observed=True)[response].count()
    if len(d) < 12 or counts.size < 3 or counts.min() < 2:
        return {"n_plots": len(d), "n_treatments": int(counts.size), "status": "insufficient data"}
    d["N Treatment"] = pd.Categorical(d["N Treatment"], categories=TREATMENT_ORDER, ordered=True)
    model = smf.ols(f'Q("{response}") ~ C(Q("N Treatment")) + C(Block)', data=d).fit()
    table = anova_lm(model, typ=2)
    trt_row = table.loc['C(Q("N Treatment"))']
    ss_trt = trt_row["sum_sq"]
    ss_err = table.loc["Residual", "sum_sq"]
    eta = ss_trt / (ss_trt + ss_err)
    shapiro_p = st.shapiro(model.resid).pvalue
    bp = sm.stats.diagnostic.het_breuschpagan(model.resid, model.model.exog)
    return {
        "n_plots": len(d),
        "n_treatments": int(counts.size),
        "df_treatment": int(trt_row["df"]),
        "df_residual": int(table.loc["Residual", "df"]),
        "F_treatment": trt_row["F"],
        "p_treatment": trt_row["PR(>F)"],
        "partial_eta_squared": eta,
        "residual_shapiro_p": shapiro_p,
        "breusch_pagan_p": bp[1],
        "status": "ok",
    }


def main() -> None:
    field_map = load_field_map()
    rows = []
    all_plot_level = []
    for rnd in ROUNDS:
        for env, path_key in [("Dark", "dark_refl"), ("Light", "light_refl")]:
            plot = ndvi_plot_level(rnd[path_key], field_map)
            if plot is None:
                rows.append({"round": rnd["key"], "label": rnd["label"], "environment": env, "status": "file
    ↪ not found"})
                continue
            plot = plot.copy()
            plot["round"] = rnd["key"]
            plot["environment"] = env
            all_plot_level.append(plot)
            result = rcbd_test(plot)
            result.update(round=rnd["key"], label=rnd["label"], environment=env)
            rows.append(result)

    results = pd.DataFrame(rows)

    # FDR across the six dark-enclosure NDVI tests -- the primary, well-powered
    # family of tests this analysis is meant to resolve (light-environment
    # tests are reported for completeness but not FDR-corrected together with
    # dark, consistent with treating light-room readings as exploratory given
    # documented missingness).
    dark_mask = (results["environment"] == "Dark") & (results["status"] == "ok")
    if dark_mask.sum():
        reject, adj, _, _ = multipletests(results.loc[dark_mask, "p_treatment"], method="fdr_bh")
        results.loc[dark_mask, "p_fdr"] = adj
        results.loc[dark_mask, "fdr_significant_0.05"] = reject

    results.to_csv(OUT_DIR / "growth_stage_ndvi_treatment_tests.csv", index=False)
    pd.concat(all_plot_level, ignore_index=True).to_csv(OUT_DIR / "growth_stage_ndvi_plot_level.csv", index=
    ↪ False)

    pd.set_option("display.width", 160)
```

```
    pd.set_option("display.max_columns", 20)
    print(results.to_string(index=False))
    print("\nSaved:", OUT_DIR / "growth_stage_ndvi_treatment_tests.csv")


if __name__ == "__main__":
    main()
```

File: `Field_experiment/Processed_Biomass_Sampling_Data/nitrate_log_transform_analysis/05_scripts/01f_regression_train_test_only.py`

Dark-enclosure log1p nitrate-N regression: baseline, ordinary least squares, ridge, kernel ridge, and neural network on the six bands and growth stage.

```
# -*- coding: utf-8 -*-
"""Train/test-only regression (no validation split, no hyperparameter
tuning), per explicit instruction. Same 8 features as
01e_regression_bands_only.py (6 raw z-scored bands + growth-stage dummies),
same complete-234-plant-minus-1-moldy feature_matrix.csv (built with 850nm
imputation for the dead-channel R3_April_0727 plants).

Each model is fit ONCE on the full training set with fixed/default
hyperparameters (nothing tuned, since there is no validation set to tune
against) and evaluated on train and test. Reported: R2_log (primary),
R2_ppm, r, RMSE for both splits, for every model.

Fixed hyperparameters used (explicitly NOT tuned):
  - Ridge: alpha=1.0 (sklearn default)
  - KernelRidge (RBF): alpha=1.0, gamma=1/n_features=0.125 (sklearn default)
  - SVR (RBF): C=1.0, epsilon=0.1 (sklearn defaults), gamma=1/n_features
  - NN: back to a plain MLP (the 1D CNN + FC head tried in between didn't
    beat OLS either and was removed), 3 hidden layers [16, 12, 8],
    dropout=0.3, Adam lr=0.01, weight_decay=0.01, fixed 300 epochs, no
    early stopping (no validation set to monitor).
"""
from pathlib import Path

import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.svm import SVR

ROOT = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment")
PBSD = ROOT / "Processed_Biomass_Sampling_Data"
OUT = PBSD / "nitrate_log_transform_analysis"
FEAT_CSV = OUT / "01_data" / "feature_matrix.csv"
OUT_DIR = OUT / "02_regression_train_test_only"
(OUT_DIR / "tables").mkdir(parents=True, exist_ok=True)
(OUT_DIR / "predictions").mkdir(parents=True, exist_ok=True)

FEATURES = ["P660_z", "P670_z", "P700_z", "P750_z", "P780_z", "P850_z", "Stage_R1", "Stage_V10"]
TARGET_RAW = "Nitrate_ppm"
RANDOM_SEED = 42
RIDGE_ALPHA_FIXED = 1.0
KRR_ALPHA_FIXED = 1.0
KRR_GAMMA_FIXED = 1.0 / len(FEATURES)
SVR_C_FIXED = 1.0
SVR_EPSILON_FIXED = 0.1
SVR_GAMMA_FIXED = 1.0 / len(FEATURES)
```

```
NN_HIDDEN = [16, 12, 8]
NN_EPOCHS_FIXED = 300
NN_LR = 1e-2
NN_WEIGHT_DECAY = 1e-2
NN_DROPOUT = 0.3

_log = []


def log(msg):
    print(msg)
    _log.append(str(msg))


def set_all_seeds(seed=RANDOM_SEED):
    np.random.seed(seed)
    torch.manual_seed(seed)


def metrics_block(y_true_ppm, y_pred_log):
    y_true_ppm = np.asarray(y_true_ppm, float)
    y_pred_log = np.asarray(y_pred_log, float)
    y_true_log = np.log1p(y_true_ppm)
    y_pred_ppm = np.clip(np.expm1(y_pred_log), 0, None)

    def r2(t, p):
        ss_res = np.sum((t - p) ** 2)
        ss_tot = np.sum((t - t.mean()) ** 2)
        return 1 - ss_res / ss_tot if ss_tot > 0 else np.nan

    def corr(t, p):
        return float(np.corrcoef(t, p)[0, 1]) if len(t) > 1 and np.std(p) > 0 else np.nan

    return {
        "R2_log": r2(y_true_log, y_pred_log), "RMSE_log": float(np.sqrt(np.mean((y_true_log - y_pred_log) **
    ↪ 2))),
        "r_log": corr(y_true_log, y_pred_log),
        "R2_ppm": r2(y_true_ppm, y_pred_ppm), "RMSE_ppm": float(np.sqrt(np.mean((y_true_ppm - y_pred_ppm) **
    ↪ 2))),
        "r_ppm": corr(y_true_ppm, y_pred_ppm), "N": len(y_true_ppm), "pred_ppm": y_pred_ppm,
    }


class MLP(nn.Module):
    def __init__(self, n_in, hidden_widths, dropout):
        super().__init__()
        layers, prev = [], n_in
        for w in hidden_widths:
            layers += [nn.Linear(prev, w), nn.ReLU(), nn.Dropout(dropout)]
            prev = w
        layers += [nn.Linear(prev, 1)]
        self.net = nn.Sequential(*layers)

    def forward(self, x):
        return self.net(x).squeeze(-1)


def main():
    log("=== Train/test-only regression (no val, no tuning): log1p(Nitrate_ppm) ===\n")
    df = pd.read_csv(FEAT_CSV)
    df["Plot_ID"] = df["Plot_ID"].astype(str)

    tr = df[df["Split"] == "train"]
    te = df[df["Split"] == "test"]
    log(f"N_train={len(tr)}, N_test={len(te)}, N_features={len(FEATURES)} ({FEATURES})")

    x_tr, x_te = tr[FEATURES].to_numpy(float), te[FEATURES].to_numpy(float)
    ppm_tr, ppm_te = tr[TARGET_RAW].to_numpy(float), te[TARGET_RAW].to_numpy(float)
```

```
y_tr_log = np.log1p(ppm_tr)

rows = []
pred_store = {}

def add_row(model_name, pred_te_log, pred_tr_log, extra=None):
    m_test = metrics_block(ppm_te, pred_te_log)
    m_train = metrics_block(ppm_tr, pred_tr_log)
    row = {
        "Model": model_name, "N_train": len(tr), "N_test": len(te),
        "R2_train_log": m_train["R2_log"], "R2_test_log": m_test["R2_log"],
        "RMSE_train_log": m_train["RMSE_log"], "RMSE_test_log": m_test["RMSE_log"],
        "r_train_log": m_train["r_log"], "r_test_log": m_test["r_log"],
        "R2_train_ppm": m_train["R2_ppm"], "R2_test_ppm": m_test["R2_ppm"],
        "RMSE_train_ppm": m_train["RMSE_ppm"], "RMSE_test_ppm": m_test["RMSE_ppm"],
    }
    if extra:
        row.update(extra)
    rows.append(row)
    pred_store[model_name] = {"pred_te_log": pred_te_log, "pred_tr_log": pred_tr_log, "m_test": m_test}
    log(f"  {model_name}: TRAIN R2_log={m_train['R2_log']:.4f} | TEST R2_log={m_test['R2_log']:.4f} "
        f"R2_ppm={m_test['R2_ppm']:.4f} RMSE_ppm={m_test['RMSE_ppm']:.1f}")
    return m_test

# Mean baseline
pred_te_base = np.full(len(te), y_tr_log.mean())
pred_tr_base = np.full(len(tr), y_tr_log.mean())
add_row("Mean_baseline", pred_te_base, pred_tr_base)

# OLS
ols = LinearRegression().fit(x_tr, y_tr_log)
add_row("OLS", ols.predict(x_te), ols.predict(x_tr))

# Ridge (fixed alpha, not tuned)
ridge = Ridge(alpha=RIDGE_ALPHA_FIXED, random_state=RANDOM_SEED).fit(x_tr, y_tr_log)
add_row("Ridge", ridge.predict(x_te), ridge.predict(x_tr), {"alpha": RIDGE_ALPHA_FIXED})

# Kernel Ridge RBF (fixed alpha/gamma, not tuned)
krr = KernelRidge(kernel="rbf", alpha=KRR_ALPHA_FIXED, gamma=KRR_GAMMA_FIXED).fit(x_tr, y_tr_log)
add_row("KernelRidge_RBF", krr.predict(x_te), krr.predict(x_tr),
        {"alpha": KRR_ALPHA_FIXED, "gamma": KRR_GAMMA_FIXED})

# SVM regression (RBF kernel, fixed C/epsilon/gamma, not tuned)
svr = SVR(kernel="rbf", C=SVR_C_FIXED, epsilon=SVR_EPSILON_FIXED, gamma=SVR_GAMMA_FIXED).fit(x_tr,
↪ y_tr_log)
add_row("SVR_RBF", svr.predict(x_te), svr.predict(x_tr),
        {"C": SVR_C_FIXED, "epsilon": SVR_EPSILON_FIXED, "gamma": SVR_GAMMA_FIXED})

# Neural network: fixed architecture, fixed epoch count, no early stopping
set_all_seeds(RANDOM_SEED)
y_scaler_mean, y_scaler_std = y_tr_log.mean(), (y_tr_log.std() if y_tr_log.std() > 1e-9 else 1.0)
model = MLP(x_tr.shape[1], NN_HIDDEN, NN_DROPOUT)
n_params = sum(p.numel() for p in model.parameters())
log(f"  MLP parameter count: {n_params}")
opt = torch.optim.Adam(model.parameters(), lr=NN_LR, weight_decay=NN_WEIGHT_DECAY)
loss_fn = nn.MSELoss()
xt = torch.tensor(x_tr, dtype=torch.float32)
yt = torch.tensor((y_tr_log - y_scaler_mean) / y_scaler_std, dtype=torch.float32)
for _ in range(NN_EPOCHS_FIXED):
    model.train()
    opt.zero_grad()
    loss = loss_fn(model(xt), yt)
    loss.backward()
    opt.step()
model.eval()
with torch.no_grad():
    pred_te_nn_log = model(torch.tensor(x_te, dtype=torch.float32)).numpy() * y_scaler_std + y_scaler_mean
    pred_tr_nn_log = model(xt).numpy() * y_scaler_std + y_scaler_mean
```

```
    add_row("NN_3hidden_16_12_8", pred_te_nn_log, pred_tr_nn_log, {"epochs": NN_EPOCHS_FIXED, "n_params":
    ↪ n_params})

    comp_df = pd.DataFrame(rows)
    comp_df.to_csv(OUT_DIR / "tables" / "model_comparison_train_test.csv", index=False)
    log(f"\n=== MODEL COMPARISON (TRAIN vs TEST, no tuning) ===\n{comp_df.to_string(index=False)}")

    candidates = comp_df[comp_df["Model"] != "Mean_baseline"]
    winner_name = candidates.loc[candidates["R2_test_log"].idxmax(), "Model"]
    log(f"\nBest model by TEST R2_log: {winner_name}")

    for name in [winner_name, "Mean_baseline"]:
        pred_log = pred_store[name]["pred_te_log"]
        pred_ppm = np.clip(np.expm1(pred_log), 0, None)
        out_df = pd.DataFrame({
            "Plant_ID": te["Plant_ID"].values, "Plot_ID": te["Plot_ID"].values,
            "Growth_Stage": te["Growth_Stage"].values, "Planting_Date": te["Planting_Date"].values,
            "Nitrate_ppm_observed": ppm_te, "Nitrate_ppm_predicted": pred_ppm,
            "log1p_observed": np.log1p(ppm_te), "log1p_predicted": pred_log,
        })
        safe_name = name.replace("/", "-")
        out_df.to_csv(OUT_DIR / "predictions" / f"test_predictions_{safe_name}.csv", index=False)

    (OUT_DIR / "regression_train_test_log.txt").write_text("\n".join(_log), encoding="utf-8")
    log("\nSaved all train/test-only regression outputs.")


if __name__ == "__main__":
    main()
```

File: `Field_experiment/Processed_Biomass_Sampling_Data/nitrate_log_transform_analysis/05_scripts/01g_regression_train_test_only_light.py`

The same regression protocol applied to the outdoor dataset, using the same plot-level train/test split.

```
# -*- coding: utf-8 -*-
"""LIGHT (outdoor/ambient) counterpart to 01f_regression_train_test_only.py
-- identical protocol, run on feature_matrix_light.csv instead of the dark
feature_matrix.csv. Same 8 features (6 raw z-scored bands + growth-stage
dummies), same 233-plant-minus-1-moldy population, and the SAME plot-level
train/test partition as the dark run (reused, not re-split, for direct
dark-vs-light comparability). Light bands were two-tier imputed (regression
for partial gaps, round-mean fallback for the 23 totally-missing rows,
mostly R1_May_0727) before this script ever sees them -- see
Imputed_bands in feature_matrix_light.csv.

Each model is fit ONCE on the full training set with fixed/default
hyperparameters (nothing tuned, since there is no validation set to tune
against) and evaluated on train and test. Reported: R2_log (primary),
R2_ppm, r, RMSE for both splits, for every model.

Fixed hyperparameters used (explicitly NOT tuned):
  - Ridge: alpha=1.0 (sklearn default)
  - KernelRidge (RBF): alpha=1.0, gamma=1/n_features=0.125 (sklearn default)
  - SVR (RBF): C=1.0, epsilon=0.1 (sklearn defaults), gamma=1/n_features
  - NN: back to a plain MLP (the 1D CNN + FC head tried in between didn't
    beat OLS either and was removed), 3 hidden layers [16, 12, 8],
    dropout=0.3, Adam lr=0.01, weight_decay=0.01, fixed 300 epochs, no
    early stopping (no validation set to monitor).
"""
from pathlib import Path
```

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.svm import SVR

ROOT = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment")
PBSD = ROOT / "Processed_Biomass_Sampling_Data"
OUT = PBSD / "nitrate_log_transform_analysis"
FEAT_CSV = OUT / "01_data" / "feature_matrix_light.csv"
OUT_DIR = OUT / "02_regression_train_test_only_light"
(OUT_DIR / "tables").mkdir(parents=True, exist_ok=True)
(OUT_DIR / "predictions").mkdir(parents=True, exist_ok=True)

FEATURES = ["P660_z", "P670_z", "P700_z", "P750_z", "P780_z", "P850_z", "Stage_R1", "Stage_V10"]
TARGET_RAW = "Nitrate_ppm"
RANDOM_SEED = 42
RIDGE_ALPHA_FIXED = 1.0
KRR_ALPHA_FIXED = 1.0
KRR_GAMMA_FIXED = 1.0 / len(FEATURES)
SVR_C_FIXED = 1.0
SVR_EPSILON_FIXED = 0.1
SVR_GAMMA_FIXED = 1.0 / len(FEATURES)
NN_HIDDEN = [16, 12, 8]
NN_EPOCHS_FIXED = 300
NN_LR = 1e-2
NN_WEIGHT_DECAY = 1e-2
NN_DROPOUT = 0.3

_log = []


def log(msg):
    print(msg)
    _log.append(str(msg))


def set_all_seeds(seed=RANDOM_SEED):
    np.random.seed(seed)
    torch.manual_seed(seed)


def metrics_block(y_true_ppm, y_pred_log):
    y_true_ppm = np.asarray(y_true_ppm, float)
    y_pred_log = np.asarray(y_pred_log, float)
    y_true_log = np.log1p(y_true_ppm)
    y_pred_ppm = np.clip(np.expm1(y_pred_log), 0, None)

    def r2(t, p):
        ss_res = np.sum((t - p) ** 2)
        ss_tot = np.sum((t - t.mean()) ** 2)
        return 1 - ss_res / ss_tot if ss_tot > 0 else np.nan

    def corr(t, p):
        return float(np.corrcoef(t, p)[0, 1]) if len(t) > 1 and np.std(p) > 0 else np.nan

    return {
        "R2_log": r2(y_true_log, y_pred_log), "RMSE_log": float(np.sqrt(np.mean((y_true_log - y_pred_log) **
    ↪ 2))),
        "r_log": corr(y_true_log, y_pred_log),
        "R2_ppm": r2(y_true_ppm, y_pred_ppm), "RMSE_ppm": float(np.sqrt(np.mean((y_true_ppm - y_pred_ppm) **
    ↪ 2))),
        "r_ppm": corr(y_true_ppm, y_pred_ppm), "N": len(y_true_ppm), "pred_ppm": y_pred_ppm,
    }
```

```
class MLP(nn.Module):
    def __init__(self, n_in, hidden_widths, dropout):
        super().__init__()
        layers, prev = [], n_in
        for w in hidden_widths:
            layers += [nn.Linear(prev, w), nn.ReLU(), nn.Dropout(dropout)]
            prev = w
        layers += [nn.Linear(prev, 1)]
        self.net = nn.Sequential(*layers)

    def forward(self, x):
        return self.net(x).squeeze(-1)


def main():
    log("=== Train/test-only regression (no val, no tuning): log1p(Nitrate_ppm) ===\n")
    df = pd.read_csv(FEAT_CSV)
    df["Plot_ID"] = df["Plot_ID"].astype(str)

    tr = df[df["Split"] == "train"]
    te = df[df["Split"] == "test"]
    log(f"N_train={len(tr)}, N_test={len(te)}, N_features={len(FEATURES)} ({FEATURES})")

    x_tr, x_te = tr[FEATURES].to_numpy(float), te[FEATURES].to_numpy(float)
    ppm_tr, ppm_te = tr[TARGET_RAW].to_numpy(float), te[TARGET_RAW].to_numpy(float)
    y_tr_log = np.log1p(ppm_tr)

    rows = []
    pred_store = {}

    def add_row(model_name, pred_te_log, pred_tr_log, extra=None):
        m_test = metrics_block(ppm_te, pred_te_log)
        m_train = metrics_block(ppm_tr, pred_tr_log)
        row = {
            "Model": model_name, "N_train": len(tr), "N_test": len(te),
            "R2_train_log": m_train["R2_log"], "R2_test_log": m_test["R2_log"],
            "RMSE_train_log": m_train["RMSE_log"], "RMSE_test_log": m_test["RMSE_log"],
            "r_train_log": m_train["r_log"], "r_test_log": m_test["r_log"],
            "R2_train_ppm": m_train["R2_ppm"], "R2_test_ppm": m_test["R2_ppm"],
            "RMSE_train_ppm": m_train["RMSE_ppm"], "RMSE_test_ppm": m_test["RMSE_ppm"],
        }
        if extra:
            row.update(extra)
        rows.append(row)
        pred_store[model_name] = {"pred_te_log": pred_te_log, "pred_tr_log": pred_tr_log, "m_test": m_test}
        log(f"  {model_name}: TRAIN R2_log={m_train['R2_log']:.4f} | TEST R2_log={m_test['R2_log']:.4f} "
            f"R2_ppm={m_test['R2_ppm']:.4f} RMSE_ppm={m_test['RMSE_ppm']:.1f}")
        return m_test

    # Mean baseline
    pred_te_base = np.full(len(te), y_tr_log.mean())
    pred_tr_base = np.full(len(tr), y_tr_log.mean())
    add_row("Mean_baseline", pred_te_base, pred_tr_base)

    # OLS
    ols = LinearRegression().fit(x_tr, y_tr_log)
    add_row("OLS", ols.predict(x_te), ols.predict(x_tr))

    # Ridge (fixed alpha, not tuned)
    ridge = Ridge(alpha=RIDGE_ALPHA_FIXED, random_state=RANDOM_SEED).fit(x_tr, y_tr_log)
    add_row("Ridge", ridge.predict(x_te), ridge.predict(x_tr), {"alpha": RIDGE_ALPHA_FIXED})

    # Kernel Ridge RBF (fixed alpha/gamma, not tuned)
    krr = KernelRidge(kernel="rbf", alpha=KRR_ALPHA_FIXED, gamma=KRR_GAMMA_FIXED).fit(x_tr, y_tr_log)
    add_row("KernelRidge_RBF", krr.predict(x_te), krr.predict(x_tr),
            {"alpha": KRR_ALPHA_FIXED, "gamma": KRR_GAMMA_FIXED})
```

```
    # SVM regression (RBF kernel, fixed C/epsilon/gamma, not tuned)
    svr = SVR(kernel="rbf", C=SVR_C_FIXED, epsilon=SVR_EPSILON_FIXED, gamma=SVR_GAMMA_FIXED).fit(x_tr,
     ↪ y_tr_log)
    add_row("SVR_RBF", svr.predict(x_te), svr.predict(x_tr),
            {"C": SVR_C_FIXED, "epsilon": SVR_EPSILON_FIXED, "gamma": SVR_GAMMA_FIXED})

    # Neural network: fixed architecture, fixed epoch count, no early stopping
    set_all_seeds(RANDOM_SEED)
    y_scaler_mean, y_scaler_std = y_tr_log.mean(), (y_tr_log.std() if y_tr_log.std() > 1e-9 else 1.0)
    model = MLP(x_tr.shape[1], NN_HIDDEN, NN_DROPOUT)
    n_params = sum(p.numel() for p in model.parameters())
    log(f"  MLP parameter count: {n_params}")
    opt = torch.optim.Adam(model.parameters(), lr=NN_LR, weight_decay=NN_WEIGHT_DECAY)
    loss_fn = nn.MSELoss()
    xt = torch.tensor(x_tr, dtype=torch.float32)
    yt = torch.tensor((y_tr_log - y_scaler_mean) / y_scaler_std, dtype=torch.float32)
    for _ in range(NN_EPOCHS_FIXED):
        model.train()
        opt.zero_grad()
        loss = loss_fn(model(xt), yt)
        loss.backward()
        opt.step()
    model.eval()
    with torch.no_grad():
        pred_te_nn_log = model(torch.tensor(x_te, dtype=torch.float32)).numpy() * y_scaler_std + y_scaler_mean
        pred_tr_nn_log = model(xt).numpy() * y_scaler_std + y_scaler_mean
    add_row("NN_3hidden_16_12_8", pred_te_nn_log, pred_tr_nn_log, {"epochs": NN_EPOCHS_FIXED, "n_params":
     ↪ n_params})

    comp_df = pd.DataFrame(rows)
    comp_df.to_csv(OUT_DIR / "tables" / "model_comparison_train_test.csv", index=False)
    log(f"\n=== MODEL COMPARISON (TRAIN vs TEST, no tuning) ===\n{comp_df.to_string(index=False)}")

    candidates = comp_df[comp_df["Model"] != "Mean_baseline"]
    winner_name = candidates.loc[candidates["R2_test_log"].idxmax(), "Model"]
    log(f"\nBest model by TEST R2_log: {winner_name}")

    for name in [winner_name, "Mean_baseline"]:
        pred_log = pred_store[name]["pred_te_log"]
        pred_ppm = np.clip(np.expm1(pred_log), 0, None)
        out_df = pd.DataFrame({
            "Plant_ID": te["Plant_ID"].values, "Plot_ID": te["Plot_ID"].values,
            "Growth_Stage": te["Growth_Stage"].values, "Planting_Date": te["Planting_Date"].values,
            "Nitrate_ppm_observed": ppm_te, "Nitrate_ppm_predicted": pred_ppm,
            "log1p_observed": np.log1p(ppm_te), "log1p_predicted": pred_log,
        })
        safe_name = name.replace("/", "-")
        out_df.to_csv(OUT_DIR / "predictions" / f"test_predictions_{safe_name}.csv", index=False)

    (OUT_DIR / "regression_train_test_log.txt").write_text("\n".join(_log), encoding="utf-8")
    log("\nSaved all train/test-only regression outputs.")


if __name__ == "__main__":
    main()
```

File: `Field_experiment/Processed_Biomass_Sampling_Data/nitrate_log_transform_analysis/05_scripts/01h_regression_ndvi_combinations.py`

Neural network with the nine red/red-edge by near-infrared NDVI combinations and growth stage as inputs.

```
# -*- coding: utf-8 -*-
```

```
"""Normalized-difference-combination neural network (dark enclosure).

Builds every visible x NIR normalized difference the sensor can form from its
relative-reflectance bands: red 660, 670 nm and red-edge 700 nm, each paired
with NIR 750, 780, 850 nm -> 3 x 3 = 9 indices, ND = (R_NIR - R_x)/(R_NIR + R_x).
The six red-based ones are NDVIs; the three 700 nm ones are red-edge (NDRE).
Also the red-edge chlorophyll index CI_RE = R_NIR / R_700 - 1 (3 combinations).

Two models: the 6 red-NIR NDVIs only, and all 9 indices (red edge added).
Indices (z-scored with training-set mean/SD only) plus the two
growth-stage dummies (Stage_R1, Stage_V10) form the inputs to the SAME
neural network, split, seed and fixed hyperparameters as
01f_regression_train_test_only.py (3 hidden layers [16, 12, 8], ReLU,
dropout 0.3, Adam lr 0.01, weight decay 0.01, 300 epochs, seed 42), so the
held-out test R2 is directly comparable with the six-band model (0.752).
"""
import importlib.util
import sys
from itertools import product
from pathlib import Path

import numpy as np
import pandas as pd
import torch
import torch.nn as nn

HERE = Path(__file__).resolve().parent
_spec = importlib.util.spec_from_file_location("base", HERE / "01f_regression_train_test_only.py")
base = importlib.util.module_from_spec(_spec)
_spec.loader.exec_module(base)

LIGHT = "--light" in sys.argv  # outdoor/ambient data (bands already imputed, same split)
FEAT_CSV = base.OUT / "01_data" / ("feature_matrix_light.csv" if LIGHT else "feature_matrix.csv")
OUT_DIR = base.OUT / ("02_regression_ndvi_combinations_light" if LIGHT else "02_regression_ndvi_combinations")
(OUT_DIR / "tables").mkdir(parents=True, exist_ok=True)
(OUT_DIR / "predictions").mkdir(parents=True, exist_ok=True)

RED = [660, 670, 700]  # 700 nm = red edge
NIR = [750, 780, 850]
COMBOS = list(product(RED, NIR))
NDVI_COLS = [f"{'NDRE' if r == 700 else 'NDVI'}_{r}_{n}" for r, n in COMBOS]
# Red-edge chlorophyll index (Gitelson et al. 2003): CI_RE = R_NIR / R_RE - 1 (one RE band -> 3 combos)
CI_COMBOS = [(700, n) for n in NIR]
CI_COLS = [f"CIRE_{r}_{n}" for r, n in CI_COMBOS]
NDVI6 = [c for c in NDVI_COLS if c.startswith("NDVI")]
SETS = {"NN_6NDVI_plus_stage": NDVI6,
        "NN_6NDVI_3CIRE_plus_stage": NDVI6 + CI_COLS,
        # LED-channel diagnosis: 660 nm channel fails the PROSPECT consistency check
        # (r(660,670) = 0.13 measured vs 1.00 expected), so drop it a priori
        "NN_3NDVI_670_plus_stage": ["NDVI_670_750", "NDVI_670_780", "NDVI_670_850"],
        "NN_2NDVI_670_trueNIR_plus_stage": ["NDVI_670_780", "NDVI_670_850"],
        "NN_9ND_rededge_plus_stage": NDVI_COLS}
STAGE_COLS = ["Stage_R1", "Stage_V10"]
SEEDS_ROBUSTNESS = list(range(10))


def train_nn(x_tr, y_tr_log, x_te, seed):
    base.set_all_seeds(seed)
    mu, sd = y_tr_log.mean(), y_tr_log.std()
    model = base.MLP(x_tr.shape[1], base.NN_HIDDEN, base.NN_DROPOUT)
    opt = torch.optim.Adam(model.parameters(), lr=base.NN_LR, weight_decay=base.NN_WEIGHT_DECAY)
    loss_fn = nn.MSELoss()
    xt = torch.tensor(x_tr, dtype=torch.float32)
    yt = torch.tensor((y_tr_log - mu) / sd, dtype=torch.float32)
    for _ in range(base.NN_EPOCHS_FIXED):
        model.train()
        opt.zero_grad()
        loss_fn(model(xt), yt).backward()
```

```
        opt.step()
    model.eval()
    with torch.no_grad():
        p_te = model(torch.tensor(x_te, dtype=torch.float32)).numpy() * sd + mu
        p_tr = model(xt).numpy() * sd + mu
    return p_te, p_tr


def main():
    df = pd.read_csv(FEAT_CSV)
    for (r, n), col in zip(COMBOS, NDVI_COLS):
        df[col] = (df[f"P{n}"] - df[f"P{r}"]) / (df[f"P{n}"] + df[f"P{r}"])
    for (r, n), col in zip(CI_COMBOS, CI_COLS):
        df[col] = df[f"P{n}"] / df[f"P{r}"] - 1
    print("NDVI(670,850) matches file NDVI column:", np.allclose(df["NDVI_670_850"], df["NDVI"]))

    tr, te = df[df["Split"] == "train"].copy(), df[df["Split"] == "test"].copy()
    ppm_tr, ppm_te = tr[base.TARGET_RAW].to_numpy(float), te[base.TARGET_RAW].to_numpy(float)
    y_tr_log, y_te_log = np.log1p(ppm_tr), np.log1p(ppm_te)

    # Table of combinations (descriptives on all 233 plants; r with log1p nitrate-N)
    rows = []
    for (r, n), col in zip(COMBOS, NDVI_COLS):
        rows.append({
            "Index": col, "Red_nm": r, "NIR_nm": n,
            "Formula": f"(R{n} - R{r}) / (R{n} + R{r})",
            "Mean": df[col].mean(), "SD": df[col].std(), "Min": df[col].min(), "Max": df[col].max(),
            "r_log_nitrate_train": np.corrcoef(tr[col], y_tr_log)[0, 1],
        })
    for (r, n), col in zip(CI_COMBOS, CI_COLS):
        rows.append({
            "Index": col, "Red_nm": r, "NIR_nm": n, "Formula": f"R{n} / R{r} - 1",
            "Mean": df[col].mean(), "SD": df[col].std(), "Min": df[col].min(), "Max": df[col].max(),
            "r_log_nitrate_train": np.corrcoef(tr[col], y_tr_log)[0, 1],
        })
    combo_df = pd.DataFrame(rows)
    ALL_IDX = NDVI_COLS + CI_COLS
    combo_df.to_csv(OUT_DIR / "tables" / "ndvi_combinations.csv", index=False)
    print(combo_df.to_string(index=False))
    print("Undefined (0/0) index values per index:", df[ALL_IDX].isna().sum().to_dict())
    print("\nInter-index correlation (all plants):\n", df[NDVI_COLS + CI_COLS].corr().round(3).to_string())

    # z-score with training statistics only
    ALL = NDVI_COLS + CI_COLS
    mu, sd = tr[ALL].mean(), tr[ALL].std(ddof=0)
    for d in (tr, te):
        for c in ALL:
            # undefined index (0/0 where both bands are at the outdoor noise floor) ->
            # training-set mean, i.e. z = 0, as for unusable outdoor bands in the Methods
            d[c + "_z"] = ((d[c] - mu[c]) / sd[c]).fillna(0.0)

    res_rows, preds = [], {}
    for name, cols in SETS.items():
        feats = [c + "_z" for c in cols] + STAGE_COLS
        x_tr, x_te = tr[feats].to_numpy(float), te[feats].to_numpy(float)
        p_te, p_tr = train_nn(x_tr, y_tr_log, x_te, base.RANDOM_SEED)
        m_te, m_tr = base.metrics_block(ppm_te, p_te), base.metrics_block(ppm_tr, p_tr)
        res_rows.append({"Model": name, "Inputs": len(feats), "N_train": len(tr), "N_test": len(te),
                         "R2_train_log": m_tr["R2_log"], "R2_test_log": m_te["R2_log"],
                         "r_test_log": m_te["r_log"], "RMSE_test_log": m_te["RMSE_log"]})
        preds[name] = p_te
    res = pd.DataFrame(res_rows)

    # Seed robustness for both index models and the six-band reference model
    seed_rows = []
    for s in SEEDS_ROBUSTNESS:
        row = {"seed": s}
        for name, cols in SETS.items():
```

```
            feats = [c + "_z" for c in cols] + STAGE_COLS
            a, _ = train_nn(tr[feats].to_numpy(float), y_tr_log, te[feats].to_numpy(float), s)
            row["R2_test_" + name] = base.metrics_block(ppm_te, a)["R2_log"]
        b, _ = train_nn(tr[base.FEATURES].to_numpy(float), y_tr_log, te[base.FEATURES].to_numpy(float), s)
        row["R2_test_bands6_stage"] = base.metrics_block(ppm_te, b)["R2_log"]
        seed_rows.append(row)
    seed_df = pd.DataFrame(seed_rows)
    seed_df.to_csv(OUT_DIR / "tables" / "seed_robustness.csv", index=False)
    res.to_csv(OUT_DIR / "tables" / "nn_ndvi_combinations_result.csv", index=False)
    print("\n", res.to_string(index=False))
    print("\n", seed_df.to_string(index=False))
    print("\nseed mean/sd:\n", seed_df.drop(columns="seed").agg(["mean", "std"]).to_string())

    pd.DataFrame({
        "Plant_ID": te["Plant_ID"].values, "Plot_ID": te["Plot_ID"].values,
        "Growth_Stage": te["Growth_Stage"].values, "log1p_observed": y_te_log, **{"pred_" + k: v for k, v in
     ↪ preds.items()},
    }).to_csv(OUT_DIR / "predictions" / "test_predictions_index_models.csv", index=False)


if __name__ == "__main__":
    main()
```

File: `Field_experiment/Processed_Biomass_Sampling_Data/nitrate_log_transform_analysis/05_scripts/01k_bands_plus_three_ndvi.py`

Neural network with the six bands, three NDVI combinations, and growth stage as inputs.

```
# -*- coding: utf-8 -*-
"""Six raw bands + three NDVI combinations + growth stage -> NN (dark enclosure).

Every 3-of-9 NDVI combination (84) is added to the six z-scored relative-reflectance
bands and the two growth-stage dummies (11 inputs), using the same NN, split and seed
as the six-band model (01f). Each combination is scored on the held-out test set
(seed 42) and by plot-grouped 5-fold CV on the training set only (unbiased selection).
"""
import importlib.util
import sys
from itertools import combinations, product
from pathlib import Path

import numpy as np
import pandas as pd
from sklearn.model_selection import GroupKFold

HERE = Path(__file__).resolve().parent
_spec = importlib.util.spec_from_file_location("h", HERE / "01h_regression_ndvi_combinations.py")
h = importlib.util.module_from_spec(_spec)
_spec.loader.exec_module(h)
base = h.base

LIGHT = "--light" in sys.argv  # outdoor data: same split; bands already imputed in feature_matrix_light.csv
FEAT_CSV = base.OUT / "01_data" / ("feature_matrix_light.csv" if LIGHT else "feature_matrix.csv")
OUT = base.OUT / ("02_regression_bands_plus_ndvi_light" if LIGHT else "02_regression_bands_plus_ndvi") / "
    ↪ tables"
OUT.mkdir(parents=True, exist_ok=True)
RED, NIR = [660, 670, 700], [750, 780, 850]
ND = [f"ND_{r}_{n}" for r, n in product(RED, NIR)]
BANDS = ["P660", "P670", "P700", "P750", "P780", "P850"]
STAGE = ["Stage_R1", "Stage_V10"]
CV_SEEDS = [0, 1]
ROBUST_SEEDS = list(range(10))


def label(c):
```

```
    r, n = c.split("_")[1:]
    return f"{'NDRE' if r == '700' else 'NDVI'}({r}, {n})"


BANDS_Z = [b + "_z" for b in BANDS]  # the exact standardized bands used for Table 13


def matrices(tr, te, nd_cols):
    nd_cols = list(nd_cols)
    mu, sd = tr[nd_cols].mean(), tr[nd_cols].std(ddof=0)
    def build(d):
        parts = [d[BANDS_Z].to_numpy(float)]
        if nd_cols:
            # undefined NDVI (0/0, both bands at the outdoor noise floor) -> training mean (z = 0)
            parts.append(((d[nd_cols] - mu) / sd).fillna(0.0).to_numpy(float))
        parts.append(d[STAGE].to_numpy(float))
        return np.column_stack(parts)
    return build(tr), build(te)


def cv_r2(tr, nd_cols):
    vals = []
    for seed in CV_SEEDS:
        ys, ps = [], []
        for a_idx, v_idx in GroupKFold(n_splits=5).split(tr, groups=tr["Plot_ID"]):
            a, v = tr.iloc[a_idx], tr.iloc[v_idx]
            xa, xv = matrices(a, v, nd_cols)
            p, _ = h.train_nn(xa, np.log1p(a["Nitrate_ppm"].to_numpy(float)), xv, seed)
            ys.append(np.log1p(v["Nitrate_ppm"].to_numpy(float))); ps.append(p)
        y, p = np.concatenate(ys), np.concatenate(ps)
        vals.append(1 - np.sum((y - p) ** 2) / np.sum((y - y.mean()) ** 2))
    return float(np.mean(vals))


def main():
    df = pd.read_csv(FEAT_CSV)
    for r, n in product(RED, NIR):
        df[f"ND_{r}_{n}"] = (df[f"P{n}"] - df[f"P{r}"]) / (df[f"P{n}"] + df[f"P{r}"])
    tr = df[df["Split"] == "train"].reset_index(drop=True)
    te = df[df["Split"] == "test"].reset_index(drop=True)
    ytr = np.log1p(tr["Nitrate_ppm"].to_numpy(float))
    ppm_te = te["Nitrate_ppm"].to_numpy(float)

    # reference: six bands + stage through this same code path
    xa, xv = matrices(tr, te, [])
    p, _ = h.train_nn(xa, ytr, xv, base.RANDOM_SEED)
    m = base.metrics_block(ppm_te, p)
    print(f"Reference six bands + stage: test R2={m['R2_log']:.3f} r={m['r_log']:.3f}")

    rows = []
    for k, combo in enumerate(combinations(ND, 3), 1):
        xa, xv = matrices(tr, te, combo)
        p, _ = h.train_nn(xa, ytr, xv, base.RANDOM_SEED)
        m = base.metrics_block(ppm_te, p)
        rows.append({"NDVI_1": label(combo[0]), "NDVI_2": label(combo[1]), "NDVI_3": label(combo[2]),
                     "cols": "|".join(combo), "Inputs": 6 + 3 + 2,
                     "R2_test": m["R2_log"], "r_test": m["r_log"], "CV_R2_train": cv_r2(tr, combo)})
        if k % 10 == 0:
            print(f"  {k}/84 done", flush=True)
    res = pd.DataFrame(rows).sort_values("R2_test", ascending=False)
    res.to_csv(OUT / "bands_plus_three_ndvi_all84.csv", index=False)

    best_test = res.iloc[0]
    best_cv = res.sort_values("CV_R2_train", ascending=False).iloc[0]
    print("\nTop 10 by test R2:\n", res.head(10)[["NDVI_1", "NDVI_2", "NDVI_3", "R2_test", "r_test", "
    ↪ CV_R2_train"]].round(3).to_string(index=False))
    print("\nTop 5 by training CV R2:\n", res.sort_values("CV_R2_train", ascending=False).head(5)[["NDVI_1", "
    ↪ NDVI_2", "NDVI_3", "R2_test", "r_test", "CV_R2_train"]].round(3).to_string(index=False))
```

```
    print(f"\nAcross all 84: test R2 median {res.R2_test.median():.3f}, range {res.R2_test.min():.3f} to {res.
     ↪ R2_test.max():.3f}; "
          f"{(res.R2_test > 0.752).sum()} of 84 exceed the six-band 0.752")

    # seed robustness: six bands alone vs the two picks
    rob = []
    for name, combo in [("six bands + stage", []), ("best by test", best_test["cols"].split("|")),
                        ("best by training CV", best_cv["cols"].split("|"))]:
        xa, xv = matrices(tr, te, combo)
        r2s = [base.metrics_block(ppm_te, h.train_nn(xa, ytr, xv, s)[0])["R2_log"] for s in ROBUST_SEEDS]
        rob.append({"Model": name, "NDVIs": ", ".join(label(c) for c in combo), "seed_mean_R2": np.mean(r2s),
     ↪ "seed_sd_R2": np.std(r2s, ddof=1)})
    rob = pd.DataFrame(rob)
    rob.to_csv(OUT / "bands_plus_three_ndvi_seed_robustness.csv", index=False)
    print("\n", rob.round(3).to_string(index=False))


if __name__ == "__main__":
    main()
```

Specular-Reflection Angle Experiment

File: `Lab_experiment/Source_codes/build_specular_reflection_dataset.py`

Converts the first-round angle-experiment readings to relative reflectance per leaf and angle.

```
# -*- coding: utf-8 -*-
"""
Specular-Reflection Angle Experiment: raw CSV -> analysis-ready dataset
========================================================================
Each Plot folder (801-806) under Lab_experiment/Data is one leaf specimen,
mounted on the servo tilt platform and read at 12 servo positions. Reuses
the same raw-frame -> detected-optical-power conversion (Equations 12-20)
and the same white-reference reflectance normalization already used
throughout Field_experiment, so this dataset is on the identical physical
basis as the rest of the thesis.

Folder layout actually used here (per author clarification, NOT the
original proposal's "10 leaves" design):
  Data/<plot>/Plant_<NN>/<lower|middle|upper>/leaf_*.csv
    plot        = one leaf specimen (801-806)
    Plant_<NN>  = servo angle position (01-12), NOT a different plant
    lower/middle/upper = sampling spots on that same leaf, pooled together
    leaf_*.csv  = repeated acquisition frames at that (leaf, angle) combo,
                  averaged (there were 7 in the original proposal; the
                  actual runs used up to 12)

Servo -> physical angle-to-normal mapping is from the author's angle
calibration sheet (specular_reflection_experiment.pdf), NOT the fixed
-60..+60 sequence in the original written proposal, which was superseded.
"""
import sys
from pathlib import Path

import numpy as np
import pandas as pd

sys.path.insert(0, str(Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\
    ↪ Masters_Thesis_Documents\Field_experiment\Source_codes\White_reference_correction")))
from power_conversion import gain_to_power_uw  # noqa: E402

DATA_ROOT = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Lab_experiment\Data")
OUT_DIR = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Lab_experiment\Processed_Data")
WHITE_REF_PATH = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment\White_reference_data")

TARGET_SENSOR = "E"
SPECTRUM_TO_WAVELENGTH_NM = {0: 780, 1: 670, 2: 700, 3: 850, 4: 660, 5: 750, 6: 840}
VALID_SPECTRUM_INDICES = [0, 1, 2, 3, 4, 5]
NOISE_SPECTRUM_INDEX = 6
WL_COLS_NM = [660, 670, 700, 750, 780, 850]
POSITIONS = ["lower", "middle", "upper"]

# Servo position (1-12) -> physical angle-to-normal (degrees), read directly
# off the author's angle-calibration sheet (specular_reflection_experiment.pdf).
# Do NOT use the proposal document's -60..+60 sequence -- superseded.
SERVO_TO_PHYSICAL_DEG = {
    1: -25, 2: -21, 3: -17, 4: -11, 5: -7, 6: -2,
    7: 0, 8: 5, 9: 10, 10: 15, 11: 18, 12: 25,
}


def collect_frames(angle_folder):
    """
```

```
    Read every leaf_*.csv across lower/middle/upper under one Plant_<NN>
    (=servo angle) folder for one leaf, pooled together (position does not
    matter per author). Returns a DataFrame: rows = one frame each, columns
    = detected power (uW) at each SpectrumIndex (0-6, including the 840nm
    noise probe), or None if nothing usable was found.
    """
    all_frames = []
    n_by_position = {}
    for position in POSITIONS:
        pos_dir = angle_folder / position
        if not pos_dir.exists():
            n_by_position[position] = 0
            continue
        csvs = sorted(pos_dir.glob("leaf_*.csv"))
        n_by_position[position] = len(csvs)
        for csv_path in csvs:
            df = pd.read_csv(csv_path)
            sensor_df = df[df["Sensor"] == TARGET_SENSOR]
            gains = sensor_df.set_index("SpectrumIndex")["Gain"]
            power = pd.Series({
                idx: gain_to_power_uw(v, SPECTRUM_TO_WAVELENGTH_NM[idx])
                for idx, v in gains.items() if idx in SPECTRUM_TO_WAVELENGTH_NM
            })
            all_frames.append(power)

    if not all_frames:
        return None, n_by_position
    return pd.concat(all_frames, axis=1), n_by_position


def parse_white_reference():
    """Same as Field_experiment/Source_codes/White_reference_correction/white_reference_correction.py."""
    plant_dirs = sorted([p for p in WHITE_REF_PATH.iterdir() if p.is_dir()], key=lambda p: p.name)
    plant_means = []
    for plant_dir in plant_dirs:
        lower_dir = plant_dir / "lower"
        csvs = sorted(lower_dir.glob("leaf_*.csv"))
        all_gains = []
        for csv_f in csvs:
            df = pd.read_csv(csv_f)
            se = (
                df[(df["Sensor"] == TARGET_SENSOR) & (df["SpectrumIndex"].isin(VALID_SPECTRUM_INDICES))]
                .set_index("SpectrumIndex")["Gain"]
            )
            power = pd.Series({
                idx: gain_to_power_uw(v, SPECTRUM_TO_WAVELENGTH_NM[idx])
                for idx, v in se.items()
            })
            all_gains.append(power)
        mean_gains = pd.concat(all_gains, axis=1).mean(axis=1)
        plant_means.append(mean_gains)
    wr = pd.concat(plant_means, axis=1).mean(axis=1)
    # index is SpectrumIndex 0-5; relabel to nm
    wr.index = [SPECTRUM_TO_WAVELENGTH_NM[i] for i in wr.index]
    return wr


def main():
    OUT_DIR.mkdir(parents=True, exist_ok=True)

    white_ref = parse_white_reference()
    print("White reference power (uW):")
    print(white_ref.round(4).to_string())

    plot_folders = sorted(
        [p for p in DATA_ROOT.iterdir() if p.is_dir()],
        key=lambda p: int(p.name),
    )
```

```
records = []
for plot_folder in plot_folders:
    leaf_id = int(plot_folder.name)
    angle_folders = sorted(
        [p for p in plot_folder.iterdir() if p.is_dir() and p.name.lower().startswith("plant_")],
        key=lambda p: int(p.name.split("_")[1]),
    )
    for angle_folder in angle_folders:
        servo_pos = int(angle_folder.name.split("_")[1])
        if servo_pos not in SERVO_TO_PHYSICAL_DEG:
            print(f"  WARNING: {plot_folder.name}/{angle_folder.name} servo position {servo_pos} "
                  f"has no calibration entry -- skipping.")
            continue

        combined, n_by_position = collect_frames(angle_folder)
        record = {
            "Leaf_ID": leaf_id,
            "Servo_Position": servo_pos,
            "Physical_Angle_deg": SERVO_TO_PHYSICAL_DEG[servo_pos],
            "N_Frames_Lower": n_by_position.get("lower", 0),
            "N_Frames_Middle": n_by_position.get("middle", 0),
            "N_Frames_Upper": n_by_position.get("upper", 0),
        }
        if combined is None:
            print(f"  WARNING: no data for {plot_folder.name}/{angle_folder.name}")
            for wl in WL_COLS_NM:
                record[f"P{wl}_uW"] = None
            record["Noise840_raw_uW"] = None
            record["Noise840_abs_uW"] = None
            records.append(record)
            continue

        record["N_Frames_Total"] = combined.shape[1]
        valid = combined.loc[combined.index.isin(VALID_SPECTRUM_INDICES)]
        mean_power = valid.mean(axis=1)
        for idx, wl in SPECTRUM_TO_WAVELENGTH_NM.items():
            if idx == NOISE_SPECTRUM_INDEX:
                continue
            record[f"P{wl}_uW"] = mean_power.get(idx, None)

        if NOISE_SPECTRUM_INDEX in combined.index:
            noise = combined.loc[NOISE_SPECTRUM_INDEX]
            record["Noise840_raw_uW"] = noise.mean()
            record["Noise840_abs_uW"] = noise.abs().mean()
        else:
            record["Noise840_raw_uW"] = None
            record["Noise840_abs_uW"] = None

        records.append(record)

df = pd.DataFrame(records)

# reflectance columns
for wl in WL_COLS_NM:
    df[f"R{wl}"] = df[f"P{wl}_uW"] / white_ref[wl]

df = df.sort_values(["Leaf_ID", "Servo_Position"]).reset_index(drop=True)

out_path = OUT_DIR / "specular_reflection_dataset.csv"
df.to_csv(out_path, index=False)
print(f"\nSaved {len(df)} rows to {out_path}")
print(df.head(15).to_string())

missing = df[df["P660_uW"].isna()]
if len(missing):
    print(f"\nWARNING: {len(missing)} (leaf, angle) combinations have no usable data:")
    print(missing[["Leaf_ID", "Servo_Position"]].to_string())
```

```
if __name__ == "__main__":
    main()
```

File: Lab_experiment/Source_codes/build_specular_reflection_dataset_data2.py

The same conversion for the second round, after the leaves were rotated by 90 degrees.

```
# -*- coding: utf-8 -*-
"""
Same as build_specular_reflection_dataset.py, applied to Data_2: the same six
leaves (801-806), same servo-angle rig and calibration, but the leaf was
rotated 90 degrees about the axis piercing the mounting-plate plane before
this second round was collected. Used to check whether the asymmetric
(+25 vs -25) reflectance trend seen in the first dataset is a property of
the leaf/rig geometry (and would rotate with the leaf) or of the sensor rig
itself (and would stay fixed regardless of leaf orientation).
"""
import sys
from pathlib import Path

import pandas as pd

sys.path.insert(0, str(Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\
    ↪ Masters_Thesis_Documents\Field_experiment\Source_codes\White_reference_correction")))
from power_conversion import gain_to_power_uw  # noqa: E402

DATA_ROOT = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Lab_experiment\Data_2")
OUT_DIR = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Lab_experiment\Processed_Data")
WHITE_REF_PATH = Path(r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Field_experiment\White_reference_data")

TARGET_SENSOR = "E"
SPECTRUM_TO_WAVELENGTH_NM = {0: 780, 1: 670, 2: 700, 3: 850, 4: 660, 5: 750, 6: 840}
VALID_SPECTRUM_INDICES = [0, 1, 2, 3, 4, 5]
NOISE_SPECTRUM_INDEX = 6
WL_COLS_NM = [660, 670, 700, 750, 780, 850]
POSITIONS = ["lower", "middle", "upper"]

SERVO_TO_PHYSICAL_DEG = {
    1: -25, 2: -21, 3: -17, 4: -11, 5: -7, 6: -2,
    7: 0, 8: 5, 9: 10, 10: 15, 11: 18, 12: 25,
}


def collect_frames(angle_folder):
    all_frames = []
    n_by_position = {}
    for position in POSITIONS:
        pos_dir = angle_folder / position
        if not pos_dir.exists():
            n_by_position[position] = 0
            continue
        csvs = sorted(pos_dir.glob("leaf_*.csv"))
        n_by_position[position] = len(csvs)
        for csv_path in csvs:
            if csv_path.stat().st_size == 0:
                print(f"  WARNING: empty frame file skipped: {csv_path}")
                n_by_position[position] -= 1
                continue
            df = pd.read_csv(csv_path)
            sensor_df = df[df["Sensor"] == TARGET_SENSOR]
            gains = sensor_df.set_index("SpectrumIndex")["Gain"]
```

```
            power = pd.Series({
                idx: gain_to_power_uw(v, SPECTRUM_TO_WAVELENGTH_NM[idx])
                for idx, v in gains.items() if idx in SPECTRUM_TO_WAVELENGTH_NM
            })
            all_frames.append(power)

    if not all_frames:
        return None, n_by_position
    return pd.concat(all_frames, axis=1), n_by_position


def parse_white_reference():
    plant_dirs = sorted([p for p in WHITE_REF_PATH.iterdir() if p.is_dir()], key=lambda p: p.name)
    plant_means = []
    for plant_dir in plant_dirs:
        lower_dir = plant_dir / "lower"
        csvs = sorted(lower_dir.glob("leaf_*.csv"))
        all_gains = []
        for csv_f in csvs:
            df = pd.read_csv(csv_f)
            se = (
                df[(df["Sensor"] == TARGET_SENSOR) & (df["SpectrumIndex"].isin(VALID_SPECTRUM_INDICES))]
                .set_index("SpectrumIndex")["Gain"]
            )
            power = pd.Series({
                idx: gain_to_power_uw(v, SPECTRUM_TO_WAVELENGTH_NM[idx])
                for idx, v in se.items()
            })
            all_gains.append(power)
        mean_gains = pd.concat(all_gains, axis=1).mean(axis=1)
        plant_means.append(mean_gains)
    wr = pd.concat(plant_means, axis=1).mean(axis=1)
    wr.index = [SPECTRUM_TO_WAVELENGTH_NM[i] for i in wr.index]
    return wr


def main():
    OUT_DIR.mkdir(parents=True, exist_ok=True)

    white_ref = parse_white_reference()
    print("White reference power (uW):")
    print(white_ref.round(4).to_string())

    plot_folders = sorted(
        [p for p in DATA_ROOT.iterdir() if p.is_dir()],
        key=lambda p: int(p.name),
    )

    records = []
    for plot_folder in plot_folders:
        leaf_id = int(plot_folder.name)
        angle_folders = sorted(
            [p for p in plot_folder.iterdir() if p.is_dir() and p.name.lower().startswith("plant_")],
            key=lambda p: int(p.name.split("_")[1]),
        )
        for angle_folder in angle_folders:
            servo_pos = int(angle_folder.name.split("_")[1])
            if servo_pos not in SERVO_TO_PHYSICAL_DEG:
                print(f"  WARNING: {plot_folder.name}/{angle_folder.name} servo position {servo_pos} "
                      f"has no calibration entry -- skipping.")
                continue

            combined, n_by_position = collect_frames(angle_folder)
            record = {
                "Leaf_ID": leaf_id,
                "Servo_Position": servo_pos,
                "Physical_Angle_deg": SERVO_TO_PHYSICAL_DEG[servo_pos],
                "N_Frames_Lower": n_by_position.get("lower", 0),
```

```
                "N_Frames_Middle": n_by_position.get("middle", 0),
                "N_Frames_Upper": n_by_position.get("upper", 0),
            }
            if combined is None:
                print(f"  WARNING: no data for {plot_folder.name}/{angle_folder.name}")
                for wl in WL_COLS_NM:
                    record[f"P{wl}_uW"] = None
                record["Noise840_raw_uW"] = None
                record["Noise840_abs_uW"] = None
                records.append(record)
                continue

            record["N_Frames_Total"] = combined.shape[1]
            valid = combined.loc[combined.index.isin(VALID_SPECTRUM_INDICES)]
            mean_power = valid.mean(axis=1)
            for idx, wl in SPECTRUM_TO_WAVELENGTH_NM.items():
                if idx == NOISE_SPECTRUM_INDEX:
                    continue
                record[f"P{wl}_uW"] = mean_power.get(idx, None)

            if NOISE_SPECTRUM_INDEX in combined.index:
                noise = combined.loc[NOISE_SPECTRUM_INDEX]
                record["Noise840_raw_uW"] = noise.mean()
                record["Noise840_abs_uW"] = noise.abs().mean()
            else:
                record["Noise840_raw_uW"] = None
                record["Noise840_abs_uW"] = None

            records.append(record)

    df = pd.DataFrame(records)

    for wl in WL_COLS_NM:
        df[f"R{wl}"] = df[f"P{wl}_uW"] / white_ref[wl]

    df = df.sort_values(["Leaf_ID", "Servo_Position"]).reset_index(drop=True)

    out_path = OUT_DIR / "specular_reflection_dataset_data2.csv"
    df.to_csv(out_path, index=False)
    print(f"\nSaved {len(df)} rows to {out_path}")
    print(df.head(15).to_string())

    missing = df[df["P660_uW"].isna()]
    if len(missing):
        print(f"\nWARNING: {len(missing)} (leaf, angle) combinations have no usable data:")
        print(missing[["Leaf_ID", "Servo_Position"]].to_string())


if __name__ == "__main__":
    main()
```

File: Lab_experiment/Source_codes/correct_850nm.py

Estimates the 850 nm reflectance of leaves 801 to 803 from their 750 nm readings.

```
# -*- coding: utf-8 -*-
"""
850nm correction for leaves 801-803.

Every band except 850nm is consistent in scale across all 6 leaves; 850nm
alone is ~10-14x lower for leaves 801-803 than for 804-806 at every angle,
a clean, leaf-wide offset rather than sporadic bad readings -- almost
certainly an 850nm LED/gain issue specific to those earlier sessions. Per
author direction: estimate R850 for 801-803 from the R750-R850 relationship
established on the unaffected leaves (804-806), applied to 801-803's own
(valid) R750 readings, rather than excluding 850nm from those leaves.
```

```
"""
import numpy as np
import pandas as pd
from scipy import stats

DATA_PATH = r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\Lab_experiment
    ↪ \Processed_Data\specular_reflection_dataset.csv"
OUT_PATH = r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\Lab_experiment\
    ↪ Processed_Data\specular_reflection_dataset_850corrected.csv"

AFFECTED_LEAVES = [801, 802, 803]
GOOD_LEAVES = [804, 805, 806]

df = pd.read_csv(DATA_PATH)

good = df[df["Leaf_ID"].isin(GOOD_LEAVES)]
slope, intercept, r, p, se = stats.linregress(good["R750"], good["R850"])
print(f"R850 ~ R750 fit on leaves {GOOD_LEAVES} (n={len(good)}):")
print(f"  R850 = {intercept:.4f} + {slope:.4f} * R750")
print(f"  r = {r:.4f}, r2 = {r**2:.4f}, p = {p:.2e}")

df["R850_original"] = df["R850"]
df["P850_original_uW"] = df["P850_uW"]
df["R850_source"] = "measured"

mask = df["Leaf_ID"].isin(AFFECTED_LEAVES)
predicted = intercept + slope * df.loc[mask, "R750"]
df.loc[mask, "R850"] = predicted
df.loc[mask, "R850_source"] = "interpolated_from_R750"
# Keep P850_uW as originally measured (it's the raw/bad channel) -- only the
# reflectance column used for analysis is corrected; P850_uW is left as-is
# with R850_source flagging which rows are not directly measured.

df.to_csv(OUT_PATH, index=False)
print(f"\nSaved corrected dataset to {OUT_PATH}")

print("\nBefore -> after, leaves 801-803:")
comp = df[mask][["Leaf_ID", "Servo_Position", "Physical_Angle_deg", "R750", "R850_original", "R850"]]
print(comp.to_string(index=False))

print("\nSanity check -- corrected R850 vs. good-leaf R850 range:")
print(f"  801-803 corrected R850: {df.loc[mask,'R850'].min():.3f} to {df.loc[mask,'R850'].max():.3f}")
print(f"  804-806 measured  R850: {df.loc[df['Leaf_ID'].isin(GOOD_LEAVES),'R850'].min():.3f} to {df.loc[df['
    ↪ Leaf_ID'].isin(GOOD_LEAVES),'R850'].max():.3f}")
```

File: Lab_experiment/Source_codes/analyze_angle_effect.py

Repeated-measures mixed-effects test of the angle effect at each wavelength (Equation 24).

```
# -*- coding: utf-8 -*-
"""
Repeated-measures mixed-effects test for an angle effect at each wavelength,
per the experiment proposal: angle as a fixed effect, leaf identity as a
random grouping effect. Run on reflectance (the thesis's standing spectral
representation).
"""
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf
import statsmodels.api as sm

DATA_PATH = r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\Lab_experiment
    ↪ \Processed_Data\specular_reflection_dataset_850corrected.csv"
WL_COLS_NM = [660, 670, 700, 750, 780, 850]

df = pd.read_csv(DATA_PATH)
```

```
df = df.dropna(subset=[f"R{wl}" for wl in WL_COLS_NM])
print(f"n = {len(df)} (leaf, angle) observations across {df['Leaf_ID'].nunique()} leaves")

results = []
for wl in WL_COLS_NM:
    col = f"R{wl}"
    d = df[["Leaf_ID", "Physical_Angle_deg", col]].copy()
    d["Leaf_ID"] = d["Leaf_ID"].astype(str)

    # Linear angle term
    md_lin = smf.mixedlm(f"{col} ~ Physical_Angle_deg", d, groups=d["Leaf_ID"])
    fit_lin = md_lin.fit(reml=False)

    # Quadratic angle term, to check nonlinearity (Fresnel reflectance is
    # strongly nonlinear/asymmetric in angle, not a straight line)
    d["Angle_sq"] = d["Physical_Angle_deg"] ** 2
    md_quad = smf.mixedlm(f"{col} ~ Physical_Angle_deg + Angle_sq", d, groups=d["Leaf_ID"])
    fit_quad = md_quad.fit(reml=False)

    # Null (intercept + random leaf effect only), for an LR test of the
    # linear angle term
    md_null = smf.mixedlm(f"{col} ~ 1", d, groups=d["Leaf_ID"])
    fit_null = md_null.fit(reml=False)

    lr_stat = -2 * (fit_null.llf - fit_lin.llf)
    from scipy import stats as spstats
    lr_p = spstats.chi2.sf(lr_stat, df=1)

    slope = fit_lin.params["Physical_Angle_deg"]
    ci = fit_lin.conf_int().loc["Physical_Angle_deg"]
    p_wald = fit_lin.pvalues["Physical_Angle_deg"]

    quad_coef = fit_quad.params.get("Angle_sq", np.nan)
    quad_p = fit_quad.pvalues.get("Angle_sq", np.nan)

    results.append({
        "wavelength_nm": wl,
        "n_obs": len(d),
        "linear_slope_per_deg": slope,
        "slope_CI_low": ci[0],
        "slope_CI_high": ci[1],
        "slope_p_wald": p_wald,
        "LR_stat_vs_null": lr_stat,
        "LR_p": lr_p,
        "quad_coef": quad_coef,
        "quad_p": quad_p,
        "leaf_random_var": fit_lin.cov_re.iloc[0, 0] if fit_lin.cov_re.shape[0] else np.nan,
        "residual_var": fit_lin.scale,
    })

res_df = pd.DataFrame(results)
pd.set_option("display.width", 160)
pd.set_option("display.max_columns", 20)
print(res_df.round(5).to_string(index=False))

out_path = r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\Lab_experiment\
    ↪ Processed_Data\angle_effect_mixed_model_results.csv"
res_df.to_csv(out_path, index=False)
print(f"\nSaved to {out_path}")

# Benjamini-Hochberg FDR correction across the 6 wavelengths (matching the
# multiplicity-correction convention already used elsewhere in the thesis)
from statsmodels.stats.multitest import multipletests
rej, p_adj, _, _ = multipletests(res_df["LR_p"], alpha=0.05, method="fdr_bh")
res_df["LR_p_FDR"] = p_adj
res_df["significant_FDR_0.05"] = rej
print("\nWith Benjamini-Hochberg FDR correction across 6 wavelengths:")
print(res_df[["wavelength_nm", "LR_p", "LR_p_FDR", "significant_FDR_0.05"]].to_string(index=False))
```

```
res_df.to_csv(out_path, index=False)
```

File: `Lab_experiment/Source_codes/fit_fresnel_with_baseline_combined.py`

Fit of the diffuse-plus-specular Fresnel model (Equation 25) to both rounds pooled.

```
# -*- coding: utf-8 -*-
"""
Same diffuse-baseline-separated Fresnel model as fit_fresnel_with_baseline.py,
now fit on BOTH bench collections pooled together:
  - specular_reflection_dataset_850corrected.csv (original leaf orientation,
    801-803's 850nm hardware anomaly already interpolated)
  - specular_reflection_dataset_data2.csv (same 6 leaves, rotated 90 degrees
    about the axis piercing the mounting-plate plane; 850nm checked and found
    NOT anomalous for 801-803 in this second collection, so used as-is)

Pooling both roughly doubles the (leaf, angle) observations available to
constrain the shared rig-level parameters (n_eff, theta0, sigma_g, k_lambda,
b_lambda), which the with-baseline single-orientation fit needed (theta0 and
sigma_g were poorly identified there). Leaf orientation is NOT included as a
model term -- both collections are treated as additional replicate
observations of the same underlying (leaf, angle) -> reflectance relationship,
per the author's own framing of Data_2 as a check on Data, not a separate
experiment.

IMPORTANT CAVEAT (communicated to the user, not silently corrected here):
the tilt rig's pivot is off-center (confirmed: ~4cm sensor-to-plate distance
at +25 deg vs ~5cm at -25 deg), so Physical_Angle_deg is confounded with
sensor-to-leaf distance. This fit still treats theta as a pure angle-of-
incidence term, same as before -- the resulting specular offset is therefore
a MIXTURE of true Fresnel specular reflectance and this uncorrected distance/
geometric-factor effect, not a distance-clean specular estimate. No distance
measurements exist at intermediate angles to separate the two.
"""
import numpy as np
import pandas as pd
from scipy.optimize import least_squares
from sklearn.metrics import r2_score

DATA_PATH_1 = r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Lab_experiment\Processed_Data\specular_reflection_dataset_850corrected.csv"
DATA_PATH_2 = r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\
    ↪ Lab_experiment\Processed_Data\specular_reflection_dataset_data2.csv"
OUT_DIR = r"C:\Users\Albert\Dropbox\Kansas_State_University\Research\Masters_Thesis_Documents\Lab_experiment\
    ↪ Processed_Data"
WL_COLS_NM = [660, 670, 700, 750, 780, 850]

df1 = pd.read_csv(DATA_PATH_1).dropna(subset=[f"R{wl}" for wl in WL_COLS_NM])
df1["Orientation"] = "original"
df2 = pd.read_csv(DATA_PATH_2).dropna(subset=[f"R{wl}" for wl in WL_COLS_NM])
df2["Orientation"] = "rotated_90deg"

# known bad acquisitions in Data_2, flagged and excluded (isolated dropped
# frames -- see 801/-11deg and 802/[15,18,25]deg discussion)
bad2 = ((df2["Leaf_ID"] == 801) & (df2["Physical_Angle_deg"] == -11)) | \
       ((df2["Leaf_ID"] == 802) & (df2["Physical_Angle_deg"].isin([15, 18, 25])))
df2 = df2[~bad2]
print(f"Data_2: excluded {bad2.sum()} flagged bad-acquisition rows, {len(df2)} remain")

df = pd.concat([df1, df2], ignore_index=True)
print(f"Combined: {len(df1)} (original) + {len(df2)} (rotated) = {len(df)} (leaf, angle) rows")

leaves = sorted(df["Leaf_ID"].unique())

long_rows = []
for _, row in df.iterrows():
```

```
    for wl in WL_COLS_NM:
        long_rows.append({"Leaf_ID": row["Leaf_ID"], "Orientation": row["Orientation"],
                          "theta": row["Physical_Angle_deg"], "wl": wl, "R": row[f"R{wl}"]})
long_df = pd.DataFrame(long_rows)


def fresnel_R(theta_deg, n_eff):
    theta_i = np.radians(theta_deg)
    sin_t = np.clip(np.sin(theta_i) / n_eff, -1.0, 1.0)
    cos_t = np.sqrt(1 - sin_t ** 2)
    cos_i = np.cos(theta_i)
    Rs = ((cos_i - n_eff * cos_t) / (cos_i + n_eff * cos_t)) ** 2
    Rp = ((n_eff * cos_i - cos_t) / (n_eff * cos_i + cos_t)) ** 2
    return 0.5 * (Rs + Rp)


def G(theta_deg, sigma_g):
    return np.exp(-(theta_deg ** 2) / (2 * sigma_g ** 2))


def specular_component(params, theta, wl_idx, n_wl):
    n_eff, theta0, sigma_g = params[:3]
    k = params[3:3 + n_wl]
    eff_theta = theta + theta0
    return k[wl_idx] * fresnel_R(eff_theta, n_eff) * G(eff_theta, sigma_g)


def model_predict(params, theta, wl_idx, n_wl):
    b = params[3 + n_wl:3 + 2 * n_wl]
    return b[wl_idx] + specular_component(params, theta, wl_idx, n_wl)


def fit_model(train_df):
    wl_index = {wl: i for i, wl in enumerate(WL_COLS_NM)}
    theta = train_df["theta"].values
    wl_idx = train_df["wl"].map(wl_index).values
    y = train_df["R"].values
    n_wl = len(WL_COLS_NM)

    def resid(params):
        return model_predict(params, theta, wl_idx, n_wl) - y

    k0 = 2.0
    b0 = [long_df[long_df["wl"] == wl]["R"].min() * 0.5 for wl in WL_COLS_NM]
    x0 = [1.4, 0.0, 30.0] + [k0] * n_wl + b0

    # Bounds needed to prevent the optimizer from running off into a
    # numerically degenerate region where b_lambda and k_lambda nearly
    # cancel (huge, unphysical individual values, e.g. b=-271, k=823) --
    # this happened with unconstrained "lm" once Data_2 was pooled in.
    # n_eff: literature-plausible dielectric range. theta0: within/near the
    # tested +-25deg sweep, not an unconstrained extrapolation. sigma_g:
    # capped at 200deg -- beyond that G(theta) is indistinguishable from a
    # flat 1.0 over any realistic angle anyway, so larger values are pure
    # numerical slack, not a meaningful fit. k_lambda >= 0 (specular can't be
    # negative). b_lambda in [0, 3] (diffuse baseline must be a plausible
    # non-negative reflectance-like value, matching the observed R range).
    lb = [1.01, -60.0, 1.0] + [0.0] * n_wl + [0.0] * n_wl
    ub = [3.0, 60.0, 200.0] + [20.0] * n_wl + [3.0] * n_wl
    x0 = np.clip(x0, lb, ub)
    result = least_squares(resid, x0, method="trf", bounds=(lb, ub), max_nfev=30000)
    return result.x


def evaluate(params, eval_df):
    wl_index = {wl: i for i, wl in enumerate(WL_COLS_NM)}
    theta = eval_df["theta"].values
    wl_idx = eval_df["wl"].map(wl_index).values
```

```
    y = eval_df["R"].values
    pred = model_predict(params, theta, wl_idx, len(WL_COLS_NM))
    resid = y - pred
    rmse = float(np.sqrt(np.mean(resid ** 2)))
    r2 = r2_score(y, pred)
    r = float(np.corrcoef(y, pred)[0, 1]) if np.std(pred) > 0 else np.nan
    return r2, rmse, r


full_params = fit_model(long_df)
n_eff, theta0, sigma_g = full_params[:3]
n_wl = len(WL_COLS_NM)
k_vals = full_params[3:3 + n_wl]
b_vals = full_params[3 + n_wl:3 + 2 * n_wl]

print("\n=== Full-dataset fit (with diffuse baseline, both orientations pooled) ===")
print(f"n_eff = {n_eff:.4f}, theta0 = {theta0:.4f} deg, sigma_g = {sigma_g:.4f} deg")
for wl, k, b in zip(WL_COLS_NM, k_vals, b_vals):
    print(f"  {wl}nm: b_lambda (diffuse) = {b:.4f}, k_lambda (specular scale) = {k:.4f}")

r2_full, rmse_full, r_full = evaluate(full_params, long_df)
print(f"\nFull-dataset fit quality: R2={r2_full:.4f}, r={r_full:.4f}, RMSE={rmse_full:.5f}")

print("\n=== Leave-one-leaf-out cross-validation (across both orientations) ===")
loo_r2 = []
for held_out in leaves:
    train = long_df[long_df["Leaf_ID"] != held_out]
    test = long_df[long_df["Leaf_ID"] == held_out]
    params = fit_model(train)
    r2, rmse, r = evaluate(params, test)
    loo_r2.append(r2)
    print(f"  held out leaf {held_out} (both orientations): test R2={r2:.4f}, RMSE={rmse:.5f}")
print(f"\nMean held-out R2 = {np.mean(loo_r2):.4f} (range {min(loo_r2):.4f} to {max(loo_r2):.4f})")

print("\n=== Specular component only, at theta=0 (field measurement angle) ===")
eff_theta = 0.0 + theta0
rf = fresnel_R(eff_theta, n_eff)
g = G(eff_theta, sigma_g)
spec_at_zero = {}
for wl, k, b in zip(WL_COLS_NM, k_vals, b_vals):
    s = k * rf * g
    spec_at_zero[wl] = s
    pct = 100 * s / (b + s) if (b + s) != 0 else float("nan")
    print(f"  {wl}nm: specular component = {s:.4f}, diffuse baseline = {b:.4f}, "
          f"specular fraction of total predicted signal = {pct:.1f}%")

out = pd.DataFrame({
    "parameter": ["n_eff", "theta0_deg", "sigma_g_deg"]
    + [f"k_{wl}nm" for wl in WL_COLS_NM] + [f"b_{wl}nm" for wl in WL_COLS_NM]
    + [f"specular_at_theta0_{wl}nm" for wl in WL_COLS_NM],
    "value": list(full_params) + [spec_at_zero[wl] for wl in WL_COLS_NM],
})
out.to_csv(f"{OUT_DIR}/fresnel_with_baseline_fit_combined.csv", index=False)
print(f"\nSaved to {OUT_DIR}/fresnel_with_baseline_fit_combined.csv")
```