Tutorial Report - Oculus Passthrough API For Drone FPV With Unity

Joshua Pulido
jpulido@oxy.edu
Occidental College

1 Tutorial Report

In an effort to familiarize myself with how to interact with a Tello drone through Unity, I chose a tutorial by Dilmer Vacelillos, an XR programmer who posts many tutorials on Youtube about development for the Oculus Quest 2. Although the tutorial did not end up being exactly what I expected (more on that later), it still provided a lot of vaulable information on how to interact between the two.

2 Methods

The tutorial I chose followed a 4 step approach to interacting with a drone in Unity, first by setting up a UI that worked to interact with a drone, then developing the backend for interacting with the drone, then interacting with the drone state, and finally interacting with the drone using the Oculus Quest 2 controllers.

2.1 UI

The development of the UI was not covered in depth in the tutorial, and for this tutorial specifically, the UI implemented was relatively simple. The main focus of the UI was getting the Oculus passthrough API to project onto a screen, so there was a large screen, as well as a smaller one to show logs and some statistics that were passed from the drone. For this tutorial, the UI was simply pulled from Github, but the creator went into detail about how he used Unity's built in UI editor to place text and other UI pieces on a canvas, which is then displayed ingame in front of the camera.

2.2 Setting Up

Unlike what I was initially planning on doing for my project, this tutorial took the approach of interacting directly with the Tello SDK over a UDP connection. To accomplish this, they essentially made their own Tello API, complete with definitions of different commands representing actions that could be taken (i.e. move left, take off, land). This was accomplished by first defining all of the possible actions as enums, to make the overall codebase significantly cleaner than just using strings. These were defined as commands, and further split into actions (for moving the drone) and attributes (for getting state info).

The next step of the tutorial was setting up some basic communication with the drone, over a UDP connection. Using the C# standard library for UDP connections, we first initialized a thread for retrieving information about the state. This thread ran a method that was always running, through the use of a "while true" loop, and constantly received information about the drone's state and logged it. The next step was to send messages to the drone over a different UDP connection (differentiated by the port number), which was ran on the main thread, as sending messages is an active part of the game rather than a background task.

From here, it was a simple task of linking the UI components to the drone through sending messages. At this point, this was simply a testing phase, so I used a debug UI that was also provided over Github, rather than trying to interact with the Oculus Quest 2 at this moment. By adding the send command method to the OnClick event of the UI buttons, it was possible to connect to the drone, takeoff, and land.

2.3 Getting Drone State

Using some of the enums defined above, it was possible to get the state of the drone, which includes a lot of various useful information such as pitch, yaw, roll, altitude, and so forth. The accuracy of these variables should be called into question as the Tello drone is not sensor-rich, but they can still be used to get some idea of what the drone is doing at all times. For the purposes of this tutorial, working with the drone state was helpful in learning about how to get information from the drone over UDP.

Interestingly, the drone had two states that it could be in. It would first give information on whether or not it was connected, and then after it was connected, it would then start giving the state info, in some sort of jumbled mess. Thankfully, the enums that were set up in the previous step now allowed for easing parsing of the state, split on the ";" character, so it was fine dealing with them.

With the information on the state parsed, passing it over to the UI is a relatively simple thing. Simply put, all of the text UI elements, which would display the state variables as strings, are stored as variables in the DroneStateManager

class, and are constantly updating their text with the state values that the drone passes.

2.4 Controlling A Drone

In order to control the drone, it was necessary to properly define the commands that were initialized during the setup stage. Unlike commands used previously, such as the takeoff/launch or state commands, the movement commands required a very specific format for them to run correctly. For example, translational movement is controlled by the rc, or "remote control" command sent over UDP, of the format x y z yaw, where all four values are between -100 and 100. In order to map these, the creator made a mapping, which mapped the different controls, including joysticks and buttons, on the Quest controllers to their corresponding commands. Thus, one of the buttons was responsible for taking off, the joysticks controlled all 3 axes of motion as well as rotation.

With the mapping complete, the next step was to define the actual controller, in a script that ran the mappings every frame. Because most of the control definition went into the actual mapping, this was as simple as calling each method in the mapping, passing the appropriate parameters (by reference) whenever needed. Each action, or command, was bound to a different button or joystick on the Quest controllers, and they were all run sequentially in such a way that would theoretically allow for smooth movement.

2.5 Running The Program

There were no detailed instructions on how to run the program on the Oculus Quest, but running the app natively was relatively simple once the Unity project was imported. However, running the app on the Oculus Quest 2 was slightly more complicated, and involved a long process of setting up both the device and Unity to be compatible with each other. Once they had the right settings, running and building was also as simple as clicking a button, but in this case the app did not quite run as expected, which I will talk about more in the evaluation section.

3 Evaluation

Since this was a tutorial, the evaluation section for this report is split into two separate sections. The first focuses on the valuable information that I was able to gain from the tutorial that will be helpful in working on my comps project next year. The second talks more about the actual end result of the tutorial, which was slightly underwhelming in my case.

3.1 Information Gained

The tutorial provided a lot of information about how the Tello drone's API actually works over UDP. Despite having worked with the drone for the past couple of months, I had yet to see the actual implementation of a wrapper API that could interact directly with the drone over UDP, and this has shown me that it is possible and within the scope of my project. I was also able to gain information about running code on separate threads in Unity, which I think will be of great help as there are many different parts to my comps project that might benefit from being run on their own threads.

Overall, the tutorial really gave me a good scope of how big my project might be. Notably, the tutorial involved no actual gameplay, and yet the diagram outlining all the different components was huge. Although the tutorial only took me a couple of hours, it was pretty clear that the amount of design work that went into the project took significantly longer, so I would imagine that such will be the case for me next semester as well.

3.2 Running the Code

The end results of the tutorial were mixed. On the one hand, I was able to get drone connected to the standalone version that was part of the tutorial, which ran on my computer. This worked with no issues, and I was able to get the drone to take off and land as I would with any other API. Unfortunately, this was where the successes more or less ended with running the program.

I was able to successfully build and run the program for the Oculus Quest 2, with no compilation errors. However, I found this a bit strange as I had not properly installed any packages to use the Quest passthrough API, or even the controllers. So, although I was able to see the UI in through the Quest 2's lenses, the code running the controller was not working properly and I couldn't tell whether or not I was connected to the drone. Moreover, since the passthrough API was not working, even if I was connected to the drone, it would be unwise to try flying it without being able to see. My suspicion is that the packages were either not installed at all, or they were installed incorrectly. Normally, when working with a Unity project, you would include only the Assets folder in the Github repository, but the tutorial author included all of the packages as well, so there might have been a bad or outdated installation for my machine.

4 Discussion

As discussed in the evaluation section above, the tutorial did not go exactly as planned, as there were a couple of features that failed. However, this is perfectly fine, and I

will be going more in depth as to why I would argue this is the case in this section. I'll also be discussing how my perspective has shifted (or not) after doing this tutorial for comps next year.

4.1 Why It's Okay That It Failed

As mentioned above, the tutorial failed to properly interact with the Oculus controllers and cameras, making the Oculus version of the program unusable. However, it is important to note that neither of these features were planned to be used in my game. For context, my goal is to make an augmented reality game using the video output from the drone, where you can control the drone using a standard Playstation or Xbox controller. So, to begin with, I had no plans to use the Quest controllers or camera passthrough. Thus, their failure did not detract from what I was able to gain out of the tutorial.

After interacting with the Quest 2 controllers, I have realized that they really do not provide the proper feel for the game I am making. The split controller layout makes it feel as though they are two separate controllers that function independently, which is certainly not the case for flying a drone. I never thought that they would be a good fit, but I had considered providing an option for the player to use the Quest controllers, if they felt more comfortable. After seeing how they are supposed to interact with the quest in code, I have realized that this is not worth the effort. Despite not being part of the initial plans for my comps project, the passthrough API did give me an idea for the game that I thought might be useful, which was a toggle for turning on passthrough to safely land the drone. Since the passthrough API was not working, I was a bit worried, but have since discovered that Oculus has a setting to enable passthrough natively by double tapping the side of the headset, while still running the game in the background. As a result, it is not a necessity to add this to the game itself, and instead should be considered something extra that I can figure out later. Notably, there were many more tutorials available by the same author which detailed how to use the Quest Passthrough API, and even just develop for the Quest in general using Unity. These almost certainly contained the information to get everything working, but given that it would take more time, I did not think it was worth the effort at this moment.

4.2 What Comps Looks Like Next Year

This tutorial gave me good context on what the process of getting the drone to interact with Unity is going to be like, and although there is a lot going on, I think that it is a very reasonable task to accomplish. My game will not be using nearly as many features of the Tello API, mostly using the movement commands to actually get the drone to fly, so I think that my version of the API will be slightly easier to implement, should I choose to go that route. There are also already existing APIs, but I specifically am curious about implementing the communication myself because this would potentially open up the project to newer drones, most notably the Tello Talent (TT). Unlike its predecessor, the TT drone does have sensors, and also allows for custom ones to be added as well.

I am still unsure whether or not using a different drone is the path I would like to take, but regardless, I have decided that implementing the communication between the controller, drone, and VR headset through Unity must be the first part of the project. Moreover, it should be more or less completed before even considering the game design, as its outcome will directly impact the type of game that can be made. There are still some uncertainties left unanswered, but this tutorial has answered one of the key ones, so I do think that my project is possible. I will continue working on the project over the summer, so that by the start of next semester, I have a good idea of what is and is not possible. I hope that the game is still a possibility, but if not I can pivot to a full VR game instead, or some other cool project with a drone.