Senior Project (Prof. Eckhardt)

Michelle Hsu & Tiger Sun

# SLO Terrain for Emergency Flight Landing

## Purpose and End Goal

The purpose of this project is to deploy the San Luis Obispo terrain to the Microsoft HoloLens. The terrain is rotated and transformed depending on the GPS coordinates and compass direction of a smartphone device. A heads-up display reads the coordinates and compass direction the smartphone is currently at.

## Use Cases

This project is designed to aid pilots that are flying through the San Luis Obispo area who may be caught between dangerous scenarios (such as clouds) that hinder visibility of the real-life terrain. They can then use the HoloLens to display a hologram of the San Luis Obispo terrain in its real-life coordinates to safely navigate and determine their surroundings.

## Tools Used

- ★ Microsoft HoloLens (1st Gen) - Used to visualize San Luis Obispo terrain hologram
- ★ HTC 10 - Sends GPS data and sensor data to the HoloLens, any Android phone with a compass will work
- ★ Unity 2019.4.17f1 - Design and deploy terrain, use of scripting to obtain sensor data
- ★ GIMP 2.10.22 - Convert heightmap from .jpg file to .raw in order to use in Unity
- ★ IP Webcam - Creates a port on IP address to post magnetometer sensor data
- ★ Share GPS - Creates a port on IP address to stream GPS coordinates in NMEA format

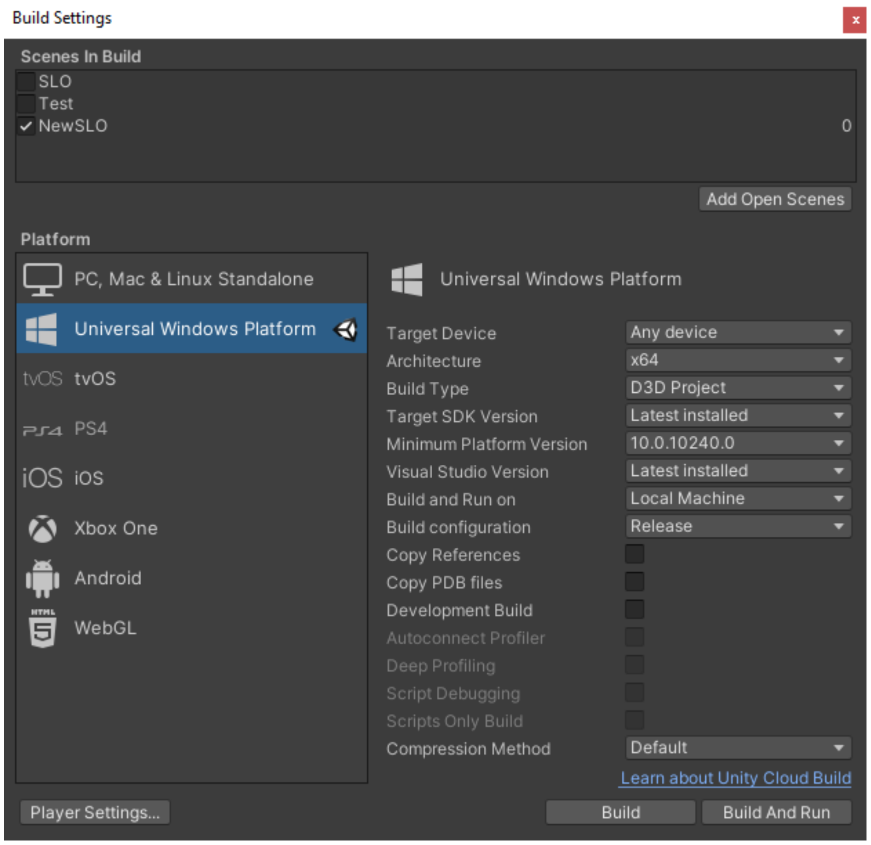
★ Visual Studio 2019 - Used for scripting to parse incoming sensor and GPS data

**How to Set-Up and Deploy Project**

1. Download the Unity project folder, add it to your Unity Hub, and open the project.
2. Ensure that the smartphone is connected to the same network as the computer that we will be using Unity on.
3. On the smartphone, go to the IP Webcam application.
    - Go to "Data logging" → select "Enable data logging" to be ON → scroll down and select "Magnetic field" to be ON
    - Go back to the main menu and scroll down to "Optional permissions" → "Allow GPS" should be enabled → "Allow streaming in background"
    - Go back to the main menu and hit "Start server"
    - Record the IP address and port number this application is running on. It should show up at the bottom of the screen when the server has been started.
        - Ex: http://192.168.68.121:8080
            - Where IP Address = 192.168.68.121 and Port Number = 8080
4. On the smartphone, go to the Share GPS application.
    - Go to "Connections" → hit "Add" at the bottom → select "Share my GPS with a laptop or tablet that does not have GPS using NMEA" → hit "OK" → choose "Connection Method" to be "Use TCP/IP to send NMEA GPS to PC. The IP addresses of the devices are known." → hit "OK" → give the connection a name and hit "OK" and then "Next"
    - On the "TCP/IP Settings" page, record the port number, then hit "OK"

- Your connection should now say “Idle”. Tap on the connection you just made and ensure the words are now yellow and say “Listening”.

5. Turn on the Microsoft HoloLens, and ensure that the device is connected to the same network as the smartphone device and computer. Go to “All Apps” → “Holographic Remoting”. Record the IP address that is shown.
6. Back to the project in Unity, ensure that all scripts are enabled. Compass Rotate and GPS Read are scripts attached to the XRRig object. The HUD script is attached to the HUD object under XRRig → Camera Offset → Main Camera → HUD.
7. Open the two scripts, Compass Rotate and GPS Read. We need to change the IP addresses and port numbers in the scripts due to a change in network and setup.
   - In the Compass Rotate script, on line 17, change the IP address and port number to match the one from Step 3. Maintain the /sensors.json at the end.
     - StartCoroutine(GetRequest(“http://IP_ADDRESS:PORT/sensors.json”));
   - In the GPS Read script, on line 24, change the IP address to match the one from Step 3, and the port number to match the one from Step 4.
     - StartCoroutine(GetRequest(“http://IP_ADDRESS:PORT”));
   - Note: The IP Addresses in both scripts should be the same. The port number should differ.
8. In Unity, ensure that the “NewSLO” scene is open. Go to “File” → “Build Settings”, a window should pop-up
   - Under “Scenes In Build”, only select “NewSLO”.
   - Under “Platform”, select “Universal Windows Platform”. Your settings should look like this:



Build Settings

Scenes In Build

- [ ] SLO
- [ ] Test
- [✓] NewSLO 0

Add Open Scenes

Platform

PC, Mac & Linux Standalone
Universal Windows Platform
tvOS
PS4
iOS
Xbox One
Android
WebGL

Universal Windows Platform

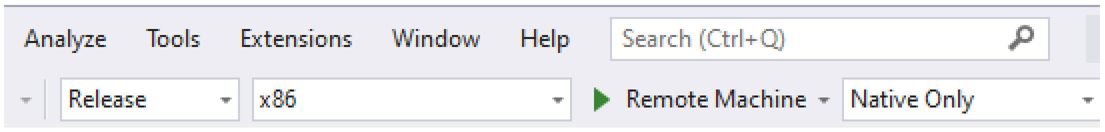
| | |
|---|---|
| Target Device | Any device |
| Architecture | x64 |
| Build Type | D3D Project |
| Target SDK Version | Latest installed |
| Minimum Platform Version | 10.0.10240.0 |
| Visual Studio Version | Latest installed |
| Build and Run on | Local Machine |
| Build configuration | Release |
| Copy References | ☐ |
| Copy PDB files | ☐ |
| Development Build | ☐ |
| Autoconnect Profiler | ☐ |
| Deep Profiling | ☐ |
| Script Debugging | ☐ |
| Scripts Only Build | ☐ |
| Compression Method | Default |

Learn about Unity Cloud Build

Player Settings... Build Build And Run

- Select “Build”, then create a new folder that will not collide with other project files and then select that folder. Your build should begin immediately and the folder with the built project should pop up once finished.

9. Open the built project (should have .sln ending). Change the settings along the top bar to match:



Analyze Tools Extensions Window Help Search (Ctrl+Q)

Release | x86 | Remote Machine | Native Only

Then, go to “Project” → “Properties” → under “Configuration Properties” select “Debugging” → change the “Machine Name” slot to the IP address from Step 5 → hit “OK”.

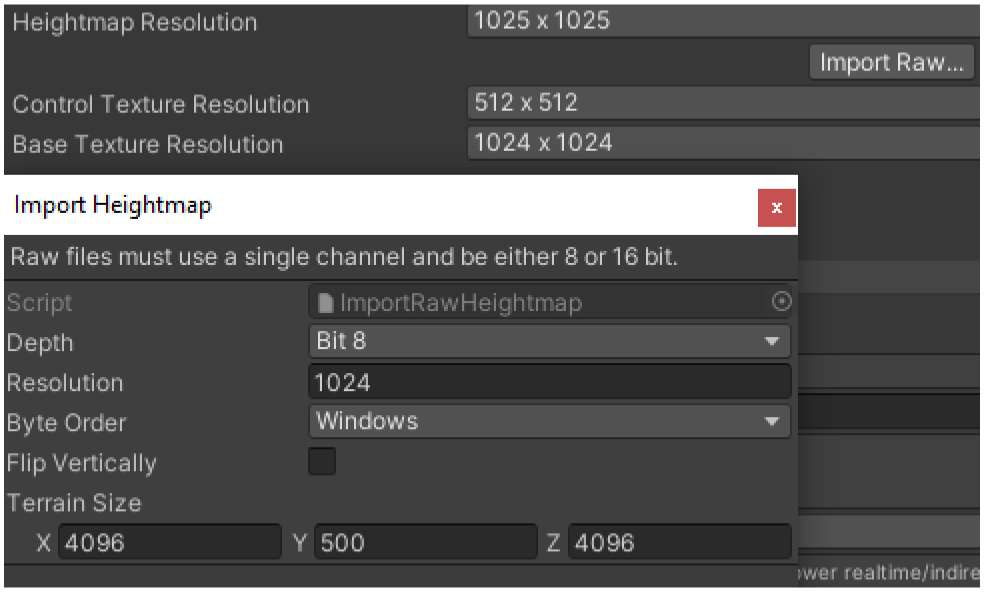
10. Still in the built project, click “Debug” → “Start Without Debugging”. The build is now being deployed to the HoloLens.
    - Note: If there are any build or deploy errors, they are likely due to invalid IP addresses. For example, if the HoloLens turns off before the deployment is finished, it will result in an error.
    - Note: In the HoloLens, a “Made in Unity” logo will appear upon successful deployment.

**Overview of Project**

This section will highlight the many steps required to begin and proceed this project. A trial and error/debugging section will be further below. These are only the steps that were successful to the progression of the project.

To start, we needed to find a mobile phone application that could send sensor data and GPS coordinates to the computer. After several trial and errors with different applications, including some dabbling in Bluetooth, two applications on the Google Play Store were found to meet our conditions. They both create their own server port on the mobile phone, which allows easy HTTP web requests to obtain their data. IP Webcam posts magnetic field sensor data to its server. Meanwhile, Share GPS streams its data in the form of NMEA (a type of GPS data format) to its server.

Next, we wanted to get the San Luis Obispo terrain and heightmap into Unity. The heightmap was given to us as a .jpg file. However, in Unity, the heightmap must be in a .raw format. We changed the file type using GIMP and created a terrain in Unity using the raw file. We then overlaid the terrain image as a layer over the terrain object and had to make slight adjustments to the size (under Tiling Settings to be 4096x4096). The raw file was imported into Unity using Terrain Tools with these specifications:



| | |
|---|---|
| Heightmap Resolution | 1025 x 1025 |
| | Import Raw... |
| Control Texture Resolution | 512 x 512 |
| Base Texture Resolution | 1024 x 1024 |

Import Heightmap

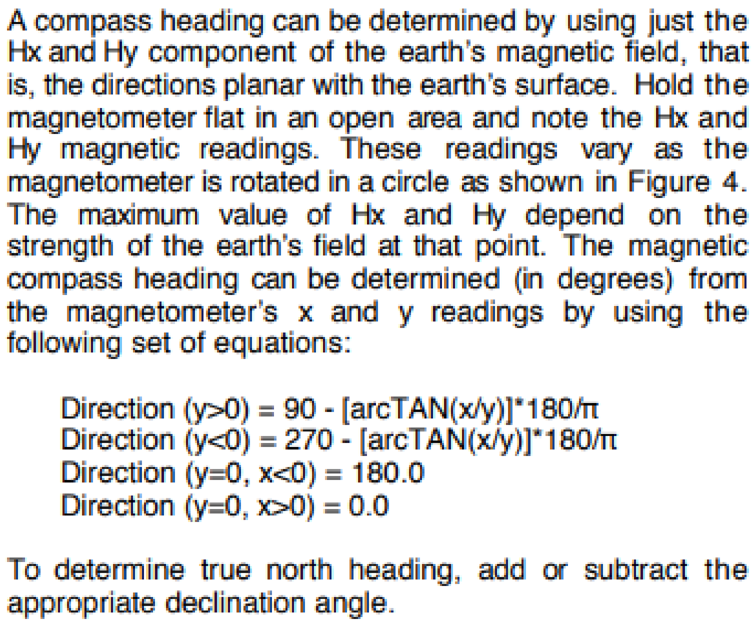
Raw files must use a single channel and be either 8 or 16 bit.

| | |
|---|---|
| Script | ImportRawHeightmap |
| Depth | Bit 8 |
| Resolution | 1024 |
| Byte Order | Windows |
| Flip Vertically | ☐ |
| Terrain Size | X 4096 Y 500 Z 4096 |

(Note: the Import Heightmap window is obtained after clicking “Import Raw” and selecting the heightmap.raw file)

After setting up the terrain, we needed to configure the project to be able to be seen and used in the HoloLens. After researching different ways to configure an AR project, we followed this tutorial for Configuring a Unity Project for XR. This results in an “XRRig” object in our Unity project. This camera rig represents our view in the HoloLens, and displays the holographic terrain accordingly when we move our head. Because we still want to see the real world behind the hologram, we changed the background attributes on the Main Camera (found as a child object of the XR Rig) to be a Solid Color with R, G, B, A values of 0, 0, 0, 0.

Next, we wanted to create scripts to rotate and transform the XRRig according to real-life changes in compass direction and location. These scripts transformed the XRRig because terrains cannot be rotated, thus, the camera rotates instead. This rotation of the camera maintains the same effect as the rotation of the terrain making no difference in the perspective of the user. In the Compass Rotate script, we followed the syntax used for a UnityWebRequest GET. This script uses a web request to obtain the magnetic field data that is being posted to the mobile phone's server. Once the data was obtained, we wrote additional parsers for the values and transformed the camera to rotate the same direction that the compass has. The math behind the transformations can be found on a datasheet made by Honeywell:



A compass heading can be determined by using just the Hx and Hy component of the earth's magnetic field, that is, the directions planar with the earth's surface. Hold the magnetometer flat in an open area and note the Hx and Hy magnetic readings. These readings vary as the magnetometer is rotated in a circle as shown in Figure 4. The maximum value of Hx and Hy depend on the strength of the earth's field at that point. The magnetic compass heading can be determined (in degrees) from the magnetometer's x and y readings by using the following set of equations:

Direction $(y>0) = 90 - [\arctan(x/y)]*180/\pi$
Direction $(y<0) = 270 - [\arctan(x/y)]*180/\pi$
Direction $(y=0, x<0) = 180.0$
Direction $(y=0, x>0) = 0.0$

To determine true north heading, add or subtract the appropriate declination angle.

Using these calculations, the XRRig successfully rotates with the compass.

In the GPS Read script, we followed the syntax for a UnityWebRequest GET again. However, the script requests data from a server that returns the data in the form of a data stream, so we decided to have the web request timeout every 1 second. This means that once a request has been made, it will timeout after 1 second which we can then read and parse the data that was coming in during that time frame. In order to parse this data, we followed the format for NMEA data sequences. We are primarily focused on "$GPGGA" sequences because they contain the longitude and latitude sequences. To understand the data, we found a breakdown of NMEA sentence information, as well as a different tutorial to explain NMEA GPS data.

The last script that we wrote was for the Heads Up Display (HUD). The HUD currently transforms with the camera. This means that the HUD will follow the rotation of your head, and will always be in your view. Using the values from the two other scripts, we can easily present longitude, latitude, and compass direction. For the compass, using the math from the Compass Rotate script, we can determine the cardinal directions. With some manipulation of the longitude and latitude data, we can present them as decimal degrees.

**Issues and Solutions**

This section focuses on the problems that we ran into during the design and development of this project and solutions or research that we found to overcome these issues.

One of the biggest issues when starting this project was that there were very limited applications that send GPS and compass data to the PC in the form that we want it. There were many applications that relate to serial or COM ports, or require a server on the PC to post data to. None of these applications would work in our project due to the HoloLens. In order to use this project efficiently, we would only need to ensure that the mobile phone and HoloLens are on the same network so that the HoloLens can access the phones' data sensors. The discovery of the two applications IP Webcam and Share GPS were tremendous in the progression of this project because both applications are able to start their own servers.

Another large issue pertains to the set-up and configuration of the project. Documentations provided on the HoloLens 1st Gen were no longer updated or were deprecated. The HoloToolkit, a toolkit used for easy configuration of the HoloLens 1st Gen, was deprecated and could no longer be found for download. Most of the tutorials for the HoloLens 1st Gen were based on the HoloToolkit. This made things problematic when all current and updated documentation used the MRToolKit (used for the HoloLens 2nd Gen). The complications continued when documentations dived into Spatial Awareness and Surface Handlers. The addition of the XR configuration solved our problems. There are many different ways to format your Unity project, but the XR configuration tutorial linked above was the one that worked for us.

An additional problem that was run into was reading the data stream from the GPS application. This ties into another problem relating to UnityWebRequests and HTTPWebRequests. In order to read data as a stream, we were required to use HTTPWebRequests (from the C# language). However, because we were designing and implementing our project in Unity, we were forced to use UnityWebRequests instead, which limits the types of methods we were allowed to use. When attempting to test the usage of HTTPWebRequests, the request would not go through and the project would timeout. In order to work around this issue, we decided to time-out our request instead so that we could access the data from the stream in parts.

## Future Works

This section highlights some aspects of the project that can be added, improved on, and fixed.

The San Luis Obispo heightmap and terrain appear to be lower resolution than the images that they were originally. This can easily be fixed by maintaining correct sizing when converting the heightmap from .jpg to .raw, and scaling the terrain correctly with a new heightmap.

The GPS Read script can be heavily improved on. We could not find a way to read the web requests' data as a stream, but there is a high possibility of a solution that exists. The current implementation where we timeout the web request slows down the script immensely. If the script can be read as a stream and parsed in real time, then we would be able to access the GPS coordinates immediately. Following the parsing of GPS coordinates, a feature that must be implemented is the mapping of GPS coordinates to the San Luis Obispo terrain. To accomplish this, we would need to assign longitude and latitude to each corner of the terrain, and then transform the XRRig to be in the correct position as a ratio between each corner. Some mathematical calculations would be involved to fulfill this.

The HUD script can be improved on by fixing the compass cardinal directions and debugging the GPS longitude and latitude feature. The current compass directions only include North, South, East, and West. North and South only appear when the compass is pointing exactly at either of those two directions. This means that the compass reads East or West most of the time. We believe that improvements can be made where the compass can additionally determine NW, NE, SW, and SE directions. On the other hand, there is a bug when the HUD updates the longitude and latitude. When we access global static variables from the GPS Read script, the flow from the HUD script gets stuck in the GPS Read script. We saw this problem arise through the usage of a step-by-step debugger. We speculate that there is some corruption in memory and that our stack pointer may be overwritten at some point.

Another improvement that can be made on the HUD is to have it rotate according to the mobile phone compass. We currently have the HUD maintain its position in front of our view, so it moves with us. The improvement would change so that the HUD stays in the compass direction spot.