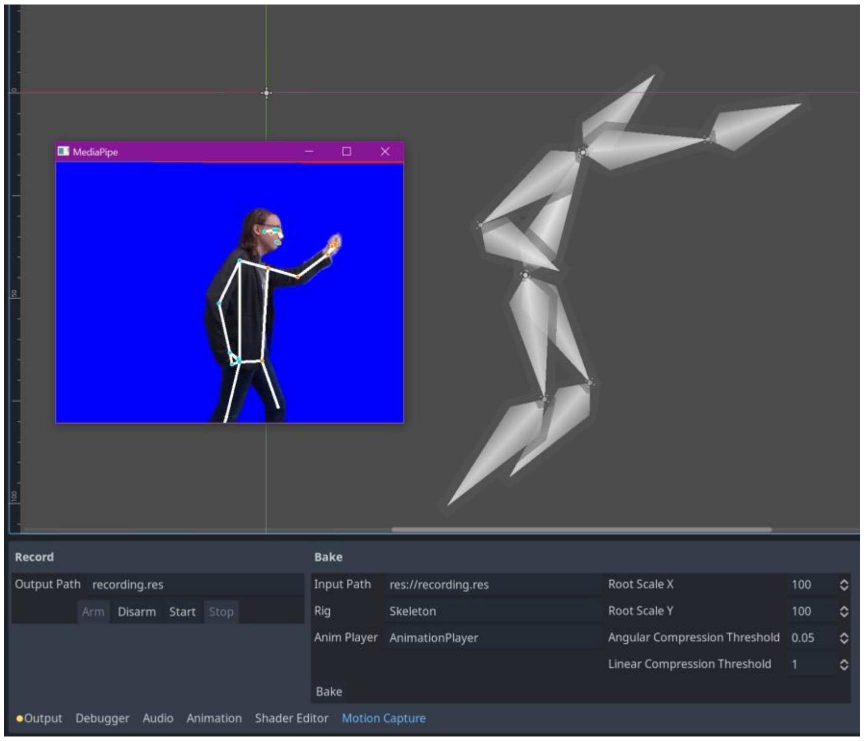
PipeDoll: A Motion Capture Addon for the Godot Game Engine

Ethan Tucker



Output Path recording.res
Input Path res://recording.res
Rig Skeleton
Anim Player AnimationPlayer
Root Scale X 100
Root Scale Y 100
Angular Compression Threshold 0.05
Linear Compression Threshold 1

Contents

Summary

Character animation is a key challenge in game development and other fields and is often created using motion capture. However, traditional motion capture requires specialized hardware and can be inaccessible for small teams. For this project, I have developed a 2D motion capture addon for the Godot game engine. The addon uses a machine learning model to perform motion capture using a standard webcam, avoiding the need for hardware. It meets all the key functional requirements identified for the project and is published as open-source software at https://github.com/ectucker1/pipedoll.

Introduction

Problem

Character animation for game development or other purposes can be an extremely time consuming process. In traditional animation, every frame had to be hand drawn with small changes to imply motion. Modern computer animation simplifies this process by allowing the keyframing of object transformations and interpolation between them. However, these keyframes can still be extremely time consuming to create. Large quantities of animations can be expensive to produce.

Many game studios have addressed this challenge by applying motion capture technology. Motion capture simplifies the animation process by recording the movement of a human actor and then applying it to an animated character. This performance often still needs to be tweaked by an animator, but it provides a strong starting point for animation work. Motion capture can be inaccessible, however, due to requirements for specialized suits, depth cameras, or other additional equipment. This makes it harder to use for independent developers or small sized studios.

Machine learning and computer vision provide an alternative to this specialized equipment. Google's MediaPipe library [1] provides an API that can identify the positions of key landmarks on a body using footage from a standard webcam. This data is not ideally formatted for use in animation, however,

Project Goals

To solve this problem, I have set out to develop a motion capture animation tool using the MediaPipe library. I have limited the scope to 2D animation, as MediaPipe's primary outputs are in 2D. I have also chosen to develop the tool as an addon for the Godot Game Engine [2]. This allows for the tool to be used within the existing UI where animators would work.

Essential functional requirements for the add-on include:

- Pose data provided by MediaPipe can be captured and saved to a recording over time.
- Recorded pose data can be applied to a skeletal rig (the setup of bones and joints used to control the character) to create an animation.

- Animations can be adjusted after recording.
- A single motion capture recording should be reusable across multiple character rigs.
- The final animation should use the existing engine animation tools and not depend on the addon at runtime.

The key non-functional requirement of the software is usability. Animators should be able to easily use the motion capture tools and quickly jump back and forth between motion capture and tweaking the resulting animation.

The resulting plugin will be published as free and open-source software.

Prior Work

Several prior projects have attempted to apply the MediaPipe library in Godot. Several of these apply it for the purpose of a movement-drive control scheme ([3], [4], [5]). Others such as [6] attempt to provide a generic interface between MediaPipe and Godot, suitable for multiple use cases. Some of this project's work connecting to the library is derived from this. My own work in [7] applied MediaPipe to the character animation in a 2D fighting game. This form of animation was limited to cropping the recorded video, rather than animating a standard character rig.

Existing software such a Reallusion's Motion Live 2D [8] provides 2D motion capture. It still requires the use of specialized hardware for pose recording. FreeMoCap [9] provides for webcam-based motion capture but requires multiple cameras to do this. It is also focused entirely on 3D rigs.

Design

Addon Structure

The first key design decision for this project was how to connect to the engine. I have referred to the project previously as an "addon," but Godot actually has several options for adding features to the engine with varying tradeoffs.

1) Modules – modules are written in C++ and can extend the engine in nearly any way. Many core engine features are developed as modules. However, they require recompilation of the complete engine binary.
2) Plugins – plugins are written in Godot's GDScript scripting engine. They can add additional UI elements to the engine editor.
3) Extensions – extensions are a new feature of Godot's 4.0 release and provide a way to load dynamically linked C++ libraries. The API is similar to that of modules, but slightly more limited.

Based on these tradeoffs I decided to develop this addon as a combination of an extension and a plugin. The extension links to MediaPipe's C++ library and implements the more performance intensive project components using this. It also exposes an API to use the new functionality from GDScript. This API is then used by the plugin to implement the user interface.

User Interface Design

The user interface of PipeDoll is split into three primary components:

1) The interface for recording motion capture data.
2) The interface for rigging a character model.
3) The interface for baking (saving data to be used by the game later) motion capture data to animate a rigged model.

The recording UI (Figure 1) is contained in a dock at the bottom of the editor. It provides basic controls to create a motion capture recording. The user can set a path to a file where recorded data will be saved. Then they can "arm" the recording to initialize the MediaPipe model and show a camera preview. Recording begins when the Start button is pressed, and the results are saved to a .res file when Stop is pressed.

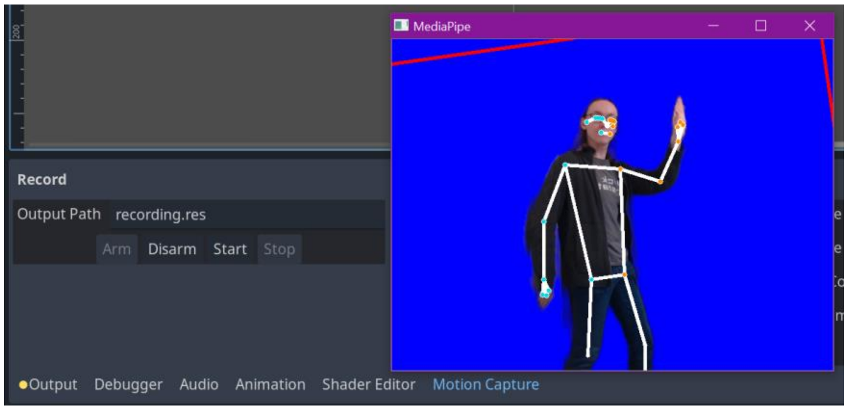
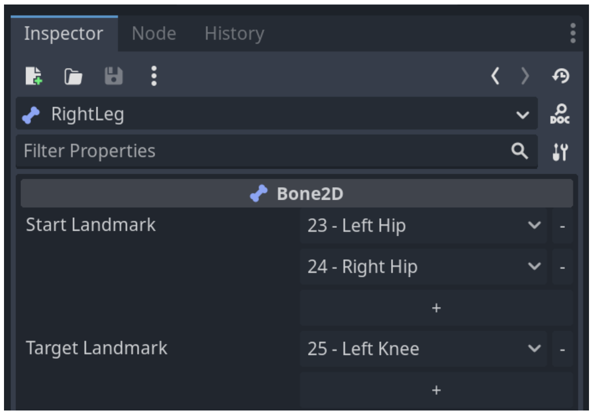

Record
Output Path recording.res

Figure 1 - Recording UI

The rigging UI (Figure 2) is displayed within the property inspector for the Bone2D node. It provides the ability to set a number of Start Landmarks and Target landmarks for the bone, which determine how it will be transformed by the motion data.



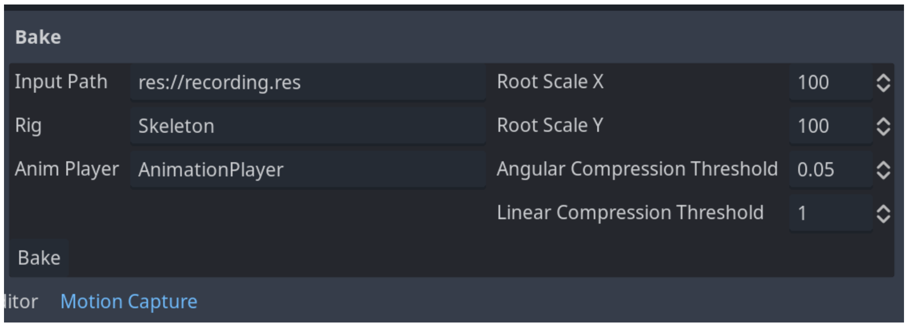
RightLeg
Filter Properties
Bone2D
Start Landmark 23 - Left Hip
24 - Right Hip
Target Landmark 25 - Left Knee

Figure 2 - Bone Rigging UI

The baking UI (Figure 3) is displayed in the bottom dock, adjacent to recording. The key controls allow the user to set the following key parameters:

- The path to the recorded motion capture data.
- The path to the rigged skeleton to animate.
- The path to the animation player to save animation data to.

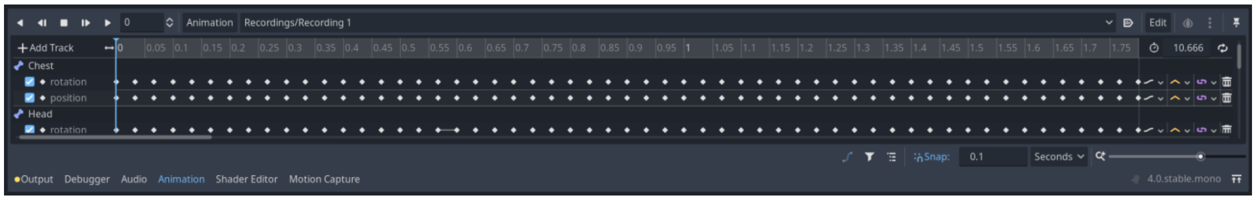
The user can then press the Bake button, which will add a new animation to the animation player which animates the transformation of the rigged skeleton over time. It is then possible to tweak this animation within Godot's built-in animation editor (Figure 4).



Bake

Input Path	res://recording.res	Root Scale X	100
Rig	Skeleton	Root Scale Y	100
Anim Player	AnimationPlayer	Angular Compression Threshold	0.05
		Linear Compression Threshold	1

Figure 3 - Baking UI



0 Animation Recordings/Recording 1

Chest: rotation, position
Head: rotation

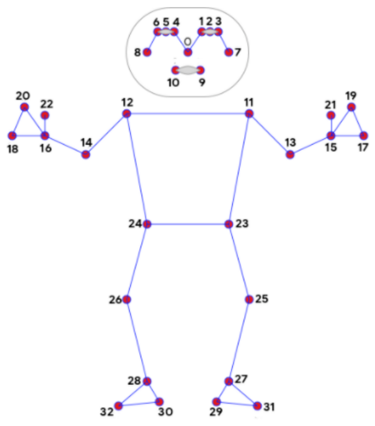
Snap: 0.1 Seconds

10.666

Figure 4 - Godot Animation Editor

Rigging

MediaPipe returns its pose data as independent positions of landmarks on the body. Distances between landmarks are not guaranteed to remain constant. In contrast, a traditional skeletal animation is defined by rotations on a hierarchy. For instance, if the upper leg is rotated the lower leg is expected to rotate with it.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Figure 5 - MediaPipe Pose Landmarks. From [1].

The animation baking process must convert the positional data into something that works with a hierarchy. My implementation solves this through the use of dynamic programming. The root bones are calculated first, with their rotation set to the angle between two user-specified landmarks. Children of that bone are then rotated similarly, but their final angle is relative to the angle of their parent. This repeats recursively until every bone in the skeleton is rotated correctly. Linear movement is also applied to the root bone to enable movement of the complete body, such as when walking or jumping.

Results

The developed addon successfully implements all the functional requirements:

- Pose data can be saved to .res files.
- The baking UI applies that data to a skeleton.
- Animations are saved to Godot's built-in format and can thus be edited by the existing tools.
- The same .res file can be reloaded for multiple bakes.
- The final animation uses only the existing parameters of Godot's format and requires no additional code for playback.

The integration with the engine editor should go a long way towards meeting the nonfunctional requirement of usability. There are also lots of minor usability features included, such as dragging and dropping paths to nodes instead of requiring them to be manually specified.

The final addon is published as open-source software at https://github.com/ectucker1/pipedoll. That repository contains the project source code, a compiled version of the extension, and an example project. A user guide to get started using the tool is hosted at https://github.com/ectucker1/pipedoll/wiki/PipeDoll-Tutorial.

References

[1] Google, "MediaPipe," 2023. [Online]. Available: https://developers.google.com/mediapipe.

[2] J. Linietsky, A. Manzur and c. , "Godot Engine," 2023. [Online]. Available: https://godotengine.org/.

[3] J. Karlsson, "2D Beats: Fast paced full body movement game using Godot Engine," 2022.

[4] S. Karlsson, "Cart: Car driving game using Godot and ML-based full-body interaction," 2022.

[5] L. Murphy and J. Malloch, "IOMapper: The Integration of Generalized Signal Control Mapping With the Godot Game Engine," 2021.

[6] P. Kumar, "godot_mediapipe_module," 2022. [Online]. Available: https://github.com/purgeme/godot_mediapipe_module.

[7] E. Tucker and L. Weeks, "MP Fighter," 2022. [Online]. Available: https://devpost.com/software/mp-fighter.

[8] Reallusion, "Motion LIVE 2D," [Online]. Available: https://www.reallusion.com/cartoon-animator/full-body-mocap/default.html.

[9] J. Matthis, "The FreeMoCap Project," [Online]. Available: https://freemocap.org/.