Final Report - CS5641

Running Simulator

Matthew Spencer

mpspence@mtu.edu

04/25/2020

Index

Abstract

This project was implemented using an Arduino Uno Wi-Fi Rev 2 microcontroller board for its native WiFi networking capabilities and internal inertial measurement unit (IMU). The Arduino is strapped onto a person's knee to record stepping movements using the IMU accelerometer data. This data is then sent over a 2.4GHz WiFi network from the Arduino into an external virtual reality[1] software application built using the Unreal Engine 4 (UE4) game engine.

The application's goal is to simulate walking or running within a virtual world created using UE4 by walking or running in-place. This world is then displayed to the user through a first-person viewpoint using a television monitor. [Fig. 1] The speed at which the user travels in the virtual world is directly correlated with how fast they are "walking" or "jogging" in-place according to the Arduino's IMU accelerometer output.

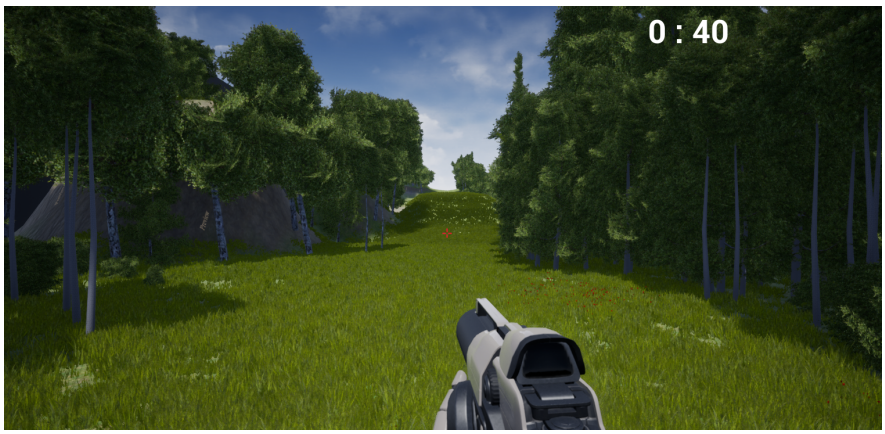

Figure 1: UE4 Running Course

A simple running course was created using open-source UE4 assets. The application has the user run in a straight line from a starting platform to a secondary platform at each ends of the virtual world. The time it takes the user to complete is tracked by a timer in the upper right hand corner of the screen that starts and stops accordingly.

[1] A head-mounted-display is not used in the final project even though it was outlined in the original proposal.

Resources

- Arduino Uno Wifi Rev 2 [1]
- Duracell 9 Volt Alkaline Battery
- Windows 10 Operating System
- Arduino IDE Software
- Visual Studio 2017[2] [2]
- Unreal Engine 4 version 4.24.3[3] [3]

Figure 2: Arduino Uno Wifi Rev 2

- Intel i5 6500 3.20 GHz Quad Core CPU
- 2 x 8GB DRR4 2133 MT/s
- Zotac GeForce GTX 1070 Mini 8GB GDDR5
- Futuro Sport Moisture Control Knee Brace[4]

[2] C++ programming for UE4 is directly integrated into Visual Studio 2017. This allows fast compilations of UE4's binary engine code for on-the-fly changes that can be seen almost immediately.
[3] For the UE4 editor to perform well, the development computer must meet a minimum spec requirement outlined by the creators of the engine, Epic Games Inc. The hardware used for the development of this project exceeds the developer's minimum recommendations.
[4] This is a standard over-the-counter cloth knee brace that can be purchased at any nearby drug store. An improvement for future iterations would be to use a 3D-printed enclosure.

Theory

It is predicted that combining the methods and technologies described below will increase a user's sense of immersiveness within a VR environment, while also giving them an alternative way to exercise within the comfort of their homes.

An Arduino is to be attached onto the user's leg with the x-axis pointing perpendicular to the ground.[5] The on-board IMU device is then initialized and used to calculate how fast the user is running in-place by collecting the x-axis acceleration data. The acceleration value is then read and assigned a value between 1 and 5 depending on the acceleration's magnitude. These values have acceleration thresholds that correspond to if the user is stopped, walking, jogging, running, or sprinting. After this, a value is sent over a private UDP WiFi connection initialized and broadcasted by the Arduino device. This signal is then picked up by the UE4 application running on a Windows 10 computer which connects to the UDP network.

The user begins the UE4 simulation by pressing start when prompted, and then controlling their virtual avatar by "running" in-place by bringing their knees up in front of their chest in rapid succession. This exercise is called "high-knees". To encourage the user to run faster through the simulated world, a stopwatch timer is placed in the top right corner of the screen.

To prevent sporadic Arduino acceleration values from affecting the movement within the UE4 simulation the following problems must be addressed; the zero acceleration values when taking a step with the leg that the Arduino is not attached to, and when the attached knee is at the peak of its stride. To solve these issues, buffer frames are added or subtracted to continue movement depending on the most recent positive acceleration values received. This frame buffer is used to help create a more immersive experience when walking or running in the simulated world.

[5] The x-axis on the Arduino runs parallel to its power terminal, which is parallel to its longest side.

Execution

Downloading the Arduino IDE is the first step, as this proprietary software easily recognizes the correct Arduino device over a Windows USB serial connection. This enables custom C code to be written in the IDE and uploaded onto the board, easily programming the device. The IDE provides an essential streamlined approach for accessing the Arduino's internal IMU and WiFi modules with its built-in C libraries. [4, 5]

The Arduino is initialized as a UDP client device broadcasting a private network that contains its own personal SSID and password. The device then sends out 1-byte characters repeatedly over the network ranging from 1 to 5 depending on the acceleration being recorded by the IMU.

After the Arduino is set up, a blank first-person UE4 default template is created and used as the starting point for the virtual world on the Windows 10 host computer. This template is then paired with Windows Visual Studio (VS) C++. Within VS, a C++ server [6] is implemented to retrieve the Arduino's messages and immediately export them to UE4's Blueprint schematics. [7] This is done by compiling UE4's binary code within VS, and making a separate "Actor" class to call the server functions within the editor through Blueprints. [Fig 3]

```
8   #include "Arduino_Functions.generated.h"
9
10  UCLASS()
11  class RUNNING_SIM_API AArduino_Functions : public AActor
```

Figure 3: Creating an "Actor" Class. Location: "./Source/Running_Sim/Arduino_Functions.h"

UE4's Blueprints are then used to virtually control the avatar's controller movement inputs, and is set to move the avatar forward at a certain Velocity according to the acceleration input received from the Arduino. To prevent glitching where the avatar moves forward and stops every frame[6], a smoothing algorithm was put into place within the Blueprints player script. [Fig. 5]

[6] The UE4 editor was set to operate at 120 frames per second rather than the typical 60 frames per second.

Smoothing Algorithm Pseudo-Code

- Bool $Moving \leftarrow False$; // See if Player is moving forward.
- Int $F_{Max} \leftarrow 0$; // Set how many frames the player moves at $Velocity$.
- Int $F_{current} \leftarrow 0$; // Iterator to keep track of the current frame
- Float $Velocity \leftarrow 0$; // Number set between 0 and 1 used to determine player speed
- Char $MsgRecv \leftarrow 0$; // Buffer to hold received messages 1 - 5 from Arduino. (char)
- Do Every Frame:
    - While ($F_{current} < F_{max}$) do
        - If $Moving == True$ then
            - $MovePlayerForward() \leftarrow Velocity$
            - $F_{current} = F_{current} + 1$;
    - $F_{current} \leftarrow 0$;
    - $Moving \leftarrow False$
    - If $Moving == False$then
        - $MsgRecv \leftarrow ReadFromArduino()$;
        - If $MsgRecv > 0$then
            - Switch ($MsgRecv$)
                - If 1: $Velocity \leftarrow 0.5$; $F_{max} \leftarrow 60$;
                - If 2: $Velocity \leftarrow 0.6$; $F_{max} \leftarrow 90$;
                - If 3: $Velocity \leftarrow 0.75$; $F_{max} \leftarrow 120$;
                - If 4: $Velocity \leftarrow 0.85$; $F_{max} \leftarrow 150$;
                - If 5: $Velocity \leftarrow 1.0$; $F_{max} \leftarrow 180$;
            - $Moving = True$;

This algorithm helps stop the player's avatar from abruptly pausing between steps and when the knee is at peak height. It also ensures that UE4's internal buffer is not being overloaded by calling the receive function more than needed, as an overloaded receive buffer causes

unnecessary lag. The engine, when backed-up, is busy trying to process past inputs that are not representing the player's actual movement, which causes the world's unresponsiveness.

UE4's editing tools, open-source meshes, textures, and animations are then applied to create a visually appealing forest landscape for the player avatar to run through. [Fig. 1] This is followed by the implementation of start and stop menus created with the heads-up-display (HUD) Blueprint modules. These menus begin and end [Fig. 6] the simulation. [8] A simple stopwatch timer was also built using the HUD Blueprint modules [9] which activates the timer once the player leaves the starting grey platform, [Fig. 4] and stops when the player reaches the second grey platform. [Fig. 8]

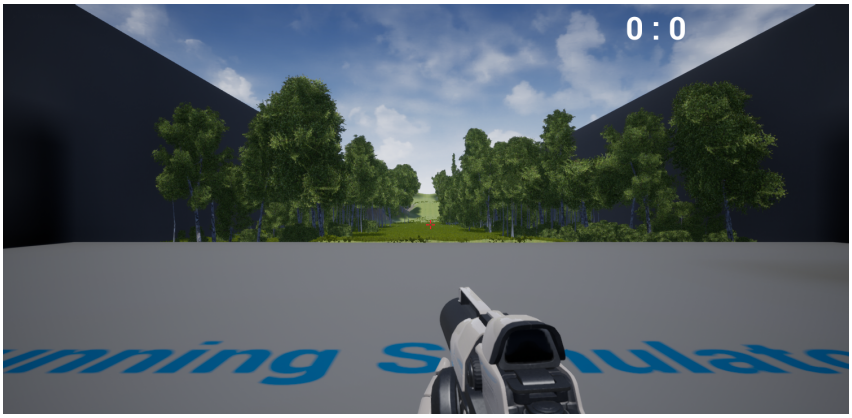

Figure 4: Starting Grey Platform

After the Arduino and UE4 simulation is set up and programmed properly, the user secures the Arduino by strapping the cloth knee brace over the device with the x-axis positioned correctly. The 9V battery is then inserted into the Arduino's battery terminal powering it on, activating the broadcast of its private WiFi network. The Arduino must be powered-on first before starting the UE4 application which automatically connects to this network when it is started. If successfully connected, the screen loads the player's avatar and virtual world.

The user then plays through the simulation by running in-place using the "high-knees method" until the second grey platform is reached. The "game over" screen[7] pops up afterwards to show the time it took to complete the simulation, and it remains displayed on-screen until the application is exited manually. [Fig. 9]

[7]To run the simulation again, exit the running simulation, reset the Arduino by pressing the active-low button next to the USB terminal, and then re-run the simulation executable.

Conclusion

If the above process is implemented correctly, Unreal Engine 4 paired with an Arduino Uno Wifi Rev2 microcontroller can be used to successfully simulate walking or running in a virtual environment by the user walking or running in-place. Immersiveness is increased when compared to using a standard analog controller method of movement. This increase in immersiveness is only possible due to the use of the smoothing algorithm, or something similar, due to the inherent issues when streaming data over a wireless network for real-time simulations. There is still much research to be done on these proposed methods.

If a full experiment is done, statistics on average acceleration rates over a wide range of test subjects who are walking, jogging, and running in real life can be used to calibrate the rate at which the avatar moves through the game world.

When testing the application over long periods heart-rate is increased, providing a good alternative to typical exercise routines. Potential future projects could include building the running simulation with a wide range of selectable environments to use in conjunction with a wireless head-mounted-display and an exercise machine, such as a treadmill or stationary bike.

Personal Notes

Overall I highly recommend that future students attempt this project idea, as it was very satisfying to complete and has a lot of potential for improvement. It pulled from a variety of past class subjects such as physics, networks, and embedded programming. As a warning though, students attempting this project must be familiar with a game engine similar to UE4, the C programming language, client/server networks over a WiFi connection, and they must be comfortable with reading obscure documentation to put the parts together. I would set the project at a medium overall difficulty for a student feeling confident after adhering to this warning.

Appendix

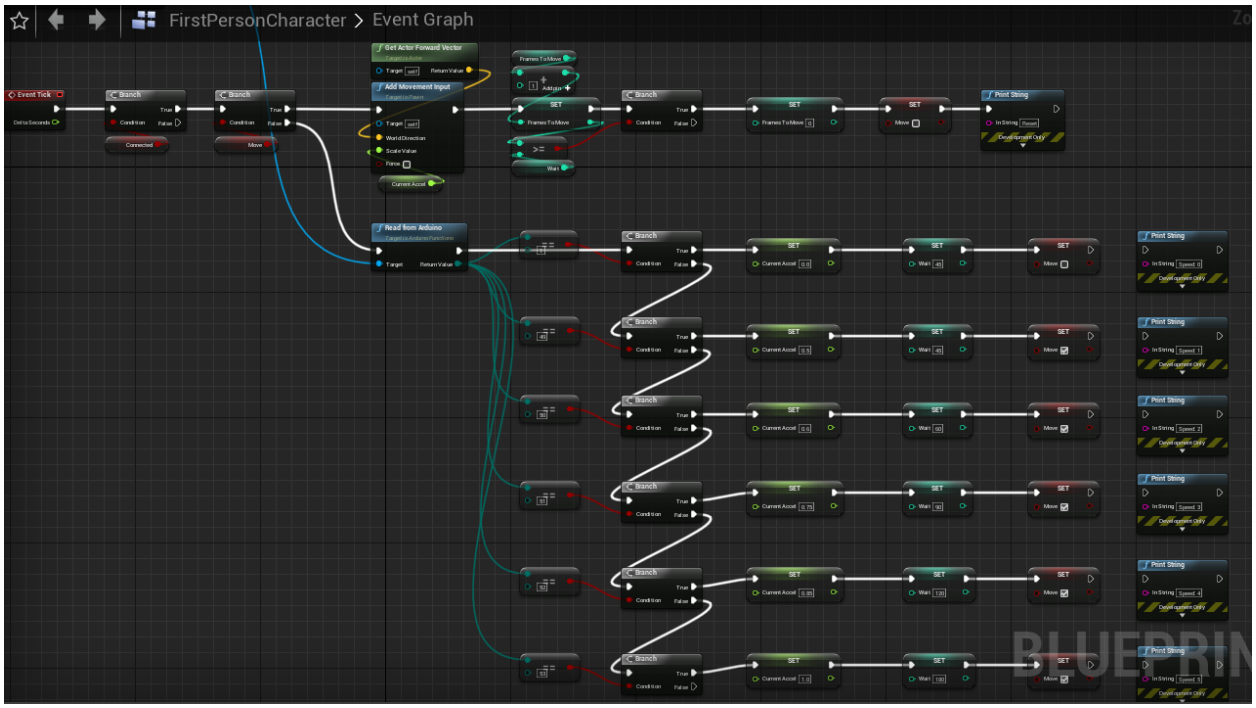

Figure 5: Smoothing Algorithm Blueprint

Figure 6: Main Menu

Figure 7: Start Stopwatch

Figure 8: Stop Stopwatch and Finish the Simulation

Figure 9: Game Over Screen

References

1. Arduino, “Getting Started with the Arduino Uno WiFi Rev 2 ”, Documentation, Revised October 30, 2018. https://www.arduino.cc/en/Guide/ArduinoUnoWiFiRev2 (accessed March 25, 2020).

2. Epic Games, “Setting Up Visual Studio for Unreal Engine”, Documentation, 2020. https://docs.unrealengine.com/en-US/Programming/Development/VisualStudioSetup/index.html (accessed March 25, 2020).

3. Epic Games, “Hardware and Software Specifications”, Documentation, 2020, https://docs.unrealengine.com/en-US/GettingStarted/RecommendedSpecifications/index.html (accessed March 25, 2020).

4. Arduino, “WiFiNINA library”, Documentation, Revised December 24, 2019. https://www.arduino.cc/en/Reference/WiFiNINA (accessed March 25, 2020).

5. Arduino, “Arduino LSM6DS3 library”, Documentation, Revised December 25, 2019. https://www.arduino.cc/en/Reference/ArduinoLSM6DS3 (accessed March 25, 2020).

6. Microsoft, “Complete Winsock Server Code”, Documentation, May 31 2018, https://docs.microsoft.com/en-us/windows/win32/winsock/complete-server-code (accessed March 25, 2020).

7. Epic Games, “Blueprints Visual Scripting”, Documentation, https://docs.unrealengine.com/en-US/Engine/Blueprints/index.html (accessed March 23, 2020).

8. CodeViper, “How To Create A Main Menu - Unreal Engine 4 Tutorial”, Youtube, October 17, 2016, https://www.youtube.com/watch?v=ulUO4EN8BG8 (accessed March 25, 2020)

9. Markom3D, “UE4 - Creating a Timer - Unreal Engine 4 Blueprints Tutorial”, Youtube, October 11, 2008, https://www.youtube.com/watch?v=UR4l_tsYcqs (accessed March 25, 2020)