4.2.4 Objective 1; Task 4

To create a control link between Unity-UR5, and operate the robotic arm through virtual reality.

My main objective for this internship period was to improve on the VR system A2labs previously had to control the ur5. I identified a few problems with their existing system;
1) The targeted position output by computer relative to position of headset, therefore slight movements with the operator's head will also move the arm.
2) It was not possible to limit or control the movement speed of the arm, as doing so will drastically reduce its accuracy.
3) There was little information that the operator can obtain about the current position of the arm, forcing you to take off the VR goggles while in operation.

After identifying these problems, I decided to completely rework the virtual reality control loop, transferring the processing to be done within unity and for the ur5 to simply output given joint target positions. In order to do this the ur5 needs to be fully simulated inside unity.
Forward kinematics are a set of equations to individually move a robot's joints, in other words known joint target angles. Inverse kinematics is a more advanced set of equations to instead derive target joint angles from a given target enfactor (x,y,z) position. To simulate UR5 within unity I had to provide an inverse kinematics solution. After studying the mathematics behind FK and IK, I converted what was learnt into C++ code. To solve Inverse kinematics, gradient descent algorithm was implemented, this finds a local minimum between angle of joint and distance from target position, modifying angle to minimize distance.

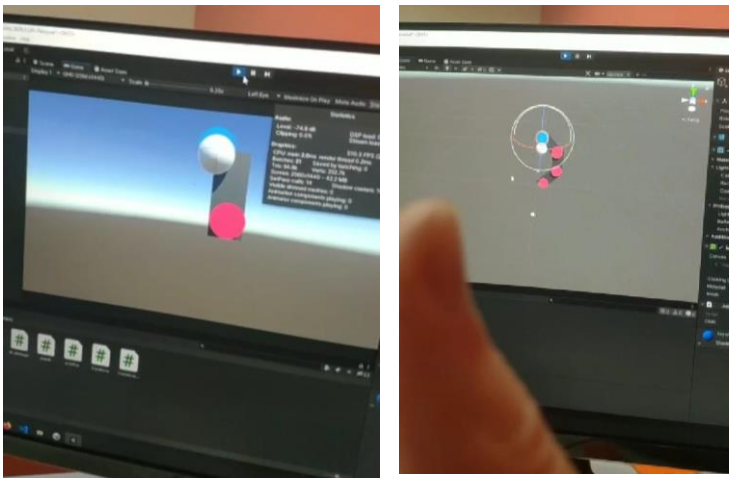

Figure 4.2.4 (a): Early inverse kinematics solution

After getting IK in unity working, I rigged an existing UR5 solidworks model to a format compatible with unity (.obj), then used this model to first create a forward kinematics scene, to do this the simulated ur5 joints were assigned a script which links it to a slider the user can control.

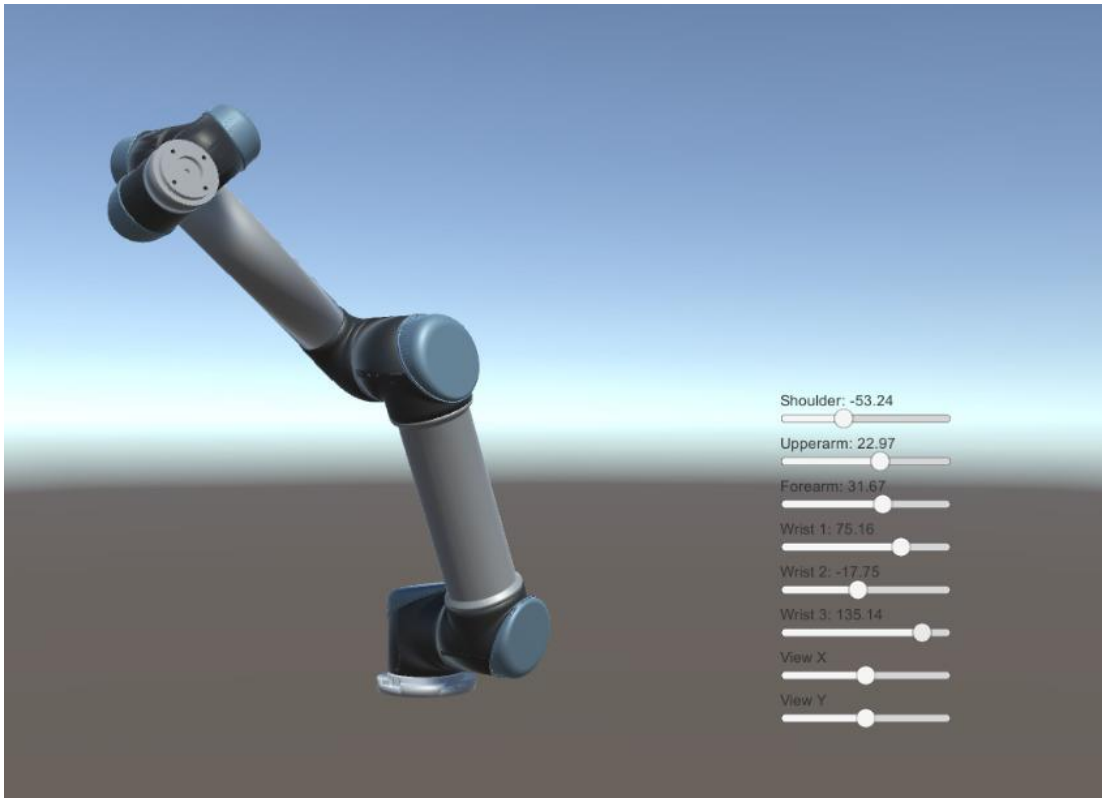

Figure 4.2.4 (b): Forward kinematics solution implemented on simulated ur5

When this worked, inverse kinematics was implemented on the simulated ur5, there were issues as gradient descent worked well in 2D but not 3D objects, so the inverse

kinematics solution was further optimized till it worked with the simulated ur5 model. At this point the simulated ur5 will move its joints to match its enfactor with the target gameobject's position.

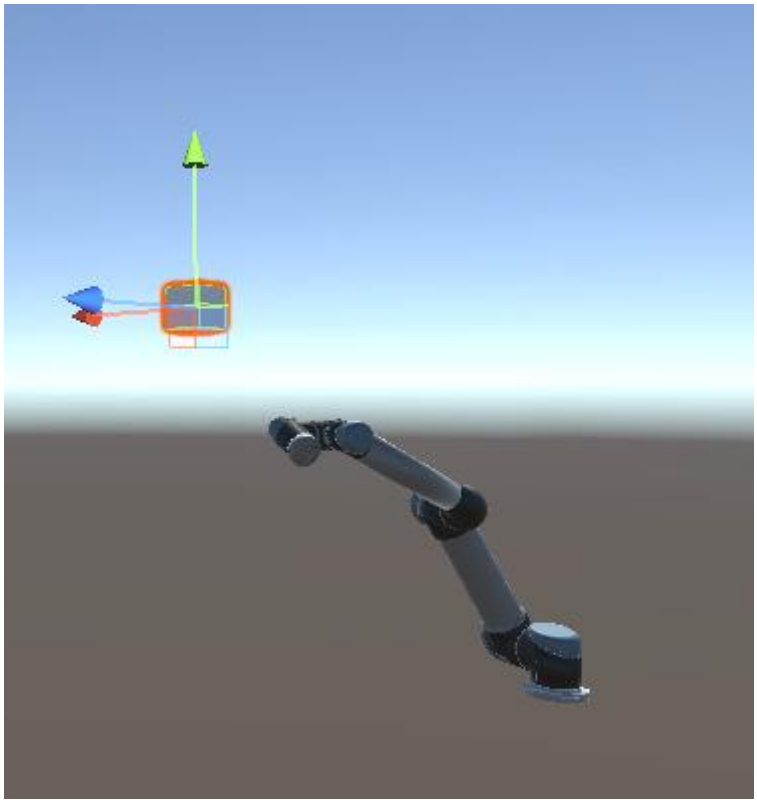

Figure 4.2.4 (c): First inverse kinematics implementation on simulated ur5

A virtual reality experience now needs to be developed to show the user the movement of the ur5 as well as for the user to control the position of target point with his hands. I spent time learning how to develop with UnityXR and made a simple virtual reality control room for the operator. It consists of the simulated ur5 in the centre, 4 virtual cameras to show the ur5 through multiple angles, and a camera feedback screen to feed data from an external camera. The user can now simply drag the target gameobject and the ur5 enfactor will follow.

Figure 4.2.4 (d): Development phase of virtual reality environment

When the simulations were complete, I worked on the connection to UR5, first I developed a joint listener script to collect the local Euler angle rotations of each joint in the simulated ur5 into an array. Then a TCP connection script is made to connect unity engine to polyscope, this is the operating system used by ur5. This TCP connection script refers to the angle listener and converts it to radians, this data array is then sent to ur5.

On polyscope side, I used block programming of the teach pendant to create a connection to TCP from the computer, this is done through ethernet wire. I then linked the built in servoJ function to link the array output by unity into targeted angles for ur5, in radians.

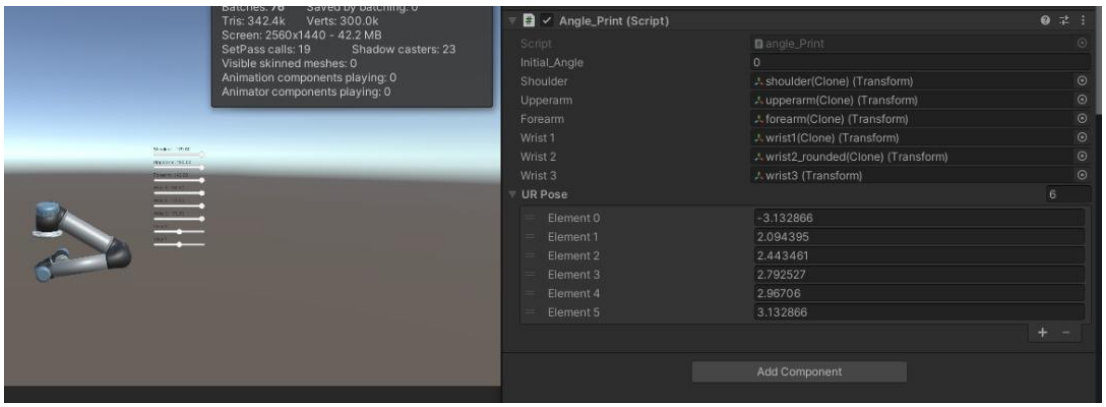

Figure 4.2.4 (e): angle listener script to output target angles in radians

For operational safety, this was first tested with the forward kinematics scene as the operator has more direct control here. The connection worked and ur5 now follows angle targets set by unity engine.

Figure 4.2.4 (f): Working connection between forward kinematics solution and ur5

I then finally implemented this system in the Virtual reality scene and it worked well, my internship objective was met.

Several improvements with my new system compared to the old system:

1) Significantly more stable movement.

2) Robot only moves when user is dragging the target gameobject (small sphere).

3) No longer reliant on VR headsets' positional data, this new system even works by controlling target gameobject on screen using a mouse.

4) There is a full virtual environment for operator to visually see robot's position.

As I have successfully connected ur5 with unity, I used several unity features to control the robot, by making a 2D inverse kinematics scene and an animation control mechanism.

The 2D IK scene works similarly to forward kinematics, but the upper arm, forearm and wrist 1 is controlled through inverse kinematics. This was much smoother for on-screen control then pure forward kinematics, as there was no stuttering.

I also implemented unity's animation system, by binding animations I made with buttons, this allows the operator to make ur5 do specific repetitive motions with a

click of button, allowing future training of ur5 to be done much faster. Normally to teach the ur5 to do repetitive movement you have to physically train it by moving the robot to desired positions, however with this system you can simply make a unity animation to bind movements to the ur5. The advantage of this is that the tasks are not fully repetitive as the user can bind multiple animations, one animation for each key on the keyboard, there is no need to manually run a new program each time the user wants a different movement performed.

Figure 4.2.4 (g): working test on project goal, controlling ur5 via virtual reality