The implementation of the second generation of a Multimodal Virtual Reality System for small animals

Undergraduate Thesis

Submitted in partial fulfillment of the requirements of
BITS F421T Thesis

By

Raghav Prasad
ID No. 2017A7TS0297G

Under the supervision of:

Dr. Mayank Mehta
&
Mr. Chinmay Purandare

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
December 2020

Declaration of Authorship

I, Raghav Prasad, declare that this Undergraduate Thesis titled, 'The implementation of the second generation of a Multimodal Virtual Reality System for small animals' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

Certificate

This is to certify that the thesis entitled, "*The implementation of the second generation of a Multimodal Virtual Reality System for small animals*" and submitted by Raghav PRASAD ID No. 2017A7TS0297G in partial fulfillment of the requirements of BITS F421T Thesis embodies the work done by him under my supervision.

Supervisor	*Co-Supervisor*
Dr. Mayank MEHTA	Mr. Chinmay PURANDARE
Professor,	PhD student,
University of California, Los Angeles	University of California, Los Angeles
Date:	Date:

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

Abstract

Bachelor of Engineering (Computer Science)

The implementation of the second generation of a Multimodal Virtual Reality System for small animals

by Raghav Prasad

Electrophysiological recordings in mobile animals in a multimodal real-world environment is challenging and has left a lot to be desired in terms of understanding the neural mechanisms of multimodal integration. Thus, a non-invasive experimental setup conducive to electrophysiological recordings is advantageous. This was accomplished in 2013 at the W.M. Keck Center for Neurophysics at UCLA, where a custom Virtual Reality setup was independently developed and used to non-invasively record rodents in a highly immersive virtual environment.

This system allows the body of the rodent to be fixed in the real world, without head fixation. The rat is placed on top of a silent, spherical treadmill, which provides movement inputs to drive the motion of the rodent in the virtual world. This opens up a multitude of possibilities to subject the rodent to a wide variety of stimuli without the constraints of the real world.

This project aims to improve upon the existing system by revamping the codebase, and introducing new features which will allow researchers to conduct experiments with new virtual worlds and complex stimuli.

Acknowledgements

I would like to thank Professor Mehta for giving me the opportunity to work on this project and for the guidance he has provided throughout the duration of the project. I would also like to thank Mr Chinmay Purandare for being an excellent mentor, for his able guidance and patience and for always leading me towards a solution whenever I hit a mental block and did not know how to proceed.

I would like to acknowledge Professor Sujith Thomas for agreeing to be my supervisor from BITS and also Professor Bharat Deshpande, for overseeing the off-campus thesis program at BITS which has presented me with this opportunity.

Lastly, I would like to express my gratitude for the open-source community and the members of various forums for being prompt and responsive towards some the queries I posted when I had difficulty using some of the software tools. Special thanks to Dr Richard J. D. Moore for helping me through the challenges I faced during the installation of FicTrac. And also to the team developing and maintaining EyeLoop who have been prompt in addressing the concerns I posed regarding bugs in the software.

Contents

List of Figures

List of Tables

Abbreviations

CMOS	**C**omplementary **M**etal **O**xide **S**emiconductor
CPU	**C**entral **P**rocessing **U**nit
FPS	**F**rames **P**er **S**econd
GPU	**G**raphics **P**rocessing **U**nit
HDR	**H**igh **D**ynamic **R**ange
VR	**V**irtual **R**eality

Constants

Centimeter to Meter conversion factor	CentimeterToMeter	=	0.01
Millisecond to Second conversion factor	MillisecondToSecond	=	0.001
Localhost IP address	LOCALHOST	=	127.0.0.1
FicTrac Port number	FictracPort	=	12346
Trackball Radius	TRACK_BALL_RADIUS_M	=	0.3048 m
Height of Avatar above ground	AvatarBaseHeight	=	0.0508 m

Chapter 1

Introduction

Virtual reality (VR) is being increasingly used in many fields, in particular in neuroscience. VR presents researchers with the opportunity to present multimodal stimuli with a high degree of ecological validity and control[3]. Moreover, it allows researchers to construct apparatuses that are unconstrained by the physical world's limits and create interesting stimuli, which would be extremely difficult to maintain in a real-world lab. VR also reduces the number of real physical components for an experimental setup, substituting them with fewer components that can be overloaded to achieve multiple functions. This reduces variability in results due to measurement or operational errors caused by an increased number of different physical components or devices, making the results of each experiment highly reproducible. This would also reduce the need to replace the apparatus due to normal wear and tear, thus making maintenance easier.

A proper understanding of multimodal cognitive integration requires highly controlled and precise stimuli while eliminating any potential variation due to noise in the form of unwanted cues. Virtual environments present researchers with this high degree of control by allowing a high level of immersion in an enclosed and controlled space. Virtual environments also allow for easier electrophysiological recording since the subject is fixated (with varying degrees of freedom depending on the setup), which means that the recording equipment need not be lightweight or mobile. Thus, researchers do not need to sacrifice recording quality for the sake of mobility. Moreover, since the environment is virtual, the recording of other parameters also becomes much more manageable. Researchers have full information regarding the subject's position and orientation at all instants of time, without the need for recording equipment since this information can be inferred from the software controlling the virtual environment. The resolution and granularity of the recording can be adjusted to suit the experimenter's needs.

The current (first) generation of the VR system has been in operation since 2013. The complete development of the system took close to 4 years, starting in 2009. Broadly, the components of the VR setup are:

- A cylindrical screen
- A roughly hemispherical convex mirror
- Projector
- Spherical treadmill (trackball)
- Data acquisition hardware
- Arduino boards

The current (first generation) implementation of the VR system was limited by both the hardware and software available to the researchers at the time. A C++ graphics engine called OGRE, which is now deprecated, formed the crux of the VR software used to generate the virtual environments. The data acquisition hardware had hardware and software redundancies, which can now be eliminated as hardware has become much cheaper. Custom fabricated tracking sensors were being used to track the movement of the spherical treadmill, which can now be replaced by much more robust solutions using computer vision.

This project aims to improve upon the existing setup and add some new features as well. This was accomplished as follows:

- Using the Unity game engine to replace the earlier version, which used OGRE. Unity uses C# (or JavaScript), so this meant that the entire codebase had to be revamped.
- Replacing the earlier custom tracking sensor implementation of motion tracking with a computer vision-based approach to tracking called FicTrac.
- Replacing redundant hardware used to synchronize electrophysiological recordings with the VR with an Arduino.
- Adding new elements to the ensemble of elements that can be used to construct a virtual world
- Formulating a template for new virtual environments that were previously not possible to build

This project will present the following benefits to researchers:

- Creation of newer, more complex virtual stimuli to study different multimodal integrations
- Easier expansion of existing virtual reality software because of a modular design

- Less variability across different runs of the same experiment because of a reduced number of hardware components
- Improved granularity and resolution of a recording of neural and positional data

Chapter 2

Materials and Methods

2.1 Current VR Setup

The current VR setup is composed of a number of different elements forming a cohesive real-time system.

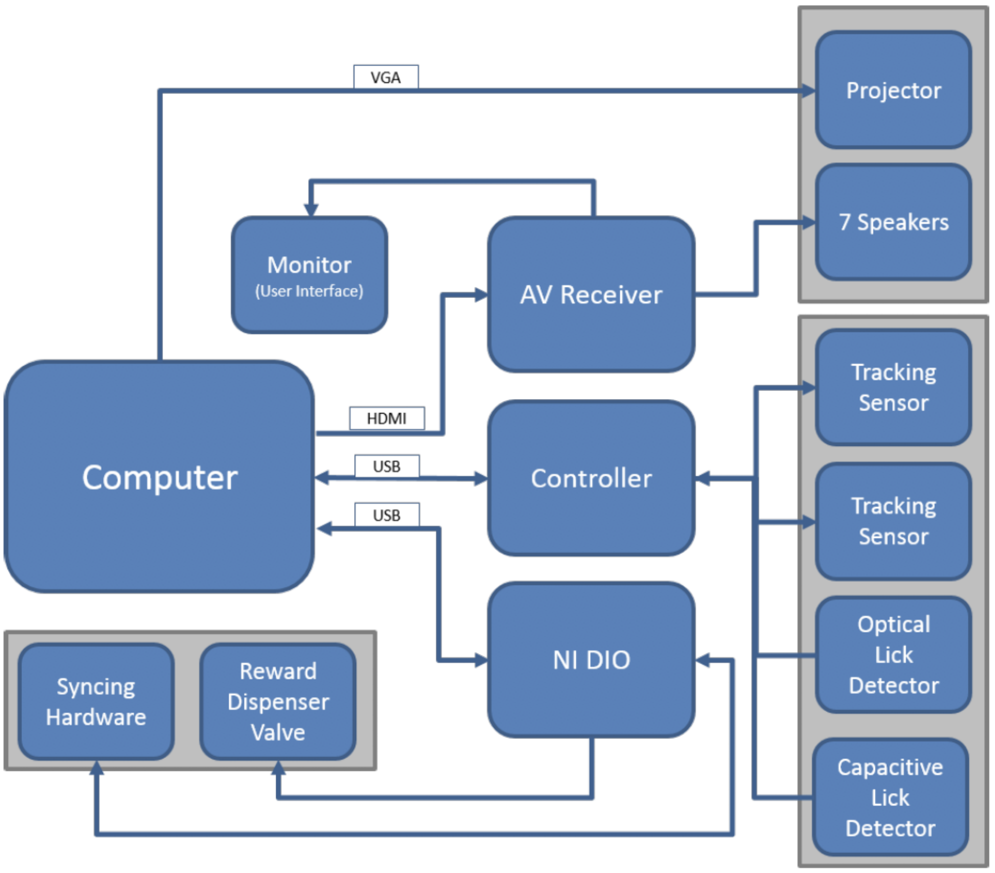


FIGURE 2.1: Schematic of the current (first) generation VR system[1]

2.1.1 Cylindrical screen and mirror

The cylinder is 75 cm tall and has a radius of 36 cm with a 13 cm opening at the center of the cylinder's ceiling. This opening holds the convex mirror and the support structure for the projector. There is another arc-like cutout on the cylinder floor, which houses the top of the spherical treadmill. The screen is made of a non-laminated lightweight fabric used to make lampshades. This allows an azimuthal visual stimulus coverage of close to 330 degrees and total coverage of nearly 4π radians[1].

The mirror is a roughly hemispherical convex piece of aluminum. The mirror is machined out of 6061 aluminum and then hand polished. The mirror's curvature is calculated from the geometry of the system using a finite element simulation method. This process only needs to be done once and yields a `.mesh` representation of the mirror. Unity cannot parse `.mesh` files, so this was converted into an XML file, which specifies the *triangles* and *vertices* necessary to generate any mesh (see procedural mesh generation section). Using this, a mesh is generated and saved as a Wavefront (`.OBJ`) file.

2.1.2 Trackball and motion tracking system

The spherical treadmill (from here on referred to as the trackball) is a hollow styrofoam sphere of radius 30.48 cm and mass roughly equal to 1.5 times the average rat's mass. This trackball has three rotational degrees of freedom and zero translational degrees of freedom. The entire trackball setup consists of an air support cushion, three centering ball bearings, and two motion tracking laser sensors.

The motion tracking system consists of a control unit and two motion tracking sensors. The tracking sensors are custom fabricated boards built around a high-end CMOS laser sensor. The sensors essentially perform the function of a pair of optic mice to give three degrees of rotational freedom, but with low latency and high precision.

2.1.3 Other peripherals and sensors

The VR setup also consists of a 7-speaker array for auditory stimulus generation. There is also a projector that projects the virtual environment onto the overhead mirror. There is a reward dispenser valve used to deliver rewards to the rat, controlled by an Arduino. The entire VR is generated and delivered from a central computer, which houses the software for generating and managing the VR behavior. The software controlling the VR is a custom-built application using a C++ graphics engine called OGRE. There is an independent data acquisition system called

Neuralynx Cheetah, synchronized with the VR using a series of data acquisition boxes (NIDAQ) and a separate computer using a custom synchronization software called RatTracker.

2.2 Second generation VR setup: Software and hardware

2.2.1 Blender

Blender is a free and open-source 3D computer graphics software built using Python[4]. Blender was used to construct complex 3D models that are integral components of the VR setup.

2.2.1.1 Virtual projection setup, predistortion and UV mapping

As described before, the lab uses a unique cylinder and mirror projection setup to achieve a 330° screen coverage in the horizontal plane. Since no projector has a horizontal field of view of 330°, the lab uses one projector aimed at the ceiling of the cylinder onto a nearly hemispherical convex mirror. The image is reflected off the mirror and onto the surface of the cylinder. This presents an intriguing challenge. When the flatscreen image, i.e., the image of the virtual world as we see it on a flat screen, is projected directly onto the mirror, the mirror will distort this image, and we will get a distorted image on the cylindrical screen. Thus, the flatscreen image of the virtual world must be transformed such that after reflection off the mirror, it appears undistorted on the cylindrical screen. This transformation is called predistortion (see Figure 2.2).

To achieve this predistortion, a virtual setup is created in the virtual world to mirror the real setup at the lab, i.e., a cylinder of matching dimensions and an overhead mirror was constructed using Blender and procedural mesh generation (ref. Section 2.2.2.2), respectively. Positioned at a height of 2 inches (approximately 5.08 centimeters) from the center of the base of the cylinder, to coincide with the position of the rodent's eyes, is an array of 6 virtual cameras, each with a vertical and horizontal field of view of 90 degrees (see Figure 2.3). The view from each of these virtual cameras is then projected onto the inside of the virtual cylinder, each of them projecting it onto the region of the cylinder that falls within the camera frustum of that camera. Thus, to ensure that the projection of each camera's view is on the correct region of the cylinder, the cylinder is cut along the points of intersection of the camera frustum with the cylinder (see Figure 2.4). Once the projection of the camera views is obtained on the inside of the virtual cylinder (see Figure 2.5), the virtual mirror will now reflect this, and there is one virtual camera pointed at the mirror from underneath to mimic the real projector projecting onto the mirror (see Figure 2.6). Thus, we are using the principle of reversibility of the path of light to perform the predistortion.

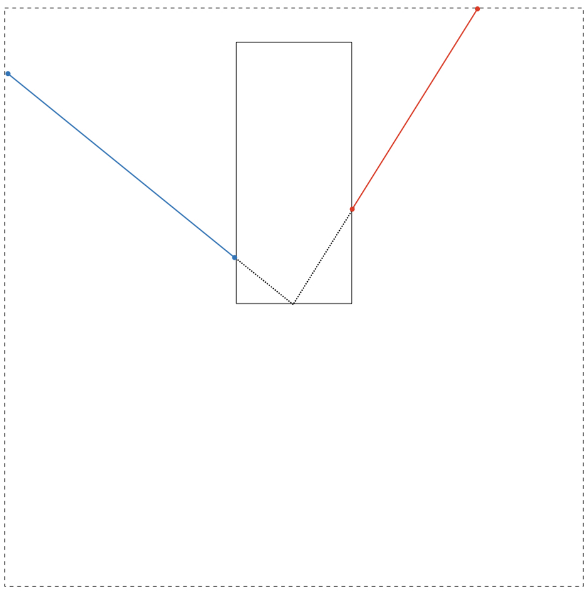
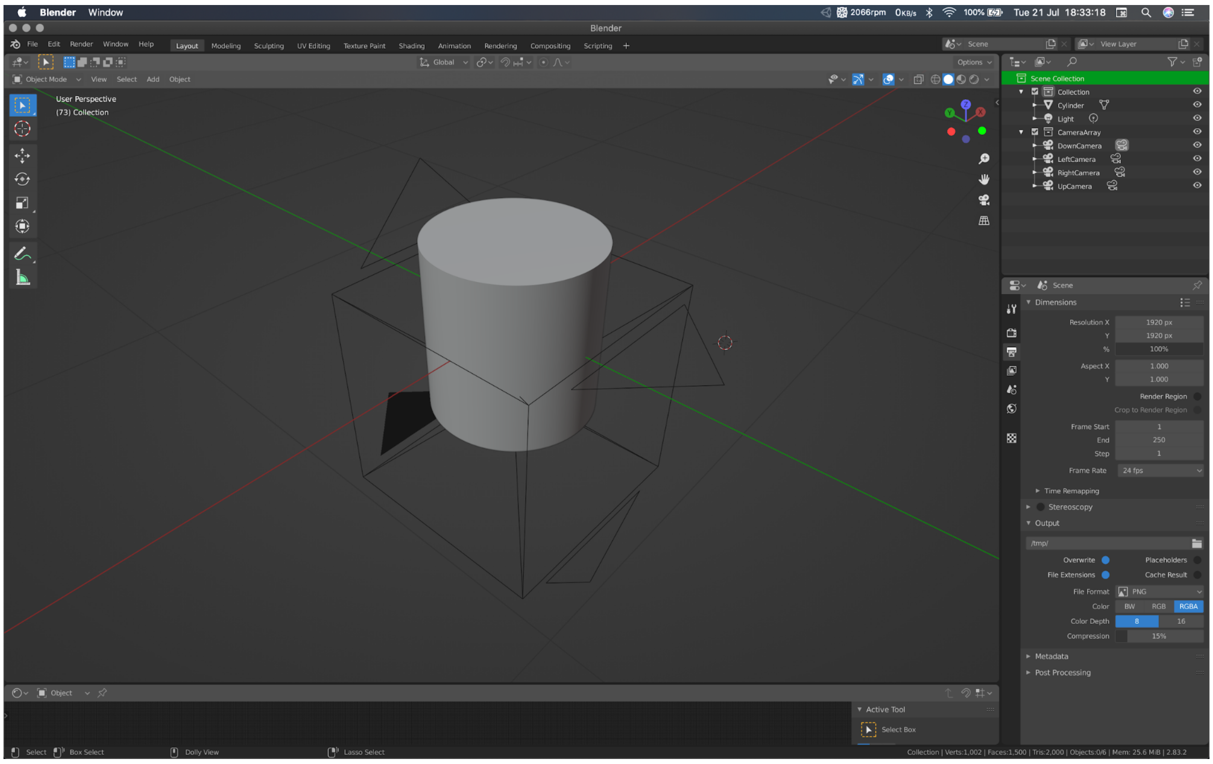


FIGURE 2.2: Predistortion; mapping the virtual world onto the virtual cylinder. This is a cross-sectional view of the virtual world + hollow cylinder setup. The red pixel is projected from the cubical room to the cylinder such that the extended ray's path passes through the centre of the base of the cylinder. Similarly for the blue pixel.


Blender Window
Tue 21 Jul 18:33:18
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
Object Mode View Select Add Object
Global
Options
User Perspective
(73) Collection
Select Box Select Dolly View Lasso Select
Active Tool
Select Box

Scene Collection
Collection
Cylinder
Light
CameraArray
DownCamera
LeftCamera
RightCamera
UpCamera

Scene
Dimensions
Resolution X 1920 px
Y 1920 px
% 100%
Aspect X 1.000
Y 1.000
Render Region
Crop to Render Region
Frame Start 1
End 250
Step 1
Frame Rate 24 fps
Time Remapping
Stereoscopy
Output
/tmp/
Overwrite Placeholders
File Extensions Cache Result
File Format PNG
Color BW RGB [✓ RGBA]
Color Depth [✓ 8] 16
Compression 15%
Metadata
Post Processing

Collection | Verts:1,002 | Faces:1,500 | Tris:2,000 | Objects:0/6 | Mem: 25.6 MiB | 2.83.2

FIGURE 2.3: Virtual cylinder setup with 6 virtual camera array

We use Unity to render each of the six virtual cameras' views onto the inside of the virtual cylinder. This rendering is done by capturing the views of each of the cameras onto a 2D texture.

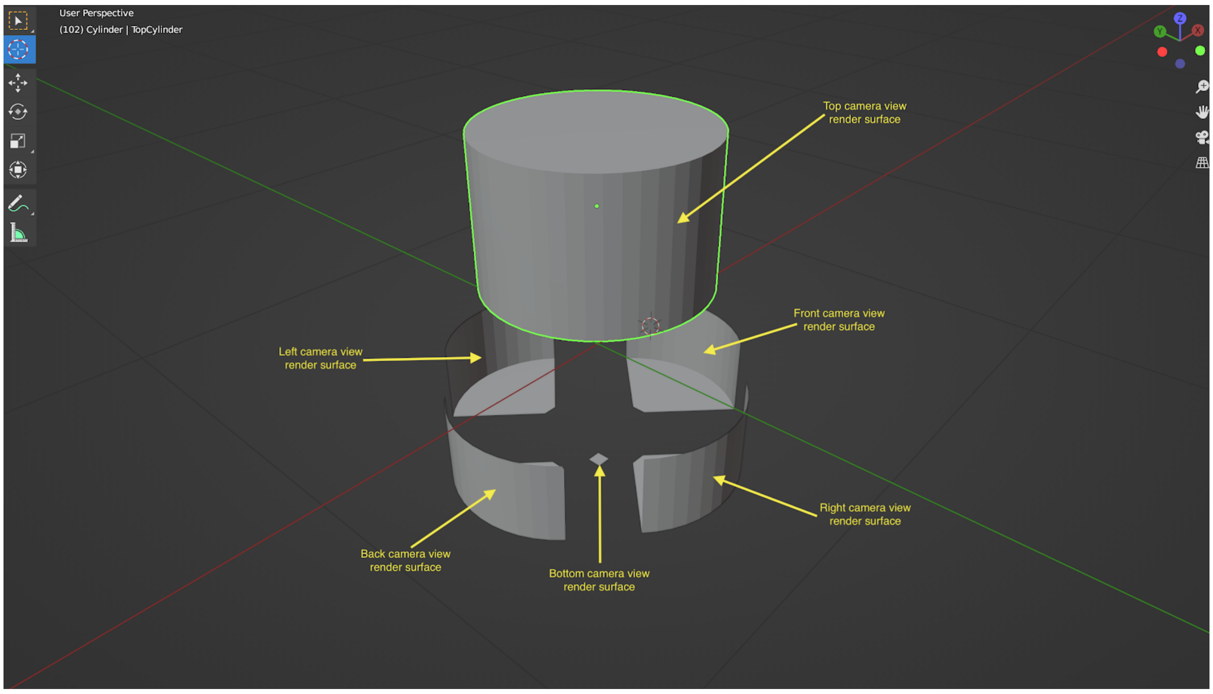

User Perspective
(102) Cylinder | TopCylinder
Top camera view render surface
Front camera view render surface
Left camera view render surface
Right camera view render surface
Back camera view render surface
Bottom camera view render surface

FIGURE 2.4: Separation of the cylinder into discrete regions for each camera to project its view

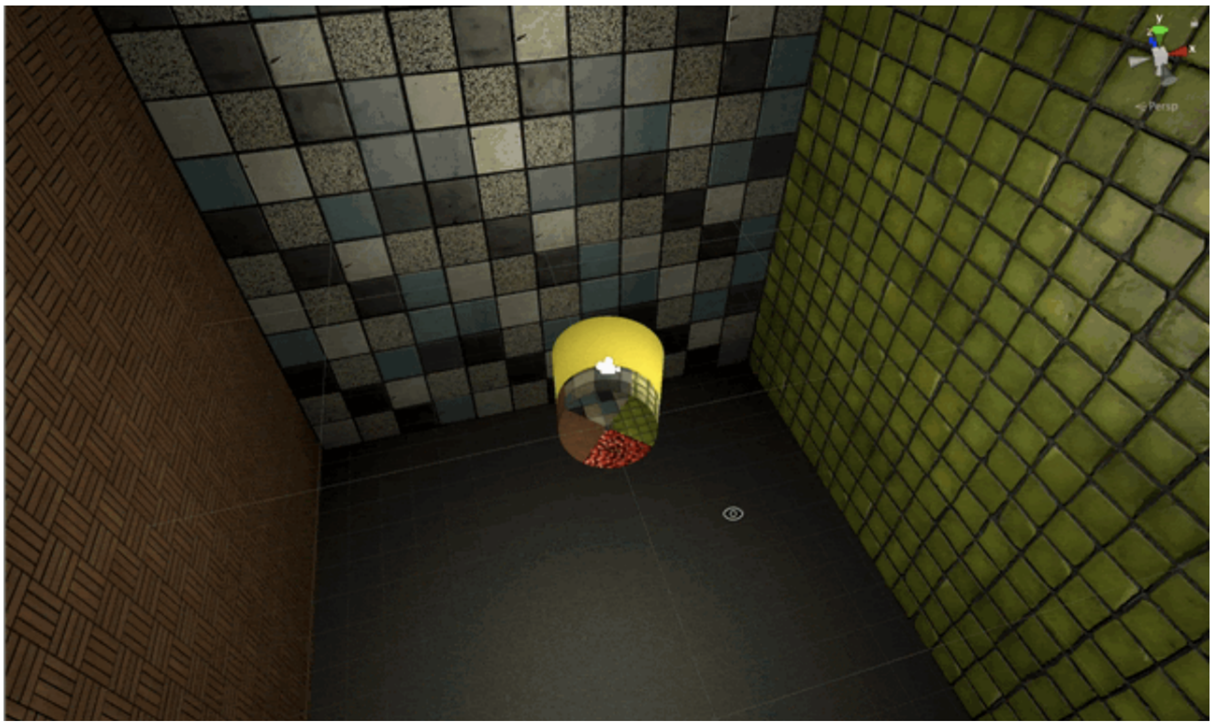


FIGURE 2.5: Projection of the world onto the inner surface of the virtual cylinder

However, this rendering happens improperly due to incorrect UV-mapping. UV-mapping is the process of mapping a 2D texture onto a 3D surface. The 2D texture is a 512 $\times$ 512 px square, whereas the surface of each of the projection regions of the cylinder is not regular, and the default mapping does not map the texture correctly in terms of the texture area covered and the

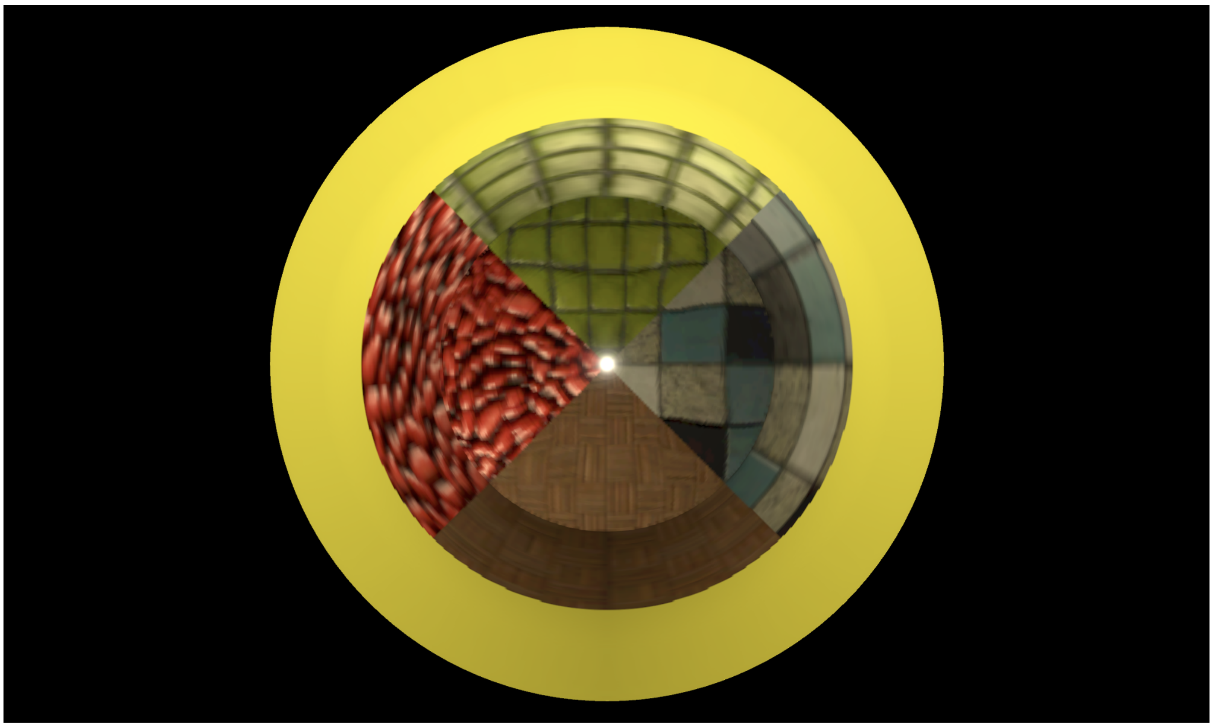


FIGURE 2.6: The view of the reflection in the virtual mirror, corresponding to the projection obtained in Figure 2.5

texture orientation with respect to the cylinder surface. This required manual correction using the UV editor in Blender.

The UV editor overlays an ”unfolded” version of the 3D surface over the 2D texture. This unfolded 3D surface is rotated and stretched symmetrically to ensure that the entire 2D texture is rendered onto the 3D surface and is appropriately aligned and oriented.

2.2.2 Unity

The Unity game engine is an industry-leading piece of software used to make professional games and XR applications[6]. Given the highly robust nature of the engine and the vast amount of online community support it has, it was an obvious candidate to replace the OGRE graphics engine. Some of the advantages of using Unity are:

- Includes a GUI to make development easier for non-programmers. This will be of great importance in the maintenance of code since the researchers using the VR application typically won’t be from a programming background.
- Includes an online store that provides a platform for a host of third party plugins (called assets), making development very easy and highly modular. This project makes use of three different assets.

- Includes lots of prebuilt functions to add and manipulate physics in an effortless manner.

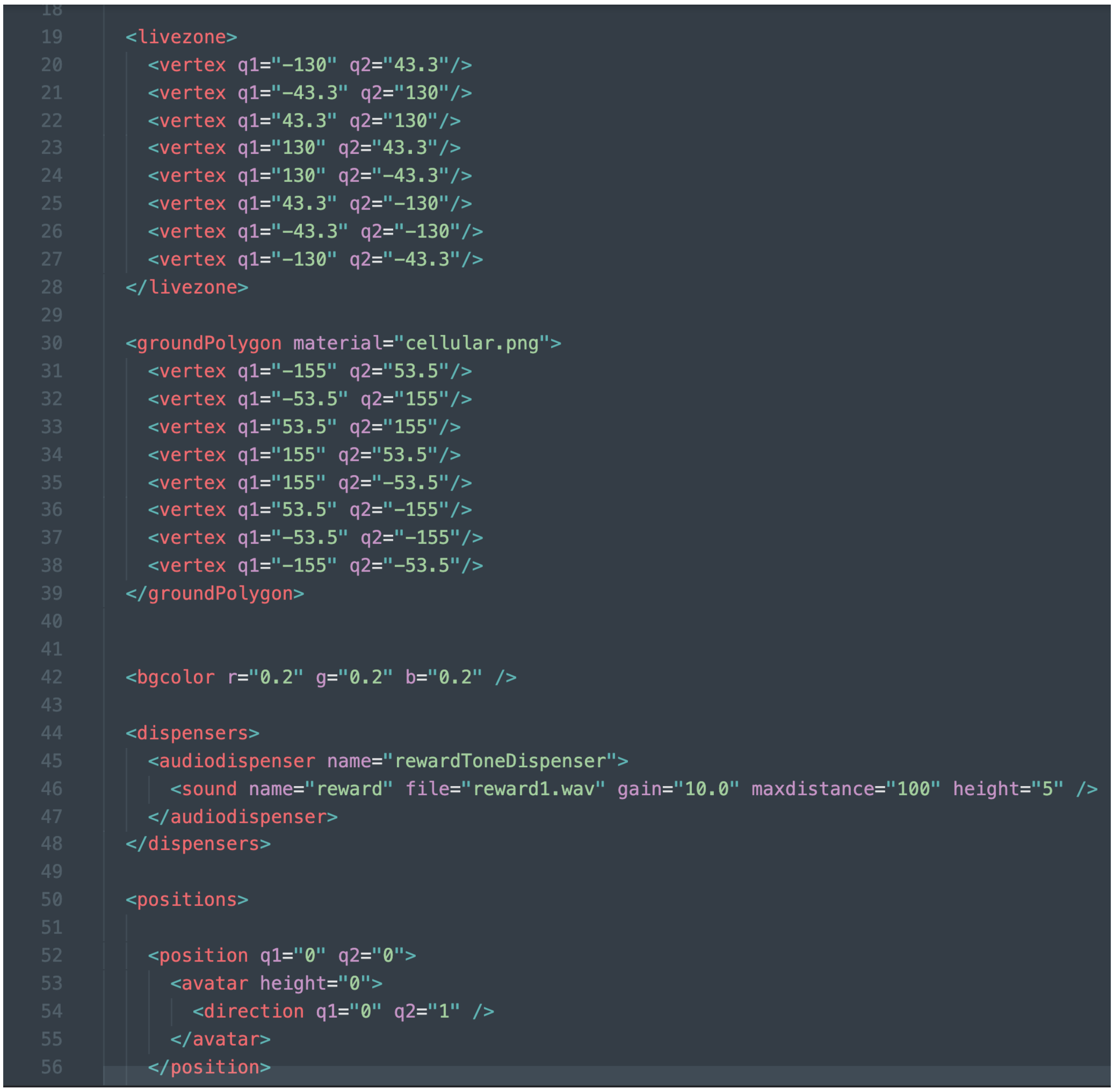
Unity uses C# or JavaScript. To keep the codebase similar to the first generation VR, this project has been written in C#. This is, however, not a simple transpilation from the existing C++ codebase to C#. The reason for this is that the Unity engine's power allows the code to be much more succinct, and it would only be prudent to use this by revamping the codebase without sacrificing any functionality but instead adding to it. A summary of the most important scripts in this project is given in Table 2.1.

File name	**Location**	**Description**
Data.cs	`Final-VR\Assets\Scripts\Data\Data.cs`	This contains two functions, LogHeaders and LogData. LogHeaders prints the column headers to the log file for the current run. Its called once when the Avatar is spawned. LogData is called every frame and it writes timestamped log data for each frame on a separate line
FictracController.cs	`Final-VR\Assets\Scripts\Utils\FictracController.cs`	Contains functions to control and stream data from Fictrac
NeuralynxController.cs	`Final-VR\Assets\Scripts\Utils\NeuralynxController.cs`	StartNeuralynxArduino() sends signal to Arduino to start sending TTL pulses to Digital Lynx SX. StopNeuralynxArduino() sends signal to Arduino to stop sending TTL pulses to Digital Lynx SX
TrackFileParser.cs	`Final-VR\Assets\Scripts\WorldBuilder\Tracks\TrackFileParser.cs`	Contains functions to parse .track files and build a Track(`Final-VR\Assets\Scripts\WorldBuilder\Tracks\Track.cs`) object
GameObjectBuilder.cs	`Final-VR\Assets\Scripts\WorldBuilder\GameObjectBuilder.cs`	Contains functions to instantiate GameObjects corresponding to all elements of the virtual world as described in a track file
TrackBuilder.cs	`Final-VR\Assets\Scripts\WorldBuilder\TrackBuilder.cs`	Opens a File Explorer to choose a track file, parses it by calling the TrackFileParser's ParseTrack function and then calls functions from GameObjectBuilder to instantiate GameObjects for the resultant Track

Table 2.1: Important source files

This project uses Unity version 2020.1.10f1

The construction of the VR world takes place from scratch, entirely using code. Since an experimenter would want to test several different hypotheses, it would only make sense to subject the rodent to different stimuli. This is achieved by using a custom world for each different type of experiment. These worlds are called ”tracks.” Each track has a different layout and is comprised of different components in different configurations. These tracks are encoded in track files. Track files are an XML-style document that encodes the quantitative and qualitative data associated with the components that constitute a track. Track files are parsed by Unity scripts and are translated into ”GameObjects” (the fundamental entities in Unity).



```xml
<livezone>
  <vertex q1="-130" q2="43.3"/>
  <vertex q1="-43.3" q2="130"/>
  <vertex q1="43.3" q2="130"/>
  <vertex q1="130" q2="43.3"/>
  <vertex q1="130" q2="-43.3"/>
  <vertex q1="43.3" q2="-130"/>
  <vertex q1="-43.3" q2="-130"/>
  <vertex q1="-130" q2="-43.3"/>
</livezone>

<groundPolygon material="cellular.png">
  <vertex q1="-155" q2="53.5"/>
  <vertex q1="-53.5" q2="155"/>
  <vertex q1="53.5" q2="155"/>
  <vertex q1="155" q2="53.5"/>
  <vertex q1="155" q2="-53.5"/>
  <vertex q1="53.5" q2="-155"/>
  <vertex q1="-53.5" q2="-155"/>
  <vertex q1="-155" q2="-53.5"/>
</groundPolygon>


<bgcolor r="0.2" g="0.2" b="0.2" />

<dispensers>
  <audiodispenser name="rewardToneDispenser">
    <sound name="reward" file="reward1.wav" gain="10.0" maxdistance="100" height="5" />
  </audiodispenser>
</dispensers>

<positions>

  <position q1="0" q2="0">
    <avatar height="0">
      <direction q1="0" q2="1" />
    </avatar>
  </position>
```

FIGURE 2.7: A sample track file snippet

The VR application allows the experimenter to choose the track file at the beginning of the experiment. This track file is then parsed to produce the virtual world in which the rodent is free to move around.

2.2.2.1 Tracks

Virtual worlds are called tracks. A track is defined by an XML style file called a track file (see Figure 2.7) and has the .track extension. Track files contain definitions for all the constituent components of the virtual world. A typical track will contain four walls and a floor, along with a ground polygon, an elevated polygonal plane surface that the avatar will move on. A subregion of the ground polygon will be demarcated as the live zone. Depending on the track, it will have zones (visible or invisible) with different actions or functions associated with them. For instance, a linear track has a plank for a ground polygon and has two reward zones, one at either end of the plank where the rat will receive a reward (sugar water). Dispensers carry out the actions or functions associated with the zones (ref. Section 2.2.2.7).

2.2.2.2 Meshes and Procedural Mesh Generation

A 3D mesh is composed of triangles arranged in 3D space. Triangles are in turn defined by three vertices. A Mesh in Unity is defined by two arrays; `vertices` and `triangles`. The `vertices` array is an array of type `Vector3` and contains the vertices that will constitute the triangles. The `triangles` array is an array of type `int` that specifies the indices in the `vertices` array used to define the triangles in 3D space, which will ultimately define a 3D mesh. The `triangles` array must have a length that's a multiple of 3. This is because the triangles array elements are considered in groups of three to constitute the triangles. For instance, a quad can be defined by the following arrays:

$$vertices = \{(0,0,0),(1,0,0),(1,1,0),(0,1,0)\}$$

$$triangles = \{0,1,2,2,3,0\}$$

The track usually consists of a room composed of 4 walls and a floor, and an elevated flat polygonal surface for the rat to move around. Thus many regular planar GameObjects need to be produced. Out of these, the ones that are planar quads are instantiated from “prefabs” (ref. Section 2.2.2.4). The elevated flat polygonal surface is usually not regular and needs to be dynamically generated once at runtime. These are produced using a technique called ”Procedural Mesh Generation.” Procedural mesh generation is a method whereby one can create meshes of any shape and size out of `vertices` and `triangles` from within a script. These primary meshes can then create more elaborate and complex structures by warping them or composing and building on top of them.

2.2.2.3 Backface Culling

Since the VR system needs to be light and low latency, every possible optimization has been made to make the VR as fast and responsive as possible. One of the most important optimizations is backface culling. Backface culling is the process by which only one side of any surface is rendered. More specifically, the back face of any surface is "culled," i.e., it is removed or not rendered. This is based on the assumption that only one side of a plane is visible at any point in time.

Furthermore, most scenarios would be such that only one side is ever presented to the character. This makes backface culling a beneficial optimization and has been used in this project as well. Backface culling manifests itself during Procedural Mesh Generation; since we are generating meshes at runtime, we must be careful about which face we are exposing. The orientation of a surface is determined by the normal of that surface. The direction of the normal determines the front face of any surface. Since the elementary unit of any surface is a triangle, a continuous surface will have, more often than not, multiple normals; one each for all its constituent triangles. This is always true for curved surfaces. Thus, when backface culling is on, one must ensure that each triangle's front face is facing in the direction opposite to the direction of vision of the character.

In Procedural Mesh Generation, since the triangles defining the meshes are themselves defined using vertices, the meshes' orientation is ultimately dependent on the vertices. The rule in Unity is that if a triangle is defined using three vertices ordered in the clockwise sense from a given point of view, the triangle's front face will be exposed to that point of view; otherwise, the backface will be exposed to that point of view. Thus, if the array of vertices is presented in a clockwise sense, the surface will be oriented correctly, else it will need to be reversed. To check whether the vertices array is clockwise or counterclockwise, the following function is used:

$$\sum_{i=0}^{N-1}(x_{(i+1)\%N} - x_i)(z_{(i+1)\%N} + z_i)\begin{cases} < 0 & counterclockwise \\ else & clockwise \end{cases}$$

where x_j and z_j are the x and z coordinates of the jth vertex,
N is the length of the array, and
% denotes the modulo operation

2.2.2.4 Prefabs

Prefabs are shorthand for "prefabrications." These refer to GameObjects that have been designed as templates to be instantiated at run time. Once prefabs are instantiated at runtime as GameObjects, the GameObject instances can be manipulated in terms of scale, position, and rotation about any of the three axes to modify the size, location, and orientation in space. Prefabs are designed and stored in a known location. This project makes use of four prefabs:

- **CircularPlane**: This prefab is used to instantiate reward zones
- **Cube**: This prefab is used to spawn invisible walls to create boundaries demarcating the live zone of the character. The live zone of the character is the region of the virtual environment within which the character is constrained to move.
- **Plane**: This prefab is used to create walls for the rooms of the virtual environment
- **Participant**

The Participant prefab is the most important and complex of the four prefabs used in this project. This prefab is used for the instantiation of the virtual rat avatar. The structure of the prefab is as follows:

```
Participant
├─ Avatar Capsule
├─ Serial Controller
├─ Main Camera
└─ Projection Setup
   ├─ final_cylinder
   │  ├─ Back
   │  ├─ Bottom
   │  ├─ Front
   │  ├─ Left
   │  ├─ Right
   │  ├─ Top
   │  ├─ Reflection Probe
   │  └─ Mirror Camera
   ├─ mirror
   │  └─ mirror
   └─ Camera Array
      ├─ Front Camera
      ├─ Back Camera
      ├─ Right Camera
      ├─ Left Camera
      ├─ Top Camera
      └─ Bottom Camera
```

The **Avatar Capsule** is representative of the rat. It contains three important components:

- A Capsule Collider
- A Rigidbody
- A script; DontGoThroughThings.cs

The Avatar Capsule is the entity that must be restricted within the live zone of the virtual environment. This is achieved with the help of the two components mentioned above. The *Capsule Collider* is a Collider that will detect collisions with other Colliders (which will be present on the invisible walls acting as the live zone boundaries). The Capsule Collider is also necessary to detect incursion into trigger zones.

The *Rigidbody* component is used to define the physics of the Avatar Capsule. We know that movement along the vertical axis, i.e., jumping upward or dropping downward, is not permitted. The Rigidbody is configured to include a constraint that freezes its position along the Y (vertical) axis. Moreover, it configures the collision detection mode for the Avatar Capsule to be Continuous Dynamic. This prevents the Avatar Capsule from breaching the boundary walls by fast repetitive collisions.

The third component is a C# script that rectifies a known Unity issue, which causes very high-velocity objects to pass through barriers. This can occur if the object's velocity is so high that the object's position changes from being on one side of the barrier to being on the other side in consecutive frames, without the chance for collision detection. The DontGoThroughThings.cs script uses RayCasting to detect objects and preempt collisions to prevent the problem described above.

SerialController is a prefab within the Participant prefab. SerialController is made available through the Ardity asset. SerialController has the SerialController.cs script, which contains fields to set parameters for the Arduino such as the Serial port name and the Baud rate. It also provides functions to send serial data to the Arduino from within the VR.

Main Camera gives a first-person perspective of the virtual environment from the point of view of the rat. This view is without predistortion and is streamed to Display 2. Display 2 refers to any video out device connected to the computer running the Unity VR application.

The Projection Setup has three major components:

- final_cylinder
- mirror
- Camera array

final_cylinder is composed of six subparts that form the whole cylinder. These are the regions obtained as the output from the cylindrical mesh-cutting process in Blender. The cutting of the cylinder was required to discretize the regions on which each of Camera Array cameras will have their view rendered. Each subpart of final_cylinder has its `Material` with its `Albedo` set to the appropriate texture on which its corresponding camera's view is rendered.

final_cylinder also houses a Reflection Probe and a Mirror Camera. The Reflection Probe is a spherical object that maps the region in a specified cuboidal volume onto itself as a reflection. This reflection is then rendered onto other "shiny" surfaces. The Reflection Probe has a Resolution setting, which determines the resolution of the reflection image it captures. This needs to be set judiciously since this reflection image refreshes every frame and can prove to be unnecessarily expensive. This project sets it to 512.

Additionally, this project chooses to have HDR turned off since the virtual environments do not have elaborate lighting that would merit the benefit of having HDR turned on. This also boosts performance. Moreover, since rats are nocturnal, having complex lighting and HDR would be wasteful.

Lastly, the **Mirror Camera** is simply a camera pointed at the mirror. It has a viewport size of 800×600. The mirror camera displays to Display 1, which corresponds to the projector in the lab.

mirror is simply the Wavefront (.OBJ) file that was ultimately obtained from the finite-element simulation process (ref Section 2.1.1). mirror has a different `Material` applied to it with maximum smoothness and metallic character. This makes it a perfect reflecting surface and hence, an ideal mirror.

Camera Array contains six virtual perspective cameras pointing along up, down, front, back, left, and right directions. Each camera has a vertical field of view of 90 degrees and an aspect ratio of 1:1, and each of their views is rendered onto their corresponding 512×512 px 2D texture. Thus, a combination of the six cameras' views yields a cubemap of the virtual world, which is projected on the cylinder.

2.2.2.5 Layers

Unity provides a feature called Layers. This feature allows one to have multiple layers and instantiate GameObjects to different layers. Each layer is assigned an integer value, starting from 0. Up to 32 layers can be defined in any project. The first eight layers are builtin and cannot be changed by a user. This project uses four user-defined layers, namely Projection Setup, Mirror, Avatar, and Play Area.

The Projection Setup layer contains only the Projection Setup GameObject. The Mirror layer contains the mirror GameObject. The Avatar layer contains the Avatar Capsule GameObject. The Play Area layer houses all other GameObjects.

The utility of Layers is seen in two cases:

- Camera culling masks
- Layer based collision detection

A Camera culling mask is essentially a 32-bit mask for each Camera GameObject. The positions where the bits are set denote the layers, and by extension, the GameObjects in those layers are visible to a particular camera. The other layers are invisible to the Camera. The default Camera culling mask for all cameras is *Everything* (all bits set to 1).

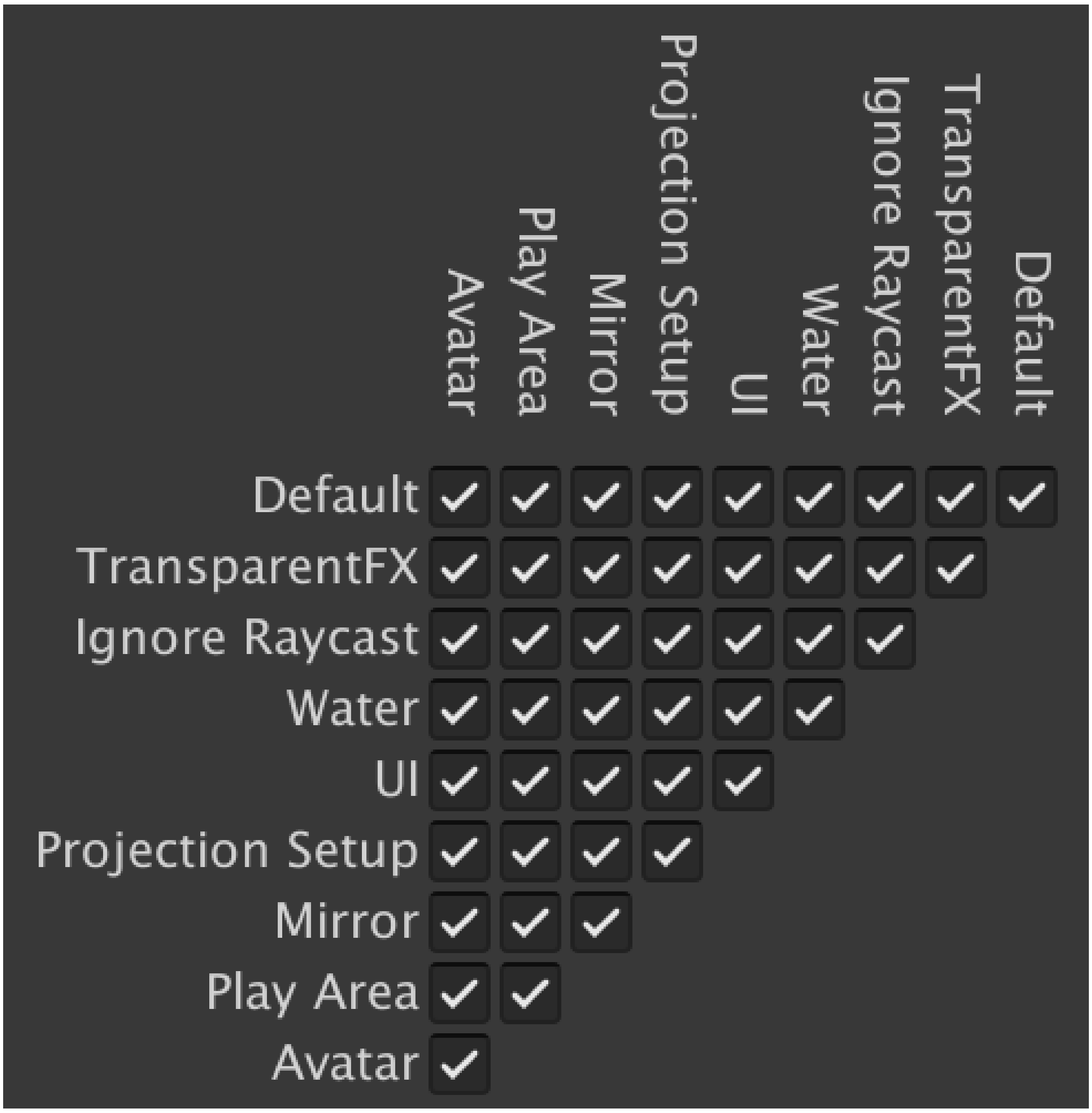
A modified Camera culling mask is used in each of the Camera Array cameras, and the Projection Setup bit is reset. This is essential because the Camera Array cameras are meant to view the virtual world while being housed inside the Projection Setup cylinder. This modified culling mask ensures that the cylinder is transparent to the cameras in the Camera Array. Similarly, the Mirror Camera has a modified culling mask with only the Mirror layer bit set. This ensures that the Mirror Camera sees nothing but the mirror.

Layers are also used in the Unity physics engine to determine collisions. Unity allows a user to configure which layers, and by extension, their GameObjects are opaque or transparent with respect to each other in terms of collisions and collision detection. In other words, one can set a pair of layers such that all the GameObjects from one layer can pass through all the GameObjects in the other layer without any collision detection, and vice-versa. This can be modified using the Physics setting in Unity, which provides a Layer Collision Matrix (see Figure 2.8). If an entry in the matrix is checked, the objects in that pair of layers will be subject to collision detection, else not.

Layer-based collision detection is used in this project to keep the Avatar within the live zone of the virtual world. Since the Avatar GameObject is spawned using the Participant prefab, the Avatar includes the Projection Setup and the Avatar Capsule, which is actually representative of the rat. Thus, we would want the collision detection to occur with the Avatar Capsule only and not the entire Avatar. This is achieved by setting the Avatar Capsule layer to Avatar and the layer of the rest of the Avatar to Projection Setup. After this, the Layer Collision Matrix is modified, and the Projection Setup - Play Area entry in the matrix is unchecked. This would allow all the Projection Setup GameObjects to pass through the Play Area GameObjects and vice-versa. However, the Avatar Capsule and the GameObjects in the Play Area layer, including the invisible live zone boundaries, will be subject to collision detection, precisely the desired behavior.

2.2.2.6 Logging

Along with electrophysiological recording, it is vital to log events occurring in the virtual world. To that end, the rat's position is recorded every frame into a CSV file (see Figure 2.9). Similarly, significant events occurring in the virtual world must also be recorded, events such as completing a subset of a foraging task or receiving a reward.

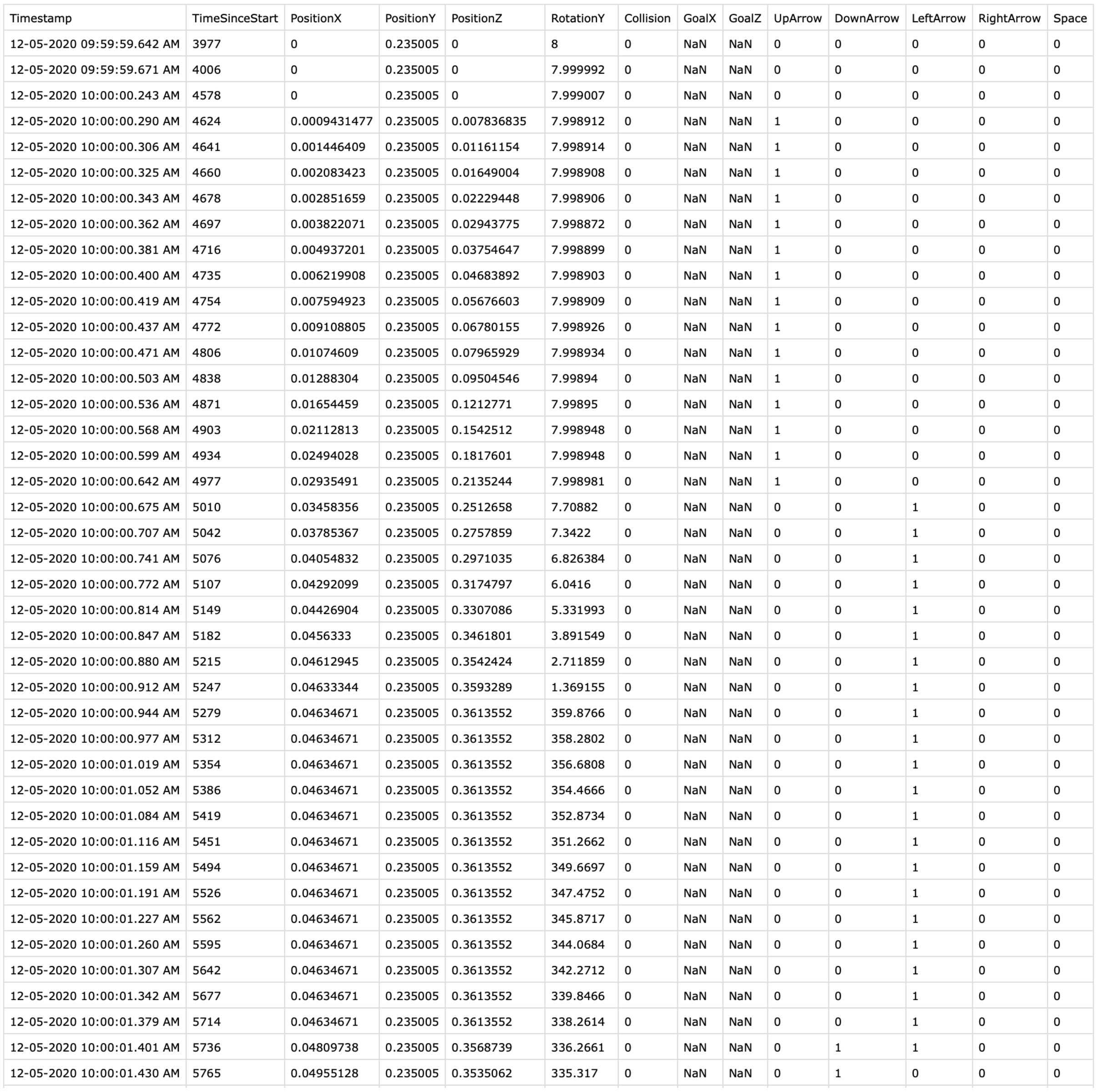


	Avatar	Play Area	Mirror	Projection Setup	UI	Water	Ignore Raycast	TransparentFX	Default
Default	✓	✓	✓	✓	✓	✓	✓	✓	✓
TransparentFX	✓	✓	✓	✓	✓	✓	✓	✓	
Ignore Raycast	✓	✓	✓	✓	✓	✓	✓		
Water	✓	✓	✓	✓	✓	✓			
UI	✓	✓	✓	✓	✓				
Projection Setup	✓	✓	✓	✓					
Mirror	✓	✓	✓						
Play Area	✓	✓							
Avatar	✓								

FIGURE 2.8: The Layer Collision Matrix

2.2.2.7 Dispensers

Dispensers is the generic term used to describe all the components that deliver stimuli to any of the modalities except vision. This includes gustatory stimuli such as sugar water reward, olfactory stimuli such as different odors, and auditory stimuli such as audio clips. The Unity scripts handle the abstract representations (GameObjects) of the dispensers. Their real-world counterparts are valves controlled using Arduinos. The hardware for the different dispensers is overloaded, with one valve and its corresponding Arduino controller being used for multiple virtual dispensers.



Timestamp	TimeSinceStart	PositionX	PositionY	PositionZ	RotationY	Collision	GoalX	GoalZ	UpArrow	DownArrow	LeftArrow	RightArrow	Space
12-05-2020 09:59:59.642 AM	3977	0	0.235005	0	8	0	NaN	NaN	0	0	0	0	0
12-05-2020 09:59:59.671 AM	4006	0	0.235005	0	7.999992	0	NaN	NaN	0	0	0	0	0
12-05-2020 10:00:00.243 AM	4578	0	0.235005	0	7.999007	0	NaN	NaN	0	0	0	0	0
12-05-2020 10:00:00.290 AM	4624	0.0009431477	0.235005	0.007836835	7.998912	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.306 AM	4641	0.001446409	0.235005	0.01161154	7.998914	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.325 AM	4660	0.002083423	0.235005	0.01649004	7.998908	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.343 AM	4678	0.002851659	0.235005	0.02229448	7.998906	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.362 AM	4697	0.003822071	0.235005	0.02943775	7.998872	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.381 AM	4716	0.004937201	0.235005	0.03754647	7.998899	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.400 AM	4735	0.006219908	0.235005	0.04683892	7.998903	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.419 AM	4754	0.007594923	0.235005	0.05676603	7.998909	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.437 AM	4772	0.009108805	0.235005	0.06780155	7.998926	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.471 AM	4806	0.01074609	0.235005	0.07965929	7.998934	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.503 AM	4838	0.01288304	0.235005	0.09504546	7.99894	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.536 AM	4871	0.01654459	0.235005	0.1212771	7.99895	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.568 AM	4903	0.02112813	0.235005	0.1542512	7.998948	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.599 AM	4934	0.02494028	0.235005	0.1817601	7.998948	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.642 AM	4977	0.02935491	0.235005	0.2135244	7.998981	0	NaN	NaN	1	0	0	0	0
12-05-2020 10:00:00.675 AM	5010	0.03458356	0.235005	0.2512658	7.70882	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:00.707 AM	5042	0.03785367	0.235005	0.2757859	7.3422	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:00.741 AM	5076	0.04054832	0.235005	0.2971035	6.826384	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:00.772 AM	5107	0.04292099	0.235005	0.3174797	6.0416	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:00.814 AM	5149	0.04426904	0.235005	0.3307086	5.331993	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:00.847 AM	5182	0.0456333	0.235005	0.3461801	3.891549	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:00.880 AM	5215	0.04612945	0.235005	0.3542424	2.711859	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:00.912 AM	5247	0.04633344	0.235005	0.3593289	1.369155	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:00.944 AM	5279	0.04634671	0.235005	0.3613552	359.8766	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:00.977 AM	5312	0.04634671	0.235005	0.3613552	358.2802	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:01.019 AM	5354	0.04634671	0.235005	0.3613552	356.6808	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:01.052 AM	5386	0.04634671	0.235005	0.3613552	354.4666	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:01.084 AM	5419	0.04634671	0.235005	0.3613552	352.8734	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:01.116 AM	5451	0.04634671	0.235005	0.3613552	351.2662	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:01.159 AM	5494	0.04634671	0.235005	0.3613552	349.6697	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:01.191 AM	5526	0.04634671	0.235005	0.3613552	347.4752	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:01.227 AM	5562	0.04634671	0.235005	0.3613552	345.8717	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:01.260 AM	5595	0.04634671	0.235005	0.3613552	344.0684	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:01.307 AM	5642	0.04634671	0.235005	0.3613552	342.2712	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:01.342 AM	5677	0.04634671	0.235005	0.3613552	339.8466	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:01.379 AM	5714	0.04634671	0.235005	0.3613552	338.2614	0	NaN	NaN	0	0	1	0	0
12-05-2020 10:00:01.401 AM	5736	0.04809738	0.235005	0.3568739	336.2661	0	NaN	NaN	0	1	1	0	0
12-05-2020 10:00:01.430 AM	5765	0.04955128	0.235005	0.3535062	335.317	0	NaN	NaN	0	1	0	0	0

Figure 2.9: Sample log CSV

2.2.2.8 Triggers

For each of the zones and their corresponding actions, there needs to be a trigger that is set off once the Avatar enters the zone. The track file specifies triggers that define which dispensers need to be activated and deactivated upon entry into a particular zone. This is handled by creating a Trigger class that can store the data specified in the track file. Unity's mechanism to detect the entry of the Avatar into a zone is also called a trigger. This project makes use of these Unity triggers to set off the track file triggers. The Unity trigger mechanism works as follows:

- Each zone is spawned using either the CircularPlanePrefab or Procedural Mesh Generation.

- In either method, the GameObject instantiated is given a Mesh Collider component and this Mesh Collider has its isTrigger field set to true.
- Now, whenever another collider collides with the Mesh Collider on the zone GameObject, it will execute a piece of code which will, in turn, execute the triggers specified in the track file.

2.2.3 Arduino and Ardity

Arduino is an open-source electronics platform based on easy-to-use hardware and software. This project uses a single Arduino Uno R3 board. The Arduino is used in conjunction with the Arduino IDE to write, debug, and upload code to the Arduino. The Arduino is overloaded and is used to control multiple hardware components. The Arduino Uno R3 has 14 five-volt digital output pins. Eight of these pins are used to send 8-bit TTL pulses to the electrophysiological recording hardware. A subset of the remaining six pins is used for other functions, such as controlling dispenser valves. The main code for the Arduino is uploaded once at the outset. The code execution is controlled via synchronous and asynchronous control signals sent from the VR application. The interface between Unity and the Arduino is provided by a third-party asset called Ardity. Ardity allows bidirectional communication over COM ports from Unity[7]. Ardity provides fields to set Arduino parameters such as the serial port name and the baud rate. It also provides functions to send serial data from the Unity application to the Arduino.

2.2.4 FicTrac

FicTrac (Fictive path Tracking software) is a novel vision-based tracking system for estimating the path an animal makes while rotating an air-supported sphere using only input from a standard camera and computer vision techniques[5]. Using FicTrac in a closed-loop configuration, i.e., the output from FicTrac is fed into the VR application to move the rodent in the VR world, the need for optic mice based motion is eliminated. This is good for the following reasons:

- The current optic mice solution for rodent motion employs circuit boards fabricated in-house at UCLA. This is a highly customized approach. While the need for the reproducibility of this setup does not arise, it is still not as good a solution as one, which requires just a camera and a patterned trackball (spherical treadmill).
- The optic mice solution requires at least two mice to cover the entire range of motion that the rodent is allowed to have in the VR world; one mouse for motion in the 2D plane and another for rotation about the vertical axis. Thus, this would require extra code to handle two mice simultaneously, on top of retrieving the data from the two mice, making the code

even more cumbersome. In contrast, given that FicTrac is a third party software tool, it makes the design much more modular and straightforward to implement.

- FicTrac is an open-source project curated by Dr. Richard J. D. Moore and receives regular community updates. Thus, as time goes on, the VR project's FicTrac module will receive updates from the community.

FicTrac is used to control the motion of the virtual character, replacing the current motion tracker system. FicTrac has the following hardware requirements:

- A patterned trackball. The pattern is a roughly regular distribution of splotches around the ball.
- A video camera. Resolution is not a constraint, and since this VR setup is meant for rats, which are relatively not very fast, the camera frame rate is also not a limiting factor.
- Adequate lighting

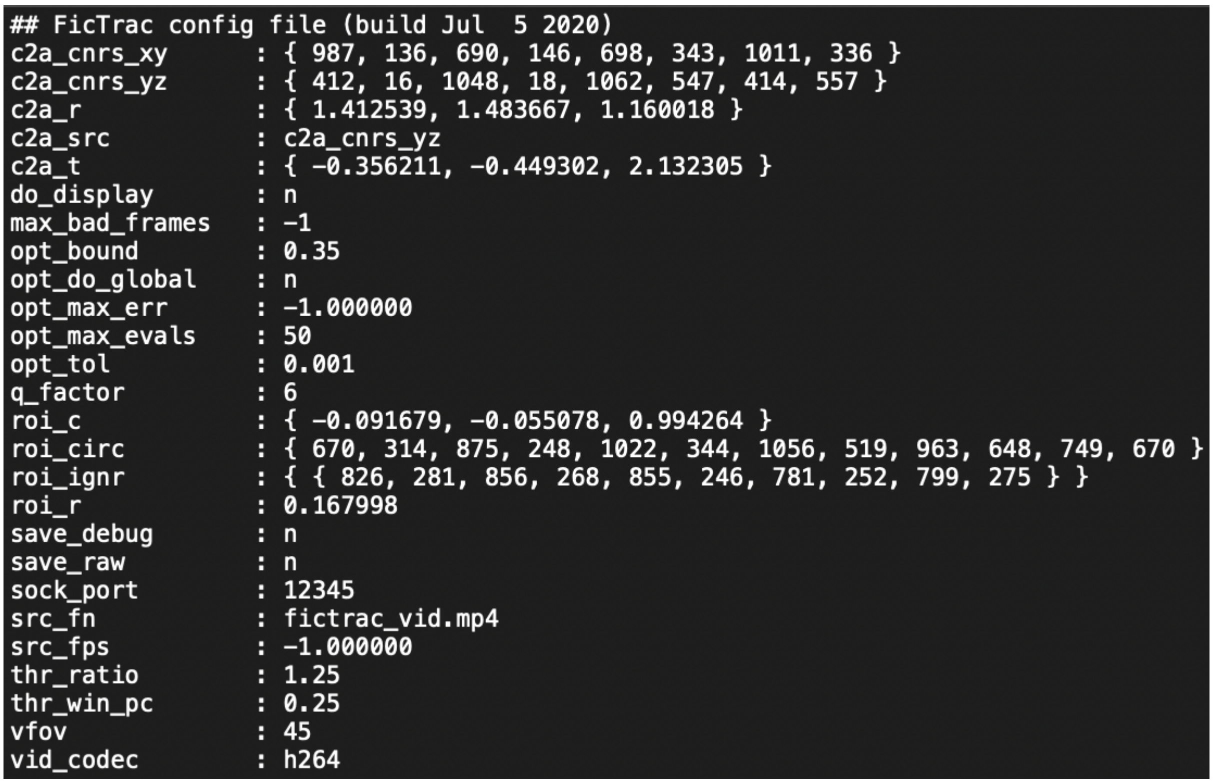
FicTrac is composed of the following software components:

- A fictrac binary which is the main application
- A configGui binary, which performs calibration
- A config.txt configuration file (see Figure 2.10) which specifies configuration and calibration parameters for configGui and later fictrac

FicTrac uses computer vision to construct a 2D map of the spherical surface and hence can estimate the position of the subject. This project employs FicTrac in a closed-loop configuration using socket programming. FicTrac is created as a server process and waits for a client to connect to it. The VR application acts as the client process; once the subject is spawned in the virtual environment, the VR application connects to the FicTrac process and streams data from the FicTrac server process (see table). The FicTrac process relays positional information regarding displacement in the XZ (horizontal) plane and orientation information regarding the change in rotation about the Y (vertical) axis from the previous frame. This information is fed to the virtual character controller script in Unity, every frame, and is used to move the virtual character around the virtual world.

2.2.5 Neuralynx Cheetah data acquisition system

Neuralynx Cheetah is the hardware system responsible for recording the neural signals from the rodent during the experiment. The VR application will interface with this using TTL pulses



```
## FicTrac config file (build Jul  5 2020)
c2a_cnrs_xy      : { 987, 136, 690, 146, 698, 343, 1011, 336 }
c2a_cnrs_yz      : { 412, 16, 1048, 18, 1062, 547, 414, 557 }
c2a_r            : { 1.412539, 1.483667, 1.160018 }
c2a_src          : c2a_cnrs_yz
c2a_t            : { -0.356211, -0.449302, 2.132305 }
do_display       : n
max_bad_frames   : -1
opt_bound        : 0.35
opt_do_global    : n
opt_max_err      : -1.000000
opt_max_evals    : 50
opt_tol          : 0.001
q_factor         : 6
roi_c            : { -0.091679, -0.055078, 0.994264 }
roi_circ         : { 670, 314, 875, 248, 1022, 344, 1056, 519, 963, 648, 749, 670 }
roi_ignr         : { { 826, 281, 856, 268, 855, 246, 781, 252, 799, 275 } }
roi_r            : 0.167998
save_debug       : n
save_raw         : n
sock_port        : 12345
src_fn           : fictrac_vid.mp4
src_fps          : -1.000000
thr_ratio        : 1.25
thr_win_pc       : 0.25
vfov             : 45
vid_codec        : h264
```

FIGURE 2.10: Sample FicTrac config file. The details of the important parameters in the config file can be found in Table 2.2

via an Arduino. The neural recording data timestamps and the corresponding TTL port values are recorded in a binary format in an “event” file generated by the Neuralynx system. The VR application logged data is later synchronized with the Neuralynx generated data offline to prepare all the data for analysis.

The current (first) generation VR setup uses two data acquisition (NIDAQ) boxes in conjunction with a custom application called RatTracker to synchronize the TTL pulses with the rest of the VR system. This was highly redundant in terms of the high number of hardware components used and the introduction of an intermediary software application. This project replaces this entire peripheral synchronization setup with a solitary Arduino board and the code that gets loaded onto it. Using the functions supplied by the Ardity asset, the VR application can send the Arduino signals in terms of integers from 0 to 255 at the appropriate time. This signal is relayed to the Neuralynx Cheetah system from the Arduino as an 8-bit digital binary TTL signal. This integer 8-bit binary encoded integer serves as the TTL port value.

Param name	**Param type**	**Default value**	**Valid range**	**Description**
src_fn	string OR int		int=$[0,\infty)$	A string that specifies the path to the input video file, OR an integer that specifies which of several connected USB cameras to use. Paths can be absolute or relative to the working directory
vfov	float		$(0,\infty)$	Vertical field of view of the input images in degrees
do_display	bool	y	y/n	Display debug screen during tracking. Slows execution very slightly
save_debug	bool	n	y/n	Record the debug screen to video file. Note that if the source frame rate is higher than FicTrac's processing frame rate, frames may be dropped from the video file
save_raw	bool	n	y/n	Record the input image stream to video file. Note that if the source frame rate is higher than FicTrac's processing frame rate, frames may be dropped from the video file
sock_host	string	127.0.0.1		Destination IP address for socket data output. Unused if sock_port is not set
sock_port	int	-1	[0,65535]	Destination socket port for socket data output. If unset or ≤ 0, FicTrac will not transmit data over sockets. Note that a number of ports are reserved and some might be in use. To avoid conflicts, you should check which UDP ports are available on your machine prior to launching FicTrac (try something like 1111)
com_port	string			Serial port over which to transmit FicTrac data. If unset, FicTrac will not transmit data over serial
com_baud	int	115200		Baud rate to use for COM port. Unused if no com_port set

Table 2.2: FicTrac config parameters

Chapter 3

Results

This final project is built into a cross-platform solution with applications for both macOS and Windows operating systems. The final application is compact; the Windows application is 224.4 MB and the macOS application is 212.7 MB.

3.1 Limitations

The application, while flexible and robust, requires a healthy amount of resources in terms of computing and graphics power. The application runs at 30 FPS on a 9th generation 6-core hyperthreaded, turbo-boost capable Intel i7 paired with an AMD Radeon Pro 5300M 4 GB with 20 compute units, with both the CPU and the GPU running at maximum speed. On systems with resources lesser than this, the application will invariably run at lower FPS. This can be alleviated to a certain extent by reducing graphics settings in the Unity project.

Chapter 4

Conclusion

This project aimed to improve the current generation of the VR system and add new features to the existing VR setup. The core graphics engine and the codebase have been revamped to produce a more efficiently performing VR application. The motion tracking system and the eye-tracking system are upgraded to use computer vision-based tools. The neural recording system has been upgraded to use less hardware efficiently.

Future improvements can include an increased variety of tracks, more varied stimuli such as virtual subjects like a person walking in the virtual world, and video cues. The projection setup and predistortion also have scope for improvement and could use a raycasting solution instead of the solution proposed in this project. This would eliminate the need to create a custom cut cylindrical mesh and instead use a standard cylinder.

4.1 EyeLoop

EyeLoop is an open-source Python-based application that easily integrates custom functions via a modular logic, tracks many eyes, including rodent, human, and non-human primate eyes, and operates well on inexpensive consumer-grade hardware[2]. EyeLoop can be used in this project to track the rodent's eyes and obtain data such as blinking, pupil dimensions, pupil direction. The current (first generation) solution is a head tracking system where a head-mounted LED is tracked. Thus, EyeLoop will be a much more robust and accurate alternative to the current solution.

Bibliography

[1] D.B. Aharoni. "Rats in Virtual Space: The development and implementation of a multimodal virtual reality system for small animals". en. In: *UCLA. ProQuest ID:* $Aharoni_ucla_0031D_11811$. Retrieved from. Merritt ID: ark:/13030/m5cn89dg, 2013. URL: https://escholarship.org/uc/item/0wd4p4mr.

[2] Simon Arvin, Rune Rasmussen, and Keisuke Yonehara. "EyeLoop: An open-source, high-speed eye-tracker designed for dynamic experiments". In: *bioRxiv* (2020). DOI: 10.1101/2020.07.03.186387. eprint: https://www.biorxiv.org/content/early/2020/07/04/2020.07.03.186387.full.pdf. URL: https://www.biorxiv.org/content/early/2020/07/04/2020.07.03.186387.

[3] C. Bohil, Bradly Alicea, and Frank A. Biocca. "Virtual reality in neuroscience research and therapy". In: *Nature Reviews Neuroscience* 12 (2011), pp. 752–762.

[4] Blender Online Community. *Blender - a 3D modelling and rendering package.* Stichting Blender Foundation, Amsterdam: Blender Foundation, 2018. URL: http://www.blender.org.

[5] Richard J.D. Moore et al. "FicTrac: A visual method for tracking spherical motion and generating fictive animal paths". In: *Journal of Neuroscience Methods* 225 (2014), pp. 106 –119. ISSN: 0165-0270. DOI: https://doi.org/10.1016/j.jneumeth.2014.01.010. URL: http://www.sciencedirect.com/science/article/pii/S0165027014000211.

[6] Unity Technologies. *Unity.* Version 2020.1.10f1. Oct. 21, 2020. URL: https://unity.com/.

[7] Daniel W. *Ardity.* Version 1.1.0. Apr. 19, 2017. URL: https://ardity.dwilches.com/.