Contents

[Final Year Project 1](#_Toc128506930)

[Setting up 1](#_Toc128506931)

[Old Design 2](#_Toc128506932)

[Whiteboard creation 2](#_Toc128506933)

[Marker grabbable interaction 4](#_Toc128506934)

[New Design 5](#_Toc128506935)

[Marker 5](#_Toc128506936)

[Whiteboard 6](#_Toc128506937)

[Texture 6](#_Toc128506938)

[Material 7](#_Toc128506939)

[Keeping colour on canvas 7](#_Toc128506940)

[Shader 8](#_Toc128506941)

[Duster 9](#_Toc128506942)

[Multiplayer 9](#_Toc128506943)

[Old Movement 11](#_Toc128506944)

[MRTK 11](#_Toc128506945)

[Hand Teleportation using MRTK 11](#_Toc128506946)

[New Movement 12](#_Toc128506947)

[Player Movement 12](#_Toc128506948)

[Hand pose 13](#_Toc128506949)

[Making the user move continuously without velocity jump 15](#_Toc128506950)

[Controller 17](#_Toc128506951)

[Movement 17](#_Toc128506952)

[Turning 17](#_Toc128506953)

# Final Year Project

# Setting up

Started off by updating my unity hub and downloading the latest version of unity which was 2021.3.11lts. The latest version also had the android check box ticked so that I could use it for VR.

Created a new unity project that used 3D universal render pipeline.

I downloaded Oculus Integration from the unity store and imported it. After that was done, I installed Oculus XR plugin. Then in my build setting I switched platforms to Android. Afterwards in my project settings in the XR plug-in management I ticked the oculus box in the android and windows tab.

Once that was completed, I created a new scene and added the OVRCameraRig prefab into the scene. In the inspector under the OVR Manager > Quest Features I changed hand tracking support from controllers only to controllers and hands. Hand tracking frequency from low to max and hand tracking version from default to V2.

I then proceeded to add OVRHandPrefab as a child to LeftHandAnchor and RightHandAnchor so that the hands would be visible when I run the scene in my oculus.

Render physical capsules was checked in both hand prefabs so that physics can be applied. Then on my right-hand prefab I switched skeleton type, mesh type, hand type from left hand to right hand option.

I ran the scene and thankfully hand tracking was working. Now I want to move on into building a scene or trying out a demo scene from github.

# Old Design

## Whiteboard creation

Created a scene and in the scene, I created a panel that I would use as my white board.

Created an empty object named marker and three children off it. One of the children is the handler, the other one is the tip and the final one was a grab points.

Adjusted all the sizes for the markers and whiteboards to be in a nice scale for the user.

Created a blue material and appointed it to the tip of the marker.

A whiteboard script was created next. In the script I set the texture resolution.

Text

Description automatically generated

Created a whiteboard tag and assigned the whiteboard panel to it.

Next a whiteboard marker script was created. This script would be used to calculate the position and the rotation of the marker and how often a 5x5 square pixel would be drawn on the whiteboard.

Text

Description automatically generated

Text

Description automatically generated

This script was added onto the marker object.

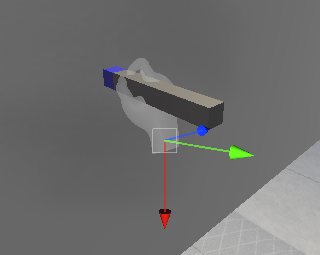
## Marker grabbable interaction

Making the marker grabbable by hand tracking took a little time to figure out. I was using the demo that the oculus integration had as a guide to help me.

I copied the scripts and the layout that the demo had onto my marker and changed all the transform, meshes, colliders and rigid body to the ones I set for the marker.

Graphical user interface, text, application

Description automatically generated

The screenshot above shows the layout for the interaction for the marker. In the visual parent, it holds the body of the marker and uses the markers body mesh as a reference on where to hold the marker essentially. The “HandGrabInteractable” is an object that shows how the marker will be placed on the players hand when they try and grab it. In that object, the marker’s scripts and rigid body is referenced so that it gets the idea of the physics that come into play. I have yet to create a mirror copy of the “HandGrabInteractable” so that people with left hand can also pick it up.

The screenshot above shows what the marker looks like when a person tries to grab it with their hands.

The issue I encountered on this was that the marker would not draw on the whiteboard. This issue was resolved when I realised that I programmed the tip to write on the board if the Y axis faces the whiteboard. So, I change the rotation from X axis facing straight to the Y axis facing straight ahead of the marker.

# New Design

My old whiteboard and marker had constant fps drop when drawing on the whiteboard, so I created a new idea for both.

## Marker

I created a Sphere object and set the scale to my desired size. I added a Rigid body, a box collider, Grabbable script, Pointable Unity Event Wrapper script, One Grab Free Transformer script and a red material. The markers material is URP Unlit.

I assigned the marker to a new layer marker. The render face of the material was set to both so that no matter where the marker touches the plane or if the camera sees the inside of the marker, it will still draw something.

A picture containing water, red, orange

Description automatically generated

Making the marker grabbable by hand tracking took a little time to figure out. I was using the demo that the oculus integration had as a guide to help me.

I copied the scripts and the layout that the demo had onto my marker and changed all the transform, meshes, colliders and rigid body to the ones I set for the marker.

Graphical user interface, text

Description automatically generated

The screenshot above shows the layout for the interaction for the marker. The “HandGrabInteractable” is an object that shows how the marker will be placed on the players hand when they try and grab it. In that object, the marker’s scripts and rigid body is referenced so that it gets the idea of the physics that come into play.

A picture containing chart

Description automatically generated

I created a mirrored version of that hand so that it can be picked up by people using their left hand.

The issue encountered with the marker is that since it is a sphere it makes the HandGrabInteractable look weird and must be set up ahead of the marker so that I can be grabbed centrally.

A picture containing logo

Description automatically generated

## Whiteboard

For the whiteboard I created a blank plane again. Afterwards created a camera child of this plane and made it face towards it. Changed the .

## Texture

I then created a render texture of 2048x2048 size so that it is not too pixelated since it is not low and will not hurt my optimization either since it is not extremely high. If I wish to change the plane in the future with the x axis being twice as large, then I would have to change my render texture size twice the size on one side. This texture was set to the camera output texture. This created the camera ratio the same as the plane.

Graphical user interface, text, application

Description automatically generated

The camera’s projection was changed so that it would be the same size as the plane. So now everything that will be touched by the plane will be seen by the camera and sent to the render texture. Rotated the camera 180 degrees so that the displayed touch of the object is exactly where the real time object is on the plane.

## Material

I created a new material and assigned it to the plane, then set the base map to the texture that I created earlier on. This now allows any object material to be displayed on the plane when it comes in contact.

## Keeping colour on canvas

So, to allow the colour to stay on canvas as the marker is being moved, I changed the cameras background type from skybox to uninitialized and set the culling mask to the marker layer. This allows the camera only to see the objects in the marker layer.

A picture containing chart

Description automatically generated

## Shader

To allow me to change the background of the canvas I needed to create a new lit shader graph. I created a colour parameter as ‘colour’ to set the background colour. Texture2D parameter as ‘Cam’ for what the camera would see where we draw and float parameter as ‘Alpha’ to allow transparency to work.

To know where the terrain is on the plane, I created a new node called sample texture2D and assigned my Cam to Texture(T2) connection. Now to check if there is paint on the texture, I made a node called comparison and dragged the alpha of the sample texture to the A(1). I also set the alpha float to B(1) and made it greater or equal.

Graphical user interface

Description automatically generated

I then created a new branch node and dragged the comparison to predicate(B) which compares if the texture has paint or not by looking at the alpha value and if it is greater than a certain threshold then uses my sample texture. If not, then uses the colour of the canvas. Out(4) of the branch is dragged to the base colour of my shader.

After that is done I changed the shader of the canvas material to the one I just made and set the alpha threshold to 0.1 and now I can change the canvas background without changing the drawings.

A screenshot of a computer

Description automatically generated with medium confidence

Now to allow the canvas to look transparent I changed the Surface to transparent in the graph inspector. I added a split node which allows me to use the alpha value of the colour and the cam texture. I connected the branch result to the split and connect the alpha split to the alpha of the fragment.

## Duster

I found a 3D Duster asset on google and imported into my unity project. I then set the bottom of the duster to the marker layer and set the shader to URP Unlit. I set the transparency to 0 so now when the camera reads the duster touching the canvas it will make everything invisible which sets the illusion of it deleting the drawing off the board. All the grabbable set up and hand grab was done the exact same way as the marker. Except I changed the Supported Grab Type from pinch to everything.

A picture containing text

Description automatically generated

Known issues include that the duster only registers your hand close to the duster when it comes in contact with the bottom of the duster, rather than centrally.

# Multiplayer

Multiplayer was created using an application called Photon. PUN 2 and Photon Voice 2 was downloaded and installed from unity asset store into the project. After installation was complete an account was made on the photon website and new application types called PUN and Voice were created to give us an App ID that will be used to connect players together.

Text

Description automatically generated with medium confidence

After that’s done a package was downloaded off GitHub where it had ‘PhotonVRManager’, ‘PhotonVRPlayerSpawner’ and ‘PhotonVRVoice’.

* PhotonVRManager: This is essentially what creates server using App ID and allows voice recordings. Sets up server location, sets room size and what objects would be the player’s head and hands. In my case the head would be the oculus tracking space and the hands would be the hand tracking objects.

Graphical user interface, application

Description automatically generated

* PhotonVRPlayerSpawner: This gameobject spawns players into the world using a prefab to represent them in game.

Graphical user interface, text, application, chat or text message

Description automatically generated

* PhotonVRVoice: This GameObject allows players to communicate with each other.



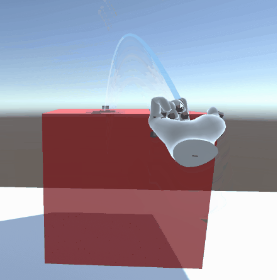
# Old Movement

## MRTK

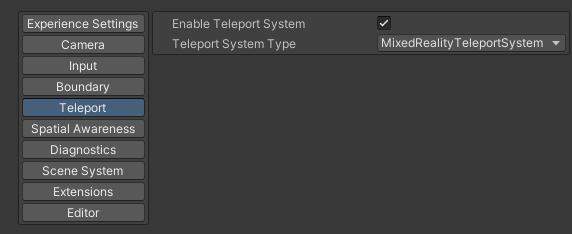
MRTK-Unity is a Microsoft-driven project that provides a set of components and features, used to accelerate cross-platform MR app development in Unity. It provides the cross-platform input system and building blocks for spatial interactions and UI. Enables rapid prototyping via in-editor simulation that allows you to see changes immediately. Operates as an extensible framework that provides developers the ability to swap out core components.

To use MRTK and foundation unity package was downloaded off GitHub and imported into the Unity project.

## Hand Teleportation using MRTK



MixedRealityToolkit object was added into the scene which was used to register hand movement in the scene.

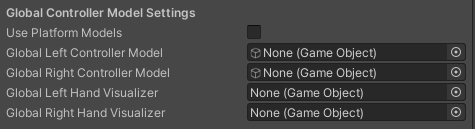


Teleport was enabled using the system type called “MixedRealityTeleportSystem”. This essentially checks the conditions that must be met in order to teleport. Your palm must be facing up, with the thumb and the index finger extended outwards in order to draw a teleportation array from your index finger. To actually teleport to the desired destination, you must curl your index finger.

A close-up of a stethoscope

Description automatically generated

Using MRTK and Oculus Integration means that 2 different systems have their own Hand visualizers and camera controllers, so in order to avoid overlapping off objects, I turned off hand visualizers for MRTK so that the Oculus Integration prefabs could be used, but kept the camera controllers on so that it could still see your hand so that it could register the conditions needed to teleport.



# New Movement

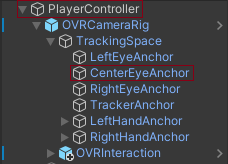
## Player Movement

Text

Description automatically generated

This script works as follows.

Two public game objects, one being the user which is going to be the “PlayerController” object and the other being the camera which is the “CentreEyeAnchor” object.



In the update method the code states that the player’s position needs to be going forward relative to the direction that the centre eye object is facing at the speed of 1. So, this means wherever the user looks, that is the direction they will be moving towards without having to use controllers to turn. This script is attached to the “PlayerController” object and is turned off.

Graphical user interface, application

Description automatically generated

## Hand pose

To allow the user to move using hand tracking I used the new hand pose gesture that oculus integration had developed recently. All that a user has to do is point their index finger straight with the palm facing down.

A screenshot of a computer

Description automatically generated with medium confidence

Figure Hand pose components

To create a pose, I created an empty object named “Point Right” as I will create it for right hand specifically for the moment.

Graphical user interface, application

Description automatically generated

Added a “Hand Ref” component where I dropped my right hand prefab that is in the scene so that it would register my right hand.

Afterwards I created a new shape by right clicking create/oculus/integration/SDK/pose detection/shape. Here I added a Curl and Flexion to each finger on the arm. Curl represents how bent the top two joints of the finger/thumb are, the Flexion is the extent that the knuckle joint is bent.

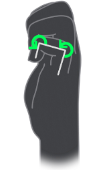


Figure 2 Curl Figure 3 Flexion

So to create a pose with the index finger pointing straight the Curl and Flexion is open meaning there are no bends to its joints.

A picture containing graphical user interface

Description automatically generated

Added a shape recognizer component and dragged the hand ref and the newly created shape into it.

Graphical user interface, text, application

Description automatically generated

To recognize the shape, I added an active state selector and dragged the shape recognizer into it.

Graphical user interface, text, application

Description automatically generated

Now that the pose is working I had to edit it so that the pose recognizer only works when the palm is facing downwards and the index is pointed forward. To do this I added a transform recognizer component, here I selected the transform feature to palm down and set it as true and set the vector type to world space. As well as dragged the hand ref into it and set the feature threshold to default.

Graphical user interface, application

Description automatically generated

Since I have two active states that need to be checked at the same time, I added an active state group component, there I dragged the active state of the shape and transform recognizer so that it can check if these two states are set to true and set the logic operator to “and”.

Graphical user interface, application

Description automatically generated

## Making the user move continuously without velocity jump

To make myself move in my application I had to write the movement script in an update method so that while the script is active the character would move forward. As well as there was an issue with my velocity which resulted in my flying up into the air whenever the pose becomes false. To fix these problems I created the following script that was attached to the hand pose object.

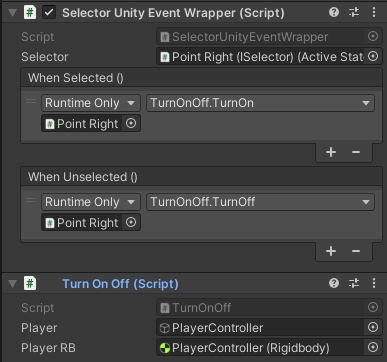
Text

Description automatically generated

In the script above I have a rigid body that will be set to the user which allows me to manipulate their gravity and kinematic on and off and a game object so that I could turn on the movement script on and off.

In the turn on method, I had the players rigid body turn on its gravity and turn off the kinematic so that it would walk on the floor and had the user’s movement script be set to true.

In the turn off method, I had the players rigid body turn off its gravity so that the sudden unexpected velocity change of the user won’t make it fly up into the air and set the kinematic to true. As well as setting the movement script to false.



I added the turn on and off script added to the hand pose object’s selector event wrapper, so when the pose is set to true it would use the turn on method and when the pose is set to false it would be using the turn off method.

# Controller

## Movement

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Created two variables, one being a reference to the users’ rigid body and another for setting the speed of the movement.

The “joystickAxis” variable gets the axis of the of the left thumb stick when it is pressed on the left controller.

To make the player move we get the players position and we add a forward motion in the direction the joystick is moved. So if the joystick goes left or right it will move the player on the x axis and if the player pushed the thumb stick up or down the player will move accordingly on the y axis at the speed that we set at the start.

To not make the player fly up into the air when the thumb stick is pushed up, we equal the players’ position to a new vector that gets the players x, y and z position.

## Turning

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

Created 3 variables, one being the speed at which we turn, another to get the users rigid body and the final one being the users rotation transform which we rotate the person around that point.

The “joystickAxis” variable gets the axis of the of the right thumb stick when it is pressed on the right controller.

Created two if statements, which essentially check how much have you actually push the thumb stick to the side. So for the first if statement if the thumb stick is 80 percent pushed to the right it will then change the players rotation around the rotators position. The second if statement is the exact same as the right one except it turns left when the thumb stick is pushed 80 percent to the left.