DEVELOPMENT OF A FIELD-BASED HIGH-THROUGHPUT MOBILE PHENOTYPING PLATFORM

by

JARED W. BARKER III

B.S., University of the Philippines Los Baños, 2010

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Biological and Agricultural Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2014

Approved by:

Major Professor
Naiqian Zhang

Copyright

Abstract

In order to meet food, fiber, and bio-fuel needs of a growing world population, crop-breeding methods must be improved and new technologies must be developed. One area under focus is the decoding of the genetic basis of complex traits, such as yield and drought stress tolerance, and predicting these traits from genetic composition of lines or cultivars. In the last three decades, significant advances in genotyping methods have resulted in a wealth of genomic information; however, little improvement has occurred for methods of collecting corresponding plant trait data, especially for agronomic crops. This study developed a mobile, field-based, high-throughput sensor platform for rapid and repeated measurement of plant characteristics. The platform consisted of three sets of sensors mounted on a high-clearance vehicle. Each set of sensors contained two infrared thermometers (IRT), one ultrasonic sensor, one Crop Circle, and one GreenSeeker. Each sensor set measured canopy temperature, crop height, and spectral reflectance. In addition to the sensors, the platform was equipped with an RTK-GPS system that provided precise, accurate position data for georeferencing sensor measurements. Software for collecting, georeferencing, and logging sensor data was developed using National Instruments LabVIEW and deployed on a laptop computer. Two verification tests were conducted to evaluate the phenotyping system. In the first test, data timestamps were analyzed to determine if the system could collect data at the required rate of 10 Hz and 5 Hz for sensor data and position data, respectively. The determination was made that, on average, IRT, ultrasonic, and Crop Circle data are received in intervals of 100 ms (SD = 10 ms), GreenSeeker data are received in intervals of 122 ms(SD=10 ms), and position data are received in intervals of 200 ms (SD = 32 ms). The second test determined that a statistically significant relationship exists between sensor readings and ambient light intensity and ambient temperatures. Whether the relationship is significant from a practical stand point should be determined based on specific application of the sensors.

Table of Contents

List of Figures vi
List of Tables x
Chapter 1 - Introduction 1
Chapter 2 - Research Objectives 3
Chapter 3 - Review of Literature 4
Phenomics and Phenotyping 4
Phenotypic Traits 5
Technologies for Phenotyping 6
Spectral and Imaging Technologies 7
Visible Light Imaging 8
Near Infrared 9
Mid Infrared 10
Hyperspectral Imaging 11
Distance Sensors 13
Satellite-Based Positioning Systems 14
Mobile Field-Based Phenotyping Platforms 15
Chapter 4 - Materials and Methods 17
System Design 17
Software Development 22
Sensor Module Software Design 25
GreenSeeker 25
Crop Circle 29
2:1 and 15:1 IRT 31
Distance Sensors 32
Positioning System 34
Main Program 35
Verification Test 41
System Timing and Position Errors 41
Determining Effects of Ambient Light Intensity and Temperature 43

Chapter 5 - Results and Discussion 45
System Timing and Position Errors 45
Position Errors 60
Effects of Ambient Light Intensity and Ambient Temperature 64
GreenSeeker 64
Crop Circle 67
Ultrasonic Sensor 72
Laser Sensor 75
IRT 78
Chapter 6 - Summary and Conclusions 83
References 85
Appendix A - R Programs for Statistical Analysis 88
Appendix B - Statistical Analysis Results Not Included in Main Text 117
Appendix D - LabVIEW Code 123
Appendix E - Sample Phenotyper Software Output File 136
Appendix F - CR850 Program For Measuring Temperature 137

List of Figures

Figure 3.1 Electromagnetic spectrum comprised of all possible wavelengths and frequencies of electromagnetic radiation (http://withfriendship.com). 8
Figure 3.2 Simplified diagram of a hyperspectral sensor (https://www.spacecomputer.com/hsi_technology.html) 12
Figure 4.1 Phenotyping platform consisting of sensors, GNSS units, and data acquistion hardware mounted on a high clearance vehicle 17
Figure 4.2 One sensor set composed of (left-to-right) two IRTs, one ultrasonic sensor, one Crop Circle, and one GreenSeeker..... 18
Figure 4.3 A) Laser sensor mounted inline with the left wheel-track to sense boom height relative to the ground. B) IRTs mounted side-by-side so measurement areas would overlap. C) GNSS receivers mounted on each end of the sensor boom. 20
Figure 4.4 Three communication hubs and one analog data acquisition module routing signals from sixteen sensors and two GNSS receivers to a personal computer. 21
Figure 4.5 The phenotyping sofware comprised of a total of 20 sensors and write-to-file loops; sensor loops place sensor data in queues write-to-file loops read data from the queues and write them to a file. 24
Figure 4.6 Real-time sensor readings displayed using custom gauges useful for monitoring purposes during phenotyping. 25
Figure 4.7 Trimble RT200 interface module for the GreenSeeker sensor head 26
Figure 4.8 If data arrives slower than the value wired to the wait function, the data rate controls the loop rate..... 28
Figure 4.9 The GreenSeeker subVI receives NDVI data sent by individual GreenSeeker sensors. 28
Figure 4.10 Five pieces of data, delimited by commas can be parsed from the Crop Circle data string. 30
Figure 4.11 The Crop Circle subVI parses five tokens from the data string sent by the Crop Circle sensor..... 30
Figure 4.12 The IRT subVI commands the IRT sensor to send temperature data, parses the data string, and outputs the temperature. 32

Figure 4.13 The NI DAQ Assistant was used to configure the NI 9207 data acquisition module. 33
Figure 4.14 The distance sensors are equipped with teach buttons for setting the lower and upper distance limits;. The corresponding sensor voltage outputs are 0 V and 9.9 V for the lower and upper limits, respectively. 34
Figure 4.15 The GNSS loop contains the GPS subVI which receives and parses GGA strings from the FmX units; the GNSS loop repeats no faster than 10 Hz. 35
Figure 4.16 The button-servicing loop checks the value of several control buttons on the user-interface. The Wait function within the loop controls the loop rate, which in this example is set to 1 Hz or a period of 1000 ms. 36
Figure 4.17 GreenSeeker write-to-file loop. Four other write-to-file loops are similar. 38
Figure 4.18 The Setup tab of the phenotyping software contains the cotnrol buttons, distance sensor calibration settings, and the field and sensor offset settings. 39
Figure 4.19 The Monitor tab contains custom gauges for real-time monitoring of the sensor readings. 39
Figure 4.20 The GPS tab plottings the position data from the GNSS receivers. 40
Figure 4.21 The raw data tab contains tables that contain the five most recent data points from all the sensors. 40
Figure 4.22 A) Sensors positioned 100 cm above a green colored carpet for the second verification test B) Thermcouples connected to a Cambell Scientific CR850 datalogger used to measure ambient and carpet surface temperatures C) Sper Scientific light meter used to measure light intensity 44
Figure 5.1 Plot of read periods for left and right GNSS receivers during the third trial showing significant delays in service of the serial port occurring less than 5 times in over 6,000 samples. 47
Figure 5.2 For almost all GNSS samples, the difference between TDif and read periods is zero. 48
Figure 5.3 Wait function inside write-to-file loops set to 10 ms (100 Hz) 49
Figure 5.4 Plots of read periods for GreenSeekers during the third trial showing significant delays in service of the serial port occurring less than 10 times in over 10,000 samples 52
Figure 5.5 Plots of read periods for Crop Circles during the third trial showing significant delays in service of the serial port occurring less than 10 times in over 10,000 samples 54

Figure 5.6 Plots of read periods for 15:1 IRTS during the third trial showing significant delays in service of the serial port occurring less than 10 times in over 10,000 samples 58
Figure 5.7 Plots of read periods for 2:1 IRTS during the third trial showing significant delays in service of the serial port occurring less than 10 times in over 10,000 samples 59
Figure 5.8 Plot of distance sensor read periods for the third trial showing that most read periods were between 90 ms and 110 ms and very few were over 120 ms 60
Figure 5.9 Fitted and residual plots of GreenSeeker data vs. ambient light intensity 65
Figure 5.10 Fitted and residual plots of GreenSeeker NDVI data with respect to ambient temperature 67
Figure 5.11 Fitted and residual plots of Crop Circle Channel 1 reflectance (670 nm) vs. light intensity 68
Figure 5.12 Fitted and residual plots of Crop Circle Channel 1 reflectance (670 nm) vs. ambient temperature 69
Figure 5.13 Fitted and residual plots of Crop Circle NDVI (670 nm, 760 nm) vs. light intensity 71
Figure 5.14 Fitted and residual plots of Crop Circle NDVI (670 nm, 760 nm) vs. ambient temperature 72
Figure 5.15 Fitted and residual plots of ultrasonic sensor data with respect to ambient light intensity 73
Figure 5.16 Fitted and residual plots of ultrasonic sensor data with respect to ambient temperature 74
Figure 5.17 Plots of raw laser sensor data vs. light intensity and ambient temperature 76
Figure 5.18 Fitted and residual plots of laser sensor data vs. ambient light intensity 77
Figure 5.19 Fitted and residual plots of laser sensor data with respect to ambient temperature .. 78
Figure 5.20 Fitted and residual plots of 15:1 IRT sensor data vs. ambient light intensity 79
Figure 5.21 Plot of 2:1 and 15:1 IRT with respect to ambient temperature 80
Figure 5.22 Fitted and residual plots of 2:1 IRT sensor data vs. ambient light intensity 81
Figure 5.23 Comparison of 2:1 and 15:1 IRT temperature measurements with respect to ambient light intensity 82
Figure B.1 Fitted and residual plots of Crop Circle Channel 2 reflectance (760 nm) vs. ambient light intensity 118

Figure B.2 Fitted and residual plots of Crop Circle Channel 2 reflectance (760 nm) vs. ambient temperature 119
Figure B.3 Fitted and residual plots of Crop Circle Channel 3 reflectance (550 nm) vs. ambient light intensity 121
Figure B.4 Fitted and residual plots of Crop Circle Channel 3 reflectance (550 nm) vs. ambient temperature 122
Figure D.1 GNSS LabVIEW module connector pane 123
Figure D.2 GNSS LabVIEW module front panel 123
Figure D.3 GNSS LabVIEW module block diagram 124
Figure D.4 GreenSeeker module connector pane 125
Figure D.5 GreenSeeker module font panel 125
Figure D.6 GreenSeeker module block diagram 126
Figure D.7 15:1 IRT module connector pane 127
Figure D.8 15:1 IRT module front panel 127
Figure D.9 15:1 IRT module block diagram 128
Figure D.10 2:1 IRT module connector pane 129
Figure D.11 2:1 IRT module front panel 129
Figure D.12 2:1 IRT module block diagram 130
Figure D.13 Crop Circle module connector pane 131
Figure D.14 Crop Circle module connector pane 131
Figure D.15 Crop Circle module connector pane 132
Figure D.16 Distance sensor module connector pane 133
Figure D.17 Distance sensor module front panel 133
Figure D.18 Distance sensor module block diagram 134
Figure D.19 Main program block diagram 135

List of Tables

Table 4.1 Summary of sensor power requirements and output signals. 22
Table 5.1 System timing results for the average, standard deviation, and range of the ready-to-read times (TI or Time In), read times (TS or Timestamp), and waiting-to-read times (TDif) of the left and right GNSS receivers. 46
Table 5.2 System timing results for write-to-file times for GNSS receivers for Trials 1, 2, and 3 49
Table 5.3 Summary of system timing results for all three trials for the average, standard deviation, and range of the read times (TS), write-to-file times (WT), data send times (GSTS), and waiting-to-read times (TDif) of GreenSeeker sensors in all three rows 51
Table 5.4 Summary of the average, standard deviation, and range of the read times (TS), write-to-file times (WT), and waiting-to-read times (TDif) of Crop Circle sensors in all three rows and for all three trials 53
Table 5.5 Summary of the average, standard deviation, and range of the read times (TS) and write-to-file times (WT) of IRT sensors in all three rows and for all three trials 56
Table 5.6 Summary of the average, standard deviation, and range of command send-to-read times (TDif) of IRT sensors in all three rows and for all three trials 57
Table 5.7 Summary of the average, standard deviation, and range of the read times (TS), write-to-file times (WT), and waiting-to-read times (TDif) of distance sensors in all three rows and for all three trials 57
Table 5.8 Summary statistics for sensor data time delays relative to position data 63
Table 5.9 Summary statistics of GreenSeeker data vs. light intensity linear regression model ... 66
Table 5.10 Summary statistics of Greenseeker NDVI vs. ambient temperature linear regression model 66
Table 5.11 Summary statistics of Crop Circle Channel 1 reflectance vs. light intensity linear regression model 69
Table 5.12 Summary statistics of Crop Circle Channel 1 reflectance vs. ambient temperature linear regression model 70
Table 5.13 Summary statistics of Crop Circle NDVI (670 nm, 760 nm) vs. light intensity linear regression model 70

Table 5.14 Summary statistics of Crop Circle NDVI (670 nm, 760 nm) vs. ambient temperature linear regression model 71
Table 5.15 Summary statistics of ultrasonic sensor distance vs. light intensity linear regression model 73
Table 5.16 Summary statistics of ultrasonic distance sensor vs. ambient temperature linear regression model 74
Table 5.17 Summary statistics of laser sensor distance vs. light intensity linear regression model 76
Table 5.18 Summary statistics of laser sensor distance vs. light intensity linear regression model 77
Table 5.19 Summary statistics of 15:1 IRT ΔT vs. light intensity linear regression model 80
Table 5.20 Summary statistics of 2:1 IRT ΔT vs. light intensity linear regression model 82
Table B.1 Summary statistics of Crop Circle Channel 2 reflectance vs. light intensity linear regression model 117
Table B.2 Summary statistics of Crop Circle Channel 2 reflectance vs. ambient temperature linear regression model 117
Table B.3 Summary statistics of Crop Circle Channel 3 reflectance vs. light intensity linear regression model 120
Table B.4 Summary statistics of Crop Circle Channel 3 reflectance vs. ambient temperature linear regression model 120
Table E.1 Columns from the GreenSeeker output data file 136

Chapter 1 - Introduction

Twenty-first century plant science and crop improvement is faced with the challenge of meeting worldwide food, fiber, and bio-fuel needs of an increasing population expected to exceed 9 billion by 2050 (http://www.un.org/en/development/desa/population/). Improvements in annual crop yields through traditional breeding programs cannot meet the projected demand of three major cereal crops: rice, maize, and wheat. New plant genotypes with intrinsic high yields and yield stability under drought and salinity stress, adaptable to future climate conditions must be developed (Furbank and Tester 2011). In order to quickly and efficiently achieve this, the genetic basis of complex plant traits, such as yield and drought stress tolerance, must be understood and methods for predicting these traits from genetic composition of lines or cultivars must be improved (Cobb et al. 2013; White et al. 2012). Understanding the genetic basis of complex plant traits requires linking genetic information with correspondingly observed phenotypic data (Montes, Melchinger, and Reif 2007b).

The genomics revolution and advances in gene technology have resulted in a wealth of genomic information (Furbank and Tester 2011). Genotyping methods have become highly mechanized, uniform across organisms, and relatively low in cost, with costs decreasing every year (Cobb et al. 2013). However, little improvement of methods for collecting plaint trait data or phenotyping have occurred in the last 30 years, especially for collecting data for single plots or plants in field-based situations (White et al. 2012). Availability of high-quality phenotypic and corresponding environmental data is becoming essential to understand the genotype-to-phenotype relationship; due to lack of phenotypic data, phenotyping is currently considered the major operational bottleneck of genetic analysis (Cobb et al. 2013; Furbank and Tester 2011). To improve complex traits through genomic selection, phenotypic data from thousands of plant varieties grown in replications under various environmental conditions is necessary, and measurements must be repeatedly taken throughout plant development in order to observe the interaction between the expression of plant traits and the environment (Cabrera Bosquet et al. 2012; Montes, Melchinger, and Reif 2007b).

Current methods for collecting phenotypic data on field-based plots require a researcher to visit each plot and manually measure specific parameters such as canopy temperature or plant height. For less than a hundred plots, manual phenotyping is feasible, but for hundreds and

thousands of plots, it is too laborious, time consuming, and costly. Consequently, many breeding programs only collect yield data taken only at the end of the growing season (Furbank and Tester 2011).

Therefore, a need exists for a flexible, robust, mobile, multi-sensor platform, capable of quickly and efficiently collecting data, from hundred to thousands of plant plots, that can be used to reliably estimate phenotypic traits. The platform should be capable of handling a variety of sensors that can collect different types of data, such as plant height, canopy temperature, and various spectral readings. The platform should also be equipped with high accuracy and precisions positioning systems, such as RTK-GPS, for georeferencing, which is necessary to correctly link data to specific plots or plants.

Chapter 2 - Research Objectives

The overall objective of this work was to advance phenomics through the development of a mobile sensor platform for field-based high-throughput phenotyping. It was envisioned that the developed platform would serve as a model and resource for future phenotyping platform designs.

The specific objectives were as follows:

1. Assemble a sensor system composed of GreenSeekers, CropCircles, infrared thermometers, distance sensors, and GPS on a high-clearance field vehicle.
2. Develop software for interfacing with various types of sensors, in addition to collecting, geo-referencing, formatting, and saving data from the sensors into a convenient format for further processing and analysis.
3. Test and verify that the developed platform timely collects, geo-references, formats, and saves data from the sensors and GPS.
4. Determine if light intensity and ambient temperature affect sensor readings.

Chapter 3 - Review of Literature

Phenomics and Phenotyping

Phenomics is the study of phenomes, or complete phenotypes, and how phenomes change over time (Mogel 2013). A phenome can be defined as the combination of all phenotypes of an individual organism. Phenotype can be broadly defined as the combination of all morphological, physiological, anatomical, chemical, developmental, and behavioral characteristics that comprise an individual organism (Pieruschka and Poorter 2012). A more specific, applicable definition for this study describes phenotype as the observable physical characteristics or traits of an organism, such as plant height, color, and yield (Fiorani and Schurr 2013; Mogel 2013). The expressed set or specific values of these observable characteristics, which are a subset of all possible values that comprise the phenome, is a result of the interaction between genetic information or genotype of the organism and the environment (Fiorani and Schurr 2013). Therefore, under different environmental conditions, organisms with identical genetic make up will have different expressed characteristics. For example, two genetically identical wheat crops will have significantly different yields if one crop experiences drought stress while the other is well watered, assuming all other environmental conditions remain equal. This interaction between genotype and the environment is complex and dynamic, and the phenotypic response of quantitative traits, such as yield, is characterized by response curves, which are continuous and typically non-linear (Fiorani and Schurr 2013).

Phenotyping is broadly defined as the set of all methods and protocols for measuring phenotypes (Fiorani and Schurr 2013). Phenotyping methods can be manual in which a person utilizes a ruler or highly-mechanized, automated ultrasonic sensors mounted to a tractor to measure plant height in a field.

Development of modern phenotyping methods clearly lags behind that of modern genotyping. Developments in next-generation sequencing methods for DNA have significantly reduced the time and cost required for genotyping, and these advances have significantly increased genotypic data. Continued development promises to make feasible the sequencing/resequencing of model plant and major crop species and parental and progeny lines of mapping populations, allowing plant breeders to link fragments of chromosome in the progeny line to the parent (Varshney et al. 2009). These genomic advancements will improve the

precision and efficiency of phenotype predictions from genotypes, consequently accelerating the development and improvement of modern crop cultivars (Cobb et al. 2013).

In order to capitalize on genomic advances, high quality phenotypic data is necessary to complete the "genotype + environment = phenotype" equation. In molecular breeding strategies such as genomic selection and marker-assisted recurrent selection, phenotypic data is used to train prediction models and identify markers for subsequent selection through generations. In transgenic studies, phenotypic data is used to identify promising events (Araus and Cairns 2013). Phenotyping is as important as genotyping for linking genes to traits; however, in contrast to genotyping, phenotyping has remained relatively primitive at the macroscopic level, especially for field-based, agronomic crops which still rely on manual measurement and assessment (Pieruschka and Poorter 2012). For example, (Yang et al. 2013) described current methods for phenotyping rice as manual, subjective, inefficient, destructive, and error-prone. Consequently, (Yang et al. 2013) suggested that in order for rice phenomics to match the level of rice genomics, reliable, automatic, multifunctional, and high-throughput phenotyping platforms must be developed. This observation is also true for other agronomic crops.

Development of field-based, high-throughput phenotyping platforms is the key to addressing the phenotyping bottleneck. Indoor or controlled environment phenotyping systems are continually being developed, but well-defined, controlled conditions within greenhouses are a stark contrast to the dynamic, outdoor environment in which agronomic crop production actually occurs (Pieruschka and Poorter 2012). Field-based platforms must be robust, non-invasive, and capable of covering hundreds and thousands of research plots in a short amount of time. Successful deployment requires the integration of automated cultivation systems, novel sensing technologies for rapid measurement and data collection, precise environmental monitoring, and a robust, well thought-out system for data management (Fiorani and Schurr 2013).

Phenotypic Traits

In addition to the development of a phenotyping platform, another key factor in solving the phenotyping bottleneck is a clear understanding of physiological plant traits that are indicative of the target performance for crop improvement. The measurement of a large number of data points which could be highly correlated or not indicative of the target performance would

be a waste of time and resources (Fiorani and Schurr 2013). A few examples of physiological traits that are highly heritable, able to be non-destructively measured, or positively associated with crop performance and yield include plant canopy evaporative cooling; vegetative, pigment, and water indices; canopy size; and morphological traits such as plant height.

Evaporative cooling from the canopy surface can be estimated by measuring plant canopy temperature (CT). CT is widely used in crop breeding because it takes into account the effects of many plants within a crop canopy, thus reducing errors associated with plant-to-plant and leaf-to-leaf variation. CT is also linked to many physiological factors such as stomatal conductance, plant water status, root/vascular capacity, and yield performance under different environments. Yield under heat and drought stress has been shown to be positively associated with cooler CT ("Physiological Breeding II: a Field Guide to Wheat Phenotyping" 2012).

Many vegetative, pigment, and water indices can be calculated from spectral reflectance measurements for crop characteristics. Water indices and vegetative indices have been shown to be reliably associated with crop performance. These indices can be used to estimate green biomass, leaf area index, photosynthetic potential, and plant water status. An example of a widely used index is the Normalized Difference Vegetative Index (NDVI), used to estimate early cover, pre-anthesis biomass, nitrogen content, and post-anthesis stay-green ("Physiological Breeding II: a Field Guide to Wheat Phenotyping" 2012). Stay-green is a term that refers to delayed senescence of plant foliage (Thomas and Ougham 2014).

Plant height is a highly heritable trait, showing minimial rank order changes across enviroments. It is strongly associated with carbohydrate storage capacity and harvest index, which is the ratio of yield to total plant weight. Plant height is also important for ease of mechanical harvesting. A longer stem has a higher carbohydrate storage capacity and is easier to harvest but increases lodging and reduces the harvest index. A shorter stem has a lower carbohydrate storage capacity and may be more difficult to harvest ("Physiological Breeding II: a Field Guide to Wheat Phenotyping" 2012).

Technologies for Phenotyping

Although development of phenotyping methods and protocols may have been stagnant, sensor, computer, communications, and software technology has continued to improve. A variety of tools, sensors, and devices useful for phenotyping have been and are continuously being

developed, such as ultrasonic and laser sensors, infrared thermometers, digital cameras, spectral reflectance sensors, positioning systems, data acquisitions systems, personal computers, and programming and software development tools. A key characteristic of sensor technologies is the capability for non-destructive, proximal, and high-throughput sensing of plant specimens in outdoor, field-based conditions. Developments in computers, communications, and software systems allow interfacing of a variety of sensors to a single computer for data collection, monitoring, and control. The combination of these technologies into a single system for phenotyping has great potential for screening a wide variety of characteristics relevant to crop improvement, such as growth rate, yield potential, adaptation to abiotic and biotic stress, and quality traits (Araus and Cairns 2013).

Spectral and Imaging Technologies

Spectral and imaging technologies measure emitted or reflected electromagnetic waves. The electromagnetic spectrum consists of all possible wavelengths of electromagnetic radiation, from radio waves to gamma rays (Figure 3.1). Most spectral and imaging technologies currently used for phenotyping utilize electromagnetic radiation within visible and infrared bands transmitted or reflected from the crop (Araus and Cairns 2013). Incoming electromagnetic radiation in the visible range, between 400-700 nm, is absorbed by crop leaves and wavelengths between 700-1100 nm, referred to as near infrared, and wavelengths up to 2500 nm, referred to as shortwave infrared, are strongly reflected (Cabrera Bosquet et al. 2012; Furbank and Tester 2011). Spectral and imaging techniques feasible for phenotyping can be grouped into three categories: visible to near infrared spectroradiometry, infrared thermometry and thermal imaging, and conventional digital photography (Araus and Cairns 2013). These techniques allow measurement of physical and chemical characteristics from plots such as canopy architecture, canopy temperature, water status, and nitrogen concentration (Montes, Melchinger, and Reif 2007a). Furthermore, measured spectral information, used to create vegetation indices (VIs) which are ratios or differences between spectral reflectance data of specific wavelengths, are related to different plant characteristics (Cabrera Bosquet et al. 2012).

Two general types of reflectance sensors exist: active and passive. Active sensors are equipped with a radiation source, thereby increasing their consistency and reliability even in dynamic ambient lighting conditions. This makes them suitable for phenotyping in field trials

with multiple locations that may have different lighting and weather conditions. However, active sensors typically measure fewer wavelengths and therefore have a lower spectral resolution, thereby limiting their use in predicting complex traits such as nitrogen-use efficiency (Montes, Melchinger, and Reif 2007a). Passive sensors do not have a built-in source of radiation, so they rely on ambient light conditions, thus limiting their use in dynamic lighting conditions. The advantage of passive sensors is their wide spectral range and high spectral resolution (Montes, Melchinger, and Reif 2007a).

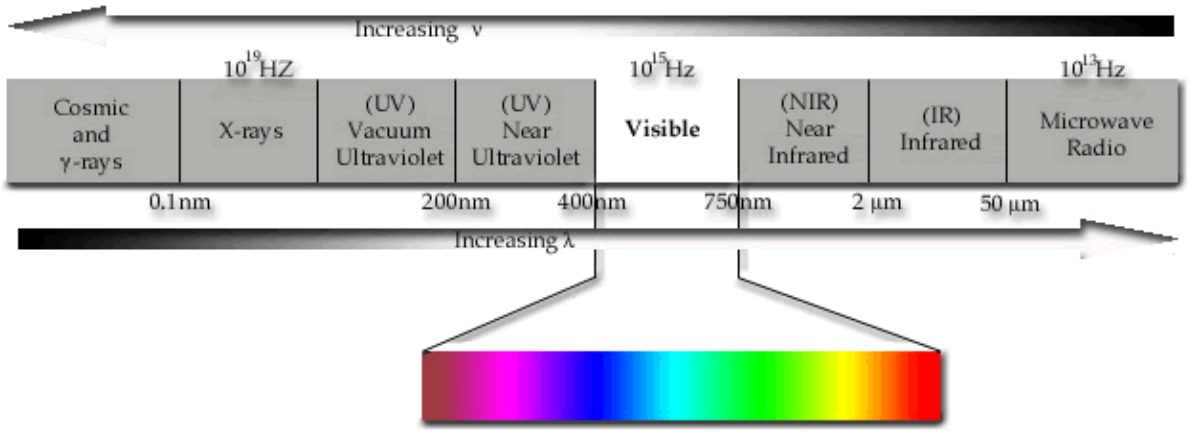

Figure 3.1 Electromagnetic spectrum comprised of all possible wavelengths and frequencies of electromagnetic radiation (http://withfriendship.com).

Visible Light Imaging

Visible light imaging technology captures wavelengths between 400-700 nm (Yang et al. 2013) and typically utilizes conventional digital photography technology, such as DSLRs, for image acquisition. Results are analyzed as two-dimensional (2D) images or as three-dimensional (3D) models from multiple 2D images combined using software. 2D photography has been used to measure plant size and leaf color to analyze growth rates and plant morphology and to quantify greenness parameters (Berger, de Regt, and Tester 2013; Fiorani and Schurr 2013; Yang et al. 2013). Color information values for red, green, and blue can be extracted from digital images which can be analyzed to provide a measure of greenness and estimate chlorophyll and nitrogen content (Berger, de Regt, and Tester 2013). (Kipp et al. 2013) used images from a consumer-grade digital SLR and correlated the proportion of green pixels to early plant vigor in

winter wheat. Results showed that using such a system would be beneficial to plant breeding because of its effectiveness and rapidity in screening large field trials.

Drawbacks of visible light imaging and imaging technologies are due to potentially difficult post-processing requirements, which may include image alignment, geometric and radiometric calibrations, atmospheric corrections, automatic mosaicking, and algorithms for automatic image segmentation (Araus and Cairns 2013). Other problems are due to physical issues with the image subject, such as overlapping leaves and background soil noise (Fiorani and Schurr 2013).

Near Infrared

Emitted infrared radiation is due to molecular movements in all objects. Near infrared radiation (NIR) refers to electromagnetic radiation with wavelengths ranging from 700-1000 nm up to 5000 nm, measured by devices known as near infrared spectrometers (NIRS). Spectrometers are instruments that measure properties of light, such as intensity and wavelength, over a certain range of the electromagnetic spectrum.

NIRS has been used in laboratory settings to analyze grains and fodder quality and to assess plant traits such as nitrogen content, moisture, fiber, carbohydrates, amino acids, minerals, and other compounds. Field use of NIRS has potential for rapid, non-destructive evaluation of biomass accumulation, radiation use efficiency, drought and nutrient use efficiency screening, and assessment of yield performance under stress conditions (Araus and Cairns 2013; Cabrera Bosquet et al. 2012). The advantages of NIRS include low cost, rapidity, high precision, and repeatability (Cabrera Bosquet et al. 2012).

NIR emitted by healthy green plants typically ranges between 800 to 1400 nm, while soil emits relatively small amounts of NIR. Soil and unhealthy plants reflect more red light compared to healthy plants (Yang et al. 2013). The difference in reflected red light is the basis for the Normalized Difference Vegetation Index (NDVI) (Equation 3.1). Thus, the more red light that is reflected, the smaller the NDVI value, indicating the presence of bare soil and unhealthy plants, prompting use of NDVI as a metric for crop health (Lan 2009). NDVI is also used to detect tiller density and soil coverage or leaf area index (Kipp et al. 2013). Examples of commercially available devices (Figure 4.2) that measure NIR include the GreenSeeker (Trimble, Westminster, Col., USA) and the Crop Circle ACS-470 (Holland Scientific, Inc., Lincoln, Neb., USA).

Equation 3.1

$$NDVI = \frac{NIR\ Reflected - Red\ Reflected}{NIR\ Reflected + Red\ Reflected}$$

Mid Infrared

Mid infrared or thermal infrared devices, typically used for non-contact temperature measurement, measure electromagnetic radiation from 7500 to 13,500 nm (Yang et al. 2013). These devices focus electromagnetic radiation emitted from an object through a lens onto a detector that outputs an electrical signal proportional to the radiation. This signal is processed and transformed to obtain a temperature measurement (MICRO-EPSILON 2013). Advantages of non-contact, non-destructive temperature measurement include the ability to measure temperatures of moving objects, overheated objects, or objects in environments hazardous to humans; fast response and exposure time; no physical interaction or influence on measured objects; and no mechanical wear on the measurement device (MICRO-EPSILON 2013). Infrared detectors can be classified into two categories: thermal detectors and quantum detectors. Examples of thermal detectors include thermopiles, pyro-electric detectors, and bolometers. In thermal detectors, absorbed electromagnetic radiation changes the temperature of the sensing element. The change in temperature modifies properties of the sensing element. The properties can then be electrically analyzed. Similar to thermocouples used for temperature measurement through contact, thermopile detectors are composed of two different metallic materials, such as bismuth and antimony. The two materials are arranged around a radiation-absorbing element, the temperature of which changes based on absorbed radiation. Temperature change is proportional to a change in voltage across the detector. In pyro-electrical detectors, absorbed radiation on the sensitive element changes the surface temperature, which consequently changes the surface loading due to the pyro-electric effect. Bolometers, used in focal plane arrays of infrared imagers, utilize the change in resistance of a heat sensitive element to measure temperatue. Heat-sensitive element temperature change due to absorbed radiation changes the resistance of the heat sensitive element. The change in resistance can be detected as a change in voltage. The change in voltage can then be related to temperature. Another category of infrared detectors, quantum detectors, is not based on change in temperature of a sensing element. In quantum detectors, infrared radiation raises the electrons of the semiconductor-sensing element to a higher

energy state. When the electrons return to a lower energy state, an electrical signal is generated which can determine the temperature of the object emitting infrared radiation. Quantum detectors react much faster to changes in temperature, compared to thermal detectors. Time constants of quantum detectors are in the order of nanoseconds and microseconds, while thermal detectors have time constants in the order of milliseconds (MICRO-EPSILON 2013).

Many applications are available for thermal imaging devices and infrared thermometers (IRTs). For phenotyping, these devices have been deployed as handheld infrared guns, thermopile detectors mounted on tractor booms, and micro-bolometer-based thermal imaging sensors mounted on mobile platforms above the crop, on model aircraft, and on manned aircraft (Furbank and Tester 2011). Thermal imaging and IRTs are commonly used to measure leaf and canopy temperature to evaluate leaf water status, quantify salinity and drought stress response in cereal crops, infer photosynthesis rates, and predict yield (Fiorani and Schurr 2013; Furbank and Tester 2011; Ingvarsson and Street 2011). Canopy temperature depression, the temperature difference between the canopy and surrounding air, is used as a selection trait for drought resistance in cereals, thus contributing to yield gains (Fiorani and Schurr 2013).

There are issues with field use of IRTs and thermal imaging, such as the need for soil background corrections and the impact of wind and transient cloudiness (Fiorani and Schurr 2013). Interpretation of crop performance and selection of breeding lines based on canopy temperature must consider plant height, soil covering, emerged spikes, leaf angle and size, and other factors related to canopy architecture. Dynamic environmental variables such as light intensity, temperature, relative humidity, wind speed, and time of measurement also affect the accuracy of thermal measurements and must be considered before using IRTs (Araus et al. 2008).

Hyperspectral Imaging

Hyperspectral images produced by imaging spectrometers are composed of data from hundreds of narrow, adjacent spectral bands. These bands have wavelengths ranging from 400 to 900 nm or 1000 to 2500 nm (Araus and Cairns 2013), depending on the configuration. They can be as narrow as 0.01 micrometers, but are typically between 0.4 to 2.4 micrometers (Randall Smith, MicroImages, Inc. 2001). Imaging spectrometers simultaneously capture these bands by passing reflected light through an optical dispersing element, such as a prism, which splits the

light into narrow, adjacent wavelengths (Figure 3.2). Separate detectors then measure the energy in each band; the larger the number of detectors utilized, the higher the spectral resolution (Randall Smith, MicroImages, Inc. 2001).

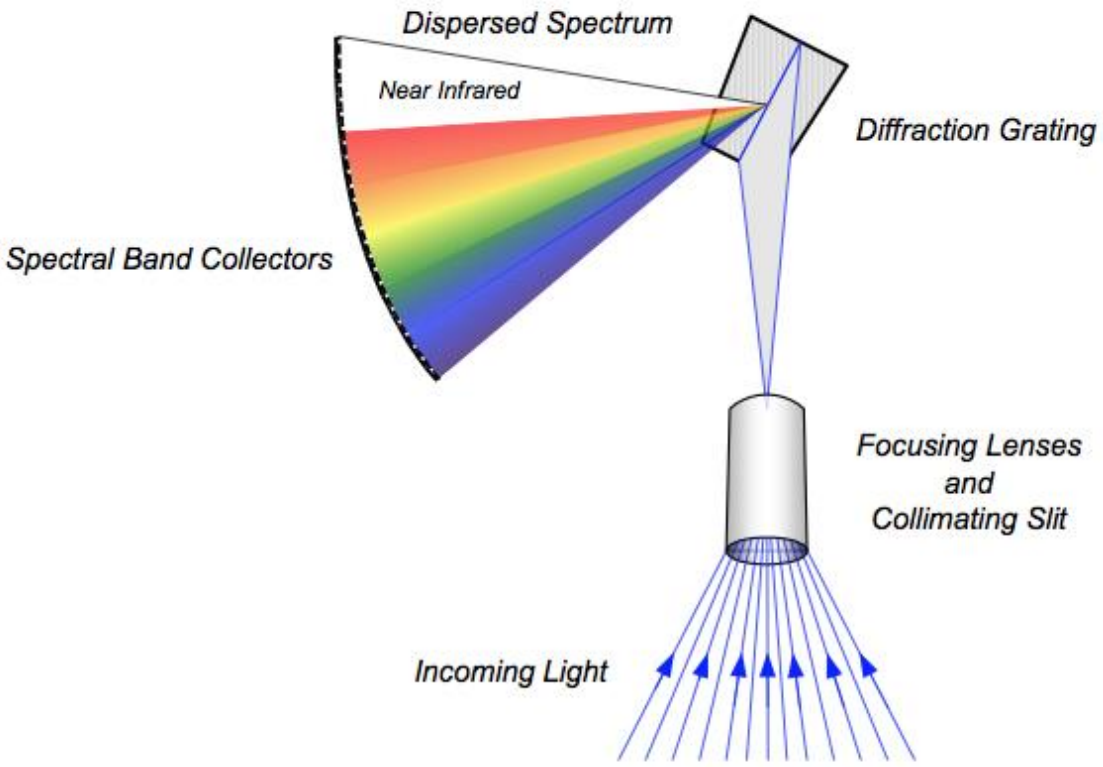

Figure 3.2 Simplified diagram of a hyperspectral sensor (https://www.spacecomputer.com/hsi_technology.html)

Hyperspectral images contain a larger portion of the electromagnetic spectrum compared to RGB imaging and infrared imaging, thereby containing more information. Hyperspectral images of crops may contain information pertaining to plant architecture, composition, and health conditions, which can be used to evaluate plant growth and development (Fiorani and Schurr 2013; Yang et al. 2013) and assess complex traits such canopy photosynthesis and fluorescence (Araus and Cairns 2013).

Disadvantages of hyperspectral imaging for field-based phenotyping include the high cost of imaging spectrometers, large data sizes, complex data analysis, and the passive characteristic of the sensors (Fiorani and Schurr 2013). However, solutions to these problems are under

investigation and field-based mobile phenotyping platforms with imaging spectrometers have been developed (Busemeyer et al. 2013).

Distance Sensors

Two common types of non-contact sensors for distance measurement are ultrasonic and laser sensors. The two sensors have a similar operating principle based on the time interval between transmission of a signal and receiving the echo or reflection of the signal from a target surface (Banner Engineering 2005).

Ultrasonic sensors use a ceramic transducer which vibrates when electrical energy is applied. The vibration produces ultrasonic sound waves greater than 20 KHz, above the human range of hearing. These waves move through the air from the transducer to the target object. To measure the distance between the sensor and target object, a short burst of ultrasonic waves is generated and then the transducer "listens" for a set period of time for the echo of transmitted waves. The speed of sound and time interval between transmitting the waves and receiving the echo are used by a micro-controller in the ultrasonic sensor to calculate distance from the sensor to the target object. The speed of sound in air at standard atmospheric conditions is approximately 769 mph (Benson 2014).

Laser distance sensors pass light emitted from a semiconductor laser diode through a lens to produce a narrow laser beam. The light is pulse modulated by an oscillator at a specific frequency. Modulation is the turning on and off of the light source, allowing the receiver to differentiate between emitted light and ambient light. The receiver consists of a photodiode or a phototransistor that detects emitted light and converts it into a voltage signal. This signal is then amplified and demodulated. Distance to an object is measured by sending a short pulse of light to the object. Some of this light is reflected back and detected by the receiver. The time required for the pulsed light to be emitted and then detected by the receiver is multiplied by the speed of light to calculate the distance from the sensor to the target object (Banner Engineering 2005). The speed of ligh in air is approximately 299,792, 458 m/s (Mendelson 2006).

Although ultrasonic and laser sensors both measure distance, they each have distinct advantages, disadvantages and suitable application areas. Unlike laser sensors, ultrasonic sensors are not affected by ambient light and light reflectance characteristics of target objects, making them more suitable for clear or translucent objects, such as clear packaging. Ultrasonic sensors

are also more suitable for wet environments since liquids can refract light. In addition, wind and air temperature changes can affect ultrasonic sensor measurements. Wind may deflect or disrupt the path of ultrasonic waves (Daigle, Piercy, and Embleton 1978); therefore, windy field conditions may affect measurement accuracy. The speed of sound through a medium, such as air, is dependent on the specific heat and temperature of the medium (Benson 2014). The speed of sound increases with temperature (Wong 1986). Expensive commercial ultrasonic sensors typically feature temperature compensation to account for temperature fluctuations.

Conversely, laser sensors are not affected by wind and temperature changes and, since the speed of light is faster than the speed of sound, laser sensors can measure distance more quickly than ultrasonic sensors. Laser sensors also have a much narrower beam compared to ultrasonic sensors for the same distance, making them useful for measuring smaller targets, such as distance-to-ground measurements along the wheel track. However, the wider target area of ultrasonic sensors is useful for measuring distance-to-plant canopy, especially for sparse plant plots through which the narrow laser beam can easily pass.

Satellite-Based Positioning Systems

Initially developed for military use, satellite-based navigation systems have found widespread use in civilian areas of application such as surveying and mapping, agriculture, transportation, machine control, and marine navigation (NovAtel 2014). Currently, the Global Navigation Satellite System (GNSS) is comprised of three independent systems: the Global Positioning System (GPS), developed and operated by the United States; the Russian-based Global Navigation Satellite System (GLONASS); and the Galileo system, developed by the European Union. GPS and GLONASS are fully operational and transmit signals for civilian use.

Location is determined by GNSS through triangulation calculations based on signals received by a GNSS receiver from GNSS satellites. Satellites send radio signals at specific times and wavelengths containing information such as date and time, satellite location and status, and information on other satellites (NovAtel 2014). Receivers must track at least four satellites in order to calculate the receivers location. Location accuracy is affected by factors such as the number of satellites in view, satellite geometry or dilution of precision (DOP), ionospheric conditions, and receiver quality (Kaplan and Hegarty 2005). Without corrections, stand-alone or

autonomous receivers can expect accuracies of approximately 13 m or smaller in the horizontal plane and approximately 22 m or smaller in the vertical plane. For precision agriculture and many other applications, more accurate position information, than what stand-alone or autonomous recievers provide, is required. Differential GNSS, developed to increase position accuracies obtained from GNSS satellites, uses a fixed base station with known and accurate position in order to calculate errors associated with satellites tracked by the base station and mobile receivers. The base station sends out a reference signal to mobile receivers; that signal is used by the receivers to correct the signal from the satellites.

Several differential-based correction systems with varying costs and accuracy are currently available. The Wide Area Augmentation System (WAAS) operated by the US Federal Aviation Administration provides 1 to 3 meter accuracy, with a total system cost ranging from $100-$500. Wide area coverage is achieved using geosynchronous satellites which transmit GNSS corrections to mobile receivers. WAAS enabled systems are useful for mapping and yield monitoring. Sub-meter accuracy can be attained with paid, subscription-based differential services such as OmniStar VBS and StarFire 1, with system costs ranging from $500-$2,500 and subscription periods ranging from several months to a year (Stephens and Rasmussen). These systems can be used for mapping, yield monitoring, applications using variable rate technology (VRT), and limited guidance. Sub-decimeter accuracy can be obtained with services such as OmniStart HP and StarFire II. These systems require additional equipment and cost, approximately $2,500-$7,500, and can be used for guidance and VRT (Stephens and Rasmussen). For accuracies of 2 to 3 centimeters, real-time kinematic (RTK) systems have been developed. RTK systems utilize a fixed, ground base station that must be within 6 to 10 miles of the receiver. These systems use complex methods to determine and calculate position errors used for calculating position corrections; corrections are sent over a radio link in real-time to a mobile receiver (“AgGPS® 432/442 GPS Receivers” 2008). RTK enabled systems, used for precision guidance, elevation mapping, and survey-grade mapping, provide the highest accuracy and repeatability of any publicly available positioning system, but at a high cost of $15,000-$50,000.

Mobile Field-Based Phenotyping Platforms

Several mobile, field-based, experimental phenotyping platforms have been developed since 2009. These platforms feature various types of sensors mounted on a mobile frame to simultaneously measure multiple types of data in a single pass. (White and Conley 2013) developed a proximal sensing pushcart consisting of a rectangular frame mounted on two bicycle frames. The cart was equipped with monochrome cameras, ultrasonic proximity sensors, infrared thermometers, radiometers, and a global positioning receiver connected to CR1000 and CR3000 data loggers (Campbell Scientific, Inc., Logan, Utah, USA) for collecting data at 1 Hz and 5 Hz, respectively (White and Conley 2013). Another platform for phenotyping small grain cereals up to a plant height of 1.6 m was developed by Busemeyer et. Al (2013) (Busemeyer et al. 2013). The platforms consisted of a variety of sensing technologies, such as 3D Time-of-Flight cameras, color cameras, laser distance sensors, hyperspectral imaging systems, light curtain imaging systems, and a GPS receiver, all mounted on a tractor-pulled trailer with a track width of 1.25 m. The system used an industrial PC with a MySQL database server for data storage (Busemeyer et al. 2013). Some platforms were equipped with more than one set of sensors for simultaneously measurement of multiple rows of plants or plots, thereby significantly increasing throughput. The system developed by Andrade-Sanchez et. al. (2013) contained four sets of sensors mounted on an open rider sprayer, capable of measuring four rows of plants simultaneously. Each sensor set included an ultrasonic sensor, two infrared thermometers (one pointing at nadir, one pointing 30° from the vertical axis), and a Crop Circle ACS-470 multi-spectral crop canopy sensor. Three data loggers were used for data collection and storage: a Holland Scientific GLS-420 for the Crop Circle ACS-470, a Campbell Scientific CR3000 for the infrared thermometers, and a Campbell Scientific CR1000 for the ultrasonic sensors. The system collected data at 1Hz (Andrade-Sanchez et al. 2013).

Chapter 4 - Materials and Methods

System Design

Basic requirements for the phenotyper included assembly of a mobile sensor platform of three sets of sensors and two GPS units, collection of data from the sensors, data georeferencing, and saving as a text file. The system must be able to measure plant canopy temperature, plant height, and canopy reflections at various spectral wavelengths at a rate of 10 samples per second.

The system consists of sensors, GPS units, and data acquisition hardware mounted on a Bowman Mudmaster sprayer (Bowman Manufacturing Co., Inc., Newport, Ariz., USA) (Figure 4.1). The spraying boom was replaced with a sensor bar. The phenotyper had a clearance of 1.6 meters, allowing it to pass over a variety of field crops, such as wheat and soybean. The sensor bar could be raised up and down while maintaining level status, thus maintaining sensor orientation relative to the ground.

Figure 4.1 Phenotyping platform consisting of sensors, GNSS units, and data acquistion hardware mounted on a high clearance vehicle

Three sets of sensors were mounted on the sensor bar for simultaneous data collection from up to three rows of plants or plots. The spacing between each sensor set was adjustable.

Each sensor set (Figure 4.2) contained one GreenSeeker (Trimble, Westminster, Colo., USA), one Crop Circle ACS-470 (Holland Scientific, Inc., Lincoln, Neb., USA), one Banner U-GAGE Q45U Ultrasonic Sensor (Banner Engineering Corp., Minneapolis, Minn,, USA), one Micro Epsilon 15:1 CX-SF15-C8 IRT (MICRO-EPSILON, Raleigh, N.C., USA), and one Micro Epsilon 2:1 CTH-SF02 IRT (MICRO-EPSILON, Raleigh, N.C., USA). Differences in the two IRTs' differed in field-of-view related to ratios in their model names (Figure 4.3B). The first number in the ratio is the distance of the measured object from the sensor, and the second number is the diameter of the measured circular spot. Thus, for the 15:1 IRT, if the object is 15 centimeters away from the sensor, the diameter of the circular spot is 1 cm. Increasing the distance increases the area of the spot. Within each set, the sensors were spaced to minimize or prevent interference with each other. The two IRT sensors were placed side-by-side so that the area measured by each sensor would overlap (Figure 4.3B).

Figure 4.2 One sensor set composed of (left-to-right) two IRTs, one ultrasonic sensor, one Crop Circle, and one GreenSeeker

In addition to the sensors, a Trimble AG 25 GNSS antenna was mounted on each end of the sensor bar (Figure 4.3C) and a Banner LT3NUQ laser distance sensor (Banner Engineering Corp., Minneapolis, Minn., USA)was mounted on the sensor bar in front of the left wheel track (Figure 4.3A). GNSS antennas were connected to two Trimble FmX integrated displays which process GNSS signals and receive RTK corrections from the Trimble AgGPS 542 RTK base station (Trimble, Westminster, Colo., USA).

Several types of hubs were mounted inside a protective box. The hubs extended the number of ports of the laptop computer that runs software to collect sensor and GPS data. Figure 4.4 illustrates the connections between the sensors, hubs, and laptop. A summary of outputs from all sensors is listed in Table 4.1. One hub was a B&B Electronics UHR307 7-port USB 2.0 hub (B&B Electronics, Ottawa, Ill., USA) which extended one USB port on the laptop. All six IRTs, two in each sensor set, were connected to this hub. The other two hubs were B&B Electronics USR604 USB to serial converters with four RS-232 ports. Three GreenSeekers were connected to the one of the serial converters and the Crop Circles were connected to the other. The remaining port on each hub was connected to a Trimble FmX unit which sends GPS data to the laptop. All B&B Electronics hubs were industrial grade with optical isolation on each port which protects the laptop, hubs, and sensors from electrical damage in case of hub or sensor malfunction.

A National Instruments 9207 analog input module (National Instruments, Austin, Texas, USA) was used to interface analog voltage signals of the three ultrasonic sensors with the laptop through a USB port. The NI 9207 module had 16 channels, 8 of which were used to measure signals between ±10 V, and the remaining channels measured current signals between ±20 mA (“NI-Datasheet-Ds-187” 2013). This added flexibility to the phenotyper by allowing it to handle analog voltage and analog current signals.

The entire system was powered by three 12 V batteries connected in parallel. A 12 V to 110 V inverter provided power to the laptop. The laptop was placed on a platform mounted to the right side of the driver’s seat. The two Trimble FMX units were located next to the laptop, each connected to one GPS antenna. A Planar PCT2265 22" Touch Screen Monitor (Planar Systems, Inc., Beaverton, Ore., USA) was mounted on a swivel arm. The touch screen monitor equipped the system with a large display for system control and monitoring, allowing the user to quickly and easily navigate the phenotyping software.

Figure 4.3 A) Laser sensor mounted inline with the left wheel-track to sense boom height relative to the ground. B) IRTs mounted side-by-side so measurement areas would overlap. C) GNSS receivers mounted on each end of the sensor boom.

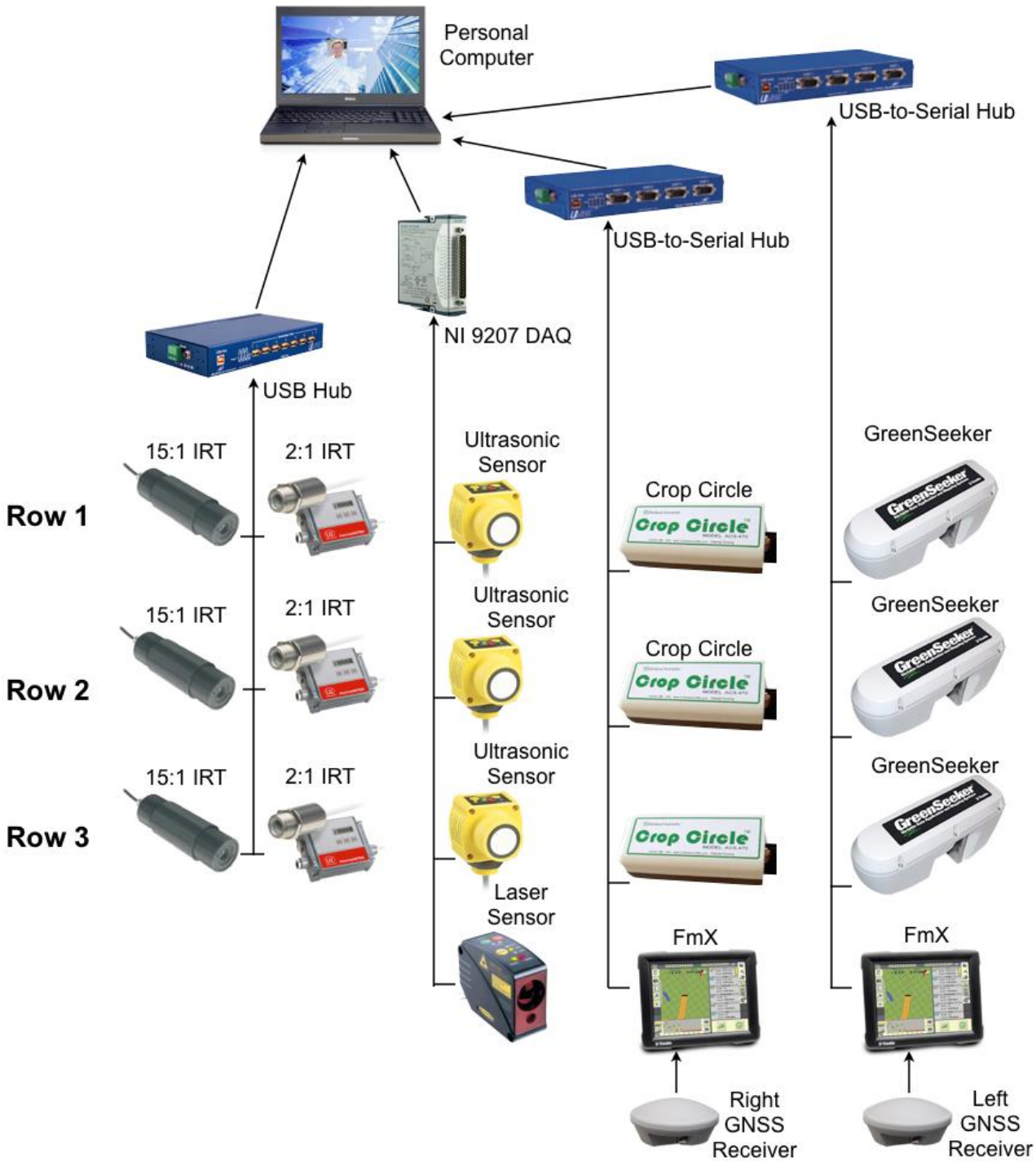

Figure 4.4 Three communication hubs and one analog data acquisition module routing signals from sixteen sensors and two GNSS receivers to a personal computer.

Table 4.1 Summary of sensor power requirements and output signals.

Sensor	Measurement	Power Input	Output Signal
Banner U-Gage Q45U Ultrasonic Sensor	0.25–3 m	15-24 V DC	0-10 V DC
Micro Epsilon CX-SF15-C8 15:1 IRT	-30–150 °C	5-28 V DC	With USB Kit – RS232, 9600 baud, 8-n-1
Micro Epsilon 2:1 CTH-SF02 IRT	-40–975°C	8-36 V DC	With USB Kit – RS232, 9600 baud, 8-n-1
Holland Scientific ACS 470 Crop Circle	Multiple Spectral Wavelengths	11-17 V DC, 350 mA	RS-232, Auto Send Mode, 76800 baud, 8-n-1
Trimble GreenSeeker 500 with RT200	Spectral Wavelengths: 770nm, 656 nm	12 V DC, 300 mA	RS-232, 38400 baud, 8-n-1
Banner LT3NUQ Laser Distance Sensor	0.3-5 m	12-24 V DC, 108 mA	0-10 V DC

Software Development

Prior to the development of software for collecting, georeferencing, and saving sensor data, consideration of the software deployment platform was essential. Two options were considered: a Campbell Scientific C3000 data loggers and a laptop computer. Due to the number of serial/communication ports required to interface with sensors, the laptop computer with extensible communication ports was preferred over the data loggers, which contain a large number of analog input ports but only four RS-232 ports. The laptop does not have any analog input ports required to collect data from ultrasonic sensors and future analog sensors, but this deficiency was solved using the NI 9207 analog input module which connects to the laptop through USB. In addition to extensibility, the laptop also allows a user interface for system control and display of real-time sensor readings using custom gauges and indicators.

The phenotyper software, developed using National Instrument's 2012 LabVIEW Professional software package (National Instruments, Austin, Texas, USA), consists of separate modules or sub-programs, referred to as virtual instruments, which are combined into the main program. Each type of module is responsible for interfacing with and collecting data from a specific type of sensor. A separate module exists for the GreenSeekers, Crop Circles, 15:1 IRTs, 2:1 IRTs, distance sensors, and GPS units. With the exception of the distance sensor module, each module communicates with only one sensor; therefore, within the main program, three instances of the GreenSeeker module occur since there are three GreenSeeker sensors, and three instances of the 15:1 IRT module occur since there are three 15:1 IRTs, etc. For each instance of a module, a unique communication port must be specified.

The main program is a collection of continuous loops, each containing one instance of a sensor module (Figure 4.5) which receives data from a specific sensor, displays the data in custom gauges and indicators (Figure 4.6), and saves the data into a formatted text file. The program saves a text file for each type of sensor.

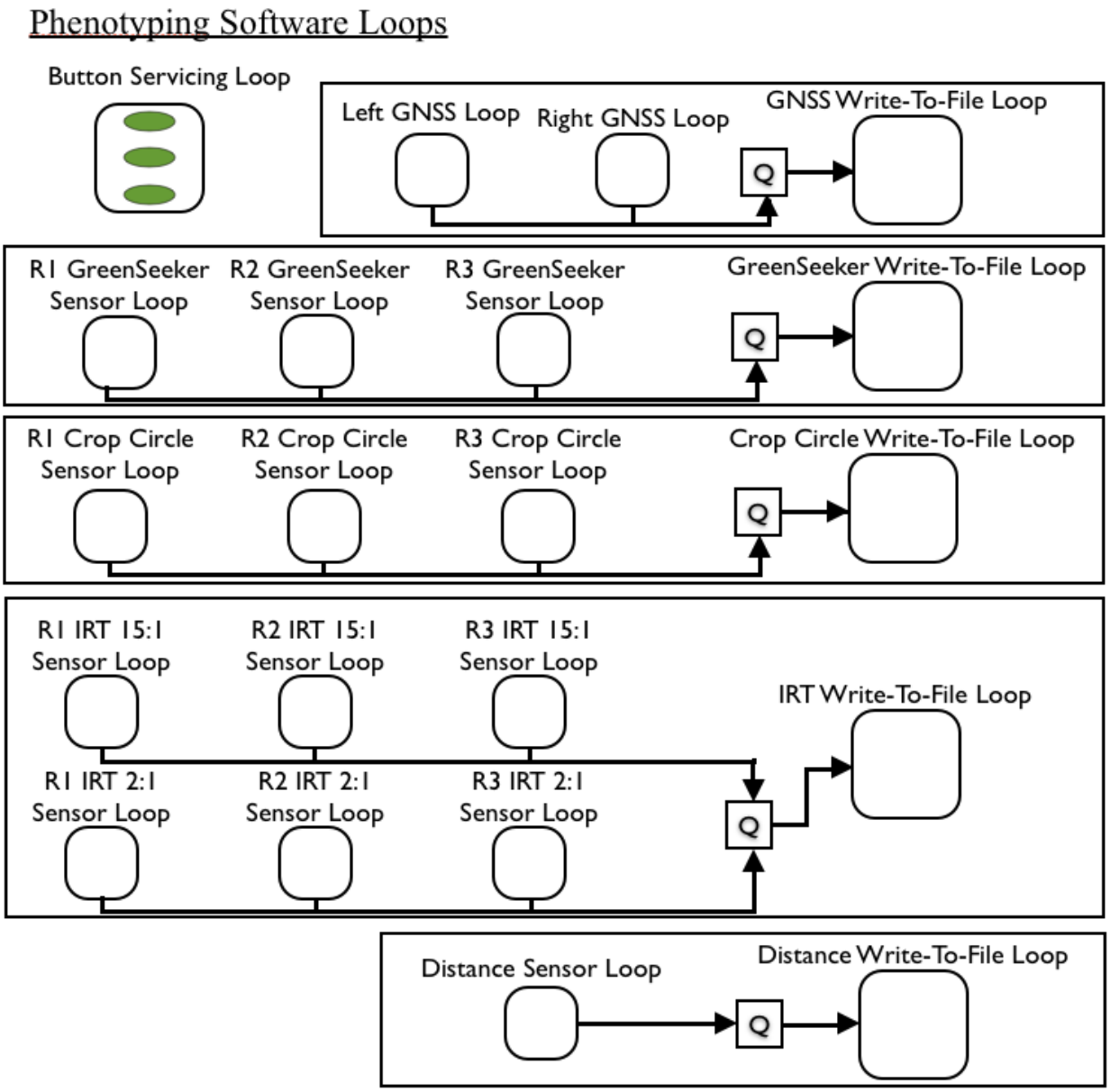

Figure 4.5 The phenotyping sofware comprised of a total of 20 sensors and write-to-file loops; sensor loops place sensor data in queues write-to-file loops read data from the queues and write them to a file.

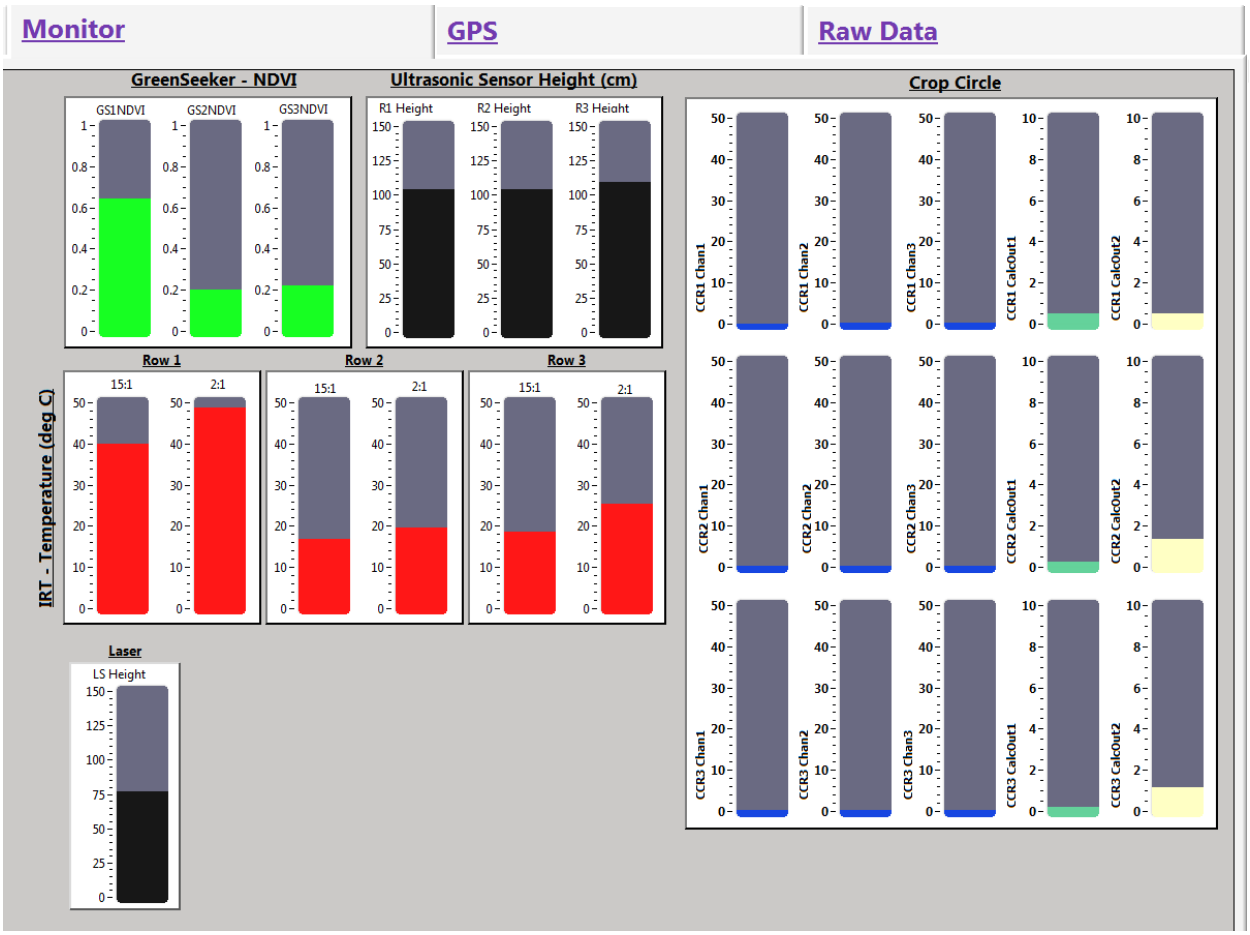

Figure 4.6 Real-time sensor readings displayed using custom gauges useful for monitoring purposes during phenotyping.

Sensor Module Software Design

The different sensor modules perform the same tasks of interfacing and receiving data from a specific sensor, but they differ in the specifics of communication with their respective sensors due to differences in baud rate and data strings. This section discusses how each module type was developed.

GreenSeeker

Each GreenSeeker sensor package consists of four pieces of hardware: the GreenSeeker sensor head, RT200 control box, a proprietary cable for connecting the sensor head to the RT200, and a propriety cable for connecting the RT200 to the laptop. The sensor head contains hardware to collect raw NDVI data. The RT200 provides power to the sensor head and processes

data which is sent to the computer through an RS-232 serial connection running at 38400 baud. The RT200 (Figure 4.7) can collect and process data from multiple GreenSeeker sensors; however, only the average of NDVI values from all sensors is sent to the computer. In this study, NDVI values of each GreenSeeker sensor were required; therefore, three RT200s were used, one for each GreenSeeker sensor. The sampling rate of the GreenSeeker can be set to several values using RT Commander software (Trimble, Westminster, Colo., USA). Each GreenSeeker sensor was set to continuously sample every 100 ms (10 Hz).

Figure 4.7 Trimble RT200 interface module for the GreenSeeker sensor head

Within LabVIEW, a new virtual instrument was created specifically for the GreenSeeker. LabVIEW's Instrument I/O Assistant (IA) was used to setup the connection with a specific communication port (COM port) to which the GreenSeeker was connected. Prior to using the IA, the COM port had to be configured to the required settings for the GreenSeeker (Table 4.1). Once the connection was established within the IA, the data string received from the GreenSeeker was parsed for NDVI data and saved as a token (Figure 4.10). The IA then automatically generated LabVIEW code (Figure D.6) to open a connection with a GreenSeeker on the specified COM port, parse GreenSeeker data for the NDVI, and save the data into a variable. This virtual instrument (VI) was added into the main program.

Within the main program, the GreenSeeker subVI is placed into a loop (Figure 4.9) which repeats no faster than 20 times a second, or 20Hz. When this loop begins to run, it enters the GreenSeeker subVI and waits for data from the GreenSeeker sensor to arrive at the serial port. It

waits until data arrives or until the timeout value of 3 seconds. Once the data is received, it is parsed, the loop then exits the GreenSeeker subVI, passes the parsed data to the front panel for display, and places it into a queue. A write-to-file loop (discussed later) accesses data in the queue and writes it to a text file. Then the loop is ready to repeat. Maximum loop rate is controlled by the wait function inside the loop which is wired with a value of 50 milliseconds. The wait function does not cause the loop to wait 50 milliseconds, instead it prevents the loop from repeating until the value of the system clock reaches a value that is a multiple of 50 milliseconds. For example, if the loop starts at time 100 ms and finishes at time 120 ms, it cannot start again until time 150 ms. If the loop starts at time 100 and finishes at time 170, it can restart immediately without waiting until time 200.

Consider a second example (Figure 4.8) in which data arrives in intervals of 100 ms and on average it takes 20 ms to parse the data. If the loop starts at 0 ms and the first data point arrives at 0 ms, the loop will finish at 20 ms. It will then wait until the clock is at 50 ms to repeat therefore, the period of the first loop is 50 ms. At 50 ms, it will run again; however, since there is no data until time 100 ms, the loop must wait inside the GreenSeeker subVI. At time 100 ms, the data will arrive and the loop will finish at time 120 ms; therefore, the period of the second loop is 70 ms. As soon as the loop finishes, it will repeat and wait for data to arrive. The loop does not have to wait until time 150 ms. The third data point will arrive at 200 ms and the third loop will finish at time 220 ms. The period of the third loop is 100 ms, equal to the period of the data; therefore, after the second loop, if the data arrives slower than 50 ms or slower than the value wired to the wait function, the loop rate is controlled by the data rate. The only time constraints imposed by the wait function come in to play is when multiple data points are buffered at the serial port and the sensor subVI does not have to wait for data to arrive.

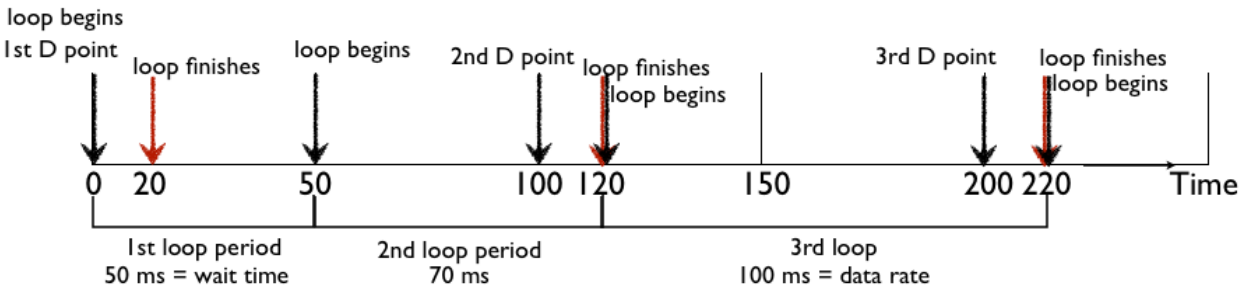

Figure 4.8 If data arrives slower than the value wired to the wait function, the data rate controls the loop rate.

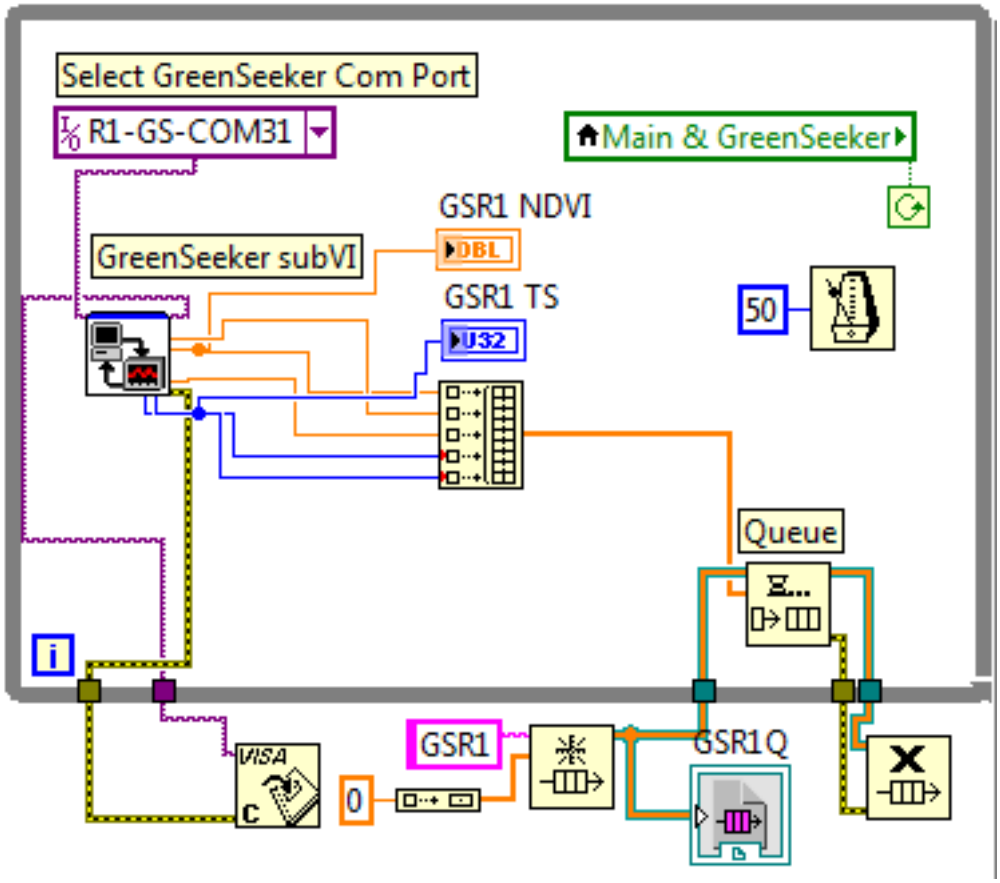

Figure 4.9 The GreenSeeker subVI receives NDVI data sent by individual GreenSeeker sensors.

Three instances of the GreenSeeker loop (Figure 4.9) were created, one for each GreenSeeker sensor. This loop rate is important and must be faster than the sampling rate of the

GreenSeeker, which was set to 10Hz. If it is slower, incoming data will accumulate in the buffer, will not be read in real-time, resulting in incorrect georeferencing. The buffer will continue to fill, eventually resulting in data loss. This also applies for the Crop Circle ACS-470 and the distance sensors.

Crop Circle

The Holland Scientific Crop Circle ACS-470 (Holland Scientific, Inc., Lincoln, Neb., USA) is equipped with RS-232 serial communication when configured in auto-send mode, but is only for sensor output (ACS-470 Spec Sheet) and not for sending commands to the sensor. It operates using a non-standard baud rate of 76800, which may not work with certain serial communication hardware on desktop computers. A proprietary cable, which can be ordered from Holland Scientific, is required. This cable has a Eurofast type, O-ring sealed connector on one end, which connects to the ACS-470, and a DB-9 connector and two power terminals on the other end. A similar, user-made cable can be assembled, but doing so may void the ACS-470 warranty. The ACS-470 was set to auto-send mode using the FieldCAL SC-1 calibrator (Holland Scientific, Inc., Lincoln, Neb., USA) and ACS-470 Configuration Software. The FieldCAL SC-1 was also used to calibrate and set the sampling rate of the ACS-470 to 10 samples per second or 10Hz.

Similar to the GreenSeeker module, the LabVIEW Instrument I/O Assistant (IA) was used to connect to the ACS-470 and parse the data string, containing five pieces of data (Figure 4.10). The first two pieces of data, calculated indices set using the ACS-470 Configuration Software, are automatically calculated by the ACS-470 from the last three pieces of data which were the reflectance readings from the three measurement channels. Resulting LabVIEW code from the IA was then saved as a VI and placed in a loop in the main program (Figure 4.11). Three instances of the VI were added to the main program, one for each ACS-470 sensor. Similar to the GreenSeeker module, the Crop Cirlce ACS-470 module loops no faster than 20 Hz but faster than the 10 Hz sampling rate.

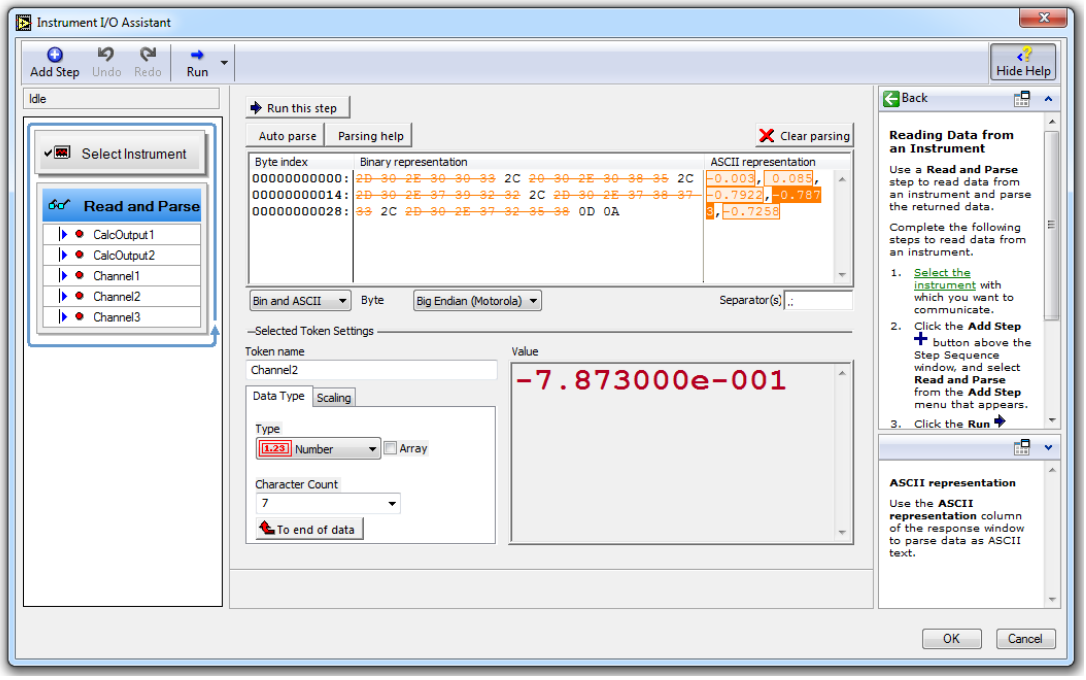

Figure 4.10 Five pieces of data, delimited by commas can be parsed from the Crop Circle data string.

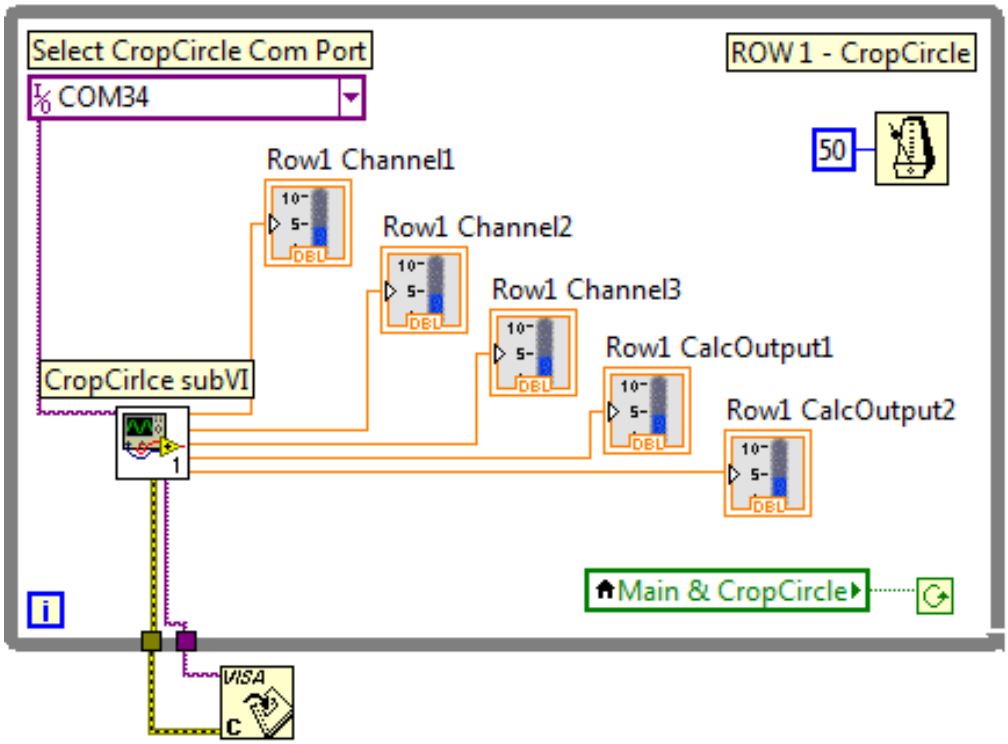

Figure 4.11 The Crop Circle subVI parses five tokens from the data string sent by the Crop Circle sensor.

2:1 and 15:1 IRT

The CX-SF15-C8 and CTH-SF02 (Figure 4.3B) IRTs (MICRO-EPSILON, Raleigh, N.C., USA) are equipped with digital USB communication when configured with their USB kits, the TM-USBK-CX and TM-USBK-CT. Both models were directly connected to the USB ports of the laptop computer through the USB kits. The IRTs can be configured using Micro-Epsilon CompactConnect software. The sensors offer two options for digital communication: bidirectional and unidirectional or burst mode. When set to bidirectional communication, data can be sent to and received from the IRT which is useful for sending specific hex commands listed in IRTs manuals. In burst mode, thermometers constantly send temperature data. Initially, the thermometers were set to burst mode, but parsing the data string in LabVIEW was difficult and unreliable due to non-standard delimiters in the data string and lack of a terminating character. Therefore, the IRTs were set to bidirectional mode. To obtain object temperature, the hex command 0x01 was sent and the IRT replied with a number that could be converted to the object's temperature in degrees Celsius. The conversion formula differed slightly between the two IRT models, so a separate software module was written for each. The LabVIEW Instrument I/O Assistant was used again to generate the code to connect to the IRT, sending the 0x01 command, and reading and parsing temperature data received. A 10 ms time delay between sending the 0x01 command and reading the reply was added to the generated code to give the IRT time to process the command and send the temperature data before the module attempted to parse it. This avoided any timeout errors and ensured that the complete data string from the IRT was received before being parsed. Resulting LabVIEW code from the IA was then saved as a virtual instrument and three instances of each module were placed in the main program (Figure 4.12), one for each model of IRT sensor. IRT modules loop no faster than 10Hz.

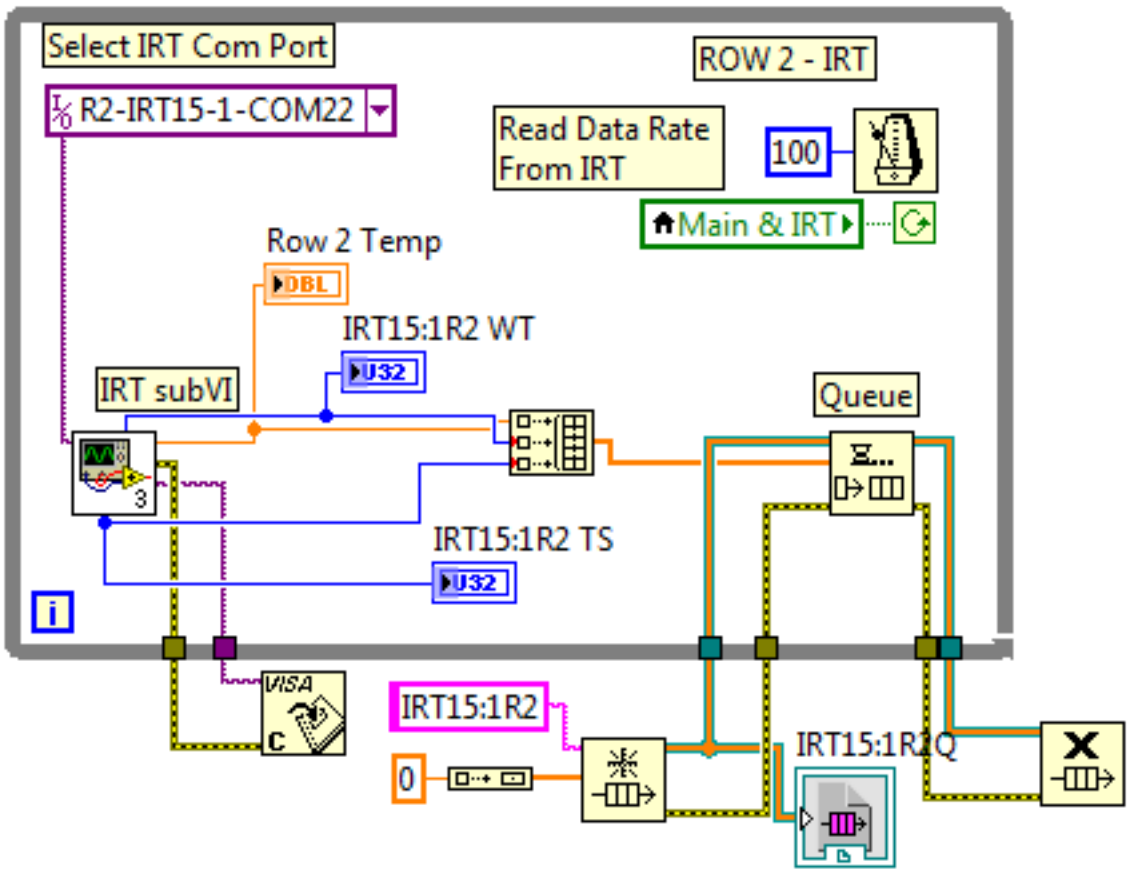

Figure 4.12 The IRT subVI commands the IRT sensor to send temperature data, parses the data string, and outputs the temperature.

Distance Sensors

The Banner U-GAGE Q45U ultrasonic sensor has a selectable analog output, either 0-10V dc or 4-20 mA. The Banner LT3NUQ laser sensor has an output of 0-10V dc. All three ultrasonic sensors were set to output 0-10V. The ultrasonic sensors and laser sensor were connected to the NI 9207 analog input module. Four DIP switches on the ultrasonic sensors set various settings on the sensors, such as output type, output slope, and loss of echo settings. The output slope was set to positive, meaning voltage output increases as distance increases. Loss of echo setting was set to hold the previous reading instead of reverting to a minimum or maximum value. The sensor response speed adjustment was set to 80 milliseconds, which is the fastest. The response speed of the laser sensor was set to MED (analog frequency response (-3dB) of 10 ms). Using the DAQ Assistant in LabVIEW (Figure 4.13), the NI 9207 module was configured to

continuously read one sample, at a rate of 10 Hz, from voltage Channels 0, 1, 2, and 3 with Channels 0, 1, and 2 connected to one ultrasonic sensor and Channel 3 connected to the laser sensor. The resulting code was then saved as a virtual instrument and placed into the main program. Unlike the other sensor modules which have one instance of the module per sensor, there is only one instance of the distance sensor module in the main program. Within the main program, sampled voltage signals from each sensor were converted to distance measurements in centimeters using a simple straight-line calibration curve defined by two calibration points (Figure 4.14). These points were the upper and lower distance limits, set to 150 cm (9.9 V) and 30 cm (0 V), respectively. Upper and lower limits can easily be reset using the teach button on the sensors.

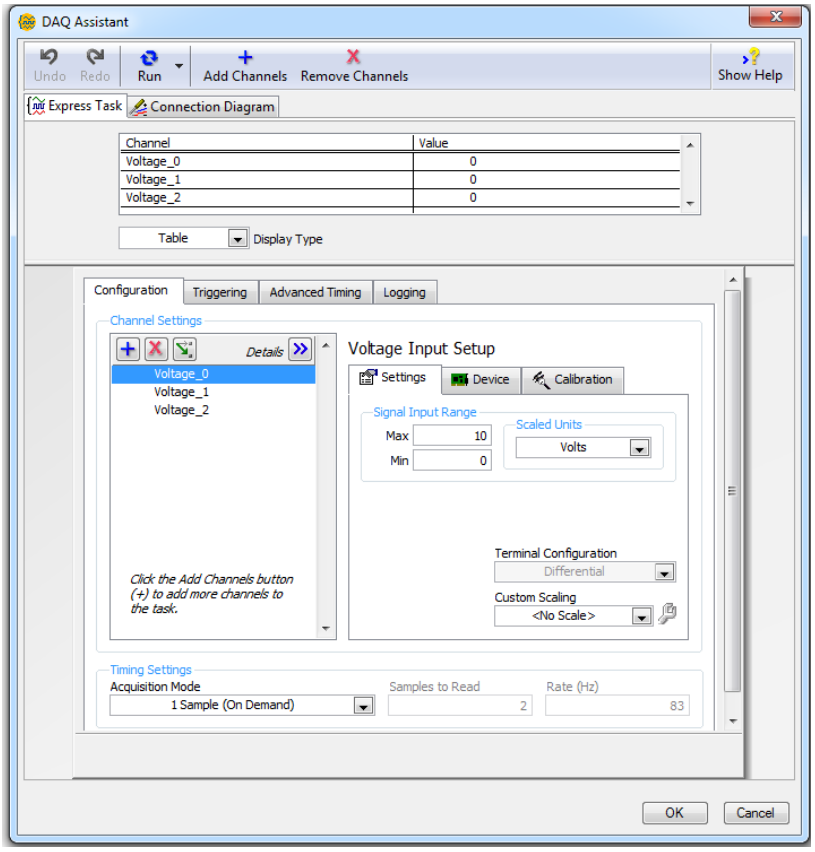

Figure 4.13 The NI DAQ Assistant was used to configure the NI 9207 data acquisition module.

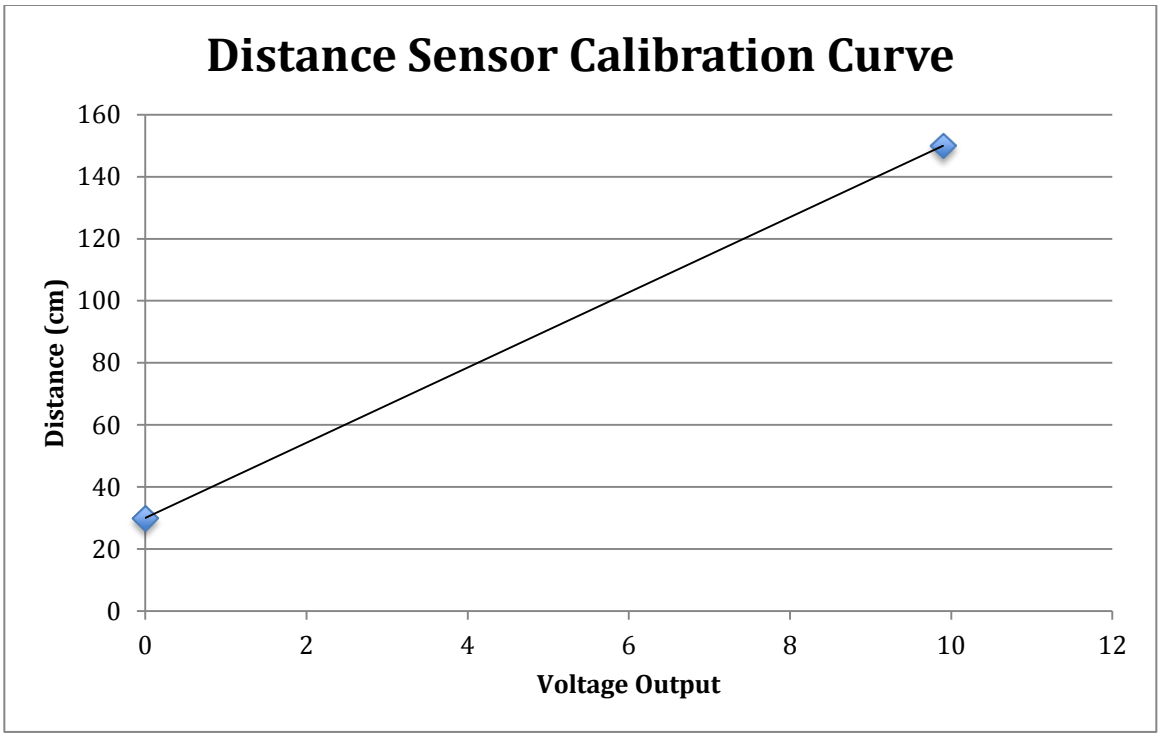

Figure 4.14 The distance sensors are equipped with teach buttons for setting the lower and upper distance limits;. The corresponding sensor voltage outputs are 0 V and 9.9 V for the lower and upper limits, respectively.

Positioning System

The source of GNSS data was the two Trimble FMX units. The FMX units were configured to output RTK corrected global positioning system fix data (GGA) strings at 5 Hz or five samples per second, which was the maximum that it could do. Data from all sensors were collected at a rate of 10 Hz, twice as fast as incoming GPS data. Therefore, more than one sensor data point could be georeferenced to a single position data point.

The FMX units were connected to the RS-232 hubs via DB-9 connectors, and the hubs were connected to the laptop computer via USB. Similar to the other sensors, the LabVIEW Instrument I/O assistant was used to generate the code to connect to the FMX units and parse GGA data strings. Four pieces of data were extracted: UTC, elevation, latitude, and longitude. A virtual instrument was created from the generated code and two instances of it were added to the main program, one for each FMX unit (Figure 4.15).

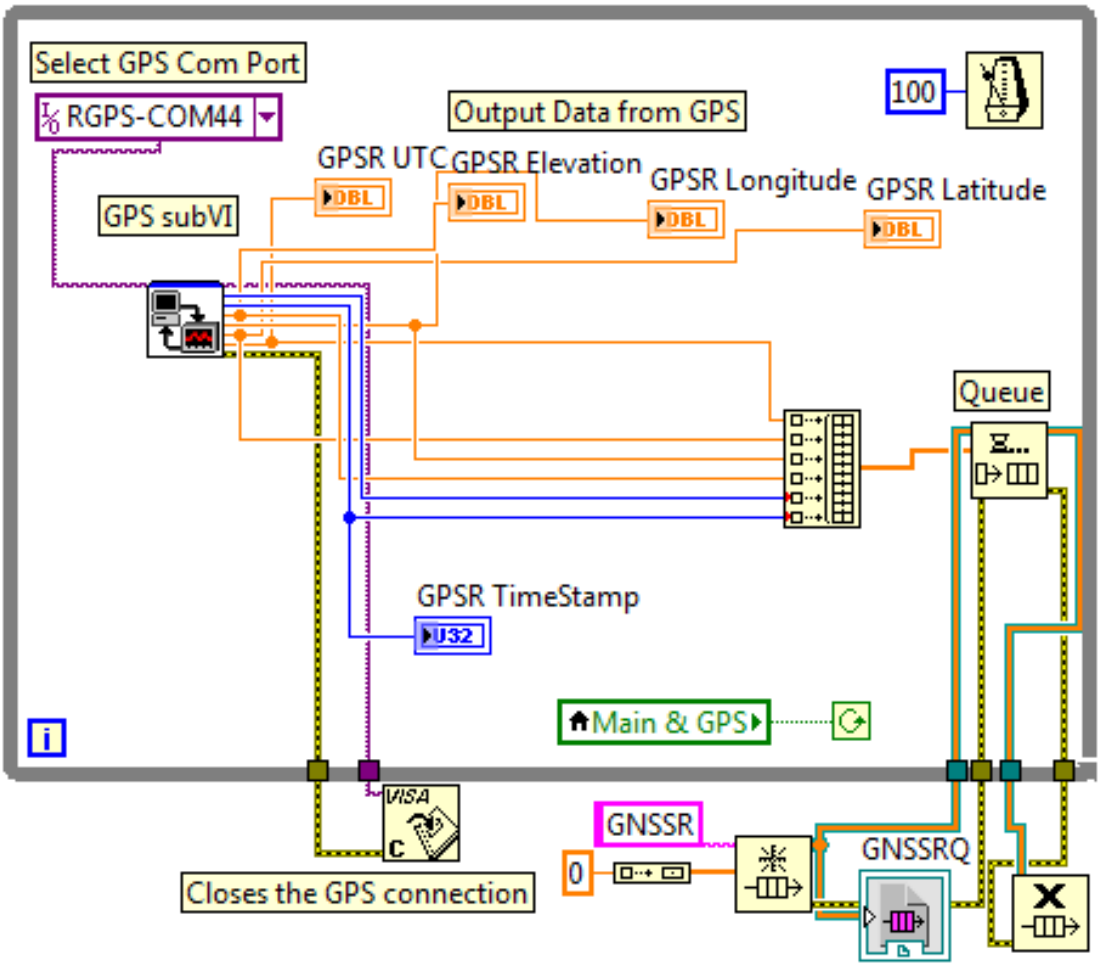

Figure 4.15 The GNSS loop contains the GPS subVI which receives and parses GGA strings from the FmX units; the GNSS loop repeats no faster than 10 Hz.

Main Program

The main program is composed of multiple loops that can be timed and run independently and simultaneously on multi-processor computers (Figure 4.5). The loops can be divided into categories such as button-servicing loops, sensor loops, and write-to-file loops. Only one button-servicing loop is present (Figure 4.16) for the primary purpose of checking the value of user-interface control buttons such as the main On/Off button and buttons for turning on/off individual types of sensors. The timing of this loop is not critical because, in most situations, delays in servicing the buttons will not lead to loss of sensor data. The other two categories, sensor loops and write-to-file loops, are critical; therefore, the loop rate of the button-servicing

loop can be slower than the other loops. A wait function was placed inside the loop (Figure 4.16) to control its rate or frequency. The wait function causes the loop to wait until the next millisecond multiple of the value wired to it, thereby limiting how fast the loop can run. The button-servicing loop rate was set to 1 Hz or a period of 1000 ms.

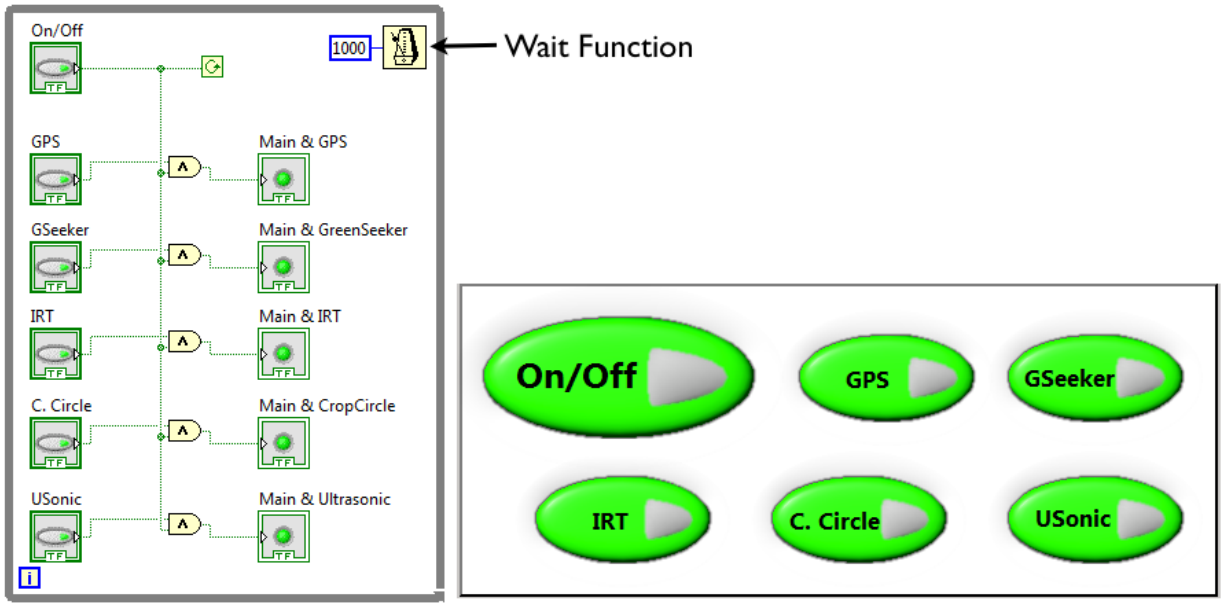

Figure 4.16 The button-servicing loop checks the value of several control buttons on the user-interface. The Wait function within the loop controls the loop rate, which in this example is set to 1 Hz or a period of 1000 ms.

There is one sensor loop for every sensor and GNSS receiver (Figure 4.5), with the exception of distance/height sensors which only have one loop that services the NI 9207 module to which these sensors are connected. Sensor loops contain the modules or sub-programs responsible for obtaining data from the sensors and GNSS receivers. Sensor loop timing is critical because if it is consistently slower than the rate at which data is sent from the sensors, data will be buffered and eventually lost. The term buffered in this text refers to a situation in which more than one data point is stored in a communication port buffer because the port is not serviced frequently enough. Another problem is that buffered data are no longer real-time and will not be correctly georeferenced. However, this does not apply to the IRTs since during each

sampling period the phenotyping software commands these sensors to send data. In this case, the loop rate for IRTs should be no faster than their sampling rate, otherwise the sensors would not be able to keep up with the software's requests. Therefore, sensor loop rates must be faster than the rate at which the sensors send data, except for IRTs. The sensors and GNSS receivers were set to send data at 10 Hz and 5 Hz, respectively. The GreenSeeker loops, Crop Circle loops, and distance sensor loop were set to a maximum of 20 Hz by wiring a value of 50 ms to the wait function. IRT loops were set to poll IRT sensors at 10 Hz.

There are a total of five write-to-file loops (Figure 4.17), one for each type of sensor and one for GNSS receivers. These loops are responsible for writing the sensor and position data to text files. After the sensor loops receive data, the data is placed in a queue. Write-to-file loops extract data from the queue and write it to a file. If the queues always contain data, the wait function controls the loop rate, which was set to 10 ms or 100 Hz for all write-to-file loops. Thus, if the write-to-file loop rate is significantly less than the sensor loop rate, the queue will eventually fill up and data will be lost. If no data is in the queue, the write-to-file loop will wait until data is present, resulting in the queue controlling the loop rate. This occurs if the sensor loop rate is slower than the write-to-file loop rate, resulting in data being extracted from the queue faster than it is filled, eventually becoming empty.

The front panel of the main program is divided into four sections or tabs: Setup, Monitor, GPS, and Raw Data. The Setup tab (Figure 4.18) contains the control buttons, distance sensor calibration settings, and the field and sensor row spacing settings. The Monitor tab (Figure 4.19) contains the custom gauges for graphically viewing incoming sensor data. The GPS tab (Figure 4.20) shows a graph of position data points from the left and right GNSS receivers. The Raw Data (Figure 4.21) tab contains tables, one for each type of sensor that displays the five most recent data points.

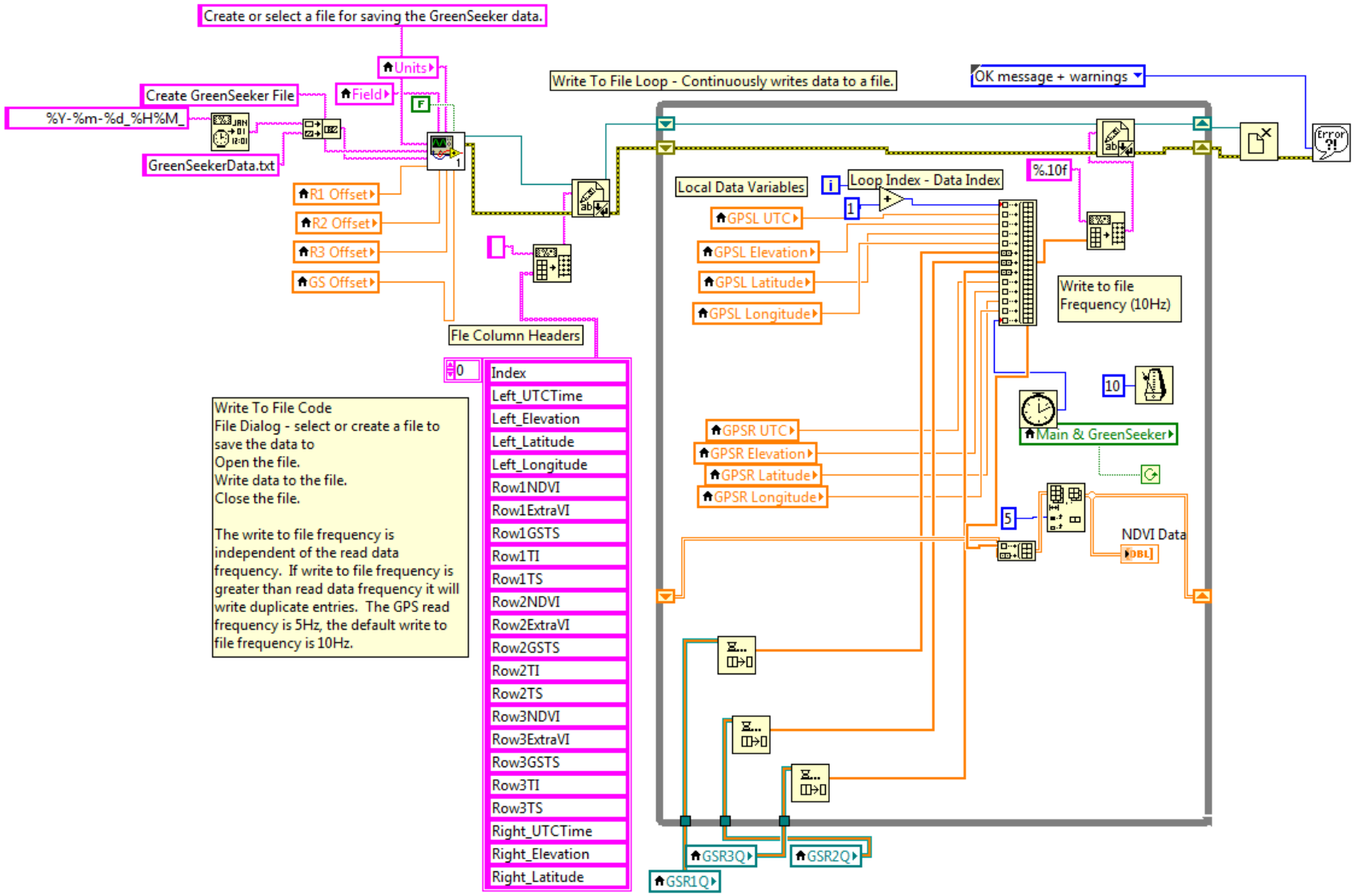

Figure 4.17 GreenSeeker write-to-file loop. Four other write-to-file loops are similar.

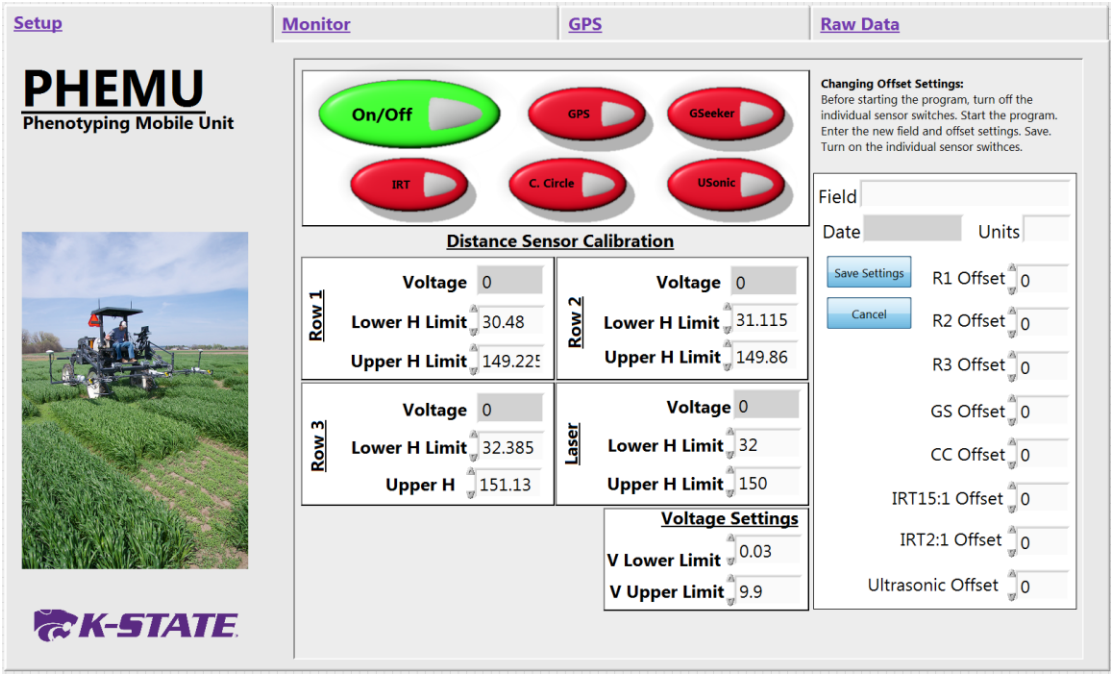

Figure 4.18 The Setup tab of the phenotyping software contains the cotnrol buttons, distance sensor calibration settings, and the field and sensor offset settings.

Figure 4.19 The Monitor tab contains custom gauges for real-time monitoring of the sensor readings.

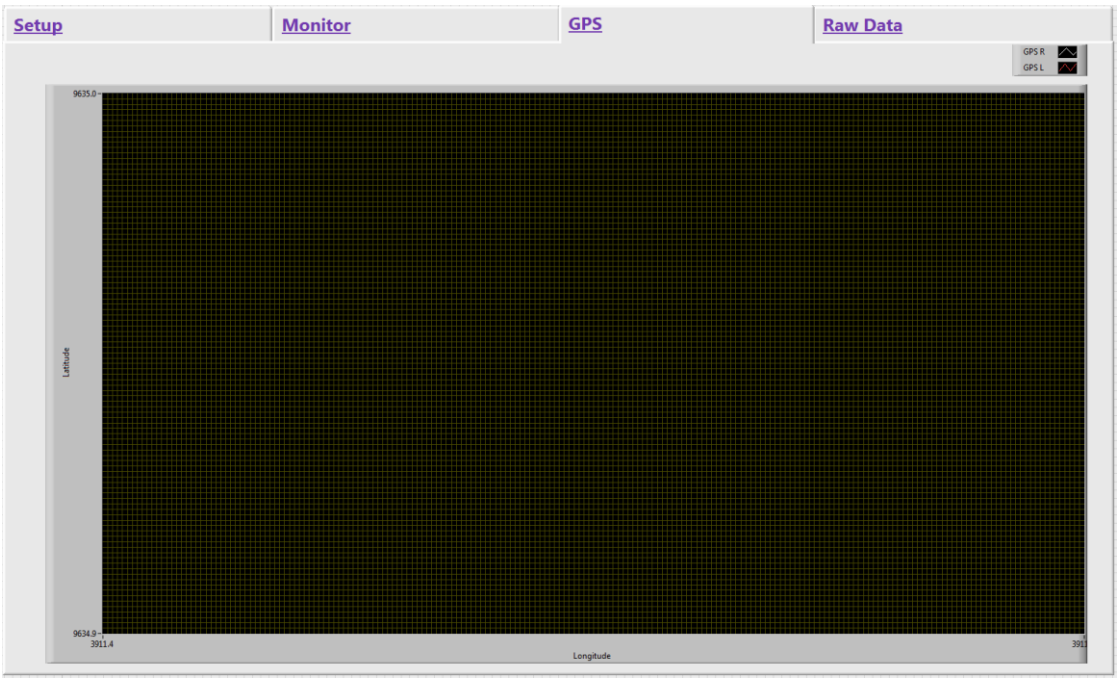

Figure 4.20 The GPS tab plottings the position data from the GNSS receivers.

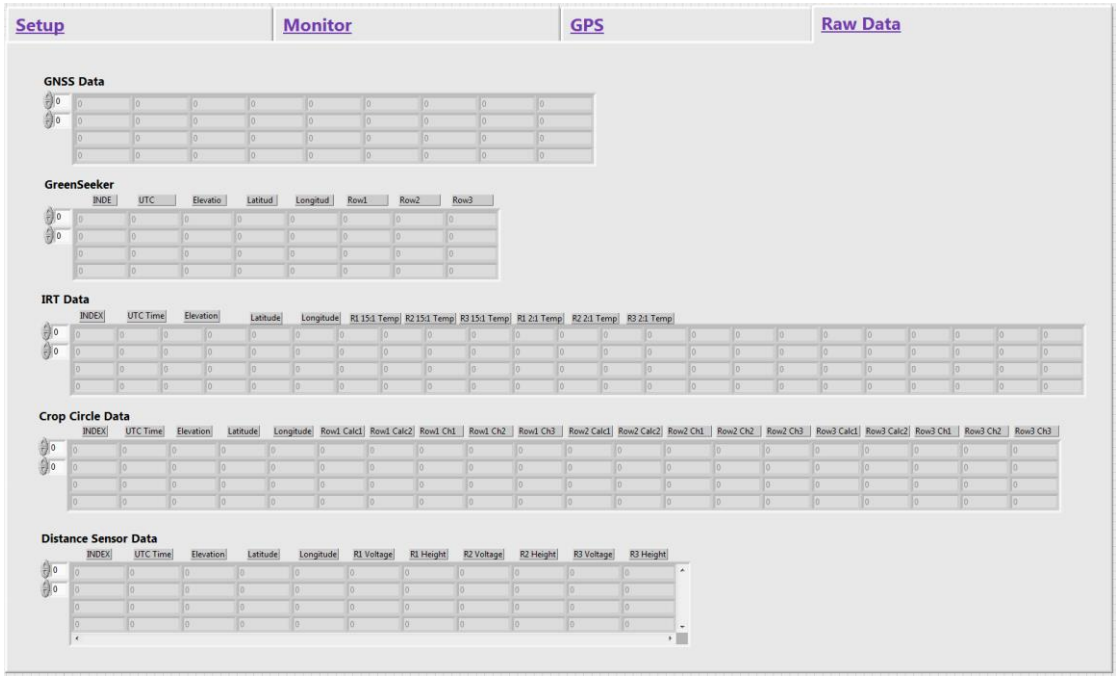

Figure 4.21 The raw data tab contains tables that contain the five most recent data points from all the sensors.

Verification Test

A variety of in-depth tests, both system-wide and component specific, could be conducted to assess and optimize the performance, reliability, robustness, accuracy, and operational limits of the phenotyping system. With the technology and requirements for phenotyping evolving it would be impractical to conduct all of these tests, many of which would be unnecessary in verifying the key objectives of this study: timely collection of sensor and position data and assessment of the effects of ambient light intensity and temperature on sensor readings. Two tests were performed to verify and accomplish these objectives.

System Timing and Position Errors

The objective of the first test was to determine the timeliness of the sensor and position data in order to determine position error of the sensor data in relation to GNSS position coordinates. Delays in reading-in sensor and GNSS data, output rate of the sensors and GNSS receivers, and ground speed of the phenotyper all affect accuracy of location data that is logged with each sensor data point. Sensor data and position data are not received simultaneously since, with the exception of polled sensors (distance sensors and IRTs), the phenotyping software has no control over when sensors send data and any delays that exist in the communication hardware. The sensors were configured to send data at a rate of 10 Hz, and the FmX units were configured to send position data at a rate of 5 Hz, which is the maximum. Furthermore, since sensor data is received twice as often as position data, two sensor data points are assigned identical position coordinates. Since the phenotyper moves forward during data collection, the position error due to differences in sampling time and throughput rates are proportional to phenotyper speed.

Since time differences in receiving sensor and positions data are independent of phenotyper speed, this test was performed while the phenotyper was stationary. Three 20-minute trials were conducted. For each trial, the phenotyping software logged the sensor and position data along with several timestamps for each sampling period. These timestamps were the time when the software was ready to read the data, the time when it read or received the data, and the time when it wrote the data to the text file. These timestamps, based on the computer system's millisecond clock available throughout the operating system, provides a reliable, consistent, and millisecond-resolution synchronization solution for the timing operations of the phenotyping software.

Raw test data were saved into five files, one for each sensor type and one for position data. An R (http://www.R-project.org) program was written for each raw data file (Appendix A - R Programs for Statistical Analysis) to calculate the average, standard deviation, and range of periods and time delays for each sensor and to georeference each sensor data point. The program outputs were saved into several text files. The R programs were similar to one another, but the program for processing GNSS position data must first be run before running any programs for processing sensor data since these require the position data that is loaded into R by the GNSS program.

To determine the time delay and to georeference the sensor data, the shortest time delay relative to position data from both the left and right GNSS receivers is first calculated. Thus, two time delays are calculated, but only one is linked to the data. The time delays are equal to the absolute value of the difference between the sensor data read-time and the position data read-time. This position data read-time is not the UTC, it is the time in milliseconds, based on the computer millisecond clock, when the phenotyping software received that specific position data. The second step is to compare the UTC associated with the position coordinates used in calculating each time delay.

There are two scenarios, the first is if the UTCs of the left-position data and right-position data are equal, then the time delay that is chosen is the one associated with the position data with the earliest read-time - the one the phenotyping software first received. Since the UTC times are identical, it can be assumed that both the left and the right GNSS receivers "generated" the position data simultaneously, but due to system delays in sending and receiving the data, the phenotyping software did not receive these at exactly the same moment. Both position data points (left and right) are used to georeference the sensor data. The second scenario is when the UTCs are not equal. In this situation, the program chooses the shortest time delay and the sensor data is georeferenced with the position data associated with this time delay. At this point, georeferencing is incomplete since the sensor data is referenced to only one position data point, but two points are required: one from the left GNSS receiver and one from the right GNSS receiver. The second data point is obtained by searching for the position data point with the identical UTC, or the nearest, to the UTC of the currently referenced position data point.

Determining Effects of Ambient Light Intensity and Temperature

Throughout the phentoyping season and during a single phenotyping session, sensors are subjected to varying ambient light intensities and ambient air temperatures due to weather, time of day, duration of phenotyping, direction of travel, and sensor position. The objective of this second test was to determine the effects of changes in ambient light intensity and ambient temperature on sensor readings.

The second test was performed outdoors, with the phenotyper parked facing south to avoid self-shadowing. The sensor boom was raised to 100 cm above the ground and supported by two 2 inch by 4 inch boards to maintain constant height. A 6 ft by 8 ft piece of green rug was placed on the ground, centered underneath the left set of sensors (Figure 4.22A). A tarp and canvas sheet was placed underneath the rug to keep it dry. A T-type thermocouple was taped on the rug's surface to measure rug temperature and another T-type thermocouple was used to measure ambient air temperature (Figure 4.22B). Both thermocouples were connected to a Campbell Scientific CR850 datalogger (Campbell Scientific, Inc., Logan, Utah, USA) which was programmed (Appendix F - CR850 Program For Measuring Temperature) to send temperature measurements through the serial port at a rate of 1 Hz. A Sper Scientific 850007 (Sper Scientific, Scottsdale, Ariz., USA) visible light logger for measuring light intensity was placed on the carpet (Figure 4.22C). This light meter can send light intensity measurements in lux at a rate of 1 Hz through its RS-232 port. All sensor data was logged on a laptop computer through a LabVIEW program. Data was initially logged once every minute, but was increased to once every ten seconds after the first 2 hours. No averaging of the logged data occurred. The test, conducted on a clear, sunny day with no wind, began at approximately 10:30 AM and finished at approximately 8:45 PM. Test data was analyzed to determine any significant relationships between sensor readings and ambient light intensity and ambient temperature.

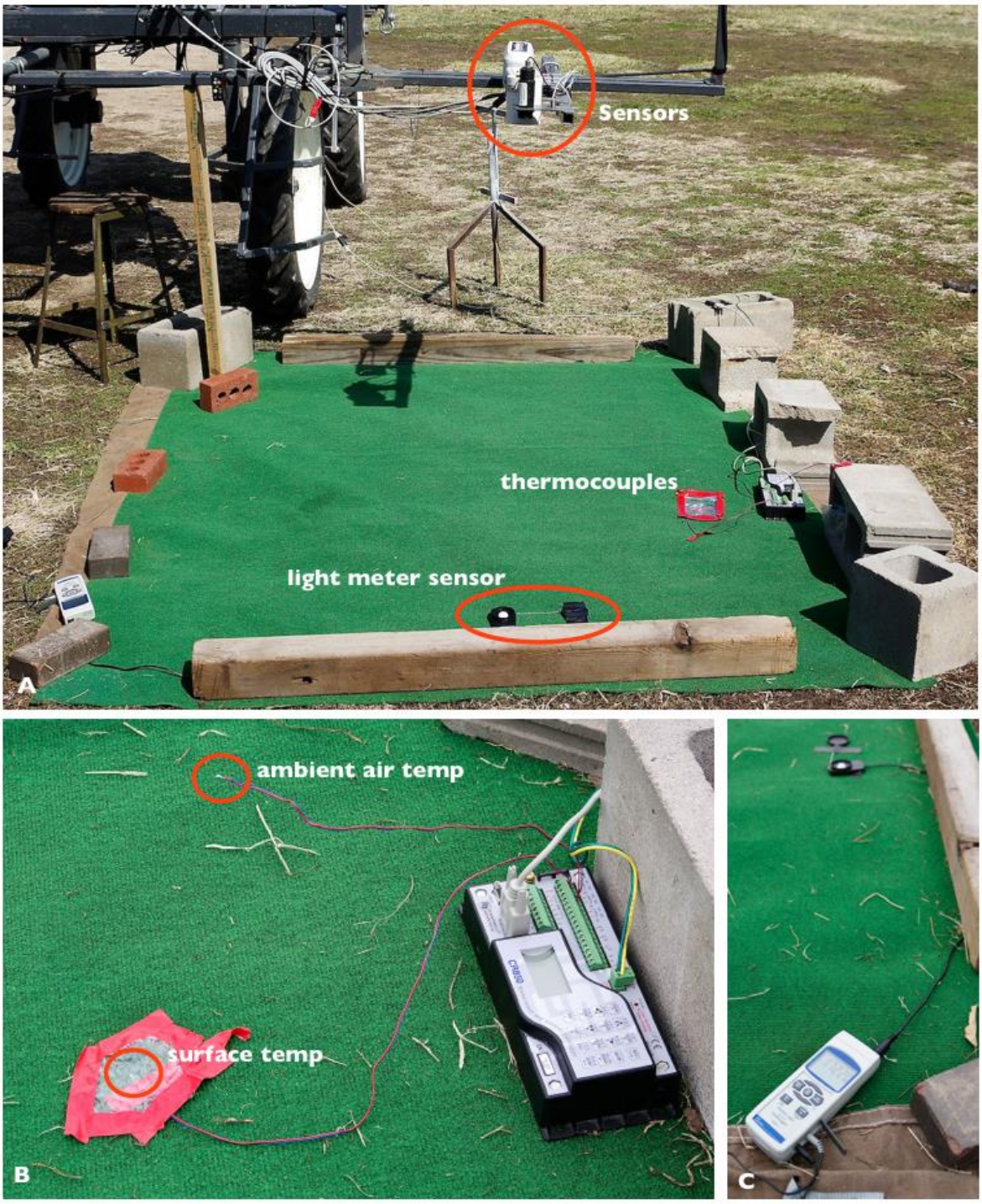

Figure 4.22 A) Sensors positioned 100 cm above a green colored carpet for the second verification test B) Thermcouples connected to a Cambell Scientific CR850 datalogger used to measure ambient and carpet surface temperatures C) Sper Scientific light meter used to measure light intensity

Chapter 5 - Results and Discussion

System Timing and Position Errors

Results of the system timing verification test show that the phenotyping software and system of hubs that connect sensors to the computer can service the sensors and GNSS receivers at the required rates of 10 Hz and 5 Hz, respectively. On average, for all three trials (Table 5.1), the phenotyping software received position data from the left GNSS receiver at a rate of 5 Hz or every 200 ms. Standard deviation for each read period was 39.5 ms, 32.6 ms, and 32.9 ms for Trials 1, 2, and 3, respectively. A greater variation of maximum and minimum read periods existed between the three trials. In the first trial, the maximum read period was 1507 ms, but only 1087 ms and 926 ms for the second and third trials, respectively. All these are much larger than the expected 200 ms periods based on the 5 Hz setting of the GNSS receivers and they are significant because they result in an accumulation of data in the buffer. More than one data point in the buffer indicates a significant delay. Figure 5.1 indicates that these long delays do not occur very often. Minimum read periods were 8 ms, 0 ms, and 0 ms for Trials 1, 2, and 3, respectively. In some instances, a large delay occurred in servicing the serial port, resulting in the long period and allowing GNSS data to accumulate in the buffer. When the serial port was serviced again, consecutive data points were quickly read (because the sensor loop did not have to wait for data), resulting in a very short period. Results for the right GNSS receiver are similar (Table 5.1) to the left GNSS receiver.

The *"TDif"* columns in Table 5.1 are the statistics for the time difference between the ready-to-read time (TI or Time In) and the read time (TS or Timestamp). This difference represents the amount of time the sensor modules wait for the serial data to arrive at the serial port. On average, GNSS modules wait for 200 ms; therefore, the GNSS loop rate is typically limited by the rate at which GNSS receivers send data, not limits imposed by the wait function. When GNSS data is buffered at the serial port, the wait function becomes the limiting factor. Figure 5.2 shows that TDif is usually the same length as the read period because the difference between the two typically is zero. Therefore, as soon as the GNSS loop reads the position data, it immediately returns to the beginning of the loop and waits for the next position data point.

Table 5.1 System timing results for the average, standard deviation, and range of the ready-to-read times (TI or Time In), read times (TS or Timestamp), and waiting-to-read times (TDif) of the left and right GNSS receivers.

		Left GNSS			*Right GNSS*		
		TI	**TS**	**TDif**	**TI**	**TS**	**TDif**
	Intended Period (ms)	200	200	200	200	200	200
	Intended Frequency (Hz)	5	5	5	5	5	5
	Ave. Period (ms)	200.001	199.991	199.831	200.031	200.034	199.968
	Ave. Freq (Hz)	5.000	5.000	5.004	4.999	4.999	5.001
Trial 1	***SD Period (ms)***	38.621	39.490	39.724	34.912	35.244	35.544
	Min (ms)	21	8	0	89	32	0
	Max (ms)	1432	1507	1432	1207	1264	1207
	Ave. Period (ms)	199.973	199.984	199.802	199.976	199.973	199.917
	Ave. Freq (Hz)	5.001	5.000	5.005	5.001	5.001	5.002
Trial 2	***SD Period (ms)***	32.527	32.612	33.003	32.072	32.077	32.074
	Min (ms)	13.000	0.000	0.000	93.000	93.000	93.000
	Max (ms)	1087.00	1087.00	1087.00	320.000	320.000	320.000
	Ave. Period (ms)	199.930	199.920	199.776	199.953	199.954	199.830
	Ave. Freq (Hz)	5.002	5.002	5.006	5.001	5.001	5.004
Trial 3	***SD Period (ms)***	32.843	32.949	33.256	33.529	33.734	33.787
	Min (ms)	13.000	0.000	0.000	10.000	0.000	0.000
	Max (ms)	925.000	926.000	925.000	495.000	495.000	495.000

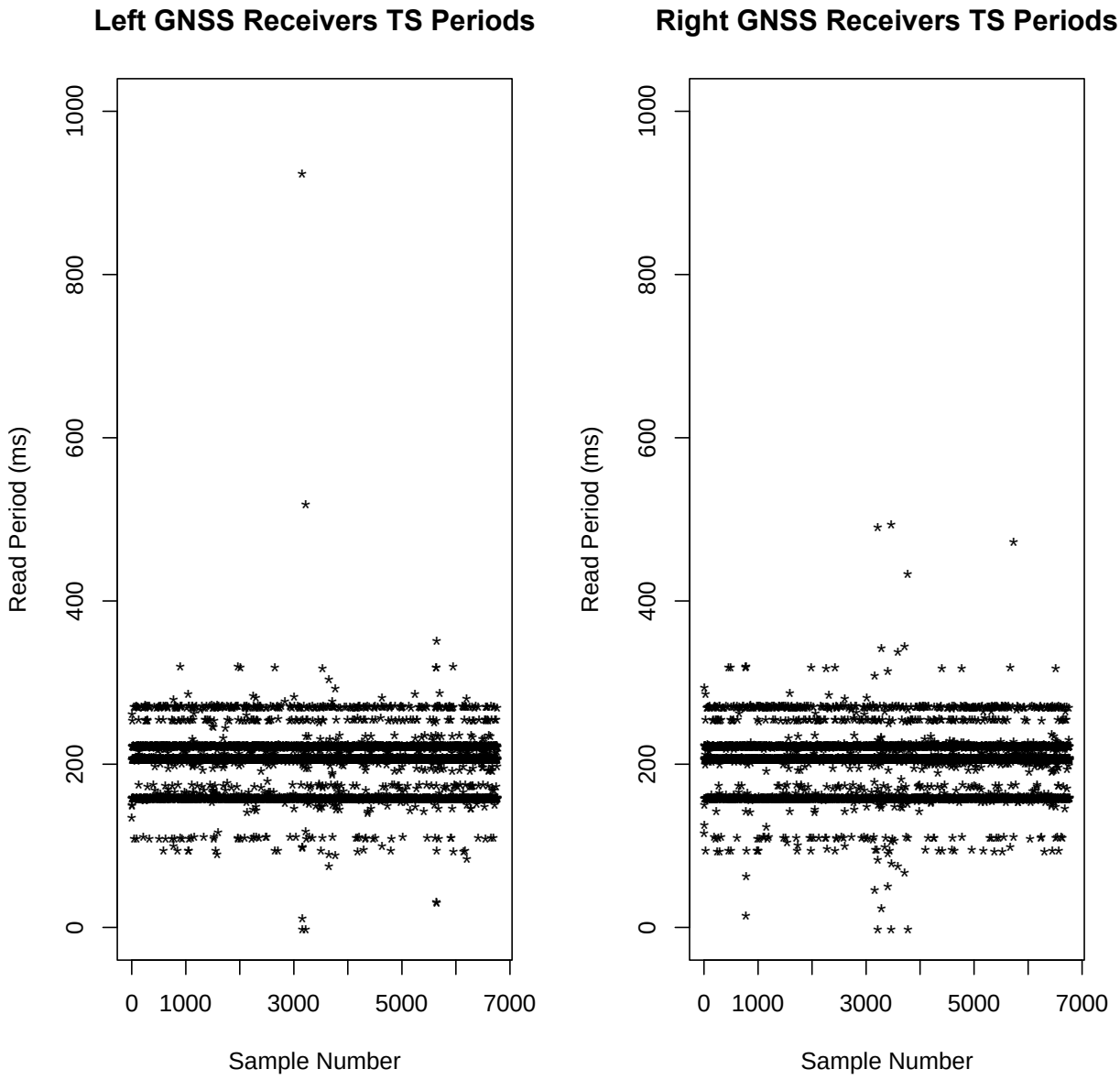

Figure 5.1 Plot of read periods for left and right GNSS receivers during the third trial showing significant delays in service of the serial port occurring less than 5 times in over 6,000 samples.

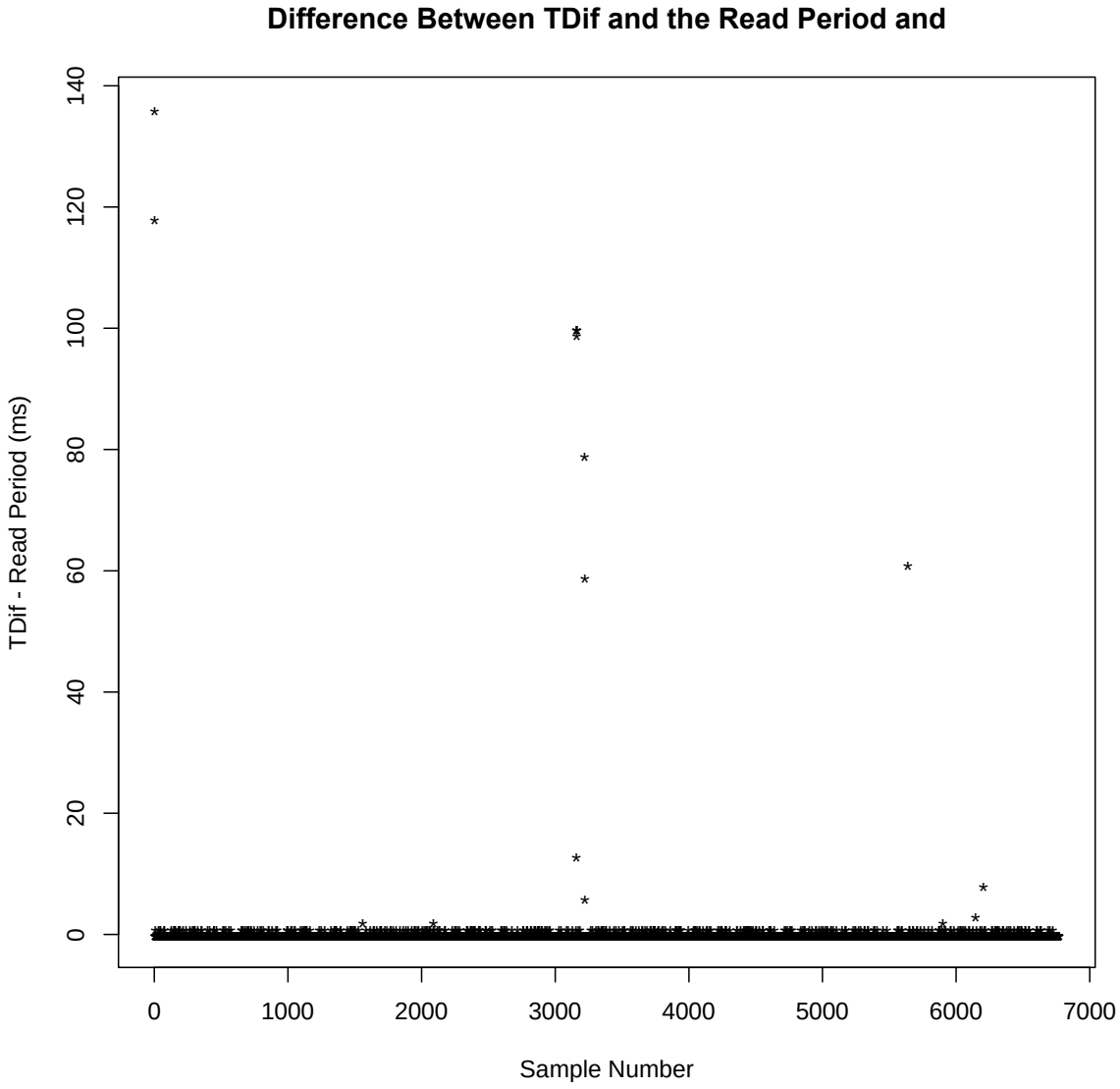

Figure 5.2 For almost all GNSS samples, the difference between TDif and read periods is zero.

Statistics for the write-to-file periods (Table 5.2) indicate that at least 95% of the time, the GNSS sensor loop limits the rate of GNSS write-to-file loop (Figure 5.3) since the average frequency is 5 Hz and not the 100 Hz it was set to. It is working the way it was designed to. The rate at which GNSS receivers send data (5 Hz) controls the rate of the sensor loop. The wait function within the write-to-file loop was set to 10 ms (100 Hz) because, in instances such as at startup when the user inputs file names and file paths for output data files, the sensor loops have

already begun to run and place data into queues. Write-to-file loops cannot begin until the filenames and paths are specified; therefore, data in the queues must be written out. Setting the write-to-file loop to 100 Hz allows the loop to quickly clear the queues and catch up to the sensor loop.

The wait function inside the write-to-file loop is necessary for situations in which the write-to-file loop must clear accumulated data in the queue. Removing the wait function essentially gives the write-to-file loop the highest priority among all the loops, allowing it to repeat as quickly as possible and potentially "steal" run time from the other loops. Adding the wait-function ensures that this does not occur. However, exact timing of the loops is difficult to analyze because LabVIEW automatically handles system scheduling.

Table 5.2 System timing results for write-to-file times for GNSS receivers for Trials 1, 2, and 3

	Trial 1	***Trial 2***	***Trial 3***
Ave. Period (ms)	199.769	199.409	199.322
Ave. Freq (Hz)	5.006	5.015	5.017
SD Period (ms)	35.838	32.582	35.715
Min (ms)	8.000	1.000	2.000
Max (ms)	509.000	516.000	562.000

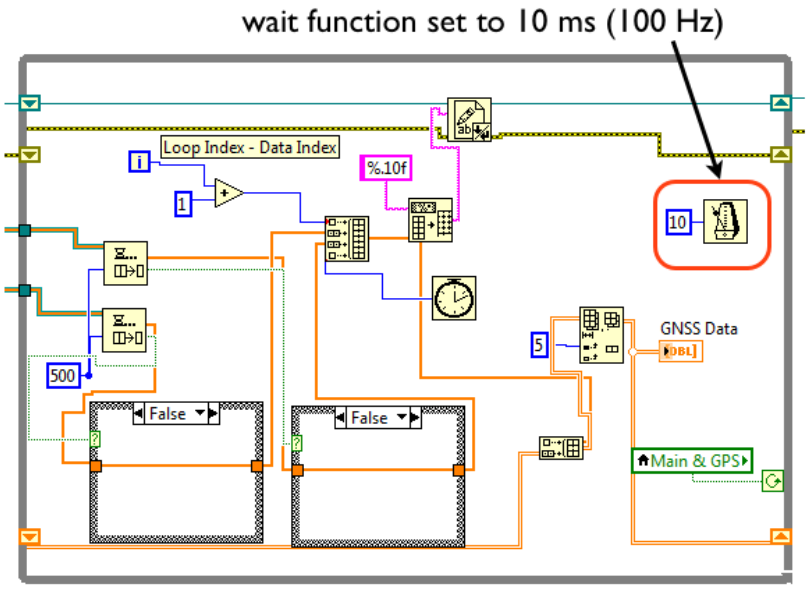

Figure 5.3 Wait function inside write-to-file loops set to 10 ms (100 Hz)

Results for all three GreenSeeker sensor loops in Trials 1, 2, and 3 indicate that the average period of reading-in data was approximately 122 ms, or a frequency of 8.15 Hz (Table 5.3). The GreenSeekers were configured using RTCommander software to sample at 100 ms. The GreenSeekers were expected to send data every 100 ms, or at 10 Hz. However, this is not a problem with the phenotyping software because internal data timestamps of the GreenSeekers, columns *"R1GSTS," "R2GSTS,"* and *"R3GSTS,"* have an average period of approximately 122 ms. However, brief instances occur in the raw data in which the period decreases to 100 ms, or a sampling frequency of 10 Hz. When this decrease occurs, the read time period also decreases to approximately 100 ms; therefore, the phenotyping software can keep up with the GreenSeekers.

Average waiting-to-read times (TDif columns) are similar to read times, indicating that, on average, as soon as a GreenSeeker data point is read from the serial port, the GreenSeeker sensor loop immediately returns to the beginning of the loop and waits for the next data point. Because the sensor loop must wait, consecutive data points are not frequently buffered in the serial port.

Maximum read periods for three tests of all three GreenSeekers occurred during the third trial. The maximum periods were 1061 ms, 913 ms, and 937 ms for rows 1, 2, and 3, respectively. These periods represent significant delays in servicing serial ports connected to the GreenSeekers, and these delays result in buffered data that cannot be accurately georeferenced. Depending on when and where these delays occur during phenotyping, the data could still be assigned to the correct field plot, as when the delay in servicing the serial port occurs when the sensor is at the beginning of the plot but resumes when the sensor is in the middle of the plot. Frequent occurrence of these delays would be a problem, but the graph of the periods of more than 10,000 samples (Figure 5.4) indicatse less than ten instances that are significantly delayed, causing other data points to accumulate in the buffer.

Table 5.3 Summary of system timing results for all three trials for the average, standard deviation, and range of the read times (TS), write-to-file times (WT), data send times (GSTS), and waiting-to-read times (TDif) of GreenSeeker sensors in all three rows

GreenSeeker		*Ave. Period (ms)*	*Ave. Freq (Hz)*	*SD Period (ms)*	*Min (ms)*	*Max (ms)*
Intended Period (ms)		100	100	100	100	100
Intended Frequency (Hz)		10	10	10	10	10
Trial 1	***R1***	122.7	8.1	9.8	81.0	145.0
	R2	123.7	8.1	9.4	81.0	144.0
	R3	122.7	8.1	9.6	85.0	144.0
	WT	123.7	8.1	9.4	81.0	144.0
	R1GSTS	122.0	8.2	6.8	99.0	124.0
	R2GSTS	122.9	8.1	6.8	99.0	125.0
	R3GSTS	122.0	8.2	6.8	99.0	124.0
	R1TDif	122.7	8.2	9.8	81.0	145.0
	R2TDif	123.7	8.1	9.4	81.0	144.0
	R3TDif	122.7	8.2	9.6	85.0	144.0
Trial 2	***R1***	122.7	8.1	10.0	1.0	313.0
	R2	123.0	8.1	10.3	0.0	342.0
	R3	122.7	8.1	10.4	0.0	395.0
	WT	123.0	8.1	10.2	3.0	343.0
	R1GSTS	122.0	8.2	6.8	99.0	124.0
	R2GSTS	122.3	8.2	6.9	99.0	125.0
	R3GSTS	122.0	8.2	6.8	99.0	124.0
	R1TDif	122.7	8.2	10.1	0.0	313.0
	R2TDif	123.0	8.1	10.3	0.0	341.0
	R3TDif	122.7	8.2	10.5	0.0	395.0
Trial 3	***R1***	122.6	8.2	13.9	0.0	1061.0
	R2	122.6	8.2	13.1	0.0	913.0
	R3	122.6	8.2	13.7	0.0	937.0
	WT	122.6	8.2	13.3	1.0	914.0
	R1GSTS	121.9	8.2	6.9	99.0	124.0
	R2GSTS	121.9	8.2	6.9	99.0	125.0
	R3GSTS	121.9	8.2	6.9	99.0	124.0
	R1TDif	122.5	8.2	14.2	0.0	1061.0
	R2TDif	122.5	8.2	13.4	0.0	912.0
	R3TDif	122.5	8.2	14.1	0.0	937.0

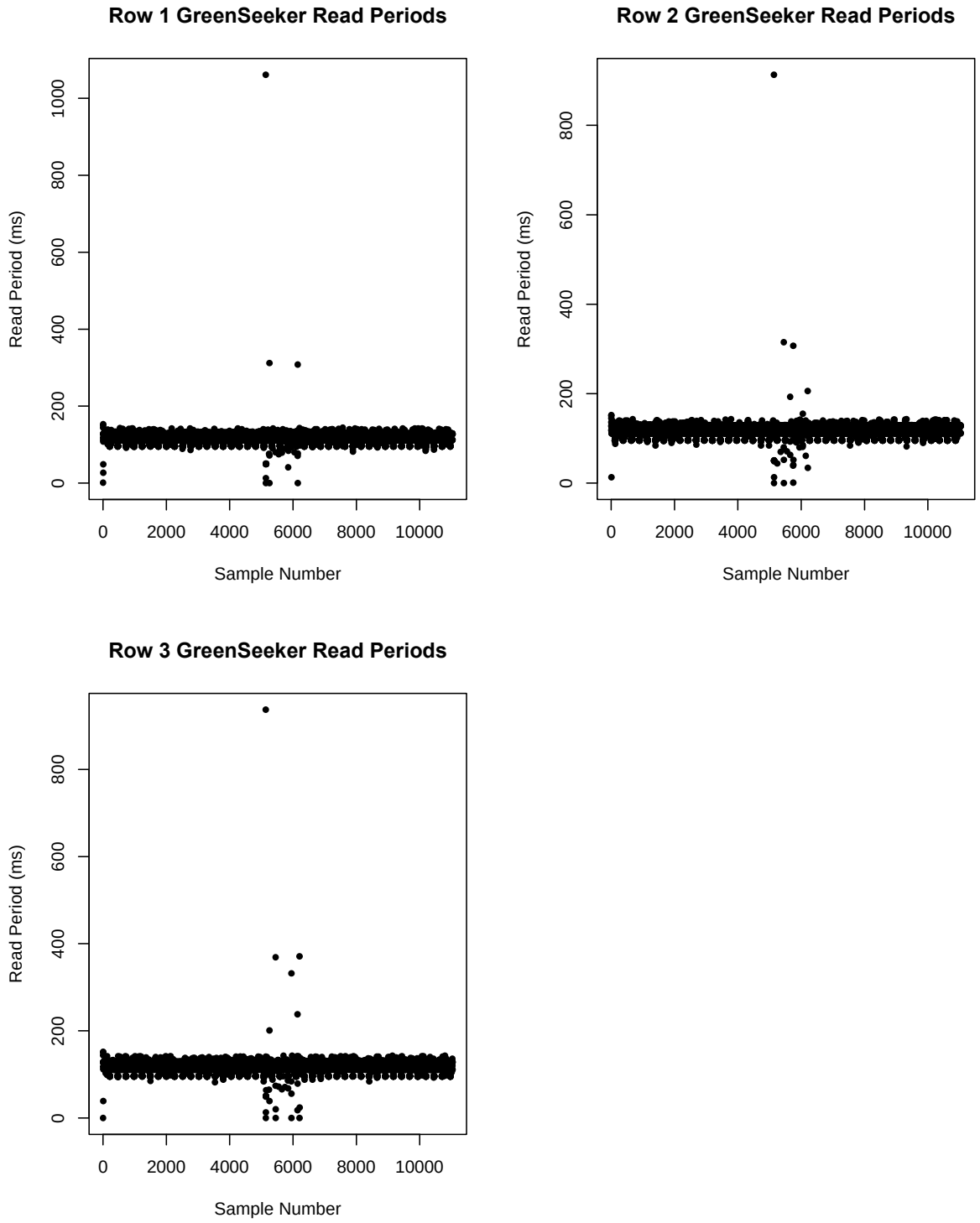

Figure 5.4 Plots of read periods for GreenSeekers during the third trial showing significant delays in service of the serial port occurring less than 10 times in over 10,000 samples

The average read, write, and wait periods for the Crop Circle sensors for all three trials were approximately 101 ms (~10 Hz) (Table 5.4). Maximum and minimum periods were identical for all Crop Circle sensors, with no significant delays and therefore no data buffering. When data was not buffered (Trial 1), the minimum delay was similar to the average value, and not zero. Maximum read periods in Trials 2 and 3 indicate significant delays for all Crop Circle sensors, with the exception of Row 1 during the second trial. Minimum read periods were zero, confirming that data was buffered, but Figure 5.5 suggests that this does not frequently occur.

Table 5.4 Summary of the average, standard deviation, and range of the read times (TS), write-to-file times (WT), and waiting-to-read times (TDif) of Crop Circle sensors in all three rows and for all three trials

Crop Circle		*Ave. Period (ms)*	*Ave. Freq (Hz)*	*SD Period (ms)*	*Min (ms)*	*Max (ms)*
Intended Period (ms)		100	100	100	100	100
Intended Frequency (Hz)		10	10	10	10	10
Trial 1	***R1***	101.5	9.9	8.1	80.0	127.0
	R2	101.0	9.9	8.0	80.0	127.0
	R3	101.0	9.9	7.9	80.0	128.0
	WT	101.5	9.9	8.1	80.0	127.0
	R1TDif	101.4	9.9	8.1	80.0	127.0
	R2TDif	100.9	9.9	8.0	80.0	127.0
	R3TDif	101.0	9.9	7.9	80.0	128.0
Trial 2	***R1***	101.5	9.9	7.9	39.0	127.0
	R2	101.0	9.9	8.4	0.0	436.0
	R3	101.0	9.9	8.8	0.0	436.0
	WT	101.5	9.9	7.9	39.0	127.0
	R1TDif	101.4	9.9	7.9	39.0	127.0
	R2TDif	100.9	9.9	8.6	0.0	435.0
	R3TDif	100.9	9.9	9.0	0.0	435.0
Trial 3	***R1***	101.4	9.9	9.3	0.0	396.0
	R2	101.0	9.9	9.2	0.0	464.0
	R3	101.0	9.9	11.1	0.0	911.0
	WT	101.4	9.9	9.3	10.0	396.0
	R1TDif	101.4	9.9	9.5	0.0	396.0
	R2TDif	100.9	9.9	9.5	0.0	464.0
	R3TDif	100.9	9.9	11.5	0.0	911.0

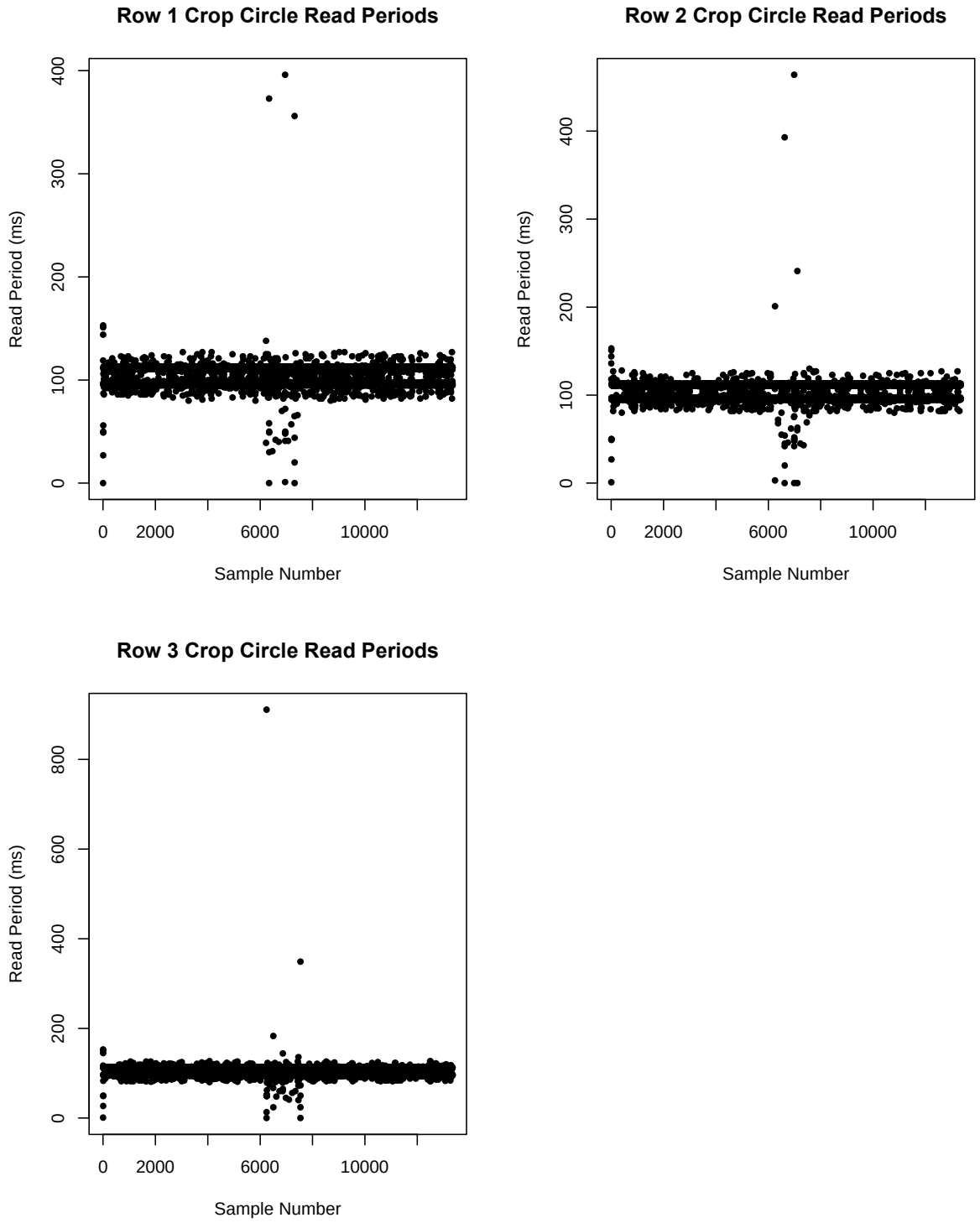

Figure 5.5 Plots of read periods for Crop Circles during the third trial showing significant delays in service of the serial port occurring less than 10 times in over 10,000 samples

The 15:1 and 2:1 set of IRT sensors were polled, meaning that, in order to receive temperature measurements, a command or request must first be sent to the IRTs for each temperature measurement. Polling provides increased control over the rate at which measurements are sent and received. The disadvantage is the added time overhead of sending a command.

Every 100 ms, a command is sent to the six IRTs (two in each sensor set). Data (Table 5.5) from all three trials show that, on average, data is received from the IRTs every 100 ms, or at rate of 10 Hz. Data in Table 5.6 indicates that, on average, temperature readings are received approximately 15 ms after the command is sent. This estimates delays involved in sending data from the sensors to the computer. Typical maximum and minimum differences between times when the command was sent and the data was received for the first and second trials were approximately 31 ms and 10 ms, respectively. For the third trial, the maximum delay was significantly greater at approximately 138 ms. This does not mean that consecutive temperature data was buffered because the IRTs were polled. However, it does suggest that either the sent command was delayed on the way to the sensor, or reading of the data from the sensor was delayed, or the data was wating to be sent at the serial port but IRT modules were delayed in reading it. Exactly which situation occurred or if another situation occurred, cannot be determined with certainty, but the minimum wait time was 10 ms, equal to the delay added between the send and receive commands inside the IRT module (Figure D.9 and Figure D.12). This delay was added to give sensors time to respond before the module expected a response. Data suggests that this delay can be shortened or removed.

The distance sensor module was initially designed to be polled, similar to the IRT. Four distance sensors, three ultrasonic sensors and one laser sensor, all output analog voltages which were measured using the NI 9207 DAQ. The DAQ was initially configured using the NI DAQ Assistant to take only one sample on demand. Initial testing revealed that when samples were requested every 100 ms (10 Hz), the average period of reading in data was approximately 125 ms (8 Hz), indicating a significant time overhead from continuously starting, stopping, and restarting data acquisition. To avoid this, the DAQ was reconfigured to continuously take samples at a rate of 10 Hz, thereby avoiding delays involved with starting and stopping data acquisition. This change made it similar to other non-polled sensors which continuously send data. If voltage data

are read less frequently from the DAQ than the rate at which the DAQ samples the sensor voltages, the sampled voltages will be buffered.

Table 5.5 Summary of the average, standard deviation, and range of the read times (TS) and write-to-file times (WT) of IRT sensors in all three rows and for all three trials

IRT Sensor		*Ave. Period (ms)*	*Ave. Freq (Hz)*	*SD Period (ms)*	*Min (ms)*	*Max (ms)*
Intended Period (ms)		100	100	100	100	100
Intended Frequency (Hz)		10	10	10	10	10
Trial 1	*R1 15:1*	100.0	10.0	12.8	42.0	156.0
	R2 15:1	100.0	10.0	12.9	43.0	157.0
	R3 15:1	100.0	10.0	12.6	41.0	159.0
	R1 2:1	100.0	10.0	13.0	43.0	163.0
	R2 2:1	100.0	10.0	13.1	43.0	161.0
	R3 2:1	100.0	10.0	12.9	45.0	157.0
	WT	100.0	10.0	13.1	43.0	161.0
Trial 2	*R1 15:1*	100.0	10.0	13.0	40.0	157.0
	R2 15:1	100.0	10.0	13.0	44.0	164.0
	R3 15:1	100.0	10.0	13.0	21.0	159.0
	R1 2:1	100.0	10.0	12.9	37.0	156.0
	R2 2:1	100.0	10.0	12.8	43.0	164.0
	R3 2:1	100.0	10.0	13.1	29.0	162.0
	WT	100.0	10.0	12.0	55.0	158.0
Trial 3	*R1 15:1*	100.0	10.0	12.9	42.0	201.0
	R2 15:1	100.0	10.0	13.1	34.0	201.0
	R3 15:1	100.0	10.0	13.3	18.0	201.0
	R1 2:1	100.0	10.0	13.1	45.0	211.0
	R2 2:1	100.0	10.0	13.0	31.0	201.0
	R3 2:1	100.0	10.0	13.1	26.0	201.0
	WT	100.0	10.0	12.2	22.0	204.0

All distance sensors share the same timestamp since only one instance of the distance sensor module occurs in the phenotyping software. Data in Table 5.7 shows that the average read period is approximately 100 ms (10 Hz), with a standard deviation of approximately 8 ms. Maximum read time period for all test trials was 145 ms, less than twice the sample period, therefore, consecutive distance sensor data were not buffered in any of the trials. As shown in Figure 5.8, the plot of read periods for the third trial shows that a majority of read periods are

between 90 ms and 110 ms, indicating that phenotyping software is capable of timely collecting distance sensor data.

Table 5.6 Summary of the average, standard deviation, and range of command send-to-read times (TDif) of IRT sensors in all three rows and for all three trials

	IRT	*Ave. Period (ms)*	*Ave. Freq (Hz)*	*SD Period (ms)*	*Min (ms)*	*Max (ms)*
Trial 1	***R1 15:1 TDif***	14.9	67.0	3.4	10.0	32.0
	R2 15:1 TDif	15.0	66.8	3.0	10.0	28.0
	R3 15:1 TDif	15.0	66.8	3.0	10.0	31.0
	R1 2:1 TDif	14.9	67.1	3.0	10.0	31.0
	R2 2:1 TDif	15.0	66.8	3.0	10.0	26.0
	R3 2:1 TDif	15.0	66.7	3.0	10.0	31.0
Trial 2	***R1 15:1 TDif***	15.0	66.6	3.1	10.0	31.0
	R2 15:1 TDif	15.2	65.9	2.6	10.0	28.0
	R3 15:1 TDif	15.1	66.1	2.6	10.0	31.0
	R1 2:1 TDif	15.1	66.2	2.7	10.0	26.0
	R2 2:1 TDif	15.1	66.2	2.6	10.0	28.0
	R3 2:1 TDif	15.2	65.9	2.6	10.0	32.0
Trial 3	***R1 15:1 TDif***	15.0	66.6	3.6	10.0	138.0
	R2 15:1 TDif	15.0	66.5	3.1	10.0	138.0
	R3 15:1 TDif	15.1	66.3	3.1	10.0	138.0
	R1 2:1 TDif	15.0	66.7	2.9	10.0	63.0
	R2 2:1 TDif	15.0	66.6	3.1	10.0	138.0
	R3 2:1 TDif	15.0	66.5	3.1	10.0	138.0

Table 5.7 Summary of the average, standard deviation, and range of the read times (TS), write-to-file times (WT), and waiting-to-read times (TDif) of distance sensors in all three rows and for all three trials

Distance Sensors		*Ave. Period (ms)*	*Ave. Freq (Hz)*	*SD Period (ms)*	*Min (ms)*	*Max (ms)*
Trial 1	***All Rows***	100.0	10.0	7.4	47.0	145.0
	WT	100.0	10.0	7.4	13.0	179.0
	RTDif	0.8	1228.1	1.0	0.0	3.0
Trial 2	***All Rows***	100.0	10.0	7.4	47.0	145.0
	WT	100.0	10.0	7.4	13.0	179.0
	RTDif	0.8	1228.1	1.0	0.0	3.0
Trial 3	***All Rows***	100.0	10.0	7.4	11.0	142.0
	WT	100.0	10.0	7.5	7.0	227.0
	RTDif	1.1	892.1	1.0	0.0	3.0

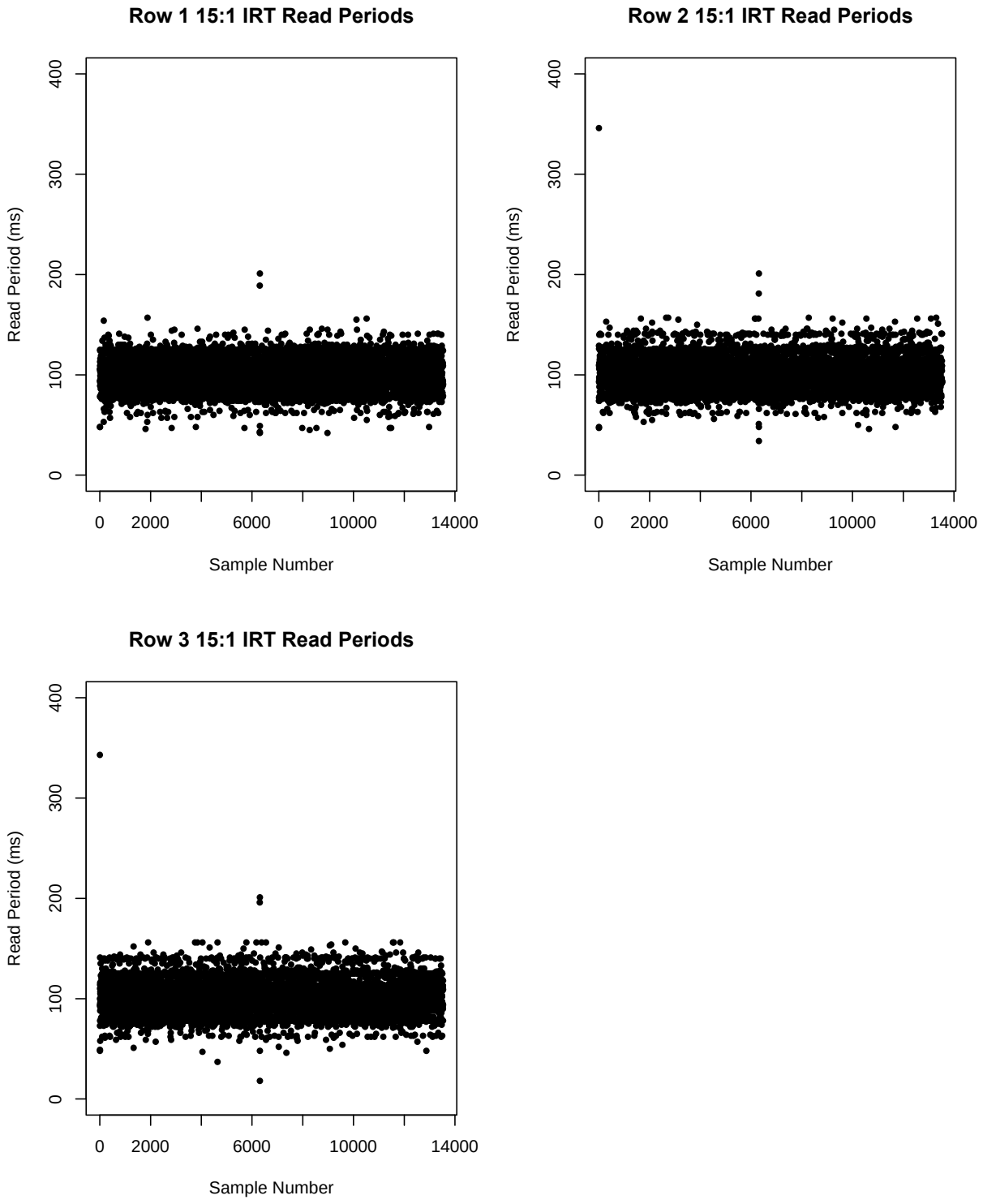

Figure 5.6 Plots of read periods for 15:1 IRTS during the third trial showing significant delays in service of the serial port occurring less than 10 times in over 10,000 samples

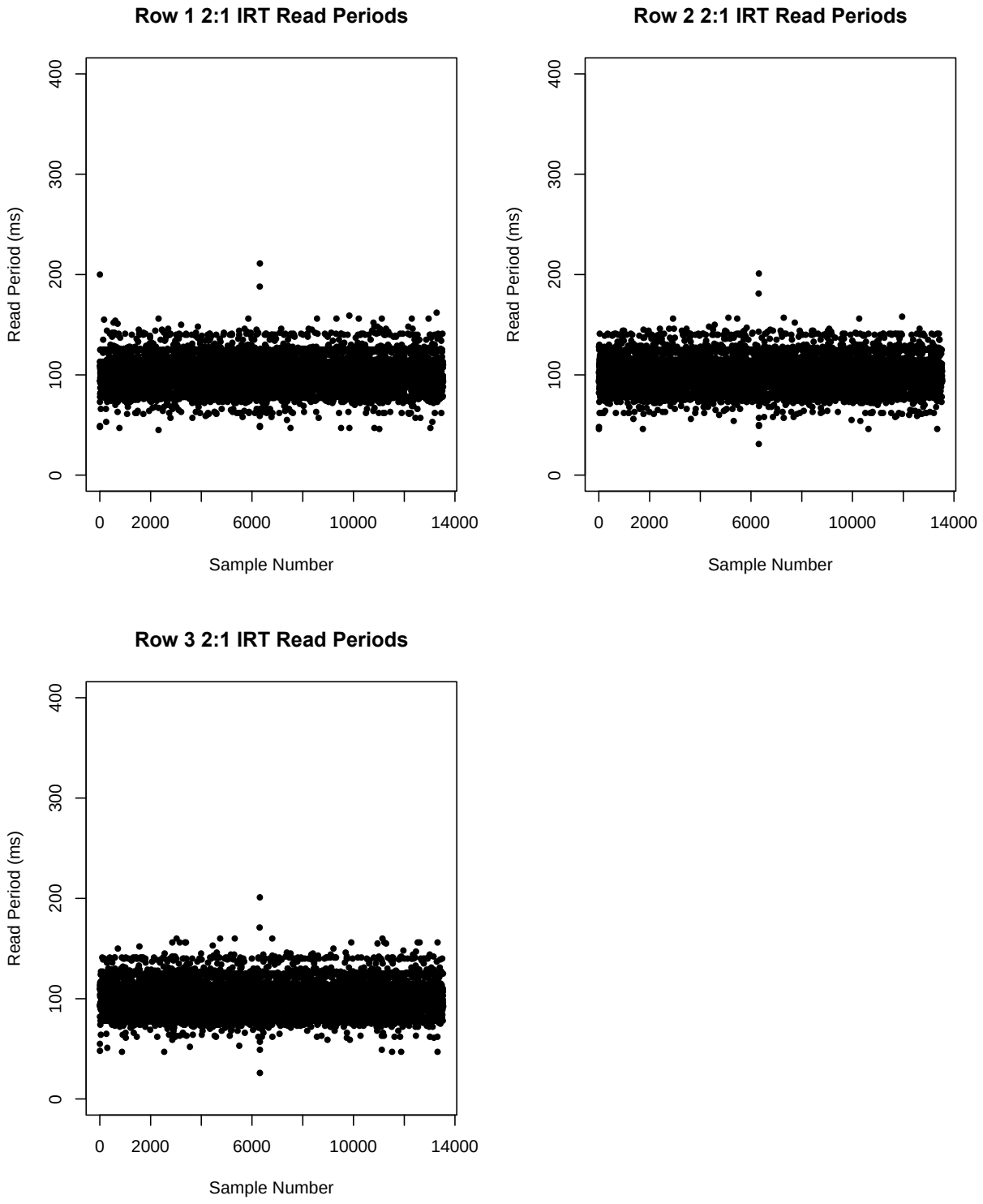

Figure 5.7 Plots of read periods for 2:1 IRTS during the third trial showing significant delays in service of the serial port occurring less than 10 times in over 10,000 samples

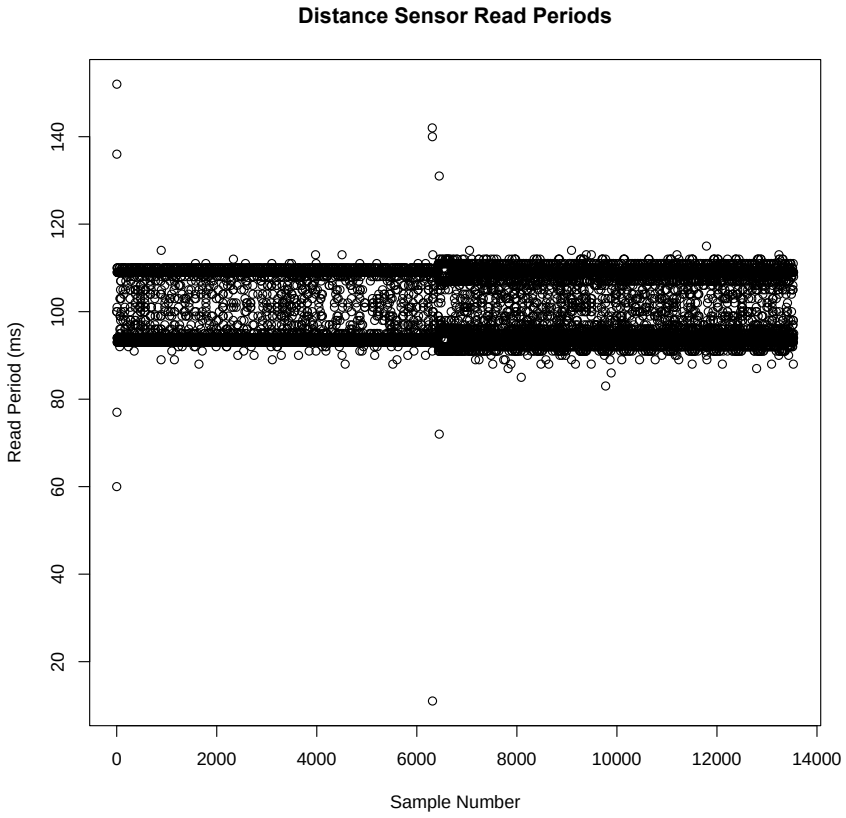

Figure 5.8 Plot of distance sensor read periods for the third trial showing that most read periods were between 90 ms and 110 ms and very few were over 120 ms

In Figure 5.8, many points have read periods of 110 ms or 90 ms. The DAQ was configured to consistently send data every 100 ms. A read period of 110 ms indicates a delay of 10 ms, assuming that the DAQ consistently sends data every 100 ms. If the distance sensor loop begins waiting at time 0 and reads the first data point generated at 100 ms, at 110 ms, the read period for that data point is 110 ms. Immediately after reading this point (at time ~110 ms), the distance sensor loop begins to wait for the next data point. This data point will be generated at time 200 ms. If no delays occur, the distance sensor loop will read this at time 200 ms, and the read period will be 90 ms.

Position Errors

Phenotyping software has no control of the timing of when data is generated and sent from the sensors and GNSS receivers and when this data arrives at the communication ports.

Data can arrive and be read in any order and at any time in regular intervals. Position data is used to georeference the sensor data, but this position is not the exact position of the sensor data. Position error is a result of the phenotyper moving forward and the time difference between when sensor data is generated and when position data is generated. If the phenotyper is not moving, no position error occurs except for the inherent error of RTK-corrected GNSS data (approximately 2-3 cm). The exact moment when sensor data is generated is unknown. An approximation of when the position data is generated is the associated UTC, with a resolution of 200 ms, but this is useless without a corresponding sensor UTC. The best approximation for the time difference between when data are generated is the time difference between when the phenotyper software receives sensor data and when it receives position data.

Results of all three tests indicate that the average time difference between sensor data and position data was approximately 50 ms (Table 5.8), with a standard deviation of approximately 30 ms. If the phenotyper is moving at 2 mph, the position error is approximately 4.5 cm. An average of 50 ms is to be expected since sensor data is received at 10 Hz and position data at 5 Hz. If the sensor data and position data were perfectly synchronized, the time differences for consecutive samples would be 0 ms, 100 ms, 0 ms, 100ms, and so on. The average of this would be 50 ms. The minimum time difference in Table 5.8 is 0 ms, indicating that instances occurred when sensor data and position data were received at the same time. This also confirms that on the specific laptop used for the tests, multiple loops were running simultaneously. Maximum time difference varies between sensors and trials, but it was typically less than 200 ms. However, maximum time difference increased to approximately 1,300 ms for all three Crop Circle sensors during Trial 2, resulting in a position error of 116 cm (at 2 mph). The large delays for all sensors occurred at approximately 6,000 samples, indicating a systemwide delay. This roughly corresponds to the 10-minute mark during the test due to the power settings on the laptop which change the laptop display state after being idle for approximately 10 minutes.

The position error of each sensor data point can be calculated and then used to correct its position and ensure that each data point is assigned to the correct field plot. The disadvantage is that it adds an additional step to the data post-processing pipeline. The alternative, also implemented by phenotyping software, is to tag each data point with the most recent position data. The phenotyping software saves five text files, each containing data collected from one type of sensor (one file for GreenSeekers, one for IRTs, etc.). Each row of sensor data also

contains the most recent position data points. Assuming no delays in generating and receiving sensor data or position data and assuming that position data and sensor data are not synchronized, the time difference between position data and sensor data ranges from 0 to approximately 200 ms. Assuming a phenotyping speed of 2 mph, the position error would be 17.9 cm. To reduce this error, the phenotyping speed can be decreased, resulting in more data per plot but reducing phenotyping throughput, or the number of plots able to be covered in a phenotyping run. Another solution would be to use a GNSS system that provides position data at a rate of 10 Hz, instead of 5 Hz, thus reducing the error to 9 cm since this reduces the time difference to 100 ms, assuming no delays.

Table 5.8 Summary statistics for sensor data time delays relative to position data

Trial	*Sensor*	*Average (ms)*	*Standard Deviation (ms)*	*Minimum (ms)*	*Maximum (ms)*
	GreenSeeker 1	49.2	30.3	0.0	179.0
	GreenSeeker 2	49.3	30.3	0.0	421.0
	GreenSeeker 3	49.2	30.1	0.0	186.0
	Crop Circle 1	49.3	29.7	0.0	173.0
	Crop Circle 2	49.4	29.6	0.0	154.0
	Crop Circle 3	49.4	29.7	0.0	179.0
Trial 1	***Height Sensors***	49.8	19.5	0.0	177.0
	IRT 15:1 1	48.2	36.5	0.0	333.0
	IRT 15:1 2	48.9	29.1	0.0	162.0
	IRT 15:1 3	48.9	29.3	0.0	179.0
	IRT 2:1 1	49.0	29.2	0.0	165.0
	IRT 2:1 2	48.9	30.0	0.0	154.0
	IRT 2:1 3	49.0	29.3	0.0	165.0
	GreenSeeker 1	49.2	30.0	0.0	138.0
	GreenSeeker 2	49.5	30.0	0.0	156.0
	GreenSeeker 3	49.3	30.1	0.0	134.0
	Crop Circle 1	49.9	39.1	0.0	1295.0
	Crop Circle 2	49.8	37.9	0.0	1295.0
	Crop Circle 3	49.9	37.4	0.0	1253.0
Trial 2	***Height Sensors***	49.5	31.3	0.0	150.0
	IRT 15:1 1	48.5	28.0	0.0	136.0
	IRT 15:1 2	48.8	28.5	0.0	136.0
	IRT 15:1 3	48.9	28.4	0.0	136.0
	IRT 2:1 1	48.9	27.9	0.0	156.0
	IRT 2:1 2	49.1	28.4	0.0	136.0
	IRT 2:1 3	48.7	28.2	0.0	136.0
	GreenSeeker 1	49.2	29.7	0.0	236.0
	GreenSeeker 2	49.4	30.2	0.0	235.0
	GreenSeeker 3	49.3	30.1	0.0	232.0
	Crop Circle 1	49.5	29.8	0.0	147.0
	Crop Circle 2	49.3	29.7	0.0	256.0
	Crop Circle 3	49.2	30.0	0.0	254.0
Trial 3	***Height Sensors***	49.3	29.4	0.0	215.0
	IRT 15:1 1	48.6	27.2	0.0	245.0
	IRT 15:1 2	48.7	30.8	0.0	244.0
	IRT 15:1 3	48.8	30.7	0.0	244.0
	IRT 2:1 1	48.7	31.0	0.0	256.0
	IRT 2:1 2	48.7	30.7	0.0	245.0
	IRT 2:1 3	48.8	30.9	0.0	245.0

Effects of Ambient Light Intensity and Ambient Temperature

If changes in ambient light intensity and ambient temperature significantly affect sensor readings, then phenotyping data will be unreliable because differences between plant varieties and cultivars may be undetected or exaggerated. The effect can be considered significant if it alters phenotypic trait conclusions. For example, consider a hypothetical situation where measured NDVI increases by 0.1 due to a change in light intensity from x_1 to x_2, and two varieties are present, v_1 and v_2, with a measured NDVI of 0.65 and 0.6, respectively, at x_1. If v_1 is measured at x_1, but v_2 is measured at x_2, measured NDVIs will be 0.65 and 0.7 for v_1 and v_2, respectively. Therefore, the influence of changes in light intensity on NDVI readings would be significant because it changes the conclusion. Now v_2 is "greener" than v_1. Consider a similar scenario where NDVI only increases by 0.01. Therefore, NDVI readings would be 0.65 and 0.61 for v_1 and v_2, respectively and the influence of changes in light intensity on NDVI readings may not be considered significant. Thus, determination of the relationship between sensor readings and influencing environmental factors is essential.

GreenSeeker

As shown in Figure 5.9 and Table 5.9, results for the second verification test indicate that statistically ambient light intensity significantly affects GreenSeeker NDVI readings, but whether it is significant from an engineering or scientific perspective depends on the specific application. (Kim et al. 2010) found no significant change in NDVI response due to changes in artificial illumination, but did observe that as solar radiation increased, NDVI response was reduced, with a maximum deflection of 0.08 due to a change in light intensity of approximately 150,000 lux. This finding is consistent with observed data in this study. Figure 5.9 was generated using R. The top graph is a plot of NDVI data with respect to light intensity and the fitted line. The bottom left graph is the standard residuals vs fitted plot, indicating whether a simple linear regression model is appropriate. If the data is randomly scattered about the fitted line and the data forms a horizontal band around the fitted line, a simple linear regression model is appropriate. The bottom right graph is a standard normal quantile-quantile (Q-Q plot) used to check whether the assumption that data is normally distributed is true. If data in the Q-Q plot forms a straight line, then the data is normally distributed. The fitted line in Figure 5.9 indicates

an inversely proportional relationship between NDVI with a slope of -4.58e-07 NDVI·lux^{-1} and an R^2 of 0.4372. Since the p-value is less than an alpha of 0.05, the slope is statistically significant at 95% confidence. Residual plots in Figure 5.9 indicate that this conclusion is reliable. Therefore, if phenotyping is conducted during the brightest part of the day (100,000 lux) and at night (0 lux), the difference between the data, as a result of the difference in light intensity, will be approximately 0.046. Typically, phenotyping is conducted between 9 AM and 3 PM and the light intensity may vary by 40,000 lux. Therefore, GreenSeeker NDVI measurements will vary by 0.018. Whether these differences of 0.046 or 0.018 are significant depends on the level of precision required for the specific application.

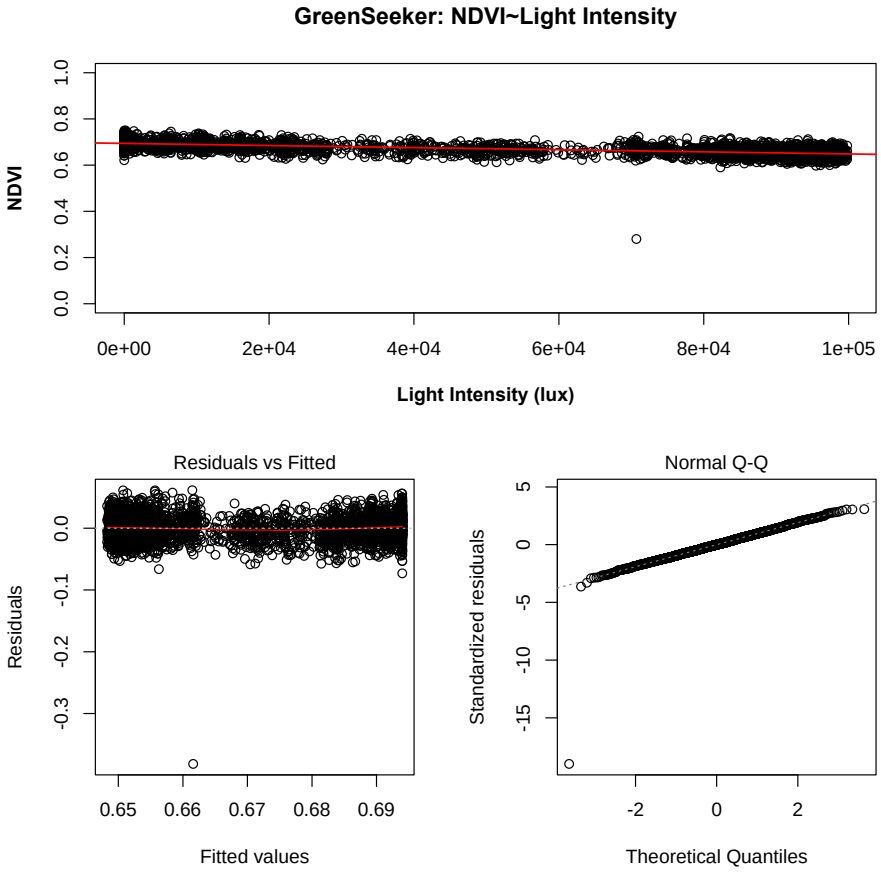

Figure 5.9 Fitted and residual plots of GreenSeeker data vs. ambient light intensity

Table 5.9 Summary statistics of GreenSeeker data vs. light intensity linear regression model

	Dependent variable: GreenSeeker NDVI			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	0.6940	0.0006	1217.45	<2e-16
LI	-4.581e-07	8.646e-09	-52.99	<2e-16
Observations	3,614			
R^2	0.4373545			
Adjusted R^2	0.4371987			
Residual Std. Error	0.020093170000 (df = 3612)			
F Statistic	$2{,}807.673^{***}$ (df = 1; 3612) (p = 0.00)			
Note:	*p **p ***p<0.01			

Statistically, the relationship between ambient temperature and GreenSeeker NDVI readings is also significant (Figure 5.10 and Table 5.10), but the R^2 value is 0.0097. Therefore, less than 1% of the variability in NDVI readings is explained by ambient temperature. The slope is approximately -0.00089 NDVI·°C^{-1}, so if the temperature varied by 20°C, the NDVI reading would only change by 0.0179.

Table 5.10 Summary statistics of Greenseeker NDVI vs. ambient temperature linear regression model

	Dependent variable: GreenSeeker NDVI			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	0.6826677	0.0022229	307.102	<2e-16
AmbTemp	-0.0008957	0.0001484	-6.037	1.73e-09
Observations	3,614			
R^2	0.010			
Adjusted R^2	0.010			
Residual Std. Error	0.027 (df = 3612)			
F Statistic	36.447^{***} (df = 1; 3612) (p = 0.000)			
Note:	*p **p ***p<0.01			

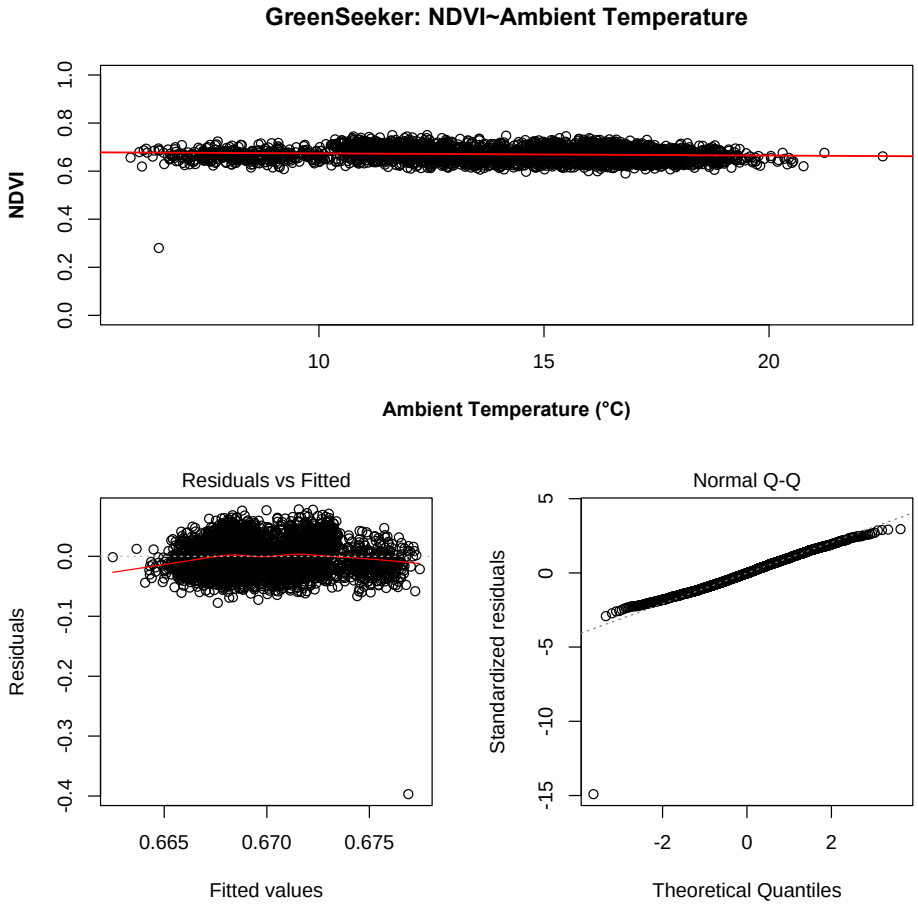

Figure 5.10 Fitted and residual plots of GreenSeeker NDVI data with respect to ambient temperature

Crop Circle

The Crop Circle data string contained five pieces of information: a user selectable vegetation index (VI), a second VI, Channel 1 reflectance (670 nm), Channel 2 reflectance (760 nm), and Channel 3 reflectance (550 nm). A statistically significant relationship exits between the readings from Channel 1 and ambient light intensity (Figure 5.11 and Table 5.11). However, the slope is only -1.351e-08 reflectance·lux^{-1}, meaning that even if phenotyping was conducted at 0 lux and 100,000 lux, the difference would only be 0.0013. Results were similar for the relationship with ambient temperature (Figure 5.12 and Table 5.12), which is also statistically significant, but the slope is only 4.167e-05 reflectance·°C^{-1}. Therefore, even if the temperature

changed by 20°C during a phenotyping run, change in reflectance would only be 0.0008. Results are similar for Channels 2 and 3 (Appendix B -).

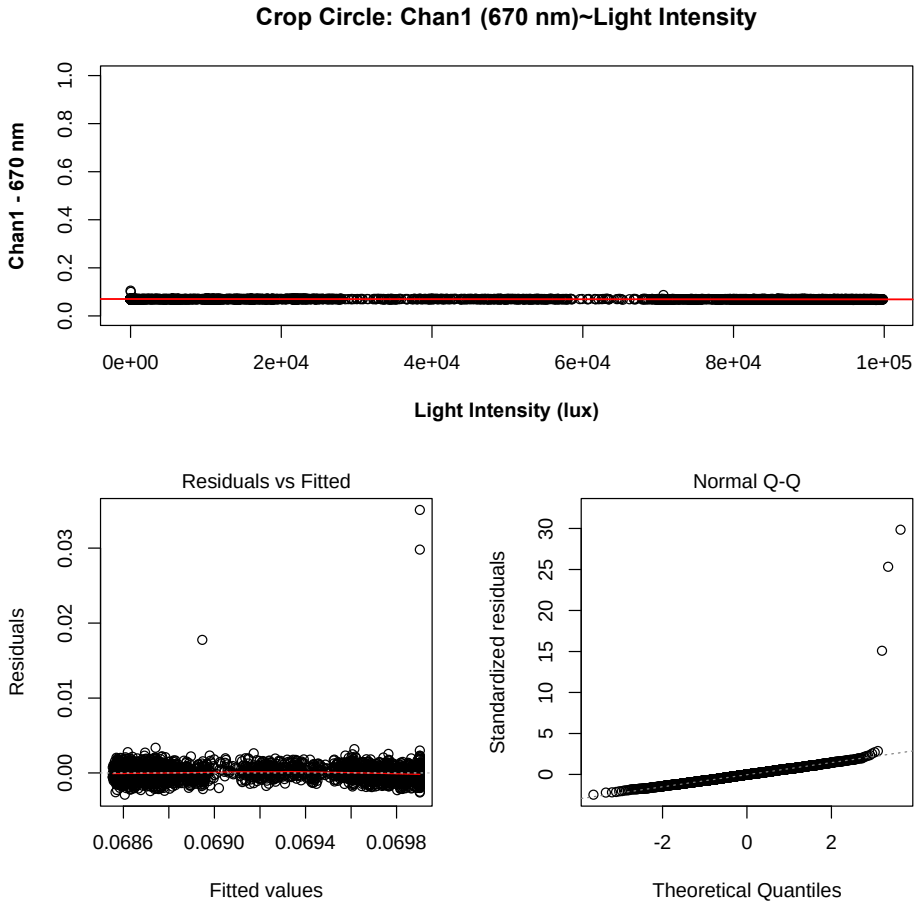

Figure 5.11 Fitted and residual plots of Crop Circle Channel 1 reflectance (670 nm) vs. light intensity

Table 5.11 Summary statistics of Crop Circle Channel 1 reflectance vs. light intensity linear regression model

	Dependent variable: Crop Circle Channel 1			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	6.990e-02	3.338e-05	2094.3	<2e-16
LI	-1.351e-08	5.062e-10	-26.7	<2e-16
Observations	3,614			
R^2	0.165			
Adjusted R^2	0.165			
Residual Std. Error	0.001 (df = 3612)			
F Statistic	712.667^{***} (df = 1; 3612) (p = 0.000)			
Note:	*p **p ***p<0.01			

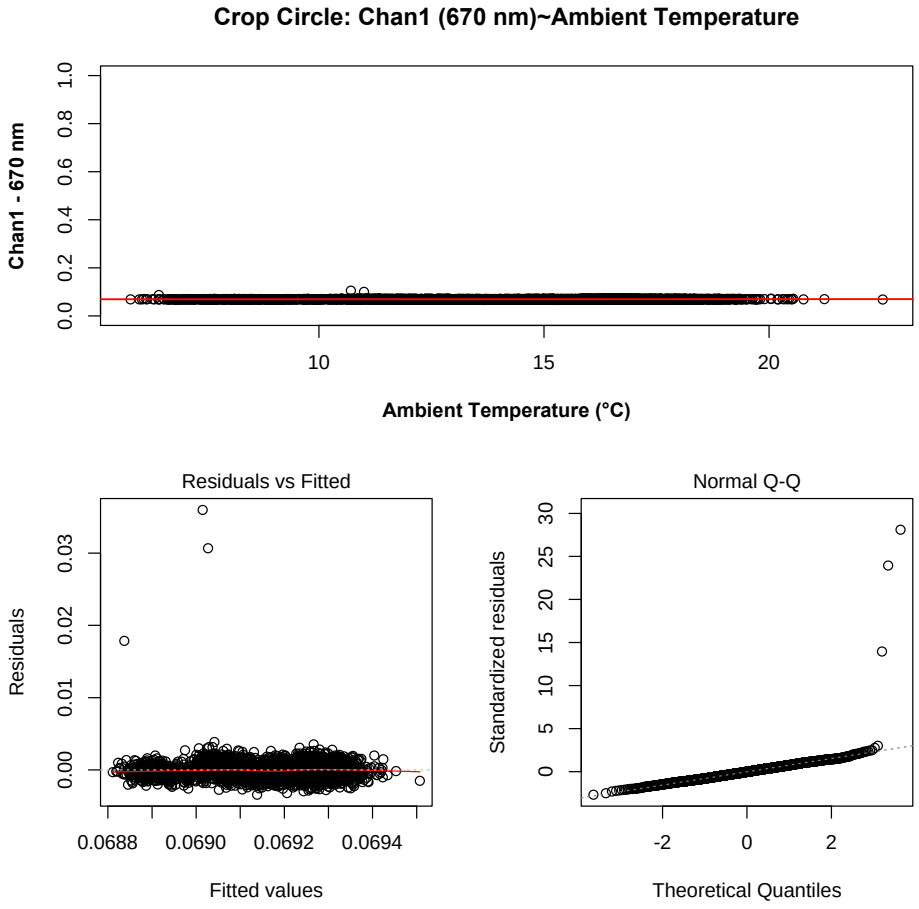

Figure 5.12 Fitted and residual plots of Crop Circle Channel 1 reflectance (670 nm) vs. ambient temperature

Table 5.12 Summary statistics of Crop Circle Channel 1 reflectance vs. ambient temperature linear regression model

	Dependent variable: Crop Circle Channel 1			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	6.857e-02	1.069e-04	641.660	<2e-16
AmbTemp	4.167e-05	7.132e-06	5.843	5.59e-09
Observations	3,614			
R^2	0.009			
Adjusted R^2	0.009			
Residual Std. Error	0.001 (df = 3612)			
F Statistic	34.139^{***} (df = 1; 3612) (p = 0.000)			
Note:	*p **p ***p<0.01			

The first VI, or the NDVI calculated using the reflectance of the 670 nm (red) and the 760 nm (near infrared) wavelengths, is not affected by ambient light intensity (Figure 5.13 and Table 5.13). Statistically, it is affected, however, by ambient temperature (Figure 5.14 and Table 5.14), with a slope of 0.000455 NDVI·°C^{-1}. Even if the temperature changes by 20°C, the NDVI would only change by 0.009, which is insignificant from a practical standpoint. Crop Circle NDVI readings cannot be directly compared to GreenSeeker NDVI (656 nm, 774 nm) because different wavelengths were used. The second Crop Circle VI was not analyzed.

Table 5.13 Summary statistics of Crop Circle NDVI (670 nm, 760 nm) vs. light intensity linear regression model

	Dependent variable: Crop Circle NDVI			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	4.863e-01	2.541e-04	1914.069	<2e-16
LI	-7.065e-09	3.854e-09	-1.833	0.0669
Observations	3,614			
R^2	0.001			
Adjusted R^2	0.001			
Residual Std. Error	0.009 (df = 3612)			
F Statistic	3.360^{*} (df = 1; 3612) (p = 0.067)			
Note:	*p **p ***p<0.01			

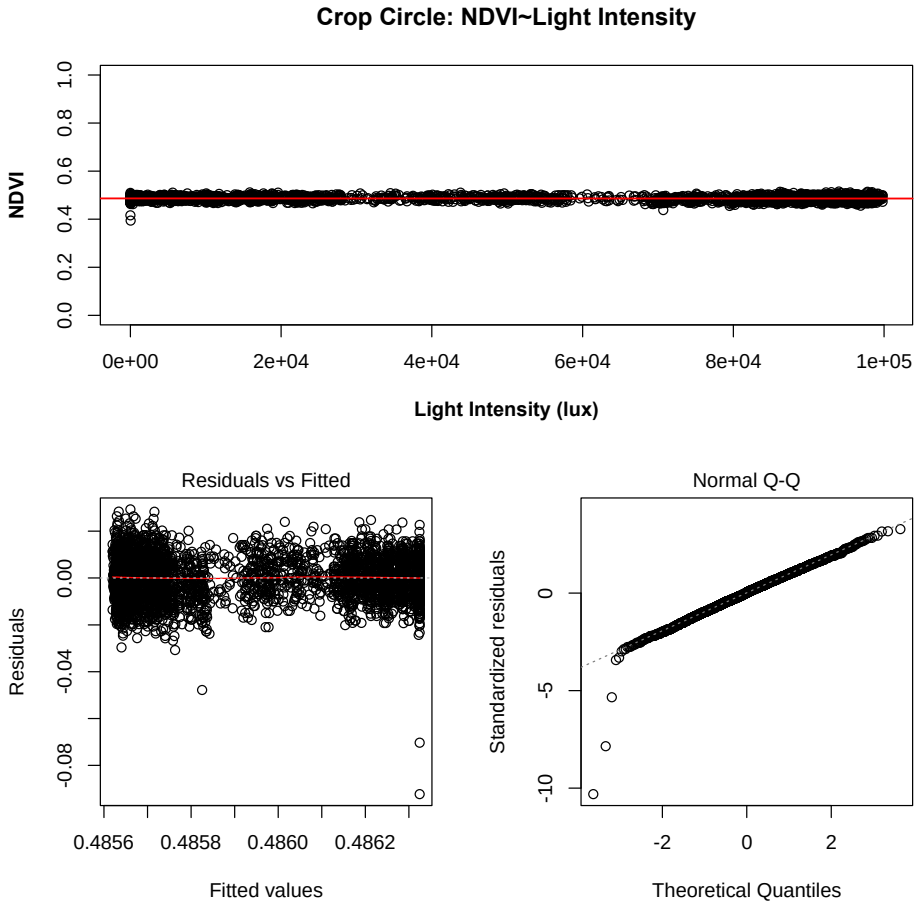

Figure 5.13 Fitted and residual plots of Crop Circle NDVI (670 nm, 760 nm) vs. light intensity

Table 5.14 Summary statistics of Crop Circle NDVI (670 nm, 760 nm) vs. ambient temperature linear regression model

	Dependent variable: Crop Circle NDVI			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	0.4792664	0.0007386	648.861	<2e-16
AmbTemp	0.0004550	0.0000493	9.231	<2e-16
Observations	3,614			
R^2	0.023			
Adjusted R^2	0.023			
Residual Std. Error	0.009 (df = 3612)			
F Statistic	85.206^{***} (df = 1; 3612) (p = 0.000)			
Note:	*p **p ***p<0.01			

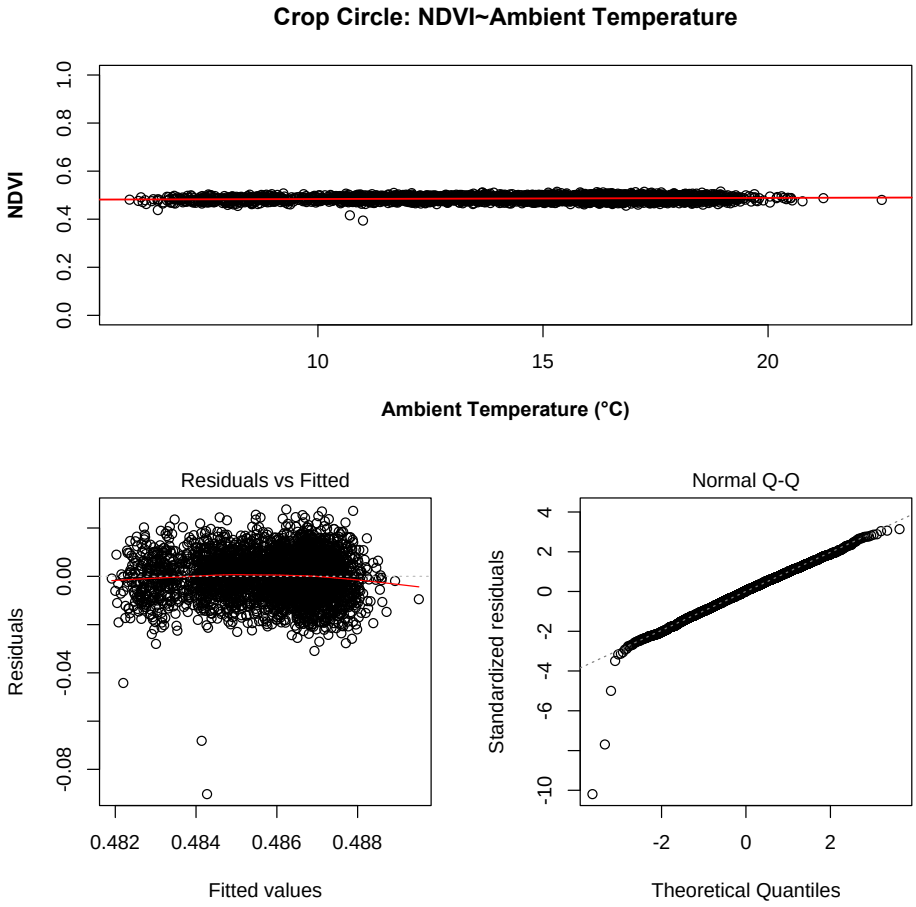

Figure 5.14 Fitted and residual plots of Crop Circle NDVI (670 nm, 760 nm) vs. ambient temperature

Ultrasonic Sensor

Similar to GreenSeeker and Crop Circle, statistically, relationships between ultrasonic sensor distance readings and ambient light intensity (Figure 5.15 and Table 5.15) and ambient temperature (Figure 5.16 and Table 5.16) are significant. The slope with respect to light intensity is 7.242e-06 cm·°C^{-1}. If the light intensity changes by 100,000 lux, the difference is 0.74 cm; however, the R^2 value is only 0.049; therefore, only 4.9 % of any change in the distance sensor reading can be explained by light intensity. The slope with respect to ambient temperature is 6.275e-02 cm·°C^{-1}. If the temperature changes by 20°C, the difference is 1.25 cm.

Table 5.15 Summary statistics of ultrasonic sensor distance vs. light intensity linear regression model

	Dependent variable: Ultrasonic Distance			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	1.034e+02	3.496e-02	2956.62	<2e-16
LI	7.242e-06	5.303e-07	13.65	<2e-16
Observations	3,614			
R^2	0.049			
Adjusted R^2	0.049			
Residual Std. Error	1.232 (df = 3612)			
F Statistic	186.469^{***} (df = 1; 3612) (p = 0.000)			
Note:	*p **p ***p<0.01			

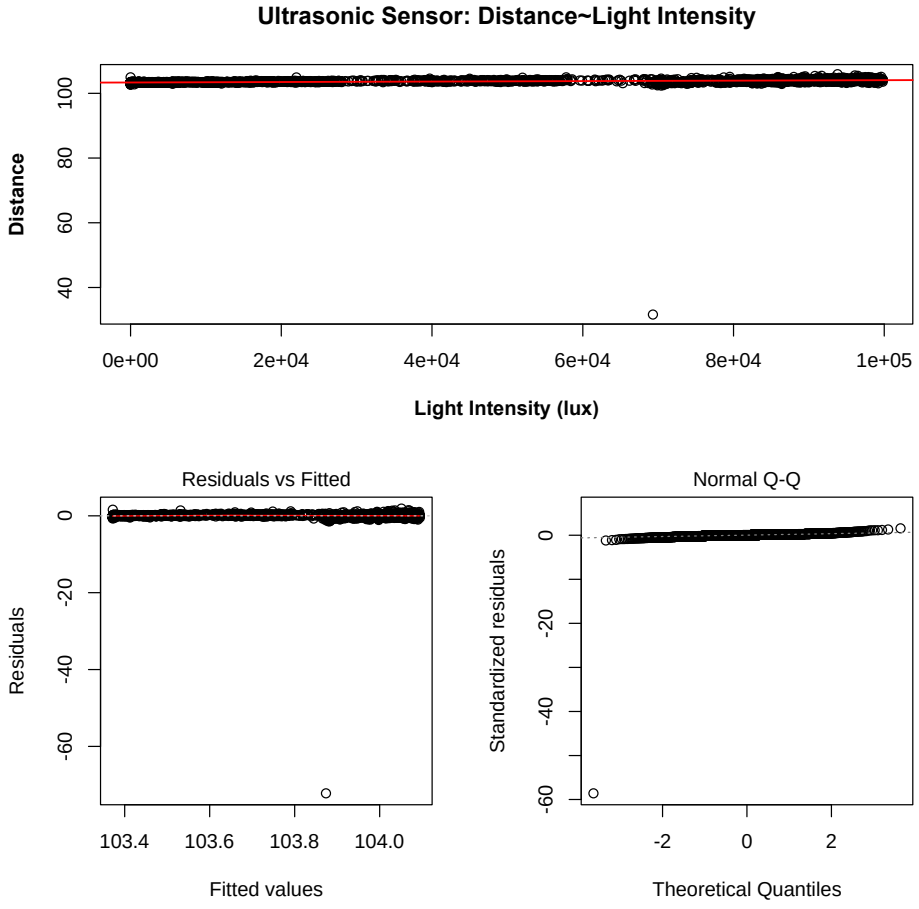

Figure 5.15 Fitted and residual plots of ultrasonic sensor data with respect to ambient light intensity

Table 5.16 Summary statistics of ultrasonic distance sensor vs. ambient temperature linear regression model

	Dependent variable: Ultrasonic Distance			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	1.028e+02	1.042e-01	986.58	<2e-16
AmbTemp	6.275e-02	6.957e-03	9.02	<2e-16
Observations	3,614			
R^2	0.022			
Adjusted R^2	0.022			
Residual Std. Error	1.250 (df = 3612)			
F Statistic	81.364*** (df = 1; 3612) (p = 0.000)			
Note:	*p **p ***p<0.01			

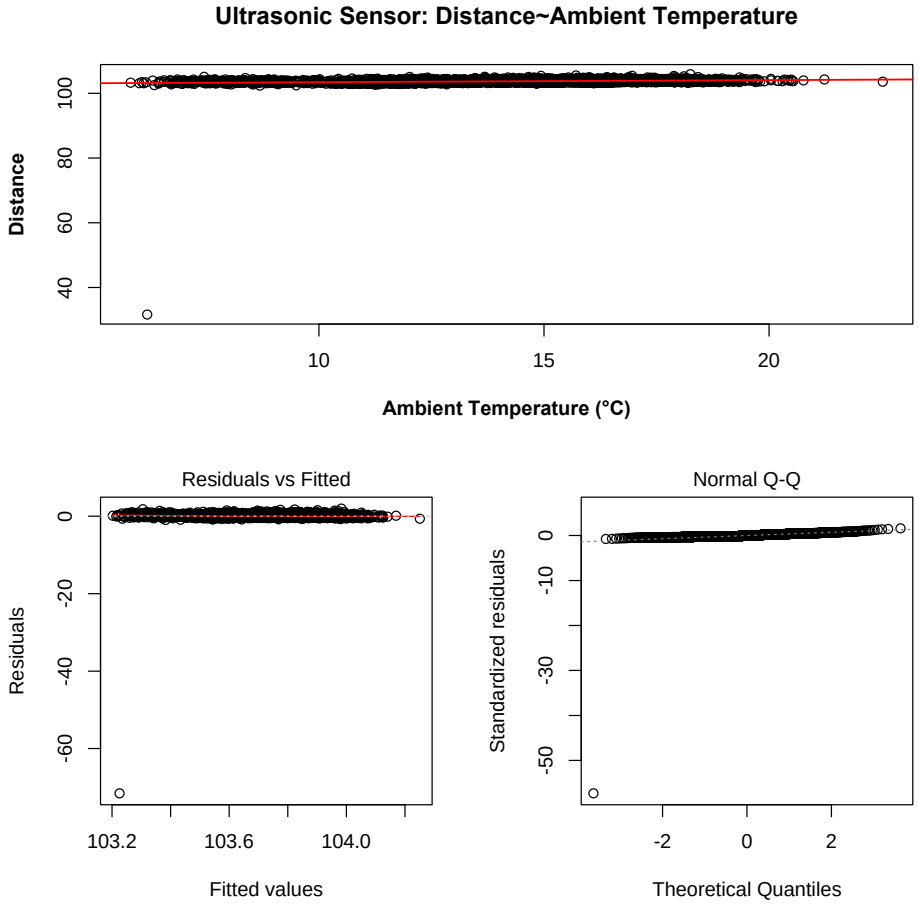

Figure 5.16 Fitted and residual plots of ultrasonic sensor data with respect to ambient temperature

Laser Sensor

Figure 5.17 indicates that ambient light intensity affects laser sensor readings. Above 20,000 lux, the laser sensor cannot consistently detect the reflected laser beam required to determine the distance. When this occurs, it defaults to the maximum distance limit set at 150 cm. A group of points with a distance of approximately 80 cm occurs at approximately 70,000 lux. These points can be ignored and are most likely a result of shadowing on just the spot where the laser was pointed. Shadowing could have occurred because the laser sensor was mounted directly on the toolbar, unlike the other sensors which were mounted away from the toolbar. The distance of this set of point is shorter than the other points, most likely because a brick was placed under the laser sensor during the first few hours of the experiment and then was later replaced with soil, the surface of which was farther from the sensor.

Linear regression with respect to ambient light intensity (Figure 5.17) and ambient temperature (Figure 5.18) was done on a subset of the laser sensor data. Distance data over 100 cm and below 80 cm were removed. Regression results (Table 5.17) indicate that no statistically significant relationship exists between the measured distance and light intensity since the p-value (0.673) is greater than 0.05. Similar to readings from the other sensors, a statistically significant relationship exists between the measured distance and ambient temperature (Table 5.18). The slope is 0.05327 $cm \cdot °C^{-1}$. A change of 20°C would result in a change in the distance measurement of approximately 1 cm. For phenotyping, this is insignificant because even a small dip or a clumped piece of soil along the wheel track will introduce a larger error.

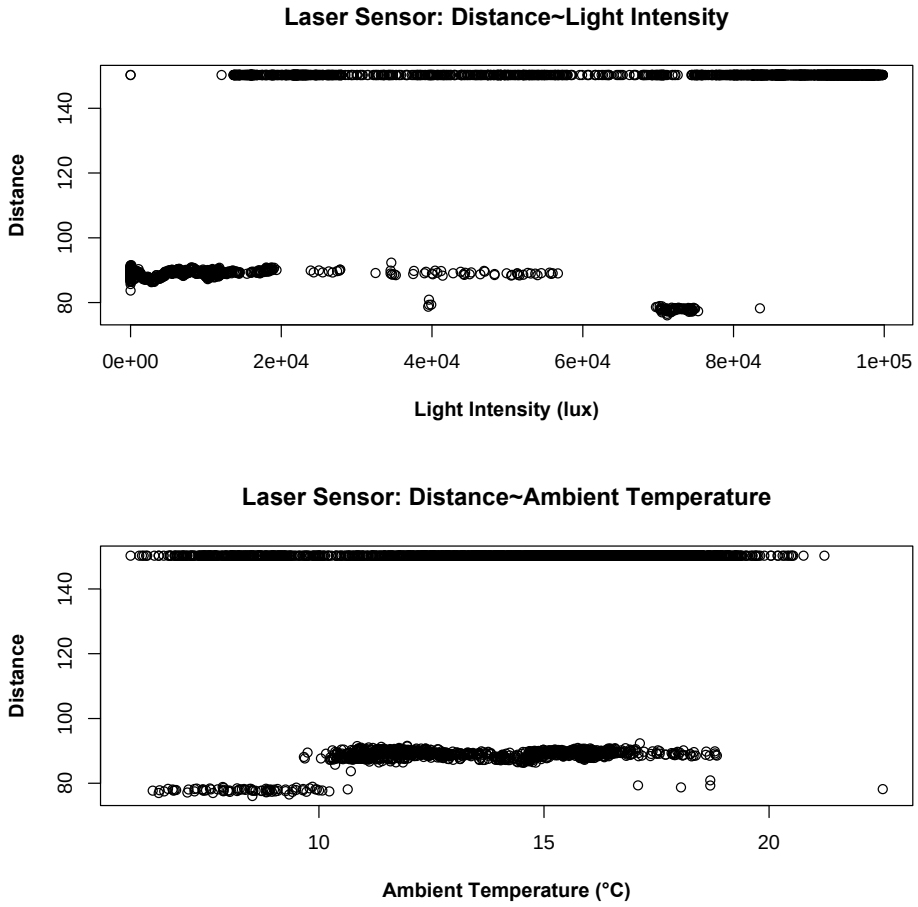

Figure 5.17 Plots of raw laser sensor data vs. light intensity and ambient temperature

Table 5.17 Summary statistics of laser sensor distance vs. light intensity linear regression model

Laser Sensor	*Dependent variable: Laser Sensor Distance*			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	8.898e+01	4.026e-02	2210.190	<2e-16
LI	1.384e-06	3.275e-06	0.423	0.673
Observations	1,102			
R^2	0.0002			
Adjusted R^2	-0.001			
Residual Std. Error	1.092 (df = 1100)			
F Statistic	0.179 (df = 1; 1100) (p = 0.673)			
Note:	*p **p ***p<0.01			

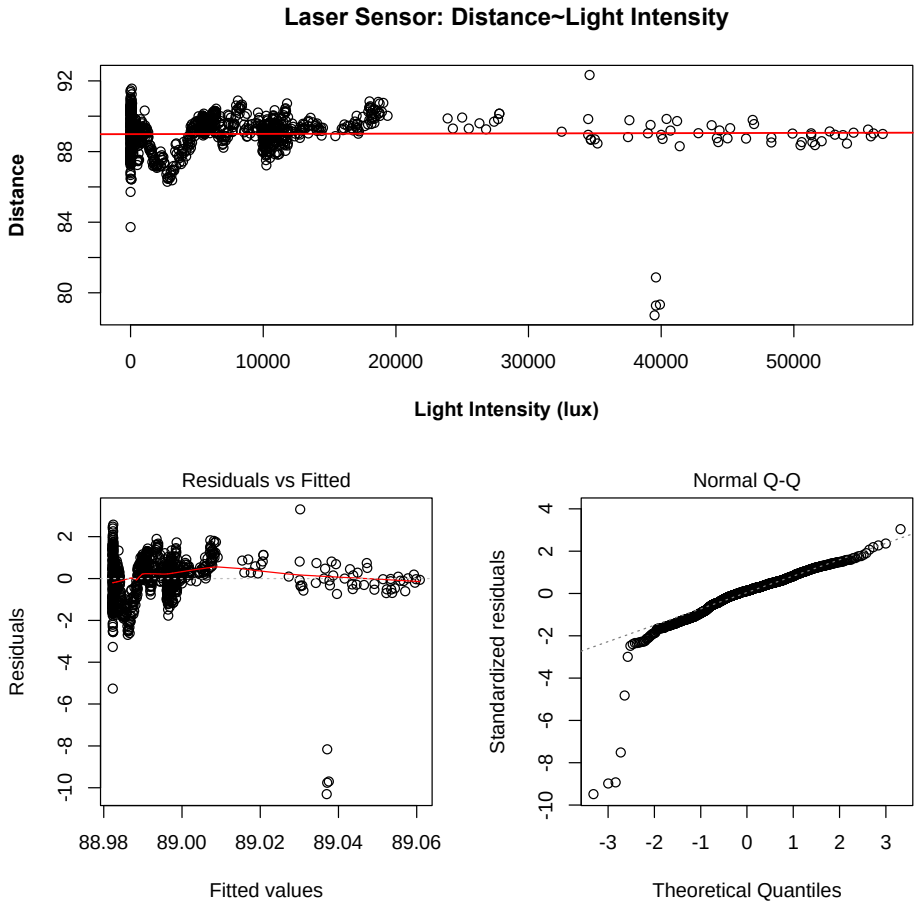

Figure 5.18 Fitted and residual plots of laser sensor data vs. ambient light intensity

Table 5.18 Summary statistics of laser sensor distance vs. light intensity linear regression model

Laser Sensor	*Dependent variable: Laser Sensor Distance*			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	88.23728	0.21804	404.684	<2e-16
AmbTemp	0.05327	0.01521	3.502	0.000481
Observations	1,102			
R^2	0.011			
Adjusted R^2	0.010			
Residual Std. Error	1.086 (df = 1100)			
F Statistic	12.261^{***} (df = 1; 1100) (p = 0.0005)			
Note:	*p **p ***p<0.01			

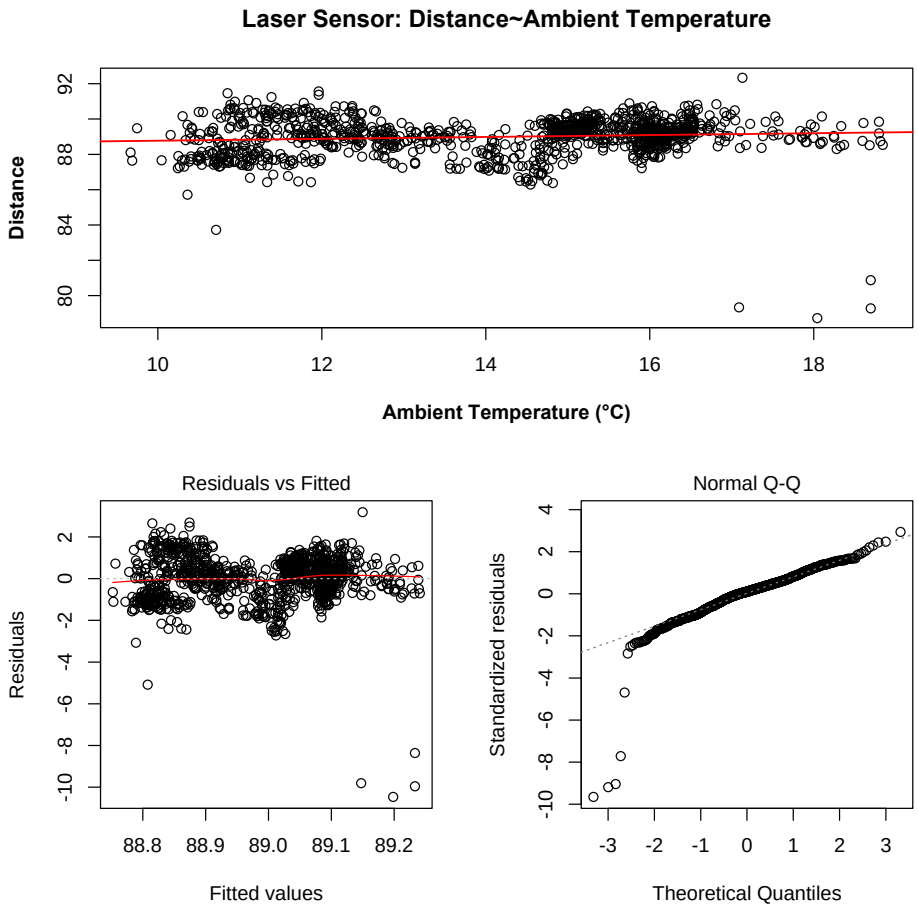

Figure 5.19 Fitted and residual plots of laser sensor data with respect to ambient temperature

IRT

Since surface temperature is directly proportional to light intensity, the difference (ΔT) between temperature measurements from thermocouple sensors and IRT sensors was used to analyze whether light intensity affects IRT measurements, assuming that thermocouples are not sensitive to changes in light intensity. Figure 5.20 indicates that a linear trend exists between ΔT and ambient light intensity. This trend is statistically significant (Table 5.19). As light intensity increases, the difference between measurements from IRT and the thermocouple increases, indicating that light intensity affects IRT measurements if temperature measurements from the thermocouples are accurate. The R^2 of the model is 0.948, meaning that 94.8% of the variability

in ΔT can be explained by changes in light intensity. The slope of the relationship is 9.453e-05 °C·lux^{-1}; therefore, a change in light intensity of 100,000 lux will result in a ΔT of 9.4°C.

Figure 5.21 indicates no linear trend between IRT ΔTs and ambient temperature; however, this does not mean that ambient temperature has no effect on IRT measurements. Based on this data, it is difficult to conclude what the effect is.

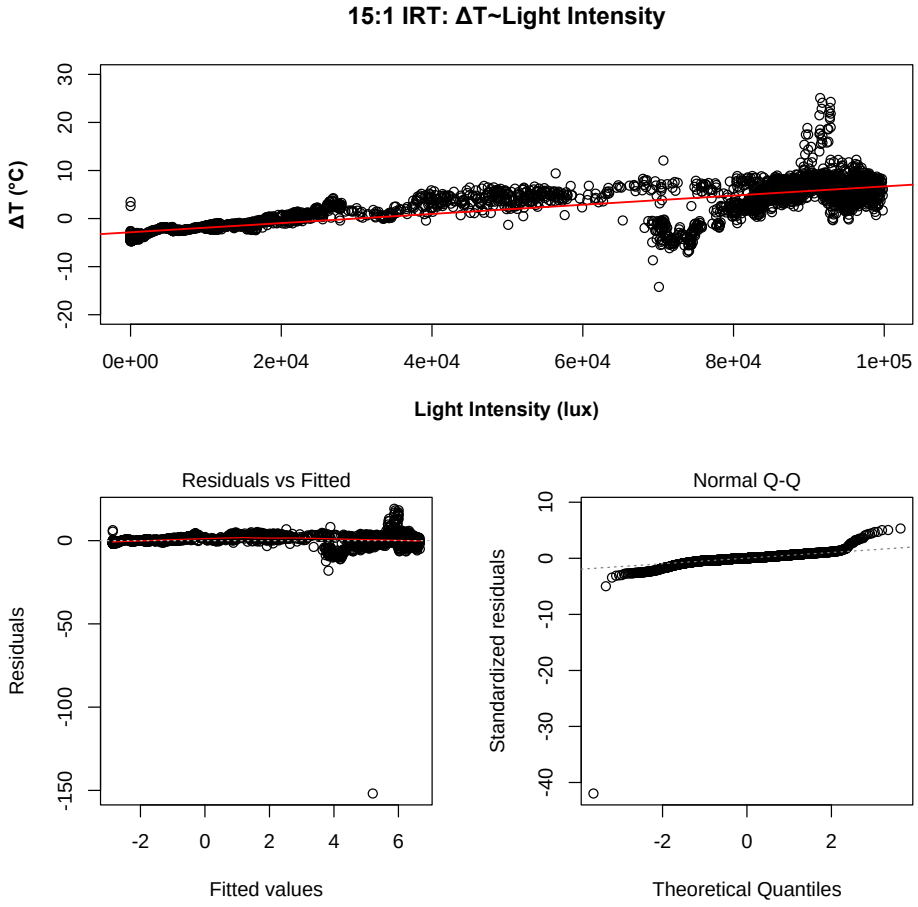

Figure 5.20 Fitted and residual plots of 15:1 IRT sensor data vs. ambient light intensity

Table 5.19 Summary statistics of 15:1 IRT ΔT vs. light intensity linear regression model

15:1 IRT	*Dependent variable: 15:1 IRT ΔT*			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	-2.860	1.027e-01	-27.85	<2e-16
LI	9.543e-05	1.558e-06	61.27	<2e-16
Observations	3,614			
R^2	0.510			
Adjusted R^2	0.509			
Residual Std. Error	3.620 (df = 3612)			
F Statistic	$3{,}753.743^{***}$ (df = 1; 3612) (p = 0.000)			
Note:	*p **p ***p<0.01			

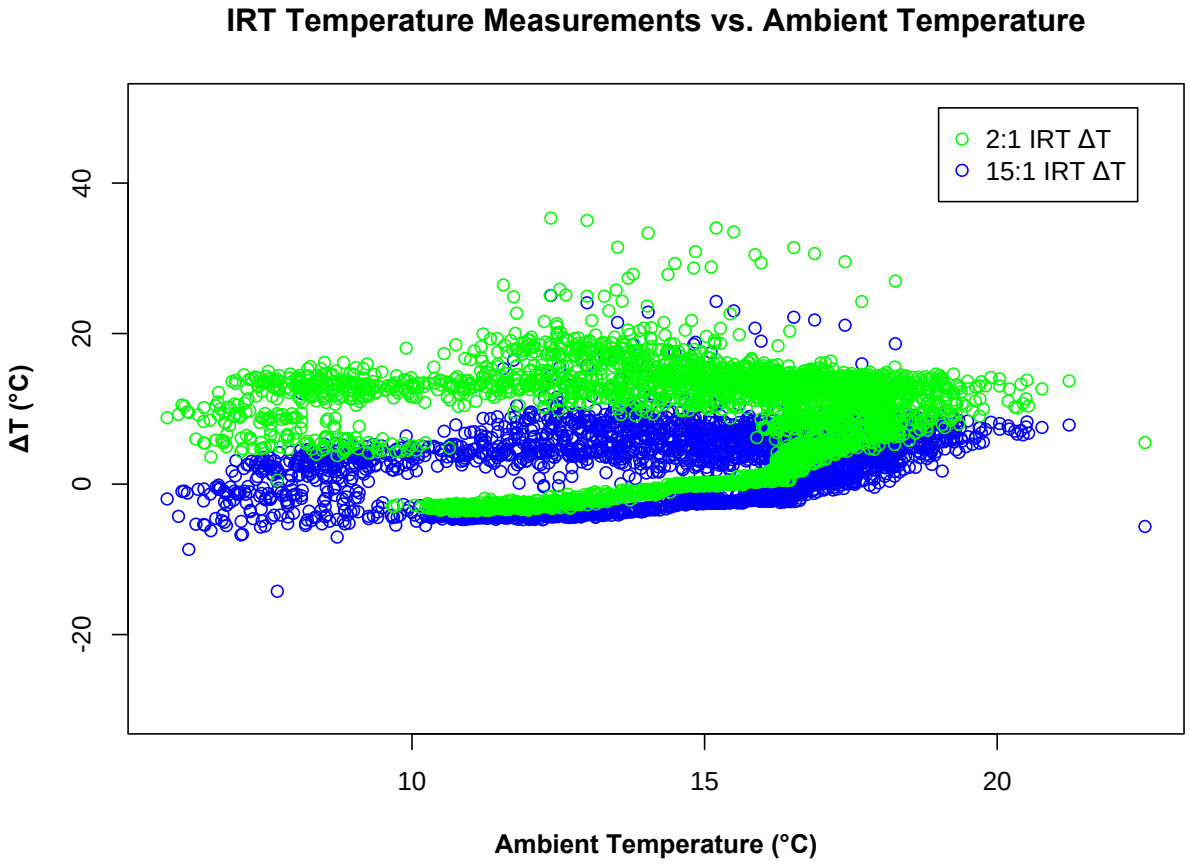

Figure 5.21 Plot of 2:1 and 15:1 IRT with respect to ambient temperature

Results for the 2:1 IRT (Figure 5.22 and Table 5.20) indicate that light intensity has a greater effect on it compared to 15:1 IRT. The slope of the fitted linear curve is larger (1.630e-04 °C·lux-1), resulting in 16.3°C change in the reading for a 100,000 lux change in light intensity. This also indicates that 2:1 IRT has larger temperature readings as light intensity increases (Figure 5.23). The relationship of 2:1 IRT with respect to ambient temperature is similar to the relationship of 15:1 IRT with respect to ambient temperature.

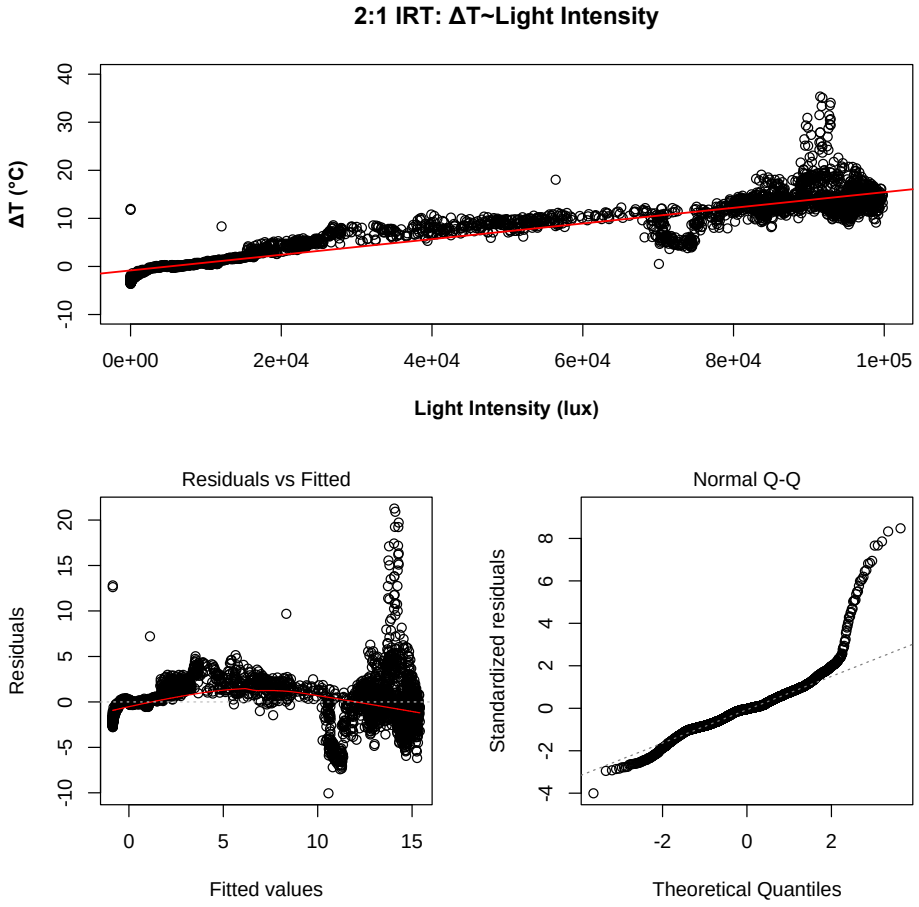

Figure 5.22 Fitted and residual plots of 2:1 IRT sensor data vs. ambient light intensity

Table 5.20 Summary statistics of 2:1 IRT ΔT vs. light intensity linear regression model

2:1 IRT	*Dependent variable: 2:1 IRT ΔT*			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	-8.567e-01	7.125e-02	-12.02	<2e-16
LI	1.630e-04	1.081e-06	150.86	<2e-16
Observations	3,614			
R^2	0.863			
Adjusted R^2	0.863			
Residual Std. Error	2.512 (df = 3612)			
F Statistic	22,759.690*** (df = 1; 3612) (p = 0.000)			
Note:	*p**p***p<0.01			

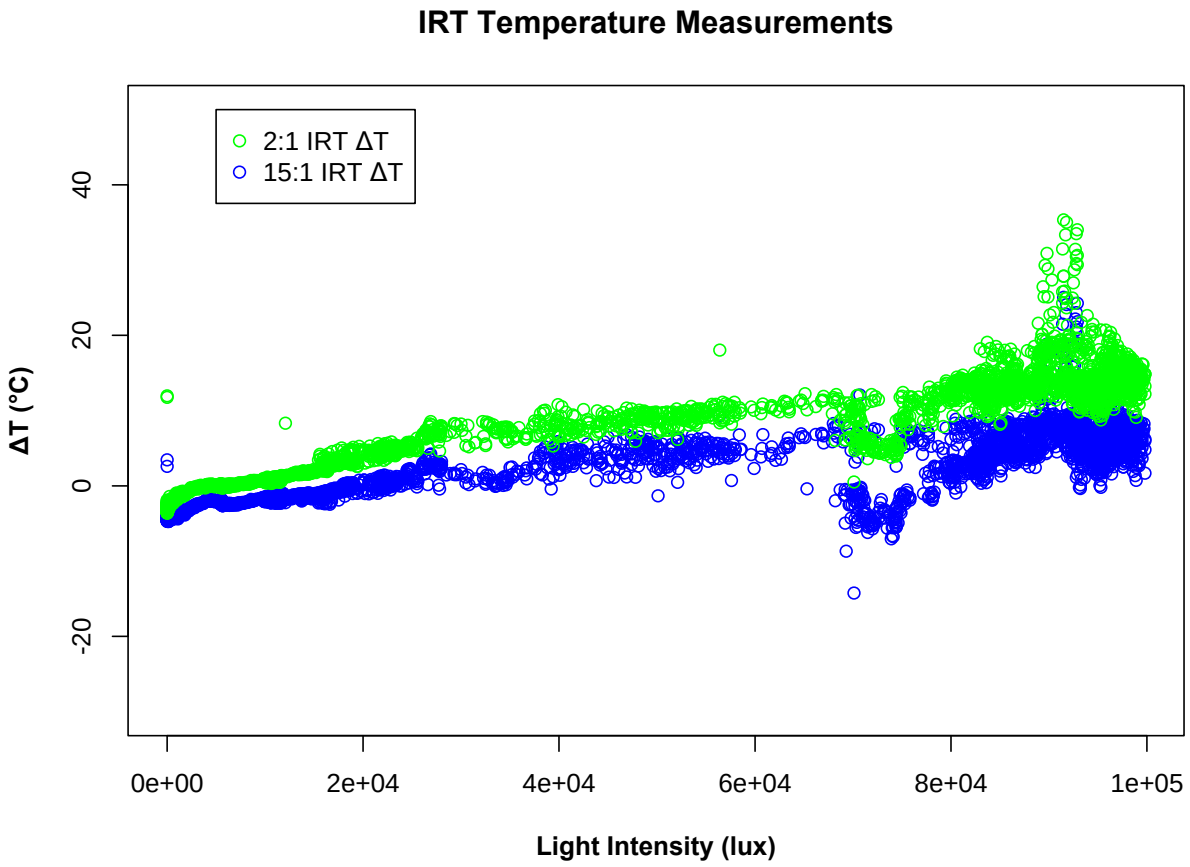

Figure 5.23 Comparison of 2:1 and 15:1 IRT temperature measurements with respect to ambient light intensity

Chapter 6 - Summary and Conclusions

A mobile field-based, high-throughput phenotyping platform was developed. A sensor platform consisting of three sensor sets was assembled and mounted on a high-clearance vehicle. Each sensor set was composed of two infrared thermometers with different field of views, one ultrasonic sensor, one ACS-470 Crop Circle, and one GreenSeeker. Two GNSS receivers and a laser sensor were also installed. The sensor sets were connected to a laptop computer through a set of hubs. This accomplished the first object of assembling the phenotyping platform.

Software to interface with the sensors, collect sensor and position data, and to georeference sensor data was developed using National Instruments LabVIEW. The main program was composed of several subVIs or modules developed to interface with each type of sensor. This accomplished the second objective.

A system verification test was conducted to ensure that the developed phenotyping platform was capable of timely collecting sensor and position data. This test logged data timestamps for each sensor and position data point. The timestamps were used to determine the average, minimum, and maximum periods of incoming position data. The determination was made that, on average, phenotyping software is able to receive sensor and position data at the rate it is generated. Maximum and minimum periods indicated that, on occasion, system delays occur that result in consecutive data points being buffered, but this does not occur frequently and should not significantly affect system performance. This accomplished the third objective.

A second test was conducted to determine the effects of ambient light intensity and ambient temperature on sensor readings. Test results indicated a statistically significant relationship between sensor readings from GreenSeekers, Crop Circles, and ultrasonic sensors and ambient light intensity and ambient temperature. Whether or not the relationship is significant from an engineering or a scientific perspective should be evaluated based on the specific application of the sensors. For the laser sensors, results indicated that when the sensor is aimed at bare soil, it cannot consistently measure distance at light intensities above 20,000 lux. The 2:1 and 15:1 IRT have statistically significant relationships with ambient light intensity. A 100,00 lux difference in light intensity results in a change of 9.4°C and 16.3°C for the 15:1 and 2:1 IRTs, respectively. The surface temperature measurement error, relative to measurements from a thermocouple, increases with increasing light intensity. The rate of increase is greater for

the 2:1 IRT; therefore, as light intensity increases, the difference between readings from the 2:1 IRT and 15:1 IRT increases. From the data collected, no conclusions could be made regarding the relationship between ambient temperature and IRT readings. Future studies should conduct a controlled test in which surface temperature and environmental factors, with the exception of ambient temperature, remain constant. The light intensity and ambient temperature test accomplished the fourth and last objective.

References

Andrade-Sanchez, Pedro, Michael A Gore, John T Heun, Kelly R Thorp, A Elizabete Carmo-Silva, Andrew N French, Michael E Salvucci, and Jeffrey W White. 2013. "Development and Evaluation of a Field-Based High-Throughput Phenotyping Platform." *Functional Plant Biology* 41 (1). CSIRO PUBLISHING: 68–79. doi:10.1071/FP13126.

Araus, José Luis, and Jill E Cairns. 2013. "Field High-Throughput Phenotyping: the New Crop Breeding Frontier." *Trends in Plant Science*. doi:10.1016/j.tplants.2013.09.008.

Araus, José Luis, Gustavo A Slafer, Conxita Royo, and M Dolores Serret. 2008. "Breeding for Yield Potential and Stress Adaptation in Cereals." *Dx.Doi.org* 27 (6). Taylor & Francis Group: 377–412. doi:10.1080/07352680802467736.

Banner Engineering. 2005. "LT3 Long-Range Time-of-Flight Sensor." Minneapolis, MN U.S.A.

Benson, Tom, ed. 2014. "Speed of Sound." *Nasa*. Accessed January 29. https://www.grc.nasa.gov/www/k-12/airplane/sound.html.

Berger, Bettina, Bas de Regt, and Mark Tester. 2013. "Applications of High-Throughput Plant Phenotyping to Study Nutrient Use Efficiency." *Plant Mineral Nutrients* (Chapter 18). Totowa, NJ: Humana Press: 277–90. doi:10.1007/978-1-62703-152-3_18.

Busemeyer, L, D Mentrup, K Möller, E Wunder, and K Alheit. 2013. "BreedVision—a Multi-Sensor Platform for Non-Destructive Field-Based Phenotyping in Plant Breeding." *Sensors*.

Cabrera Bosquet, Llorenç, José Crossa, Jarislav von Zitzewitz, María Dolors Serret, and José Luis Araus. 2012. "High- Throughput Phenotyping and Genomic Selection: the Frontiers of Crop Breeding ConvergeF." *Journal of Integrative Plant Biology* 54 (5). Blackwell Publishing Asia: 312–20. doi:10.1111/j.1744-7909.2012.01116.x.

Cobb, Joshua N, Genevieve DeClerck, Anthony Greenberg, Randy Clark, and Susan McCouch. 2013. "Next-Generation Phenotyping: Requirements and Strategies for Enhancing Our Understanding of Genotype–Phenotype Relationships and Its Relevance to Crop Improvement." *Theoretical and Applied Genetics* 126 (4). Springer-Verlag: 867–87. doi:10.1007/s00122-013-2066-0.

Daigle, G A, J E Piercy, and T F W Embleton. 1978. "Effects of Atmospheric Turbulence on the Interference of Sound Waves Near a Hard Boundary." *The Journal of the Acoustical Society of America* 64 (2). Acoustical Society of America: 622–30. doi:10.1121/1.381998.

Fiorani, Fabio, and Ulrich Schurr. 2013. "Future Scenarios for Plant Phenotyping." *Dx.Doi.org*. Annual Reviews. doi:10.1146/annurev-arplant-050312-120137.

Furbank, Robert T, and Mark Tester. 2011. “Phenomics--Technologies to Relieve the Phenotyping Bottleneck..” *Trends in Plant Science* 16 (12): 635–44. doi:10.1016/j.tplants.2011.09.005.

Ingvarsson, Pär K, and Nathaniel R Street. 2011. “Association Genetics of Complex Traits in Plants.” *New Phytologist* 189 (4). Blackwell Publishing Ltd: 909–22. doi:10.1111/j.1469-8137.2010.03593.x.

Kaplan, Elliott D, and Christopher J Hegarty. 2005. *Understanding GPS*. Artech House.

Kim, Y, D M Glenn, J Park, H K Ngugi, and B L LEHMAN. 2010. “Active Spectral Sensor Evaluation Under Varying Condition.” *ASABE Paper*.

Kipp, S, B Mistele, P Baresel, and U Schmidhalter. 2013. “High-Throughput Phenotyping Early Plant Vigour of Winter Wheat.” *European Journal of …*.

Lan, Yubin. 2009. “Development of an Integration Sensor and Instrumentation System for Measuring Crop Conditions.” *Agricultural Engineering International: CIGR Journal* 0 (0).

Mendelson, Kenneth S. 2006. “The Story of C.” *American Journal of Physics* 74 (11): 995. doi:10.1119/1.2238887.

MICRO-EPSILON. 2013. *Basics of Non Contact Temperature Measurement*.

Mogel, von, K H. 2013. “Phenomics Revolution.”

Montes, J M, A E Melchinger, and J C Reif. 2007a. “Novel Throughput Phenotyping Platforms in Plant Genetic Studies.” *Trends in Plant Science*.

Montes, Juan M, Albrecht E Melchinger, and Jochen C Reif. 2007b. “Novel Throughput Phenotyping Platforms in Plant Genetic Studies.” *Trends in Plant Science* 12 (10): 433–36. doi:10.1016/j.tplants.2007.08.006.

NovAtel. 2014. “An Introduction to GNSS | Chapter 1 - GNSS Overview | NovAtel.” *NovAtel*. Accessed March 4. http://www.novatel.com/an-introduction-to-gnss/chapter-1-gnss-overview/.

Pieruschka, R, and H Poorter. 2012. “Phenotyping Plants: Genes, Phenes and Machines.” *Functional Plant Biology*.

Randall Smith, MicroImages, Inc. 2001. “Hyperspectral Imaging”: 1–24.

Stephens, S Chod, and V Philip Rasmussen. “High-End DGPS and RTK Systems.”

Thomas, Howard, and Helen Ougham. 2014. “The Stay-Green Trait..” *Journal of Experimental Botany*. Oxford University Press: eru037. doi:10.1093/jxb/eru037.

Varshney, Rajeev K, Spurthi N Nayak, Gregory D May, and Scott A Jackson. 2009. “Next-Generation Sequencing Technologies and Their Implications for Crop Genetics and Breeding.” *Trends in Biotechnology* 27 (9): 522–30. doi:10.1016/j.tibtech.2009.05.006.

White, Jeffrey W, and Matthew M Conley. 2013. “A Flexible, Low-Cost Cart for Proximal Sensing.” *Crop Science* 53 (4). The Crop Science Society of America, Inc.: 1646–49. doi:10.2135/cropsci2013.01.0054.

White, Jeffrey W, Pedro Andrade-Sanchez, Michael A Gore, Kevin F Bronson, Terry A Coffelt, Matthew M Conley, Kenneth A Feldmann, et al. 2012. “Field-Based Phenomics for Plant Genetics Research.” *Field Crops Research* 133: 101–12. doi:10.1016/j.fcr.2012.04.003.

Wong, George S K. 1986. “Speed of Sound in Standard Air.” *The Journal of the Acoustical Society of America* 79 (5). Acoustical Society of America: 1359–66. doi:10.1121/1.393664.

Yang, Wanneng, Lingfeng Duan, Guoxing Chen, Lizhong Xiong, and Qian Liu. 2013. “Plant Phenomics and High-Throughput Phenotyping: Accelerating Rice Functional Genomics Using Multidisciplinary Technologies.” *Current Opinion in Plant Biology* 16 (2): 180–87. doi:10.1016/j.pbi.2013.03.005.

2008. “AgGPS® 432/442 GPS Receivers.” Trimble Navigation Limited.

2012. “Physiological Breeding II: a Field Guide to Wheat Phenotyping.”

2013. “NI-Datasheet-Ds-187”: 1–7.

Appendix A - R Programs for Statistical Analysis

GreenSeeker

```
#This program processes the GreenSeeker Data
#
#sensorData - table where the raw sensor data is stored
#          the GNSS/GPS coordinates are not included in this table
#sensorReadPeriods - data frame where the calculated time periods are stored
#          Column Names: R1, R2, R3, WT, R1GSTS, R2GSTS, R3GSTS
#          ... R1 is the column name for the read/timestamp periods of the row 1
 GreenSeeker
#summaryTable - summary of the period statiscs for each sensor
#          Column Names Ave. Period, Ave. Freq, SD Period, Min, Max


#set working directory
setwd("directory")
filename = '2014-03-21_1240_GreenSeekerData.txt'
sensorData = read.table(file=filename, header=T, sep = '\t', row.names = "Index",
 skip = 2)

#Clean up the data in sensorData - remove extra columns
colsToDelete = c('Left_UTCTime','Left_Elevation',
 'Left_Latitude','Left_Longitude','Right_UTCTime','Right_Elevation',
 'Right_Latitude', 'Right_Longitude')
colsToDelete = !(names(sensorData) %in% colsToDelete)
sensorData = sensorData[,colsToDelete]  #columns to delete


########################################
########## Calculate Periods ############
########################################

#column names for sensorReadPeriods
columnNames = c('R1','R2','R3','WT','R1GSTS', 'R2GSTS', 'R3GSTS',
 'R1TDif','R2TDif','R3TDif')
#create sensorReadPeriods data frame
sensorReadPeriods = data.frame(matrix(nrow=nrow(sensorData)-1,
 ncol=length(columnNames)))
#sensorReadPeriods[1,]=c(sensorData$Row1TS[1],sensorData$Row2TS[1],sensorData$Row3TS[
 1]) #assign values for Row1
colnames(sensorReadPeriods) = columnNames

#Calculate periods from the timestamp data
for(i in 1:nrow(sensorReadPeriods)){
 sensorReadPeriods$R1[i] = sensorData$Row1TS[i+1]-sensorData$Row1TS[i]
 sensorReadPeriods$R2[i] = sensorData$Row2TS[i+1]-sensorData$Row2TS[i]
 sensorReadPeriods$R3[i] = sensorData$Row3TS[i+1]-sensorData$Row3TS[i]
 sensorReadPeriods$WT[i] = sensorData$Write_TS[i+1]-sensorData$Write_TS[i]
 sensorReadPeriods$R1GSTS[i] = sensorData$Row1GSTS[i+1]-sensorData$Row1GSTS[i]
 sensorReadPeriods$R2GSTS[i] = sensorData$Row2GSTS[i+1]-sensorData$Row2GSTS[i]
 sensorReadPeriods$R3GSTS[i] = sensorData$Row3GSTS[i+1]-sensorData$Row3GSTS[i]
```

```
 sensorReadPeriods$R1TDif[i] = sensorData$Row1TS[i]-sensorData$Row1TI[i]
 sensorReadPeriods$R2TDif[i] = sensorData$Row2TS[i]-sensorData$Row2TI[i]
 sensorReadPeriods$R3TDif[i] = sensorData$Row3TS[i]-sensorData$Row3TI[i]
}

#print(summary(sensorReadPeriods[-(1:20),]),)

##### Period Statistics Calculations #####
# Average, Standard Deviation, and Range
#avePeriod = c(mean(sensorReadPeriods$R1),mean(sensorReadPeriods$R2),
 mean(sensorReadPeriods$R3))
avePeriod = apply(sensorReadPeriods[-(1:100),], 2, mean) #shorter method
aveFrequency = 1/avePeriod*1000
stdevPeriod = apply(sensorReadPeriods[-(1:100),], 2, sd)
rangePeriod = apply(sensorReadPeriods[-(1:100),], 2, range)

rowNames = c("Ave. Period (ms)", "Ave. Freq (Hz)", "SD Period (ms)", "Min (ms)", "Max
 (ms)")

#join the above vectors into a table
summaryTable = rbind(avePeriod,aveFrequency,stdevPeriod,rangePeriod)
rownames(summaryTable) = rowNames  #assign rownames
print(summaryTable)


############################################################
########## Link GreenSeeker Data With GPS Data ############
############################################################

#calculate the average time delay, the min and maximum time delay
#final output table of linked data

columnNames = c('SIndex','LGPSIndex','RGPSIndex','TimeDelay', 'WhichGPS')
#GreenSeeker Row 1

S1 = data.frame(matrix(nrow = nrow(sensorData), ncol=5))
S2 = data.frame(matrix(nrow = nrow(sensorData), ncol=5))
S3 = data.frame(matrix(nrow = nrow(sensorData), ncol=5))

Slist = list(S1=S1, S2=S2, S3=S3)
TSlist = list(sensorData$Row1TS,sensorData$Row2TS, sensorData$Row3TS)

for(L in 1:length(Slist)) {
 names(Slist[[L]]) = columnNames
}

#colnames(S1) = columnNames

for(L in 1:length(Slist)) {
 LstartingIndex = 1
 LGPSIndex = 1
 RstartingIndex = 1
 RGPSIndex = 1
 for (i in 1:nrow(Slist[[L]])){
```

```
      #sensorTS = sensorData$Row1TS[i]
      sensorTS = TSlist[[L]][i]
      LcurDiff = abs(sensorTS - LeftGPSData$TS[LstartingIndex])
      RcurDiff = abs(sensorTS - RightGPSData$TS[RstartingIndex])
      LprevDiff = LcurDiff
      RprevDiff = RcurDiff
      #print(i)

      #Left GPS Loop
      for (j in LstartingIndex:nrow(LeftGPSData)){
            LcurDiff = abs(sensorTS - LeftGPSData$TS[j])
            if (LcurDiff < LprevDiff) {
                  LprevDiff = LcurDiff
                  LGPSIndex = j
                  print(j)
            }
            else if(LcurDiff >LprevDiff) {
                  print("stop")
                  break
            }
      }

      #Right GPS Loop
      for (k in RstartingIndex:nrow(RightGPSData)){
            RcurDiff = abs(sensorTS - RightGPSData$TS[k])
            if (RcurDiff < RprevDiff) {
                  RprevDiff = RcurDiff
                  RGPSIndex = k
                  print(k)
            }
            else if(RcurDiff >RprevDiff) {
                  print("stop")
                  break
            }
      }

      saveLGPSIndex = LGPSIndex # for use later in updating the starting index used
above
      saveRGPSIndex = RGPSIndex # for use later in updating the starting index used
above

      #compare time delay(RprevDiff and LprevDiff) values
      #first - check if UTC of GPS's are the same
      if(LeftGPSData$UTC[LGPSIndex] == RightGPSData$UTC[RGPSIndex]){
            #choose the time delay based on the GPS with the earliest time stamp
            if(LeftGPSData$TS[LGPSIndex]<=RightGPSData$TS[RGPSIndex]){
                  Slist[[L]]$TimeDelay[i] = LprevDiff
                  Slist[[L]]$WhichGPS[i] = 'L'
            }
            else {
                  Slist[[L]]$TimeDelay[i] = RprevDiff
                  Slist[[L]]$WhichGPS[i] = 'R'
            }
```

```
            Slist[[L]]$LGPSIndex[i] = LGPSIndex
            Slist[[L]]$RGPSIndex[i] = RGPSIndex
      }
      else { #if the UTC's are not equal
            #Go with the GPS associated with the shortest delay
            if(LprevDiff<=RprevDiff){ #choose Left GPS
                  Slist[[L]]$TimeDelay[i] = LprevDiff
                  Slist[[L]]$WhichGPS[i] = 'L'
                  Slist[[L]]$LGPSIndex[i] = LGPSIndex

                  #search for the Right Side GPS with equal or nearest UTC
                  UTCDiff = LeftGPSData$UTC[LGPSIndex] - RightGPSData$UTC[RGPSIndex]
                  IndexAdjust = abs(UTCDiff/0.2) #use an adjusted index for an
efficient starting point to search

                  if(UTCDiff < 1) {
                        RGPSIndex = RGPSIndex - IndexAdjust - 5
                  }
                  else if(UTCDiff > nrow(RightGPSData)) {RGPSIndex = 1}
                  else {
                        RGPSIndex = RGPSIndex + IndexAdjust -5
                  }
                  if (RGPSIndex < 1) RGPSIndex = 1
                  else if(RGPSIndex >= nrow(RightGPSData)) RGPSIndex =
nrow(RightGPSData)-10 #the -10 probably isn't necessary


                  #do the search
                  UTCDiff = abs(LeftGPSData$UTC[LGPSIndex] -
RightGPSData$UTC[RGPSIndex])
                  prevUTCDiff = UTCDiff
                  Index = RGPSIndex
                  for (t in RGPSIndex:nrow(RightGPSData)){
                        UTCDiff = abs(LeftGPSData$UTC[LGPSIndex] -
RightGPSData$UTC[t])
                        if (UTCDiff == 0) {
                              Index = t
                              break
                        }
                        else if(UTCDiff < prevUTCDiff) {
                              prevUTCDiff = UTCDiff
                              Index = t
                        }
                        else if(UTCDiff > prevUTCDiff) break
                  }

                  #assign the index to the S1 data frame
                  Slist[[L]]$RGPSIndex[i] = Index
            } #end of If statement
            else{ #choose Right GPS
                  Slist[[L]]$TimeDelay[i] = RprevDiff
                  Slist[[L]]$WhichGPS[i] = 'R'
                  Slist[[L]]$RGPSIndex[i] = RGPSIndex
```

```
                #search fo the Left Side GPS with equal or nearest UTC
                UTCDiff = RightGPSData$UTC[RGPSIndex] - LeftGPSData$UTC[LGPSIndex]
                IndexAdjust = abs(UTCDiff/0.2) #use an adjusted index for an
efficient starting point to search

                if(UTCDiff < 1) {
                      LGPSIndex = LGPSIndex - IndexAdjust - 5
                }
                else if(UTCDiff > nrow(LeftGPSData)) {LGPSIndex = 1}
                else {
                      LGPSIndex = LGPSIndex + IndexAdjust -5
                }
                if (LGPSIndex < 1) LGPSIndex = 1
                else if(LGPSIndex >= nrow(LeftGPSData)) LGPSIndex =
nrow(LeftGPSData)-10 #the -10 probably isn't necessary


                #do the search
                UTCDiff = abs(RightGPSData$UTC[RGPSIndex] -
LeftGPSData$UTC[LGPSIndex])
                prevUTCDiff = UTCDiff
                Index = LGPSIndex
                for (t in LGPSIndex:nrow(LeftGPSData)){
                      UTCDiff = abs(RightGPSData$UTC[RGPSIndex] -
LeftGPSData$UTC[t])
                      if (UTCDiff == 0) {
                            Index = t
                            break
                      }
                      else if(UTCDiff < prevUTCDiff) {
                            prevUTCDiff = UTCDiff
                            Index = t
                      }
                      else if(UTCDiff > prevUTCDiff) break
                }

                #assign the index to the S1 data frame
                Slist[[L]]$LGPSIndex[i] = Index
          } #end of else statement
     }#end of else statement - UTC are not equal statement

     Slist[[L]]$SIndex[i] = i
     LstartingIndex = saveLGPSIndex
     RstartingIndex = saveRGPSIndex
     #if(startingIndex < 1) startingIndex = 1
}
} #end of for loop


##############################################
########## Save GS Data in a Table  ##########
##############################################

columnNames = c('NDVI', 'Index2', 'TimeDelay', 'WhichGPS',
```

```
 'L_UTC','L_Lat','L_Lon','L_Elev','R_UTC','R_Lat','R_Lon','R_Elev')
S1Filled = data.frame(matrix(nrow = nrow(S1), ncol = length(columnNames)))
S2Filled = data.frame(matrix(nrow = nrow(S2), ncol = length(columnNames)))
S3Filled = data.frame(matrix(nrow = nrow(S3), ncol = length(columnNames)))

SFlist = list(S1Filled = S1Filled, S2Filled = S2Filled, S3Filled = S3Filled)

for(L in 1:length(SFlist)) {
 colnames(SFlist[[L]]) = columnNames
}

for(L in 1:length(SFlist)){
 if(L == 1){
      SFlist[[L]]$NDVI = sensorData$Row1NDVI
      SFlist[[L]]$Index2 = sensorData$Row1ExtraVI
 }
 else if(L == 2){
      SFlist[[L]]$NDVI = sensorData$Row2NDVI
      SFlist[[L]]$Index2 = sensorData$Row2ExtraVI
 }
 else if(L == 3){
      SFlist[[L]]$NDVI = sensorData$Row3NDVI
      SFlist[[L]]$Index2 = sensorData$Row3ExtraVI
 }
 SFlist[[L]]$TimeDelay = Slist[[L]]$TimeDelay
 SFlist[[L]]$WhichGPS = Slist[[L]]$WhichGPS
 SFlist[[L]][,(length(columnNames)-8):length(columnNames)]=
 LeftGPSData[Slist[[L]]$LGPSIndex,1:4]
 SFlist[[L]][,(length(columnNames)-
 4):length(columnNames)]=RightGPSData[Slist[[L]]$RGPSIndex, 1:4]

 ##############################################
 #########  Write GS Data to a File #########
 ##############################################
 filedate = substring(filename,1,16)
 filename = paste0(filedate,'GreenSeeker')
 filename = paste0(filename,names(SFlist[L]))
 filename = paste0(filename,'.txt')
 write.table(SFlist[[L]], file = filename, append = FALSE, quote = TRUE, sep = "\t",
             eol = "\n", na = "NA", dec = ".", row.names = FALSE,
             col.names = TRUE, qmethod = c("escape", "double"),
             fileEncoding = "")
}


##############################################
########## Analyze Time Delays ##############
##############################################
AveTimeDelay = c(mean(Slist$S1$TimeDelay),mean(Slist$S2$TimeDelay),
 mean(Slist$S3$TimeDelay) )
SDTimeDelay = c(sd(Slist$S1$TimeDelay),sd(Slist$S2$TimeDelay), sd(Slist$S3$TimeDelay)
 )
MinTimeDelay = c(min(Slist$S1$TimeDelay), min(Slist$S2$TimeDelay),
 min(Slist$S3$TimeDelay) )
```

```
MaxTimeDelay = c(max(Slist$S1$TimeDelay), max(Slist$S2$TimeDelay),
 max(Slist$S3$TimeDelay) )

TimeDelayTable = rbind(AveTimeDelay,SDTimeDelay,MinTimeDelay, MaxTimeDelay)
colnames(TimeDelayTable) = c("GreenSeeker 1", "GreenSeeker 2", "GreenSeeker 3")
rownames(TimeDelayTable) = c("Average (ms)", "Standard Deviation (ms)","Minimum
 (ms)", "Maximum (ms)")

##############################################
######## Write Summary Results To File ########
##############################################
 filename = paste0(filedate,'GreenSeekerStatSummary')
 filename = paste0(filename,'.txt')
write.table(summaryTable, file = filename, append = FALSE, quote = TRUE, sep = "\t",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"),
            fileEncoding = "")

write.table(TimeDelayTable, file = filename, append = TRUE, quote = TRUE, sep = "\t",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"),
            fileEncoding = "")


##############################################
######## Print Plots and Graphs ########
##############################################
dev.new(width = 8, height = 10)
op=par(mfrow=c(2,2))
plot(rownames(sensorReadPeriods), sensorReadPeriods$R1, xlab = "Sample Number", ylab
 = "Read Period (ms)", main="Row 1 GreenSeeker Read Periods")
plot(rownames(sensorReadPeriods), sensorReadPeriods$R2, xlab = "Sample Number", ylab
 = "Read Period (ms)", main="Row 2 GreenSeeker Read Periods")
plot(rownames(sensorReadPeriods), sensorReadPeriods$R3, xlab = "Sample Number", ylab
 = "Read Period (ms)", main="Row 3 GreenSeeker Read Periods")
par(op)

dev.new(width = 8, height = 8)
      plot(1:length(Slist[[1]]$TimeDelay), Slist[[1]]$TimeDelay, xlab = "Sample
Number", ylab = "Time Difference (ms)", main="GreenSeeker")
```

Crop Circle

```
#This program processes the Crop Circle Data
#
#sensorData - table where the raw sensor data is stored
#          the GNSS/GPS coordinates are not included in this table
#sensorReadPeriods - data frame where the calculated time periods are stored
#          Column Names: R1, R2, R3, WT, R1TDif, R2TDif, R3TDif
#          ... R1 is the column name for the read/timestamp periods of the row 1
 Crop Circle
#          ... R1TDif is the column name for the difference in TS - TI for Row 1,
#          ... Technically, R1TDif, R2TDif, and R3TDif and not periods, they are
 wait times
#summaryTable - summary of the period statiscs for each sensor
#          Column Names Ave. Period, Ave. Freq, SD Period, Min, Max


#set working directory
setwd("directory")
filename = '2014-03-21_1240_CropCircleData.txt'
sensorData = read.table(file=filename, header=T, sep = '\t', row.names = "Index",
 skip = 2)


#Clean up the data in sensorData - remove extra columns
colsToDelete = c('Left_UTCTime','Left_Elevation',
 'Left_Latitude','Left_Longitude','Right_UTCTime','Right_Elevation',
 'Right_Latitude', 'Right_Longitude')
colsToDelete = !(names(sensorData) %in% colsToDelete)
sensorData = sensorData[,colsToDelete]  #columns to delete


########################################
########## Calculate Periods ###########
########################################

#column names for sensorReadPeriods
columnNames = c('R1','R2','R3','WT','R1TDif','R2TDif','R3TDif')
#create sensorReadPeriods data frame
sensorReadPeriods = data.frame(matrix(nrow=nrow(sensorData)-1,
 ncol=length(columnNames)))
#sensorReadPeriods[1,]=c(sensorData$Row1TS[1],sensorData$Row2TS[1],sensorData$Row3TS[
 1]) #assign values for Row1
colnames(sensorReadPeriods) = columnNames

#Calculate periods from the timestamp data
for(i in 1:nrow(sensorReadPeriods)){
 sensorReadPeriods$R1[i] = sensorData$Row1TS[i+1]-sensorData$Row1TS[i]
 sensorReadPeriods$R2[i] = sensorData$Row2TS[i+1]-sensorData$Row2TS[i]
 sensorReadPeriods$R3[i] = sensorData$Row3TS[i+1]-sensorData$Row3TS[i]
 sensorReadPeriods$WT[i] = sensorData$Write_TS[i+1]-sensorData$Write_TS[i]
 sensorReadPeriods$R1TDif[i] = sensorData$Row1TS[i]-sensorData$Row1TI[i]
 sensorReadPeriods$R2TDif[i] = sensorData$Row2TS[i]-sensorData$Row2TI[i]
 sensorReadPeriods$R3TDif[i] = sensorData$Row3TS[i]-sensorData$Row3TI[i]
```

```
}

#print(summary(sensorReadPeriods[-(1:20),]),)

##### Period Statistics Calculations #####
# Average, Standard Deviation, and Range
#avePeriod = c(mean(sensorReadPeriods$R1),mean(sensorReadPeriods$R2),
 mean(sensorReadPeriods$R3))
avePeriod = apply(sensorReadPeriods[-(1:100),], 2, mean) #shorter method
aveFrequency = 1/avePeriod*1000
stdevPeriod = apply(sensorReadPeriods[-(1:100),], 2, sd)
rangePeriod = apply(sensorReadPeriods[-(1:100),], 2, range)

rowNames = c("Ave. Period (ms)", "Ave. Freq (Hz)", "SD Period (ms)", "Min (ms)", "Max
 (ms)")

#join the above vectors into a table
summaryTable = rbind(avePeriod,aveFrequency,stdevPeriod,rangePeriod)
rownames(summaryTable) = rowNames  #assign rownames
print(summaryTable)



############################################################
########## Link Crop Circle Data With GPS Data ############
############################################################

#calculate the average time delay, the min and maximum time delay
#final output table of linked data

columnNames = c('SIndex','LGPSIndex','RGPSIndex','TimeDelay', 'WhichGPS')
#GreenSeeker Row 1

S1 = data.frame(matrix(nrow = nrow(sensorData), ncol=5))
S2 = data.frame(matrix(nrow = nrow(sensorData), ncol=5))
S3 = data.frame(matrix(nrow = nrow(sensorData), ncol=5))

Slist = list(S1=S1, S2=S2, S3=S3)
TSlist = list(sensorData$Row1TS,sensorData$Row2TS, sensorData$Row3TS)

for(L in 1:length(Slist)) {
 names(Slist[[L]]) = columnNames
}



for(L in 1:length(Slist)) {
 LstartingIndex = 1
 LGPSIndex = 1
 RstartingIndex = 1
 RGPSIndex = 1
 for (i in 1:nrow(Slist[[L]])){

      #sensorTS = sensorData$Row1TS[i]
```

```
	sensorTS = TSlist[[L]][i]
	LcurDiff = abs(sensorTS - LeftGPSData$TS[LstartingIndex])
	RcurDiff = abs(sensorTS - RightGPSData$TS[RstartingIndex])
	LprevDiff = LcurDiff
	RprevDiff = RcurDiff
	#print(i)

	#Left GPS Loop
	for (j in LstartingIndex:nrow(LeftGPSData)){
		LcurDiff = abs(sensorTS - LeftGPSData$TS[j])
		if (LcurDiff < LprevDiff) {
			LprevDiff = LcurDiff
			LGPSIndex = j
			print(j)
		}
		else if(LcurDiff >LprevDiff) {
			print("stop")
			break
		}
	}

	#Right GPS Loop
	for (k in RstartingIndex:nrow(RightGPSData)){
		RcurDiff = abs(sensorTS - RightGPSData$TS[k])
		if (RcurDiff < RprevDiff) {
			RprevDiff = RcurDiff
			RGPSIndex = k
			print(k)
		}
		else if(RcurDiff >RprevDiff) {
			print("stop")
			break
		}
	}

	saveLGPSIndex = LGPSIndex # for use later in updating the starting index used
above
	saveRGPSIndex = RGPSIndex # for use later in updating the starting index used
above

	#compare time delay(RprevDiff and LprevDiff) values
	#first - check if UTC of GPS's are the same
	if(LeftGPSData$UTC[LGPSIndex] == RightGPSData$UTC[RGPSIndex]){
		#choose the time delay based on the GPS with the earliest time stamp
		if(LeftGPSData$TS[LGPSIndex]<=RightGPSData$TS[RGPSIndex]){
			Slist[[L]]$TimeDelay[i] = LprevDiff
			Slist[[L]]$WhichGPS[i] = 'L'
		}
		else {
			Slist[[L]]$TimeDelay[i] = RprevDiff
			Slist[[L]]$WhichGPS[i] = 'R'
		}
		Slist[[L]]$LGPSIndex[i] = LGPSIndex
		Slist[[L]]$RGPSIndex[i] = RGPSIndex
```

```
      }
      else { #if the UTC's are not equal
            #Go with the GPS associated with the shortest delay
            if(LprevDiff<=RprevDiff){ #choose Left GPS
                  Slist[[L]]$TimeDelay[i] = LprevDiff
                  Slist[[L]]$WhichGPS[i] = 'L'
                  Slist[[L]]$LGPSIndex[i] = LGPSIndex

                  #search for the Right Side GPS with equal or nearest UTC
                  UTCDiff = LeftGPSData$UTC[LGPSIndex] - RightGPSData$UTC[RGPSIndex]
                  IndexAdjust = abs(UTCDiff/0.2) #use an adjusted index for an
efficient starting point to search

                  if(UTCDiff < 1) {
                        RGPSIndex = RGPSIndex - IndexAdjust - 5
                  }
                  else if(UTCDiff > nrow(RightGPSData)) {RGPSIndex = 1}
                  else {
                        RGPSIndex = RGPSIndex + IndexAdjust -5
                  }
                  if (RGPSIndex < 1) RGPSIndex = 1
                  else if(RGPSIndex >= nrow(RightGPSData)) RGPSIndex =
nrow(RightGPSData)-10 #the -10 probably isn't necessary


                  #do the search
                  UTCDiff = abs(LeftGPSData$UTC[LGPSIndex] -
RightGPSData$UTC[RGPSIndex])
                  prevUTCDiff = UTCDiff
                  Index = RGPSIndex
                  for (t in RGPSIndex:nrow(RightGPSData)){
                        UTCDiff = abs(LeftGPSData$UTC[LGPSIndex] -
RightGPSData$UTC[t])
                        if (UTCDiff == 0) {
                              Index = t
                              break
                        }
                        else if(UTCDiff < prevUTCDiff) {
                              prevUTCDiff = UTCDiff
                              Index = t
                        }
                        else if(UTCDiff > prevUTCDiff) break
                  }

                  #assign the index to the GS1 data frame
                  Slist[[L]]$RGPSIndex[i] = Index
            } #end of If statement
            else{ #choose Right GPS
                  Slist[[L]]$TimeDelay[i] = RprevDiff
                  Slist[[L]]$WhichGPS[i] = 'R'
                  Slist[[L]]$RGPSIndex[i] = RGPSIndex

                  #search fo the Left Side GPS with equal or nearest UTC
                  UTCDiff = RightGPSData$UTC[RGPSIndex] - LeftGPSData$UTC[LGPSIndex]
```

```
                IndexAdjust = abs(UTCDiff/0.2) #use an adjusted index for an
efficient starting point to search

                if(UTCDiff < 1) {
                      LGPSIndex = LGPSIndex - IndexAdjust - 5
                }
                else if(UTCDiff > nrow(LeftGPSData)) {LGPSIndex = 1}
                else {
                      LGPSIndex = LGPSIndex + IndexAdjust -5
                }
                if (LGPSIndex < 1) LGPSIndex = 1
                else if(LGPSIndex >= nrow(LeftGPSData)) LGPSIndex =
nrow(LeftGPSData)-10 #the -10 probably isn't necessary


                #do the search
                UTCDiff = abs(RightGPSData$UTC[RGPSIndex] -
LeftGPSData$UTC[LGPSIndex])
                prevUTCDiff = UTCDiff
                Index = LGPSIndex
                for (t in LGPSIndex:nrow(LeftGPSData)){
                      UTCDiff = abs(RightGPSData$UTC[RGPSIndex] -
LeftGPSData$UTC[t])
                      if (UTCDiff == 0) {
                            Index = t
                            break
                      }
                      else if(UTCDiff < prevUTCDiff) {
                            prevUTCDiff = UTCDiff
                            Index = t
                      }
                      else if(UTCDiff > prevUTCDiff) break
                }

                #assign the index to the GS1 data frame
                Slist[[L]]$LGPSIndex[i] = Index
          } #end of else statement
     }#end of else statement - UTC are not equal statement

     Slist[[L]]$SIndex[i] = i
     LstartingIndex = saveLGPSIndex
     RstartingIndex = saveRGPSIndex
     #if(startingIndex < 1) startingIndex = 1
}
} #end of for loop



##############################################
########## Save GS Data in a Table  ##########
##############################################

columnNames = c('CalcOut1', 'CalcOut2','Chan1','Chan2','Chan3', 'TimeDelay',
 'WhichGPS', 'L_UTC','L_Lat','L_Lon','L_Elev','R_UTC','R_Lat','R_Lon','R_Elev')
```

```
S1Filled = data.frame(matrix(nrow = nrow(S1), ncol = length(columnNames)))
S2Filled = data.frame(matrix(nrow = nrow(S2), ncol = length(columnNames)))
S3Filled = data.frame(matrix(nrow = nrow(S3), ncol = length(columnNames)))

SFlist = list(S1Filled = S1Filled, S2Filled = S2Filled, S3Filled = S3Filled)

for(L in 1:length(SFlist)) {
 colnames(SFlist[[L]]) = columnNames
}

for(L in 1:length(SFlist)){
 if(L==1){
      SFlist[[L]]$CalcOut1 = sensorData$Row1CalcOutput1
      SFlist[[L]]$CalcOut2 = sensorData$Row1CalcOutput2
      SFlist[[L]]$Chan1 = sensorData$Row1Channel1
      SFlist[[L]]$Chan2 = sensorData$Row1Channel2
      SFlist[[L]]$Chan3 = sensorData$Row1Channel3
 }
 else if(L==2){
      SFlist[[L]]$CalcOut1 = sensorData$Row2CalcOutput1
      SFlist[[L]]$CalcOut2 = sensorData$Row2CalcOutput2
      SFlist[[L]]$Chan1 = sensorData$Row2Channel1
      SFlist[[L]]$Chan2 = sensorData$Row2Channel2
      SFlist[[L]]$Chan3 = sensorData$Row2Channel3
 }
 else if(L==3){
      SFlist[[L]]$CalcOut1 = sensorData$Row3CalcOutput1
      SFlist[[L]]$CalcOut2 = sensorData$Row3CalcOutput2
      SFlist[[L]]$Chan1 = sensorData$Row3Channel1
      SFlist[[L]]$Chan2 = sensorData$Row3Channel2
      SFlist[[L]]$Chan3 = sensorData$Row3Channel3
 }

 SFlist[[L]]$TimeDelay = Slist[[L]]$TimeDelay
 SFlist[[L]]$WhichGPS = Slist[[L]]$WhichGPS
 SFlist[[L]][,(length(columnNames)-8):length(columnNames)]=
 LeftGPSData[Slist[[L]]$LGPSIndex,1:4]
 SFlist[[L]][,(length(columnNames)-
 4):length(columnNames)]=RightGPSData[Slist[[L]]$RGPSIndex, 1:4]


 ##############################################
 #########  Write GS Data to a File #########
 ##############################################
 filedate = substring(filename,1,16) #grab the date
 filename = paste0(filedate,'CropCircle')
 filename = paste0(filename,names(SFlist[L]))
 filename = paste0(filename,'.txt')
 write.table(SFlist[[L]], file = filename, append = FALSE, quote = TRUE, sep = "\t",
             eol = "\n", na = "NA", dec = ".", row.names = FALSE,
             col.names = TRUE, qmethod = c("escape", "double"),
             fileEncoding = "")
}
```

```
##############################################
########## Analyze Time Delays ##############
##############################################
AveTimeDelay = c(mean(Slist$S1$TimeDelay),mean(Slist$S2$TimeDelay),
 mean(Slist$S3$TimeDelay) )
SDTimeDelay = c(sd(Slist$S1$TimeDelay),sd(Slist$S2$TimeDelay), sd(Slist$S3$TimeDelay)
 )
MinTimeDelay = c(min(Slist$S1$TimeDelay), min(Slist$S2$TimeDelay),
 min(Slist$S3$TimeDelay) )
MaxTimeDelay = c(max(Slist$S1$TimeDelay), max(Slist$S2$TimeDelay),
 max(Slist$S3$TimeDelay) )

TimeDelayTable = rbind(AveTimeDelay,SDTimeDelay,MinTimeDelay, MaxTimeDelay)
colnames(TimeDelayTable) = c("CropCircle 1", "CropCircle 2", "CropCircle 3")
rownames(TimeDelayTable) = c("Average (ms)", "Standard Deviation (ms)","Minimum
 (ms)", "Maximum (ms)")

##############################################
######## Write Summary Results To File ########
##############################################
 filename = paste0(filedate,'CropCircleStatSummary')
 filename = paste0(filename,'.txt')
write.table(summaryTable, file = filename, append = FALSE, quote = TRUE, sep = "\t",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"),
            fileEncoding = "")

write.table(TimeDelayTable, file = filename, append = TRUE, quote = TRUE, sep = "\t",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"),
            fileEncoding = "")



##############################################
######## Print Plots and Graphs ########
##############################################
dev.new(width = 8, height = 10)
op=par(mfrow=c(2,2))
plot(rownames(sensorReadPeriods), sensorReadPeriods$R1, xlab = "Sample Number", ylab
 = "Read Period (ms)", main="Row 1 Crop Circle Read Periods")
plot(rownames(sensorReadPeriods), sensorReadPeriods$R2, xlab = "Sample Number", ylab
 = "Read Period (ms)", main="Row 2 Crop Circle Read Periods")
plot(rownames(sensorReadPeriods), sensorReadPeriods$R3, xlab = "Sample Number", ylab
 = "Read Period (ms)", main="Row 3 Crop Circle Read Periods")
par(op)

dev.new(width = 8, height = 8)
plot(1:length(Slist[[1]]$TimeDelay), Slist[[1]]$TimeDelay, xlab = "Sample Number",
 ylab = "Time Difference (ms)", main="Crop Circle")
```

Distance Sensor

```
#This program processes the Distance Data From the Ultrasonic and Laser Sensors
#
#sensorData - table where the raw sensor data is stored
#          the GNSS/GPS coordinates are not included in this table
#sensorReadPeriods - data frame where the calculated time periods are stored
#          Column Names: R123, WT, RTDif
#          ... R123 is the column name for the read/timestamp periods of all the
 Distance Data
#          ... RTDif is the column name for the difference in TS - TI
#          ... Technically RTDif is not a period,but a wait times
#summaryTable - summary of the period statiscs for each sensor
#          Column Names Ave. Period, Ave. Freq, SD Period, Min, Max

#set working directory
setwd("Directory")
filename = '2014-03-21_1240_HeightSensorData.txt'
sensorData = read.table(file=filename, header=T, sep = '\t', row.names = "Index",
 skip = 2)


#Clean up the data in sensorData - remove extra columns
colsToDelete = c('Left_UTCTime','Left_Elevation',
 'Left_Latitude','Left_Longitude','Right_UTCTime','Right_Elevation',
 'Right_Latitude', 'Right_Longitude')
colsToDelete = !(names(sensorData) %in% colsToDelete)
sensorData = sensorData[,colsToDelete]  #columns to delete


########################################
########## Calculate Periods ###########
########################################

#column names for sensorReadPeriods
columnNames = c('R123','WT','RTDif')
#create sensorReadPeriods data frame
sensorReadPeriods = data.frame(matrix(nrow=nrow(sensorData)-1,
 ncol=length(columnNames)))
#sensorReadPeriods[1,]=c(sensorData$Row1TS[1],sensorData$Row2TS[1],sensorData$Row3TS[
 1]) #assign values for Row1
colnames(sensorReadPeriods) = columnNames

#Calculate periods from the timestamp data
for(i in 1:nrow(sensorReadPeriods)){
 sensorReadPeriods$R123[i] = sensorData$TS[i+1]-sensorData$TS[i]
 sensorReadPeriods$WT[i] = sensorData$Write_TS[i+1]-sensorData$Write_TS[i]
 sensorReadPeriods$RTDif[i] = sensorData$TS[i]-sensorData$WT[i]
}

#print(summary(sensorReadPeriods[-(1:20),]),)

##### Period Statistics Calculations #####
# Average, Standard Deviation, and Range
```

```
#avePeriod = c(mean(sensorReadPeriods$R1),mean(sensorReadPeriods$R2),
 mean(sensorReadPeriods$R3))
avePeriod = apply(sensorReadPeriods[-(1:100),], 2, mean) #shorter method
aveFrequency = 1/avePeriod*1000
stdevPeriod = apply(sensorReadPeriods[-(1:100),], 2, sd)
rangePeriod = apply(sensorReadPeriods[-(1:100),], 2, range)

rowNames = c("Ave. Period (ms)", "Ave. Freq (Hz)", "SD Period (ms)", "Min (ms)", "Max
 (ms)")

#join the above vectors into a table
summaryTable = rbind(avePeriod,aveFrequency,stdevPeriod,rangePeriod)
rownames(summaryTable) = rowNames  #assign rownames
print(summaryTable)


############################################################
########## Link Distance Data With GPS Data ############
############################################################

#calculate the average time delay, the min and maximum time delay
#final output table of linked data

columnNames = c('SIndex','LGPSIndex','RGPSIndex','TimeDelay', 'WhichGPS')
#GreenSeeker Row 1

S1 = data.frame(matrix(nrow = nrow(sensorData), ncol=5))

Slist = list(S1=S1)
TSlist = list(sensorData$TS)

for(L in 1:length(Slist)) {
 names(Slist[[L]]) = columnNames
}

for(L in 1:length(Slist)) {
 LstartingIndex = 1
 LGPSIndex = 1
 RstartingIndex = 1
 RGPSIndex = 1
 for (i in 1:nrow(Slist[[L]])){

      #sensorTS = sensorData$Row1TS[i]
      sensorTS = TSlist[[L]][i]
      LcurDiff = abs(sensorTS - LeftGPSData$TS[LstartingIndex])
      RcurDiff = abs(sensorTS - RightGPSData$TS[RstartingIndex])
      LprevDiff = LcurDiff
      RprevDiff = RcurDiff
      #print(i)

      #Left GPS Loop
      for (j in LstartingIndex:nrow(LeftGPSData)){
            LcurDiff = abs(sensorTS - LeftGPSData$TS[j])
            if (LcurDiff < LprevDiff) {
```

```
                LprevDiff = LcurDiff
                LGPSIndex = j
                print(j)
          }
          else if(LcurDiff >LprevDiff) {
                print("stop")
                break
          }
     }

     #Right GPS Loop
     for (k in RstartingIndex:nrow(RightGPSData)){
          RcurDiff = abs(sensorTS - RightGPSData$TS[k])
          if (RcurDiff < RprevDiff) {
                RprevDiff = RcurDiff
                RGPSIndex = k
                print(k)
          }
          else if(RcurDiff >RprevDiff) {
                print("stop")
                break
          }
     }

     saveLGPSIndex = LGPSIndex # for use later in updating the starting index used
above
     saveRGPSIndex = RGPSIndex # for use later in updating the starting index used
above

     #compare time delay(RprevDiff and LprevDiff) values
     #first - check if UTC of GPS's are the same
     if(LeftGPSData$UTC[LGPSIndex] == RightGPSData$UTC[RGPSIndex]){
          #choose the time delay based on the GPS with the earliest time stamp
          if(LeftGPSData$TS[LGPSIndex]<=RightGPSData$TS[RGPSIndex]){
                Slist[[L]]$TimeDelay[i] = LprevDiff
                Slist[[L]]$WhichGPS[i] = 'L'
          }
          else {
                Slist[[L]]$TimeDelay[i] = RprevDiff
                Slist[[L]]$WhichGPS[i] = 'R'
          }
          Slist[[L]]$LGPSIndex[i] = LGPSIndex
          Slist[[L]]$RGPSIndex[i] = RGPSIndex
     }
     else { #if the UTC's are not equal
          #Go with the GPS associated with the shortest delay
          if(LprevDiff<=RprevDiff){ #choose Left GPS
                Slist[[L]]$TimeDelay[i] = LprevDiff
                Slist[[L]]$WhichGPS[i] = 'L'
                Slist[[L]]$LGPSIndex[i] = LGPSIndex

                #search for the Right Side GPS with equal or nearest UTC
                UTCDiff = LeftGPSData$UTC[LGPSIndex] - RightGPSData$UTC[RGPSIndex]
                IndexAdjust = abs(UTCDiff/0.2) #use an adjusted index for an
```

```
efficient starting point to search

                if(UTCDiff < 1) {
                      RGPSIndex = RGPSIndex - IndexAdjust - 5
                }
                else if(UTCDiff > nrow(RightGPSData)) {RGPSIndex = 1}
                else {
                      RGPSIndex = RGPSIndex + IndexAdjust -5
                }
                if (RGPSIndex < 1) RGPSIndex = 1
                else if(RGPSIndex >= nrow(RightGPSData)) RGPSIndex =
nrow(RightGPSData)-10 #the -10 probably isn't necessary


                #do the search
                UTCDiff = abs(LeftGPSData$UTC[LGPSIndex] -
RightGPSData$UTC[RGPSIndex])
                prevUTCDiff = UTCDiff
                Index = RGPSIndex
                for (t in RGPSIndex:nrow(RightGPSData)){
                      UTCDiff = abs(LeftGPSData$UTC[LGPSIndex] -
RightGPSData$UTC[t])
                      if (UTCDiff == 0) {
                            Index = t
                            break
                      }
                      else if(UTCDiff < prevUTCDiff) {
                            prevUTCDiff = UTCDiff
                            Index = t
                      }
                      else if(UTCDiff > prevUTCDiff) break
                }

                #assign the index to the GS1 data frame
                Slist[[L]]$RGPSIndex[i] = Index
          } #end of If statement
          else{ #choose Right GPS
                Slist[[L]]$TimeDelay[i] = RprevDiff
                Slist[[L]]$WhichGPS[i] = 'R'
                Slist[[L]]$RGPSIndex[i] = RGPSIndex

                #search fo the Left Side GPS with equal or nearest UTC
                UTCDiff = RightGPSData$UTC[RGPSIndex] - LeftGPSData$UTC[LGPSIndex]
                IndexAdjust = abs(UTCDiff/0.2) #use an adjusted index for an
efficient starting point to search

                if(UTCDiff < 1) {
                      LGPSIndex = LGPSIndex - IndexAdjust - 5
                }
                else if(UTCDiff > nrow(LeftGPSData)) {LGPSIndex = 1}
                else {
                      LGPSIndex = LGPSIndex + IndexAdjust -5
                }
                if (LGPSIndex < 1) LGPSIndex = 1
```

```
                else if(LGPSIndex >= nrow(LeftGPSData)) LGPSIndex =
 nrow(LeftGPSData)-10 #the -10 probably isn't necessary


                #do the search
                UTCDiff = abs(RightGPSData$UTC[RGPSIndex] -
 LeftGPSData$UTC[LGPSIndex])
                prevUTCDiff = UTCDiff
                Index = LGPSIndex
                for (t in LGPSIndex:nrow(LeftGPSData)){
                      UTCDiff = abs(RightGPSData$UTC[RGPSIndex] -
 LeftGPSData$UTC[t])
                      if (UTCDiff == 0) {
                            Index = t
                            break
                      }
                      else if(UTCDiff < prevUTCDiff) {
                            prevUTCDiff = UTCDiff
                            Index = t
                      }
                      else if(UTCDiff > prevUTCDiff) break
                }

                #assign the index to the GS1 data frame
                Slist[[L]]$LGPSIndex[i] = Index
           } #end of else statement
     }#end of else statement - UTC are not equal statement

     Slist[[L]]$SIndex[i] = i
     LstartingIndex = saveLGPSIndex
     RstartingIndex = saveRGPSIndex
     #if(startingIndex < 1) startingIndex = 1
 }
} #end of for loop



############################################################
########## Save Distance Sensor Data in a Table  ##########
############################################################

columnNames = c('R1H', 'R2H','R3H','LSH','TimeDelay', 'WhichGPS',
 'L_UTC','L_Lat','L_Lon','L_Elev','R_UTC','R_Lat','R_Lon','R_Elev')
S1Filled = data.frame(matrix(nrow = nrow(S1), ncol = length(columnNames)))

SFlist = list(S1Filled = S1Filled)

for(L in 1:length(SFlist)) {
 colnames(SFlist[[L]]) = columnNames
}

for(L in 1:length(SFlist)){

 SFlist[[L]]$R1H = sensorData$Row1Height
```

```
 SFlist[[L]]$R2H = sensorData$Row2Height
 SFlist[[L]]$R3H = sensorData$Row3Height
 SFlist[[L]]$LSH = sensorData$LSHeight
 SFlist[[L]]$TimeDelay = Slist[[L]]$TimeDelay
 SFlist[[L]]$WhichGPS = Slist[[L]]$WhichGPS
 SFlist[[L]][,(length(columnNames)-8):length(columnNames)]=
 LeftGPSData[Slist[[L]]$LGPSIndex,1:4]
 SFlist[[L]][,(length(columnNames)-
 4):length(columnNames)]=RightGPSData[Slist[[L]]$RGPSIndex, 1:4]

 ##############################################
 #########  Write Height Data to a File #########
 ##############################################
 filedate = substring(filename,1,16) #grab the date
 filename = paste0(filedate,'HeightSensor')
 filename = paste0(filename,'.txt')
 write.table(SFlist[[L]], file = filename, append = FALSE, quote = TRUE, sep = "\t",
             eol = "\n", na = "NA", dec = ".", row.names = FALSE,
             col.names = TRUE, qmethod = c("escape", "double"),
             fileEncoding = "")
}


##############################################
########## Analyze Time Delays ##############
##############################################
AveTimeDelay = mean(Slist$S1$TimeDelay)
SDTimeDelay = sd(Slist$S1$TimeDelay)
MinTimeDelay = min(Slist$S1$TimeDelay)
MaxTimeDelay = max(Slist$S1$TimeDelay)

TimeDelayTable = rbind(AveTimeDelay,SDTimeDelay,MinTimeDelay, MaxTimeDelay)
colnames(TimeDelayTable) = "HeightSensors"
rownames(TimeDelayTable) = c("Average (ms)", "Standard Deviation (ms)","Minimum
 (ms)", "Maximum (ms)")


##############################################
######## Write Summary Results To File ########
##############################################
 filename = paste0(filedate,'HeightSensorStatSummary')
 filename = paste0(filename,'.txt')
write.table(summaryTable, file = filename, append = FALSE, quote = TRUE, sep = "\t",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"),
            fileEncoding = "")

write.table(TimeDelayTable, file = filename, append = TRUE, quote = TRUE, sep = "\t",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"),
            fileEncoding = "")


##############################################
```

```
######## Print Plots and Graphs ########
################################################
dev.new(width = 8, height = 8)
plot(rownames(sensorReadPeriods), sensorReadPeriods$R123, xlab = "Sample Number",
 ylab = "Read Period (ms)", main="Distance Sensor Read Periods")

dev.new(width = 8, height = 8)
      plot(1:length(Slist[[1]]$TimeDelay), Slist[[1]]$TimeDelay, xlab = "Sample
Number", ylab = "Time Difference (ms)", main="Distance Sensors")
```

IRT

```
#This program processes the IRT Data
#
#sensorData - table where the raw sensor data is stored
#          the GNSS/GPS coordinates are not included in this table
#sensorReadPeriods - data frame where the calculated time periods are stored
#          Column Names: R1.15.1, R2.15.1, R3.15.1, WT, R1.15.1TDif, R2.15.1TDif,
 R3.15.1TDif
#          ... R1.15.1 is the column name for the read/timestamp periods of the row
 1 15:1 IRT
#          ... R1TDif is the column name for the difference in TS - TI for Row 1,
#          ... Technically, R1TDif, R2TDif, and R3TDif and not periods, they are
 wait times
#summaryTable - summary of the period statiscs for each sensor
#          Column Names Ave. Period, Ave. Freq, SD Period, Min, Max


#set working directory
setwd("Directory")
filename = '2014-03-21_1240_IRTData.txt'
sensorData = read.table(file=filename, header=T, sep = '\t', row.names = "Index",
 skip = 2)


#Clean up the data in sensorData - remove extra columns
colsToDelete = c('Left_UTCTime','Left_Elevation',
 'Left_Latitude','Left_Longitude','Right_UTCTime','Right_Elevation',
 'Right_Latitude', 'Right_Longitude')
colsToDelete = !(names(sensorData) %in% colsToDelete)
sensorData = sensorData[,colsToDelete]  #columns to delete


########################################
########## Calculate Periods ###########
########################################

#column names for sensorReadPeriods
columnNames =
 c('R1.15.1','R2.15.1','R3.15.1','R1.2.1','R2.2.1','R3.2.1','WT','R1.15.1TDif','R2.15
 .1TDif','R3.15.1TDif','R1.2.1TDif','R2.2.1TDif','R3.2.1TDif')
#create sensorReadPeriods data frame
sensorReadPeriods = data.frame(matrix(nrow=nrow(sensorData)-1,
 ncol=length(columnNames)))
#sensorReadPeriods[1,]=c(sensorData$Row1TS[1],sensorData$Row2TS[1],sensorData$Row3TS[
 1]) #assign values for Row1
colnames(sensorReadPeriods) = columnNames

#Calculate periods from the timestamp data
for(i in 1:nrow(sensorReadPeriods)){
 sensorReadPeriods$R1.15.1[i] = sensorData$Row1TS15.1[i+1]-sensorData$Row1TS15.1[i]
 sensorReadPeriods$R2.15.1[i] = sensorData$Row2TS15.1[i+1]-sensorData$Row2TS15.1[i]
 sensorReadPeriods$R3.15.1[i] = sensorData$Row3TS15.1[i+1]-sensorData$Row3TS15.1[i]
 sensorReadPeriods$R1.2.1[i] = sensorData$Row1TS2.1[i+1]-sensorData$Row1TS2.1[i]
```

```
 sensorReadPeriods$R2.2.1[i] = sensorData$Row2TS2.1[i+1]-sensorData$Row2TS2.1[i]
 sensorReadPeriods$R3.2.1[i] = sensorData$Row3TS2.1[i+1]-sensorData$Row3TS2.1[i]
 sensorReadPeriods$WT[i] = sensorData$Write_TS[i+1]-sensorData$Write_TS[i]
 sensorReadPeriods$R1.15.1TDif[i] = sensorData$Row1TS15.1[i]-sensorData$Row1WT15.1[i]
 sensorReadPeriods$R2.15.1TDif[i] = sensorData$Row2TS15.1[i]-sensorData$Row2WT15.1[i]
 sensorReadPeriods$R3.15.1TDif[i] = sensorData$Row3TS15.1[i]-sensorData$Row3WT15.1[i]
 sensorReadPeriods$R1.2.1TDif[i] = sensorData$Row1TS2.1[i]-sensorData$Row1WT2.1[i]
 sensorReadPeriods$R2.2.1TDif[i] = sensorData$Row2TS2.1[i]-sensorData$Row2WT2.1[i]
 sensorReadPeriods$R3.2.1TDif[i] = sensorData$Row3TS2.1[i]-sensorData$Row3WT2.1[i]
}

#print(summary(sensorReadPeriods[-(1:20),]),)

##### Period Statistics Calculations #####
# Average, Standard Deviation, and Range
#avePeriod = c(mean(sensorReadPeriods$R1),mean(sensorReadPeriods$R2),
 mean(sensorReadPeriods$R3))
avePeriod = apply(sensorReadPeriods[-(1:100),], 2, mean) #shorter method
aveFrequency = 1/avePeriod*1000
stdevPeriod = apply(sensorReadPeriods[-(1:100),], 2, sd)
rangePeriod = apply(sensorReadPeriods[-(1:100),], 2, range)

rowNames = c("Ave. Period (ms)", "Ave. Freq (Hz)", "SD Period (ms)", "Min (ms)", "Max
 (ms)")

#join the above vectors into a table
summaryTable = rbind(avePeriod,aveFrequency,stdevPeriod,rangePeriod)
rownames(summaryTable) = rowNames  #assign rownames
print(summaryTable)




############################################################
########## Link IRT Data With GPS Data ############
############################################################

#calculate the average time delay, the min and maximum time delay
#final output table of linked data

columnNames = c('SIndex','LGPSIndex','RGPSIndex','TimeDelay', 'WhichGPS')
#GreenSeeker Row 1

S1.15.1 = data.frame(matrix(nrow = nrow(sensorData), ncol=5))
S2.15.1 = data.frame(matrix(nrow = nrow(sensorData), ncol=5))
S3.15.1 = data.frame(matrix(nrow = nrow(sensorData), ncol=5))
S1.2.1 = data.frame(matrix(nrow = nrow(sensorData), ncol=5))
S2.2.1 = data.frame(matrix(nrow = nrow(sensorData), ncol=5))
S3.2.1 = data.frame(matrix(nrow = nrow(sensorData), ncol=5))

Slist = list(S1.15.1=S1.15.1, S2.15.1=S2.15.1, S3.15.1=S3.15.1, S1.2.1=S1.2.1,
 S2.2.1=S2.2.1, S3.2.1=S3.2.1)
TSlist = list(sensorData$Row1TS15.1,sensorData$Row2TS15.1, sensorData$Row3TS15.1,
 sensorData$Row1TS2.1,sensorData$Row2TS2.1, sensorData$Row3TS2.1)
```

```
for(L in 1:length(Slist)) {
 names(Slist[[L]]) = columnNames
}

for(L in 1:length(Slist)) {
 LstartingIndex = 1
 LGPSIndex = 1
 RstartingIndex = 1
 RGPSIndex = 1
 for (i in 1:nrow(Slist[[L]])){

      #sensorTS = sensorData$Row1TS[i]
      sensorTS = TSlist[[L]][i]
      LcurDiff = abs(sensorTS - LeftGPSData$TS[LstartingIndex])
      RcurDiff = abs(sensorTS - RightGPSData$TS[RstartingIndex])
      LprevDiff = LcurDiff
      RprevDiff = RcurDiff
      #print(i)

      #Left GPS Loop
      for (j in LstartingIndex:nrow(LeftGPSData)){
            LcurDiff = abs(sensorTS - LeftGPSData$TS[j])
            if (LcurDiff < LprevDiff) {
                  LprevDiff = LcurDiff
                  LGPSIndex = j
                  print(j)
            }
            else if(LcurDiff >LprevDiff) {
                  print("stop")
                  break
            }
      }

      #Right GPS Loop
      for (k in RstartingIndex:nrow(RightGPSData)){
            RcurDiff = abs(sensorTS - RightGPSData$TS[k])
            if (RcurDiff < RprevDiff) {
                  RprevDiff = RcurDiff
                  RGPSIndex = k
                  print(k)
            }
            else if(RcurDiff >RprevDiff) {
                  print("stop")
                  break
            }
      }

      saveLGPSIndex = LGPSIndex # for use later in updating the starting index used
 above
      saveRGPSIndex = RGPSIndex # for use later in updating the starting index used
 above

      #compare time delay(RprevDiff and LprevDiff) values
```

```
      #first - check if UTC of GPS's are the same
      if(LeftGPSData$UTC[LGPSIndex] == RightGPSData$UTC[RGPSIndex]){
            #choose the time delay based on the GPS with the earliest time stamp
            if(LeftGPSData$TS[LGPSIndex]<=RightGPSData$TS[RGPSIndex]){
                  Slist[[L]]$TimeDelay[i] = LprevDiff
                  Slist[[L]]$WhichGPS[i] = 'L'
            }
            else {
                  Slist[[L]]$TimeDelay[i] = RprevDiff
                  Slist[[L]]$WhichGPS[i] = 'R'
            }
            Slist[[L]]$LGPSIndex[i] = LGPSIndex
            Slist[[L]]$RGPSIndex[i] = RGPSIndex
      }
      else { #if the UTC's are not equal
            #Go with the GPS associated with the shortest delay
            if(LprevDiff<=RprevDiff){ #choose Left GPS
                  Slist[[L]]$TimeDelay[i] = LprevDiff
                  Slist[[L]]$WhichGPS[i] = 'L'
                  Slist[[L]]$LGPSIndex[i] = LGPSIndex

                  #search for the Right Side GPS with equal or nearest UTC
                  UTCDiff = LeftGPSData$UTC[LGPSIndex] - RightGPSData$UTC[RGPSIndex]
                  IndexAdjust = abs(UTCDiff/0.2) #use an adjusted index for an
efficient starting point to search

                  if(UTCDiff < 1) {
                        RGPSIndex = RGPSIndex - IndexAdjust - 5
                  }
                  else if(UTCDiff > nrow(RightGPSData)) {RGPSIndex = 1}
                  else {
                        RGPSIndex = RGPSIndex + IndexAdjust -5
                  }
                  if (RGPSIndex < 1) RGPSIndex = 1
                  else if(RGPSIndex >= nrow(RightGPSData)) RGPSIndex =
nrow(RightGPSData)-10 #the -10 probably isn't necessary


                  #do the search
                  UTCDiff = abs(LeftGPSData$UTC[LGPSIndex] -
RightGPSData$UTC[RGPSIndex])
                  prevUTCDiff = UTCDiff
                  Index = RGPSIndex
                  for (t in RGPSIndex:nrow(RightGPSData)){
                        UTCDiff = abs(LeftGPSData$UTC[LGPSIndex] -
RightGPSData$UTC[t])
                        if (UTCDiff == 0) {
                              Index = t
                              break
                        }
                        else if(UTCDiff < prevUTCDiff) {
                              prevUTCDiff = UTCDiff
                              Index = t
                        }
```

```
                        else if(UTCDiff > prevUTCDiff) break
                  }

                  #assign the index to the GS1 data frame
                  Slist[[L]]$RGPSIndex[i] = Index
            } #end of If statement
            else{ #choose Right GPS
                  Slist[[L]]$TimeDelay[i] = RprevDiff
                  Slist[[L]]$WhichGPS[i] = 'R'
                  Slist[[L]]$RGPSIndex[i] = RGPSIndex

                  #search fo the Left Side GPS with equal or nearest UTC
                  UTCDiff = RightGPSData$UTC[RGPSIndex] - LeftGPSData$UTC[LGPSIndex]
                  IndexAdjust = abs(UTCDiff/0.2) #use an adjusted index for an
efficient starting point to search

                  if(UTCDiff < 1) {
                        LGPSIndex = LGPSIndex - IndexAdjust - 5
                  }
                  else if(UTCDiff > nrow(LeftGPSData)) {LGPSIndex = 1}
                  else {
                        LGPSIndex = LGPSIndex + IndexAdjust -5
                  }
                  if (LGPSIndex < 1) LGPSIndex = 1
                  else if(LGPSIndex >= nrow(LeftGPSData)) LGPSIndex =
nrow(LeftGPSData)-10 #the -10 probably isn't necessary


                  #do the search
                  UTCDiff = abs(RightGPSData$UTC[RGPSIndex] -
LeftGPSData$UTC[LGPSIndex])
                  prevUTCDiff = UTCDiff
                  Index = LGPSIndex
                  for (t in LGPSIndex:nrow(LeftGPSData)){
                        UTCDiff = abs(RightGPSData$UTC[RGPSIndex] -
LeftGPSData$UTC[t])
                        if (UTCDiff == 0) {
                              Index = t
                              break
                        }
                        else if(UTCDiff < prevUTCDiff) {
                              prevUTCDiff = UTCDiff
                              Index = t
                        }
                        else if(UTCDiff > prevUTCDiff) break
                  }

                  #assign the index to the GS1 data frame
                  Slist[[L]]$LGPSIndex[i] = Index
            } #end of else statement
      }#end of else statement - UTC are not equal statement

      Slist[[L]]$SIndex[i] = i
      LstartingIndex = saveLGPSIndex
```

```
      RstartingIndex = saveRGPSIndex
      #if(startingIndex < 1) startingIndex = 1
 }
} #end of for loop



##############################################
########## Save IRT Data To a Table  ##########
##############################################

columnNames = c('Temp','TimeDelay', 'WhichGPS',
 'L_UTC','L_Lat','L_Lon','L_Elev','R_UTC','R_Lat','R_Lon','R_Elev')
S1.15.1Filled = data.frame(matrix(nrow = nrow(S1.15.1), ncol = length(columnNames)))
S2.15.1Filled = data.frame(matrix(nrow = nrow(S2.15.1), ncol = length(columnNames)))
S3.15.1Filled = data.frame(matrix(nrow = nrow(S3.15.1), ncol = length(columnNames)))
S1.2.1Filled = data.frame(matrix(nrow = nrow(S1.2.1), ncol = length(columnNames)))
S2.2.1Filled = data.frame(matrix(nrow = nrow(S2.2.1), ncol = length(columnNames)))
S3.2.1Filled = data.frame(matrix(nrow = nrow(S3.2.1), ncol = length(columnNames)))

SFlist = list(S1.15.1Filled=S1.15.1Filled, S2.15.1Filled=S2.15.1Filled,
 S3.15.1Filled=S3.15.1Filled, S1.2.1Filled=S1.2.1Filled, S2.2.1Filled=S2.2.1Filled,
 S3.2.1Filled=S3.2.1Filled)

for(L in 1:length(SFlist)) {
 colnames(SFlist[[L]]) = columnNames
}

for(L in 1:length(SFlist)){
 if(L==1){
      SFlist[[L]]$Temp = sensorData$Row1IRT15.1

 }
 else if(L==2){
      SFlist[[L]]$Temp = sensorData$Row2IRT15.1
 }
 else if(L==3){
      SFlist[[L]]$Temp = sensorData$Row3IRT15.1
 }
 else if(L==4){
      SFlist[[L]]$Temp = sensorData$Row1IRT2.1
 }
 else if(L==5){
      SFlist[[L]]$Temp = sensorData$Row2IRT2.1
 }
 else if(L==6){
      SFlist[[L]]$Temp = sensorData$Row3IRT2.1
 }


 SFlist[[L]]$TimeDelay = Slist[[L]]$TimeDelay
 SFlist[[L]]$WhichGPS = Slist[[L]]$WhichGPS
 SFlist[[L]][,(length(columnNames)-8):length(columnNames)]=
```

```
 LeftGPSData[Slist[[L]]$LGPSIndex,1:4]
 SFlist[[L]][,(length(columnNames)-
 4):length(columnNames)]=RightGPSData[Slist[[L]]$RGPSIndex, 1:4]

 ##########################################
 #########  Write GS Data to a File #########
 ##########################################
 filedate = substring(filename,1,16) #grab the date
 filename = paste0(filedate,'IRT')
 filename = paste0(filename,names(SFlist[L]))
 filename = paste0(filename,'.txt')
 write.table(SFlist[[L]], file = filename, append = FALSE, quote = TRUE, sep = "\t",
             eol = "\n", na = "NA", dec = ".", row.names = FALSE,
             col.names = TRUE, qmethod = c("escape", "double"),
             fileEncoding = "")
}



##############################################
########## Analyze Time Delays ##############
##############################################
AveTimeDelay = c(mean(Slist$S1.15.1$TimeDelay),mean(Slist$S2.15.1$TimeDelay),
 mean(Slist$S3.15.1$TimeDelay),
 mean(Slist$S1.2.1$TimeDelay),mean(Slist$S2.2.1$TimeDelay),
 mean(Slist$S3.2.1$TimeDelay) )
SDTimeDelay = c(sd(Slist$S1.15.1$TimeDelay),sd(Slist$S2.15.1$TimeDelay),
 sd(Slist$S3.15.1$TimeDelay), sd(Slist$S1.2.1$TimeDelay),sd(Slist$S2.2.1$TimeDelay),
 sd(Slist$S3.2.1$TimeDelay) )
MinTimeDelay = c(min(Slist$S1.15.1$TimeDelay), min(Slist$S2.15.1$TimeDelay),
 min(Slist$S3.15.1$TimeDelay), min(Slist$S1.2.1$TimeDelay),
 min(Slist$S2.2.1$TimeDelay), min(Slist$S3.2.1$TimeDelay) )
MaxTimeDelay = c(max(Slist$S1.15.1$TimeDelay), max(Slist$S2.15.1$TimeDelay),
 max(Slist$S3.15.1$TimeDelay), max(Slist$S1.2.1$TimeDelay),
 max(Slist$S2.2.1$TimeDelay), max(Slist$S3.2.1$TimeDelay) )

TimeDelayTable = rbind(AveTimeDelay,SDTimeDelay,MinTimeDelay, MaxTimeDelay)
colnames(TimeDelayTable) = c("IRT15:1 1", "IRT15:1 2", "IRT15:1 3","IRT2:1 1",
 "IRT2:1 2", "IRT2:1 3")
rownames(TimeDelayTable) = c("Average (ms)", "Standard Deviation (ms)","Minimum
 (ms)", "Maximum (ms)")

##############################################
######## Write Summary Results To File ########
##############################################
 filename = paste0(filedate,'IRTStatSummary')
 filename = paste0(filename,'.txt')
write.table(summaryTable, file = filename, append = FALSE, quote = TRUE, sep = "\t",
             eol = "\n", na = "NA", dec = ".", row.names = TRUE,
             col.names = TRUE, qmethod = c("escape", "double"),
             fileEncoding = "")

write.table(TimeDelayTable, file = filename, append = TRUE, quote = TRUE, sep = "\t",
             eol = "\n", na = "NA", dec = ".", row.names = TRUE,
```

```
            col.names = TRUE, qmethod = c("escape", "double"),
            fileEncoding = "")


##############################################
######## Print Plots and Graphs ########
##############################################
dev.new(width = 8, height = 10)
op=par(mfrow=c(2,2))
plot(rownames(sensorReadPeriods), sensorReadPeriods$R1.15.1, xlab = "Sample Number",
 ylab = "Read Period (ms)", main="Row 1 15:1 IRT Read Periods")
plot(rownames(sensorReadPeriods), sensorReadPeriods$R2.15.1, xlab = "Sample Number",
 ylab = "Read Period (ms)", main="Row 2 15:1 IRT Read Periods")
plot(rownames(sensorReadPeriods), sensorReadPeriods$R3.15.1, xlab = "Sample Number",
 ylab = "Read Period (ms)", main="Row 3 15:1 IRT Read Periods")
par(op)

dev.new(width = 8, height = 10)
op=par(mfrow=c(2,2))
plot(rownames(sensorReadPeriods), sensorReadPeriods$R1.2.1, xlab = "Sample Number",
 ylab = "Read Period (ms)", main="Row 1 2:1 IRT Read Periods")
plot(rownames(sensorReadPeriods), sensorReadPeriods$R2.2.1, xlab = "Sample Number",
 ylab = "Read Period (ms)", main="Row 2 2:1 IRT Read Periods")
plot(rownames(sensorReadPeriods), sensorReadPeriods$R3.2.1, xlab = "Sample Number",
 ylab = "Read Period (ms)", main="Row 3 2:1 IRT Read Periods")
par(op)

dev.new(width = 8, height = 8)
plot(1:length(Slist[[1]]$TimeDelay), Slist[[2]]$TimeDelay, xlab = "Sample Number",
 ylab = "Time Difference (ms)", main="IRT")
```

Appendix B - Statistical Analysis Results Not Included in Main Text

Table B.1 Summary statistics of Crop Circle Channel 2 reflectance vs. light intensity linear regression model

	Dependent variable: Crop Circle Channel 2			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	2.023e-01	1.195e-04	1692.65	<2e-16
LI	-4.199e-08	1.812e-09	-23.17	<2e-16
Observations	3,614			
R^2	0.129			
Adjusted R^2	0.129			
Residual Std. Error	0.004 (df = 3612)			
F Statistic	536.823^{***} (df = 1; 3612) (p = 0.000)			
Note:	*p **p ***p<0.01			

Table B.2 Summary statistics of Crop Circle Channel 2 reflectance vs. ambient temperature linear regression model

	Dependent variable: Crop Circle Channel 2			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	1.947e-01	3.657e-04	532.57	<2e-16
AmbTemp	3.596e-04	2.441e-05	14.73	<2e-16
Observations	3,614			
R^2	0.057			
Adjusted R^2	0.056			
Residual Std. Error	0.004 (df = 3612)			
F Statistic	217.082^{***} (df = 1; 3612) (p = 0.000)			
Note:	*p **p ***p<0.01			

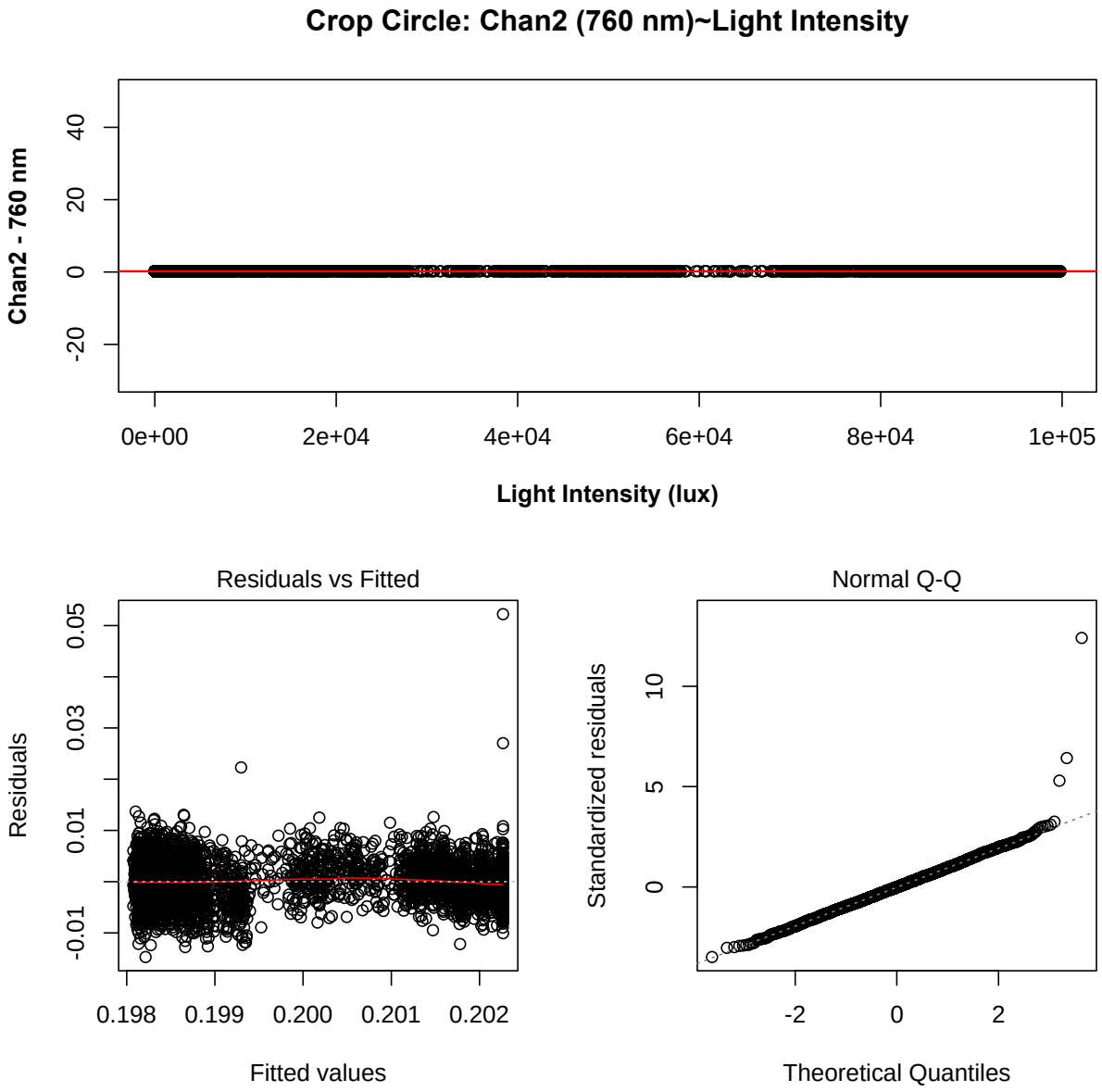

Figure B.1 Fitted and residual plots of Crop Circle Channel 2 reflectance (760 nm) vs. ambient light intensity

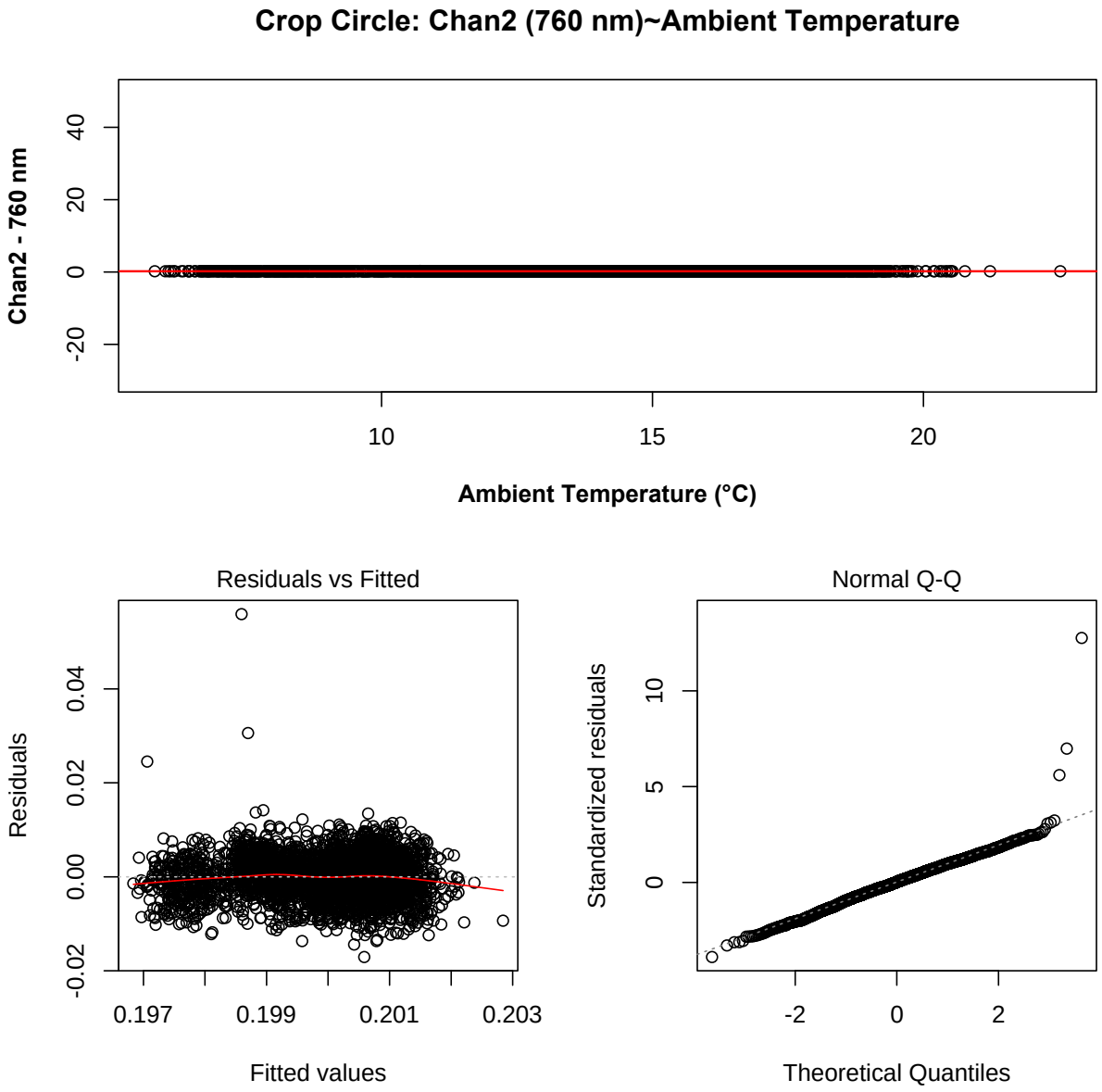

Figure B.2 Fitted and residual plots of Crop Circle Channel 2 reflectance (760 nm) vs. ambient temperature

Table B.3 Summary statistics of Crop Circle Channel 3 reflectance vs. light intensity linear regression model

	Dependent variable: Crop Circle Channel 3			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	1.372e-01	1.476e-04	929.72	<2e-16
LI	-2.952e-08	2.239e-09	-13.19	<2e-16
Observations	3,614			
R^2	0.046			
Adjusted R^2	0.046			
Residual Std. Error	0.005 (df = 3612)			
F Statistic	173.858^{***} (df = 1; 3612) (p = 0.000)			
Note:	*p **p ***p<0.01			

Table B.4 Summary statistics of Crop Circle Channel 3 reflectance vs. ambient temperature linear regression model

	Dependent variable: Crop Circle Channel 3			
	Estimate	***Std. Error***	***t value***	***Pr(>|t|)***
(Intercept)	1.377e-01	4.428e-04	311.098	<2e-16
AmbTemp	-1.432e-04	2.955e-05	-4.846	1.31e-06
Observations	3,614			
R^2	0.006			
Adjusted R^2	0.006			
Residual Std. Error	0.005 (df = 3612)			
F Statistic	23.486^{***} (df = 1; 3612) (p = 0.00001)			
Note:	*p **p ***p<0.01			

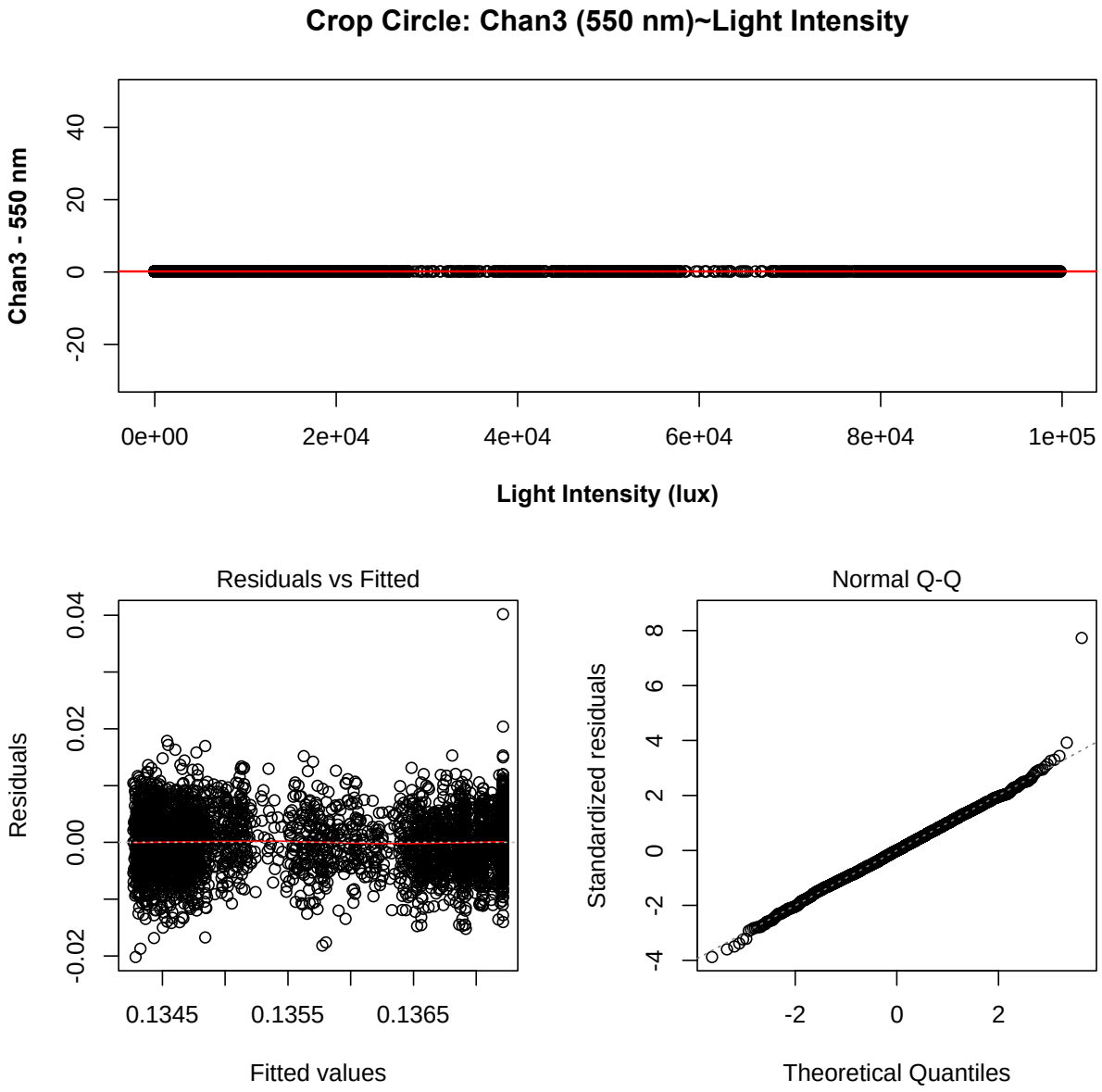

Figure B.3 Fitted and residual plots of Crop Circle Channel 3 reflectance (550 nm) vs. ambient light intensity

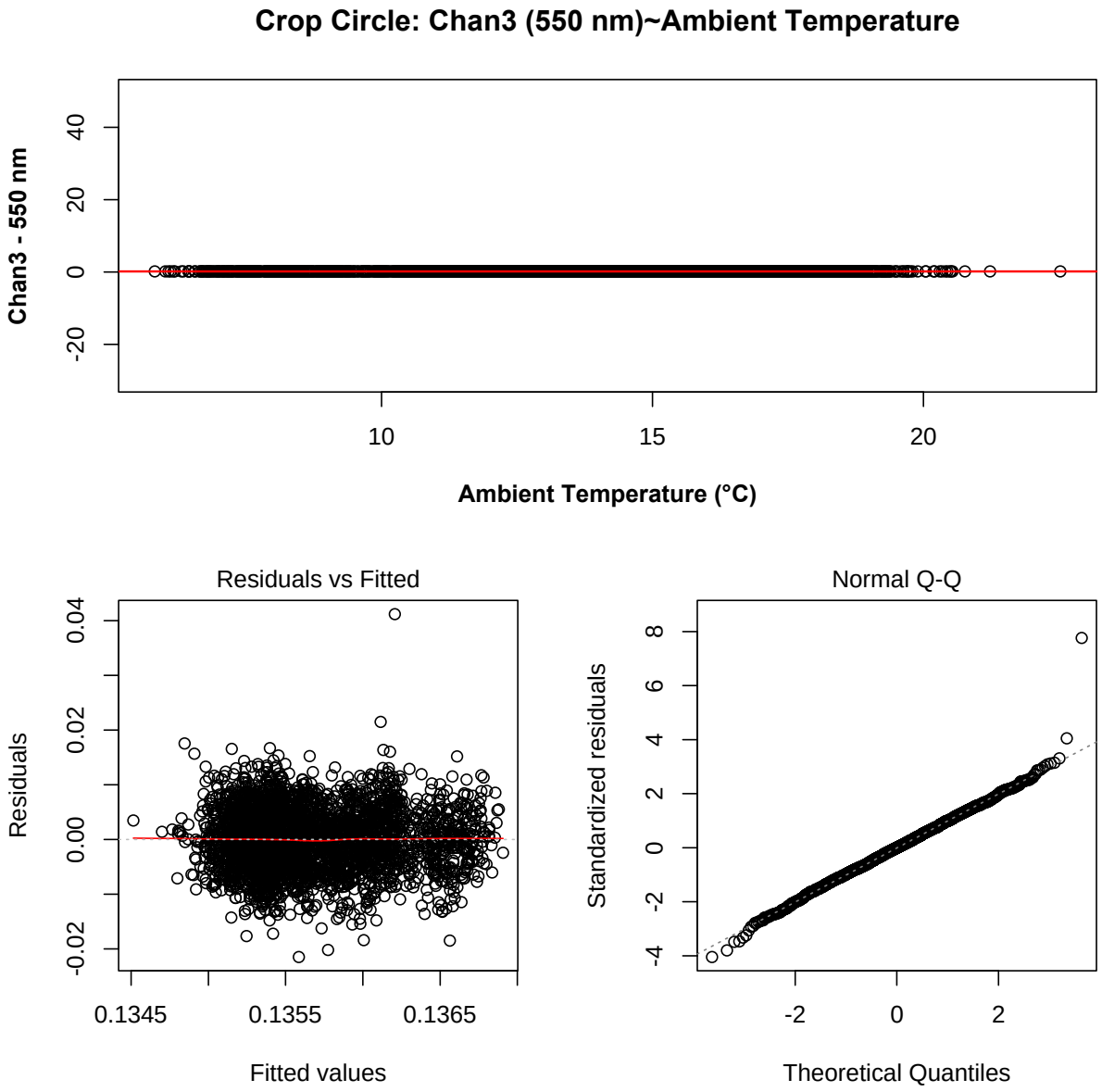

Figure B.4 Fitted and residual plots of Crop Circle Channel 3 reflectance (550 nm) vs. ambient temperature

Appendix D - LabVIEW Code

GPSRead_v3_TimeStamp.vi

Reads and parses the GNSS data strings from the FmX units. It returns the UTC Time, Latitude, Longitude, Elevetion, Time In (time when VI began waiting for data), Time Read (time VI received the data).

Connector Pane

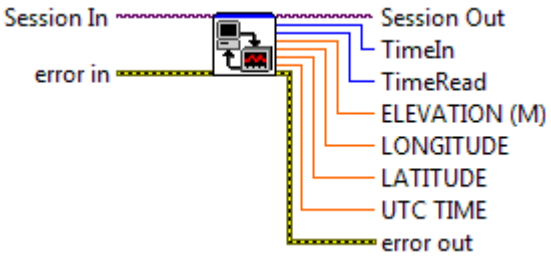

Figure D.1 GNSS LabVIEW module connector pane

Front Panel

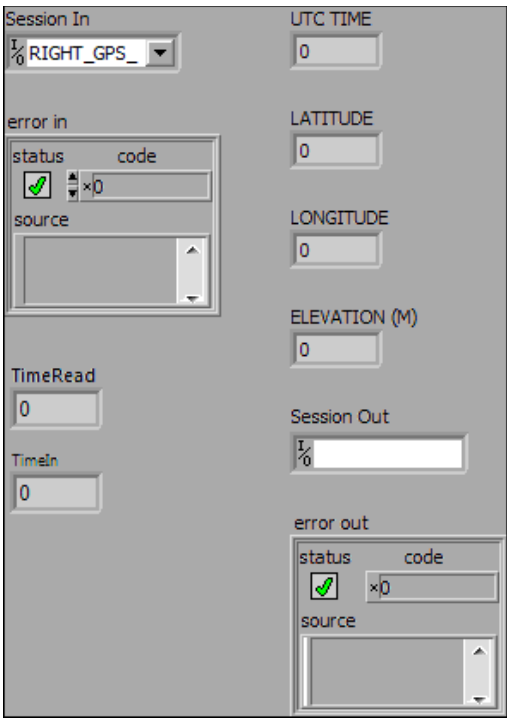

Figure D.2 GNSS LabVIEW module front panel

GPSRead_v3_TimeStamp.vi

Block Diagram

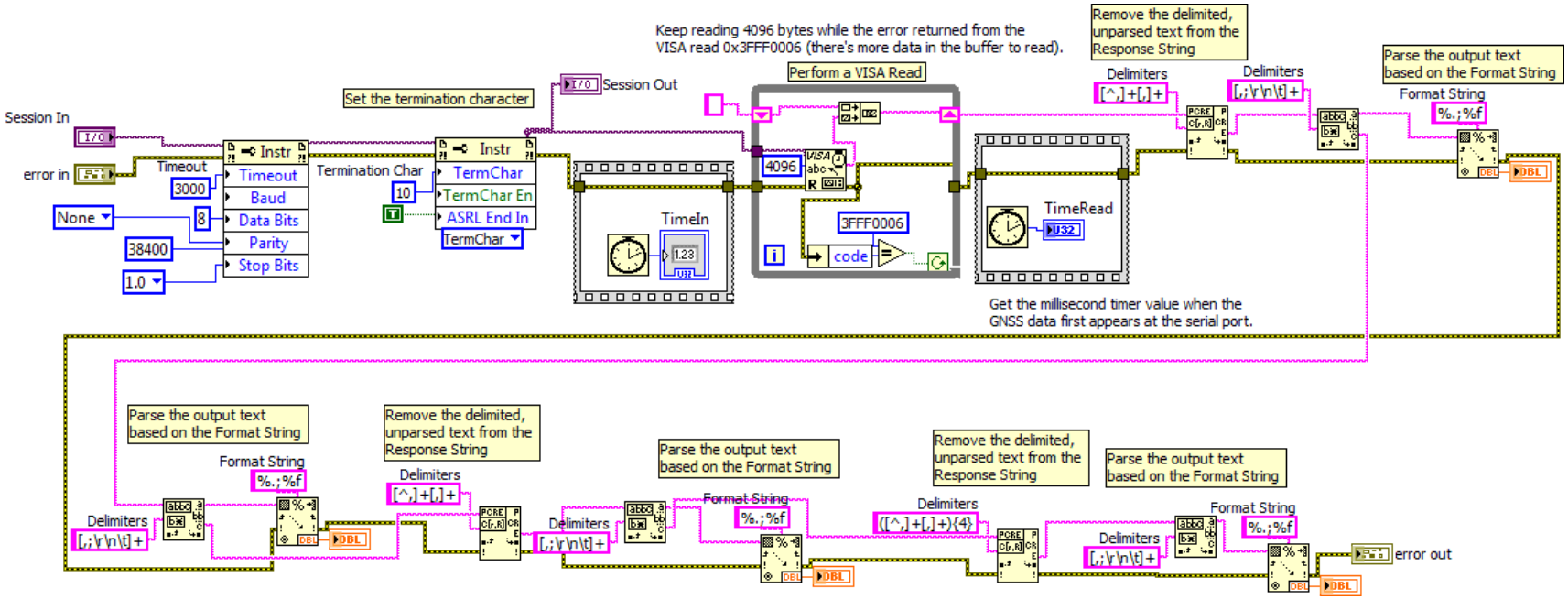

Figure D.3 GNSS LabVIEW module block diagram

GreeSeekerRead_ReturnAll_v1.vi

Receives and parses the data string from the GreenSeeker sensor. It outputs the GreenSeeker data timestampe, the Trigger Count, the Sample Count, the NDVI, and a VI.

Connector Pane

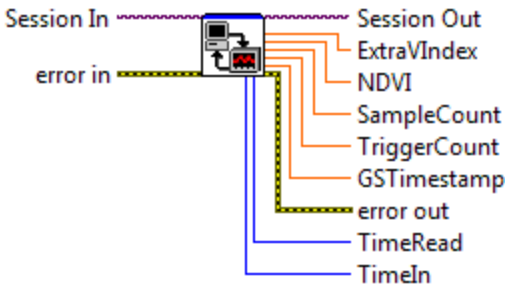

Figure D.4 GreenSeeker module connector pane

Front Panel

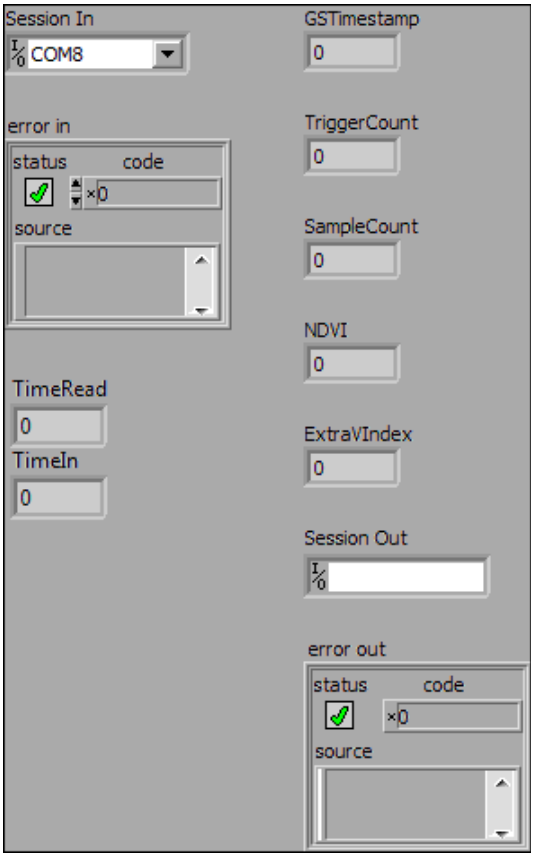

Figure D.5 GreenSeeker module font panel

Block Diagram

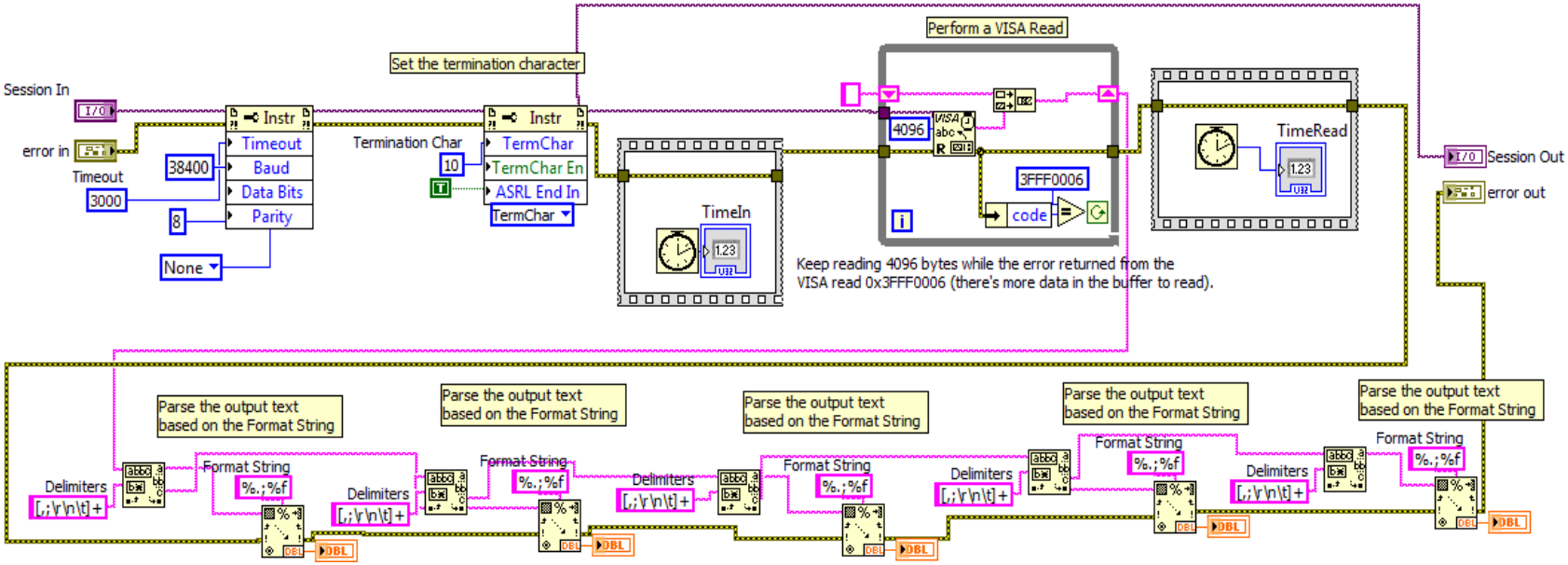

Figure D.6 GreenSeeker module block diagram

IRT.vi

Sends the read command (0x01) to the Micro Epsilon 15:1 IRT, parses the IRTs response, and converts it to the temperature in degrees C.

Connector Pane

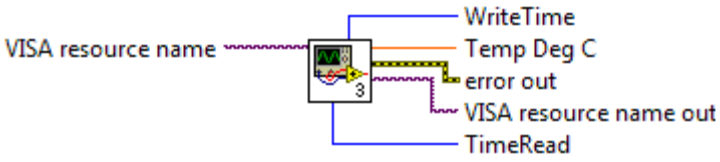

Figure D.7 15:1 IRT module connector pane

Front Panel

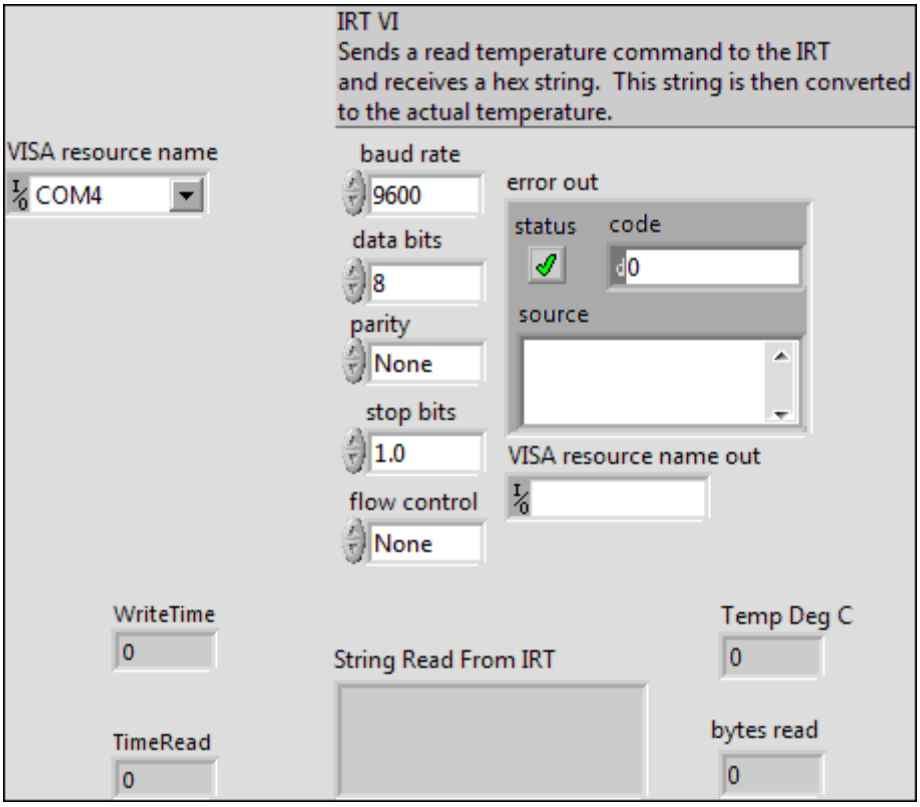

Figure D.8 15:1 IRT module front panel

Block Diagram

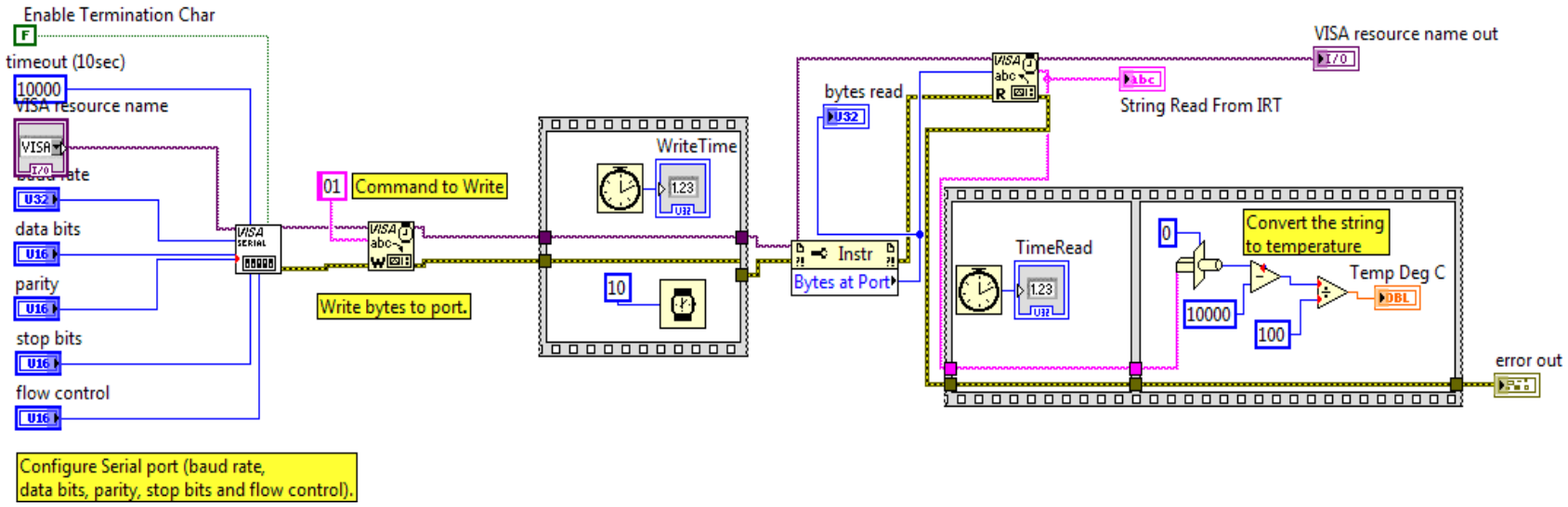

Figure D.9 15:1 IRT module block diagram

IRT_2-1_v2.vi

Sends the read command (0x01) to the Micro Epsilon 2:1 IRT, parses the IRTs response, and converts it to the temperature in degrees C.

Connector Pane

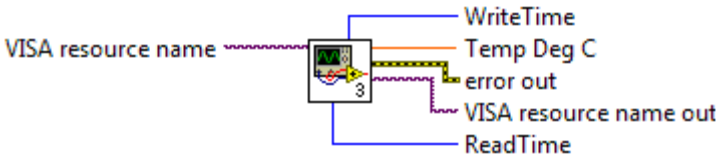

Figure D.10 2:1 IRT module connector pane

Front Panel

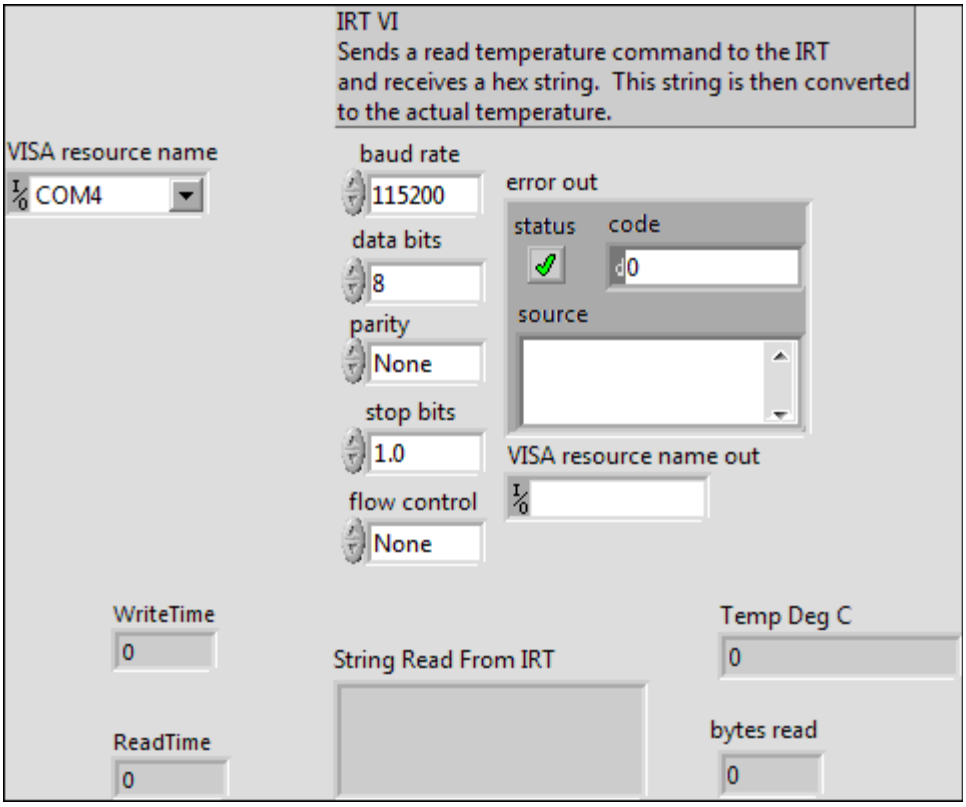

Figure D.11 2:1 IRT module front panel

IRT_2-1_v2.vi

Block Diagram

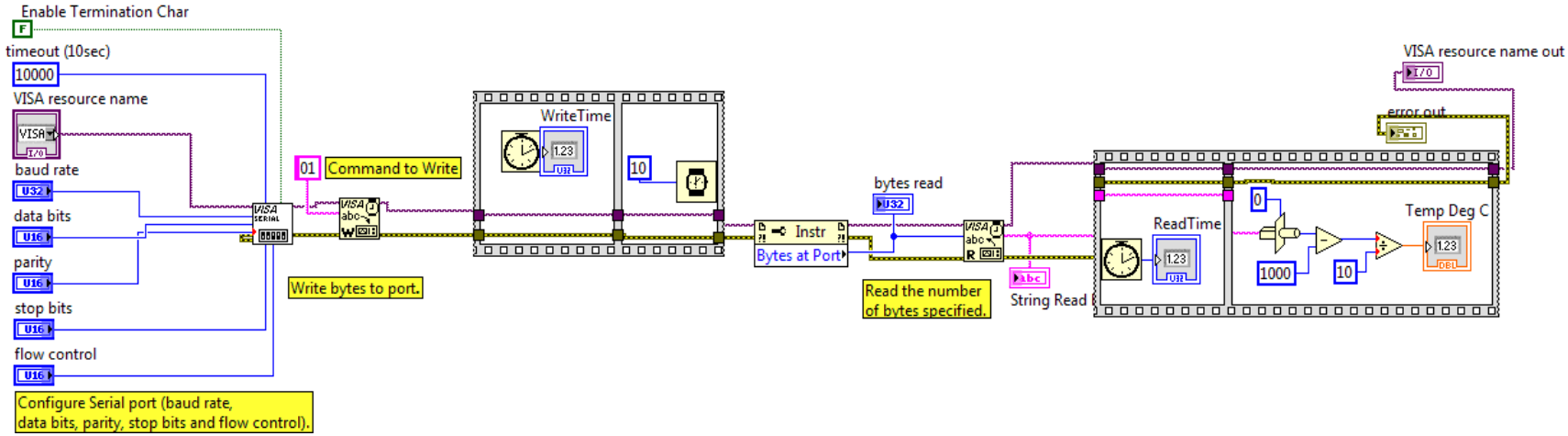

Figure D.12 2:1 IRT module block diagram

CropCircleRead_v3.vi

Receives data from a Crop Circle sensor, parses it, and returns the first calculated index, the second calculated index, the raw value from Channel 1, the raw value from Channel 2, and the raw value from Channel 3.

Connector Pane

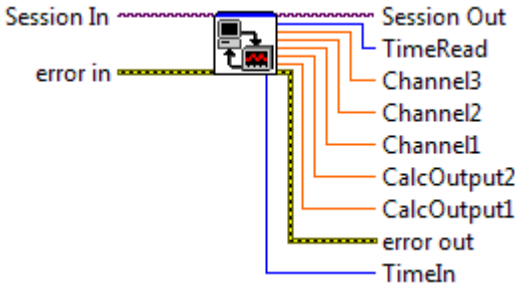

Figure D.13 Crop Circle module connector pane

Front Panel

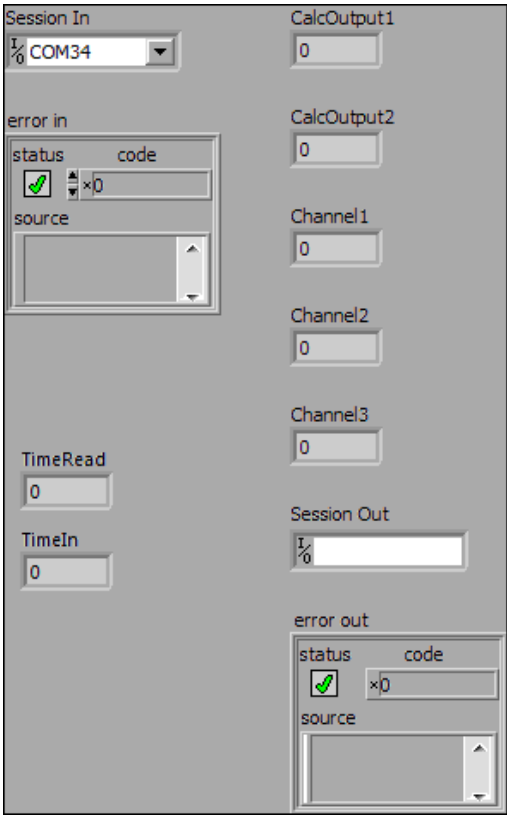

Figure D.14 Crop Circle module connector pane

CropCircleRead_v3.vi

Block Diagram

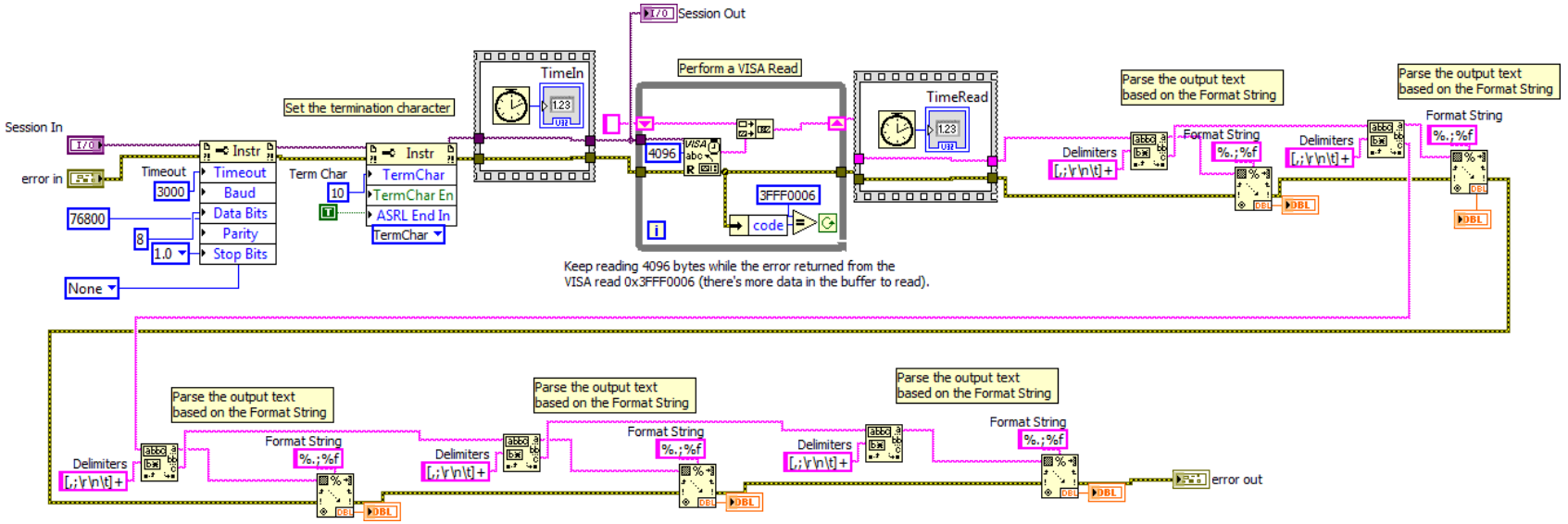

Figure D.15 Crop Circle module connector pane

DistanceSensorRead_v2_ReplacedwithDAQAssistant.vi

Connects to the NI 9207 DAQ and outputs the voltages from the three ultrasonic sensors and the laser sensor.

Connector Pane

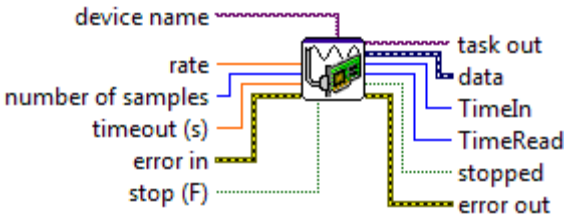

Figure D.16 Distance sensor module connector pane

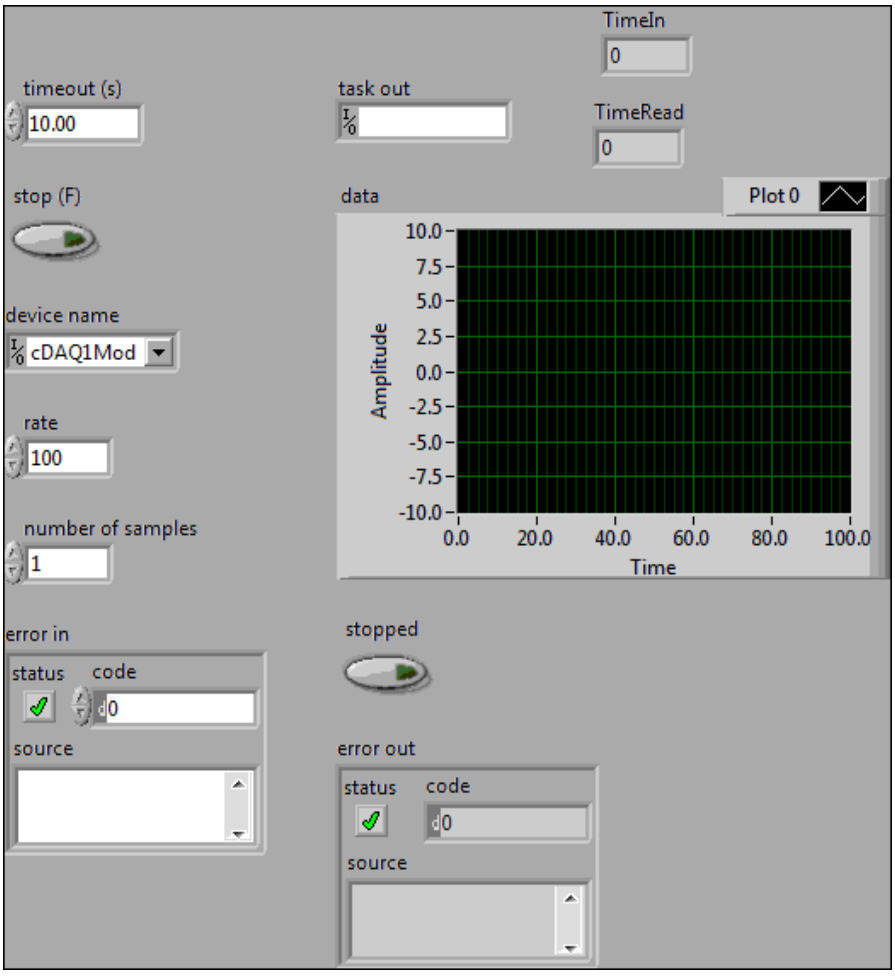

Figure D.17 Distance sensor module front panel

DistanceSensorRead_v2_ReplacedwithDAQAssistant.vi:1

Block Diagram

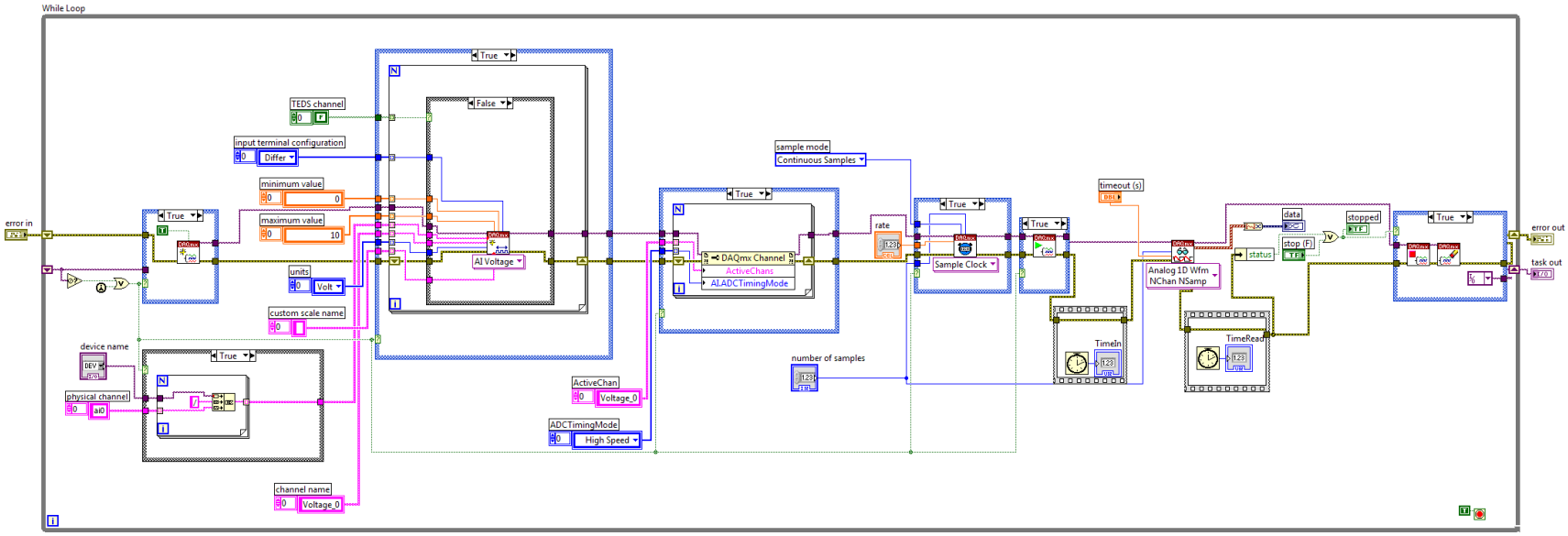

Figure D.18 Distance sensor module block diagram

Main Program

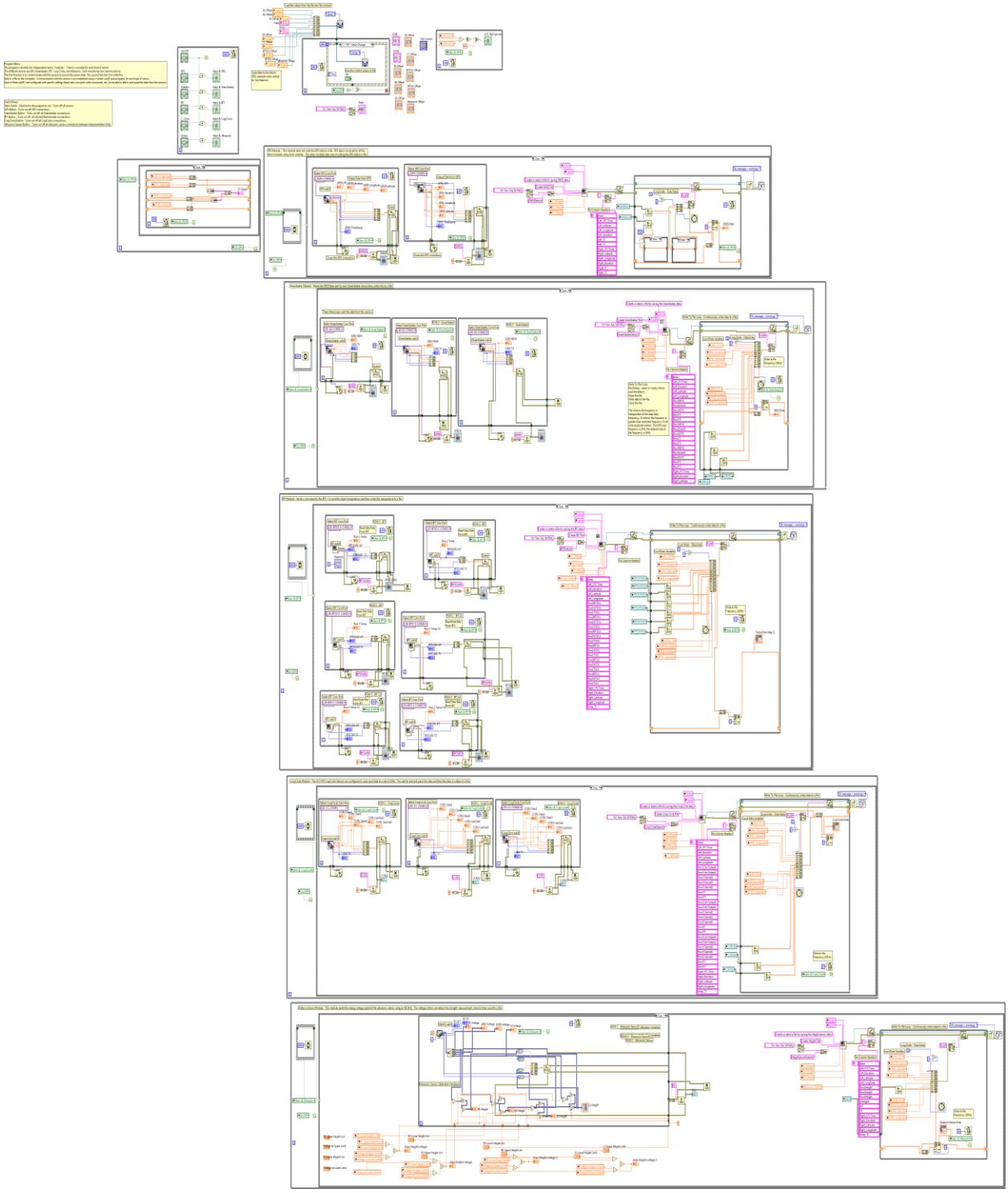

Figure D.19 Main program block diagram

Appendix E - Sample Phenotyper Software Output File

Table E.1 Columns from the GreenSeeker output data file containing a sample of data from the GreenSeeker data file created by phenotyping software. More columns exist than can be displayed on a single sheet of paper. The first two rows contain sensor set locations on the sensor tool bar. The OffsetFromBar is the position of the sensor perpendicular to the toolbar. The remaining rows contain the sensor, position, and timestamp data.

Table E.1 Columns from the GreenSeeker output data file

Row1	Row2	Row3	OffsetFromBar							
0	0	0	0							
Index	Left_UTCTime	Left_Elevation	Left_Latitude	Left_Longitude	Row1NDVI	Row1ExtraVI	Row1GSTS	Row1TI	Row1TS	Row2NDVI
1	164953.6	326.16	3911.411526	9634.989255	0.116	0.116	1714810	1998153	1998154	-0.013
2	164953.6	326.16	3911.411526	9634.989255	0.11	0.11	1714934	1998154	1998154	-0.013
3	164953.6	326.16	3911.411526	9634.989255	0.113	0.113	1715058	1998298	1998299	-0.013
4	164953.6	326.16	3911.411526	9634.989255	0.114	0.114	1715182	1998300	1998300	-0.013
5	164953.6	326.16	3911.411526	9634.989255	0.113	0.113	1715306	1998442	1998442	-0.014
6	164953.6	326.16	3911.411526	9634.989255	0.114	0.114	1715430	1998586	1998586	-0.012
7	164953.8	326.16	3911.411526	9634.989255	0.112	0.112	1715554	1998586	1998587	-0.014
8	164953.8	326.16	3911.411526	9634.989255	0.109	0.109	1715678	1998600	1998737	-0.013
9	164953.8	326.16	3911.411526	9634.989255	0.113	0.113	1715802	1998737	1998889	-0.013
10	164953.8	326.16	3911.411526	9634.989255	0.113	0.113	1715926	1998890	1998966	-0.013

Appendix F - CR850 Program For Measuring Temperature

This program was written to measure temperature with thermcouples connected to a Campbell Scientific CR850 data logger. The program reads temperature data once every second and transmits it via the serial port.

```
Public PTemp
Public SurfaceTemp
Public AmbientTemp

Const TERMCHAR = 35
'Main Program
BeginProg
  SerialOpen(ComRS232,9600,0,0,500)
      Scan (1,Sec,0,0)
            PanelTemp(PTemp,_60Hz)
            TCDiff(SurfaceTemp,1,mV7_5,1,TypeT,PTemp,1,0,_60Hz,1.0,0)
            TCDiff(AmbientTemp,1,mV7_5,2,TypeT,PTemp,1,0,_60Hz,1.0,0)
            SerialOut (COMRS232,SurfaceTemp,"",0,0)
            SerialOut (COMRS232,",","",0,0)
            SerialOut (COMRS232,AmbientTemp,"",0,0)
            SerialOut (COMRS232,CHR(10)+CHR(13),"",0,0)
      NextScan
EndProg
```