A Haptic Glove Prototype for Tactile Feedback in 3D Interactive Applications

Michalis Busho

Diploma Thesis
July 2017
Department of Electrical and Computer Engineering
Technical University of Crete
Supervisor: K. Mania

Abstract

Although Haptic technologies have been in existence for the last couple of decades, the recent rise of virtual reality applications has intensified the demand for consumer-grade haptics as well as sophisticated methods of haptic implementation. Besides wearable technology, haptics can already be encountered in a number of everyday consumer electronics. From mobile devices, to video game controllers the simulation of touch is used to bridge the gap between reality and the virtual world.

Motivated to develop a cheap and portable system which offers the haptic sensation, this thesis offers an approach to haptic feedback provision by developing a prototype system, able to supply vibrotactile feedback through a glove. The wearable glove was designed as a CAD model using a 3D modeling application and brought to life through the process of 3D printing. As all haptic systems the implemented system comprised of a software suit determining the forces that result when a user's virtual identity interacts with an object and a device through which those forces can be applied to the user. The haptic forces were supplied to the user with the help of electric motors and the control of these motors was achieved using the Arduino microcontroller. The haptic glove is able to offer 10 different points of haptic simulation on the user's hand, two points on each finger. A haptic point exists on the tip of each finger and a second on the bottom part. Each point is enabled independently when a collision with a virtual object occurs. In addition, the glove is able to provide different vibration strengths by decreasing or increasing the voltage supplied on the motors. A 3D interactive game was also developed with the purpose of showcasing the glove's features and capabilities. The system utilizes the Leap Motion controller for hand and finger tracking and the Unity software framework for collision detection and graphics rendering.

Keywords: Haptic, Leap Motion, Tactile Feedback, Unity, Arduino, Haptic Glove

Acknowledgements

The completion of this thesis could not have been possible without the help and support from people around me, to only some of whom it is possible to give particular mention here.

First and foremost, I would like to thank my supervisor, Associate Professor Aikaterini Mania for her support, motivation, excitement and all the resources she provided me with.

I would like to thank the thesis committee, e.g., Associate Professor Ioannis Papaefstathiou and Assistant Professor Vasilis Samoladas for their time reviewing and reading this work.

In addition, I would like to thank the lab members of the graphics group and fellow students for their suggestions, feedback and interest they showed in this project.

Last but not least, my family who patiently supported me all these years in all aspects.

Contents

List of Figures

Chapter One

Introduction

1 Introduction

The recent emergence of touchless interaction applications has enabled new ways of interaction that extend traditional input mechanisms such as using keyboard and mouse. These touchless interaction applications offer opportunities for exploring new ways of interacting with the digital world without touch, and provide opportunities for humans to manipulate digital objects as though they were real-world objects.

Touchless interaction is a type of interaction that can take place without mechanical contact between the human and the artificial system. Touchless interaction involves bodily gestures and movements. Although there has been a number of recent studies on touchless interaction giving implication on how these technologies could work and presenting applications of these technologies, these studies cover very little the use of haptic technologies in touchless interaction.

Haptics are key technologies found as an essential feature enhancing the user experience in a plethora of familiar products. Whether as notification in a vibrating smartphone, tension building in a video game controller, or input confirmation in an industrial scanner, haptics technologies have now reached billions of electronics devices. Haptics can be defined as the science of applying touch sensation and control to interact with computer applications by using special input or output devices such as joysticks and data glove. Users receive feedback from computer applications in the form of felt sensations in the hand or other parts of the body.

1.1 Motivation

Haptics is critical for normal human functioning in many levels, from controlling the body to perceiving, learning, and interacting with the environment. Touchless interaction with virtual objects has contrasting properties compared to touch-based interactions with physical objects. One of the fundamental properties which touchless interaction lacks is haptic feedback. In addition, whereas touch-based interaction enables visual, auditory, and haptic feedback, touchless interaction leaves haptic feedback out of the user's interaction experience. Therefore, the motivation of this thesis lies in exploring a new way to overcome the lack of haptic feedback in touchless interaction.

1.2 Aim and Objectives

This thesis aims to present our approach to a haptic feedback method by designing and implementing a glove that would be able to provide the user with vibrotactile feedback in VR or Desktop applications. After investigating the state of the art of the haptics field, the initial and crucial step was to decide the necessary technologies, both software and hardware related, to be used. These technologies will be analyzed thoroughly in the next chapters of this thesis.

Important features of the implemented glove that are achieved are the followings:

- **Ten different points of haptic simulation**. The haptic glove is able to offer 10 different points of haptic simulation on the user's hand, two points on each finger. A haptic point exists on the tip of each finger and a second on the bottom part.

- **Scalable vibration intensity**. The types of stimulus received with real touch can vary including sense of mild touch, heavy touch, pain, pressure and more. Giving a variation of vibration strength helps us simulate in a small level the different sensations that real touch offers.

- **Independent hand regions of collision**. When grasping an object not all parts of the hand make contact with it. Tactile sensation appears only on the part of the hand that makes contact. Thus making each vibrating sensor being enabled or disabled independently from each other was another objective this implementation achieved.

- **Unity support**. By creating a prefab of a 3D hand embedded with Unity's rigid body and script components this thesis offers the possibility to use the haptic glove easily in any new Unity project. Any Unity developer can import the prefab hand, attach rigid bodies to the objects of the virtual world, connect the glove to the pc via USB and enjoy its haptic feedback capabilities.

The end result was a system which consisted of a 3D environment suite, a hand tracking device, a microcontroller for the glove and the glove itself which had on it attached 10 coin shaped vibration motors, 2 on each finger. In addition, a 3D game was also developed taking advantage of the haptic system features.

1.3 Thesis Outline

Each chapter of this thesis presents a different aspect of the implementation process. The next two chapters focus on the technological background and technical terms the reader needs to be aware of.

Chapter 2 focuses on haptics in general and the hardware devices commonly used in this field of science. In addition, related work is presented.

Since the implemented system is used as an interaction device in an interactive 3D application implemented in the Unity framework, basic concepts associated with 3D computer graphics are reviewed on Chapter 3. Animations, the rendering process, 3D models and game engines are some of the topics included.

On Chapter 4 the thought process followed when designing and implementing the haptic system is explained. The main focus of this chapter is to present the way in which software and hardware work in conjunction to achieve a haptic feedback capability. The creation of the glove, programming of the software and hardware related components are some of the sections this chapter includes

Chapter 5 covers the development process of the Demo application. The application developed is a whack-a-mole clone game. The goal of the game is to touch with the hand or hit as many rabbits as possible in a small amount of time. When the time is over the final score appears. Concepts related to 3D objects, animations, User interfaces, programming and general game development are this chapter's focus point.

Concepts related to 3D objects, programming and general game development are this chapter's focus point.

Finally, in Chapter 6 the thesis is concluded and future ideas for upgrades are presented.

Chapter Two

Haptic Technologies

2 Haptic Technologies

This chapter provides a theoretical background on the field of haptics as well as an overview on related research including a Wearable Tactile Device, a Vibrotactile Head-Mounted Display and an effort to explore the Effects of Vibrotactile Feedback on Social Presence. The section begins by introducing the definition of haptics and how human beings perceive haptic feedback. Then, a range of technologies capable of simulating the sense of touch are explained. In order to achieve a good haptic feedback result, it is important to be aware of the different methods we have in our disposal for tactile feedback provision and hand tracking. This section closes by reviewing related literature, research and implementation methods.

2.1 Touch Interaction

The human hand, the primary structure associated with the sense of touch, is extraordinary complex. With 27 bones and 40 muscles the hand offers tremendous dexterity. Scientists quantify this dexterity using a concept known as degrees of freedom. A degree of freedom is movement afforded by a single joint. Because the human hand contains 22 joints it allows movement with 22 degrees of freedom. The skin covering the hand is also rich with receptors and nerves (Figure 1), components of the nervous system that communicate touch sensations to the brain and spinal cords.

Those receptors are structures that can get information from the environment. The information when received is changed into a signal that can be understood by the nervous system. Receptors that let the body sense touch are located in the top layers of the skin called dermis and epidermis [1]. While there are many types of receptors it is worth mentioning the following ones.

Mechanoreceptors. Those receptors perceive sensations such as pressure, vibrations and texture.

Thermoreceptors. As their name suggests, these receptors perceive sensations related to the temperature of objects the skin feels.

Pain receptors. The role of these receptors is to detect pain or stimuli that can or does cause damage to the skin and other tissues of the body.

Proprioceptors. They can sense the position of the different parts of the body in relation to each other and the surrounding environment. They are found in tendons, muscles and joint capsules.

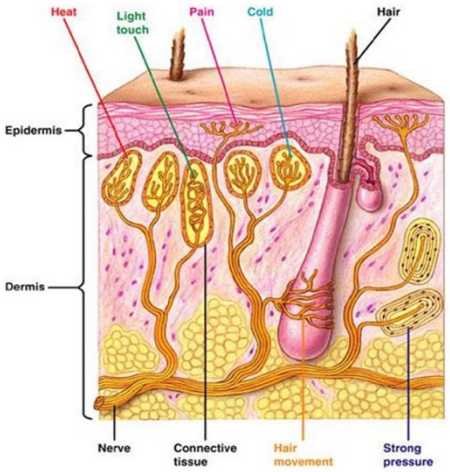


Figure 1. Skin receptors.

The haptic sensation offered by the skin's receptors includes several categories and subcategories most of which appear in the next diagram (Figure 2).

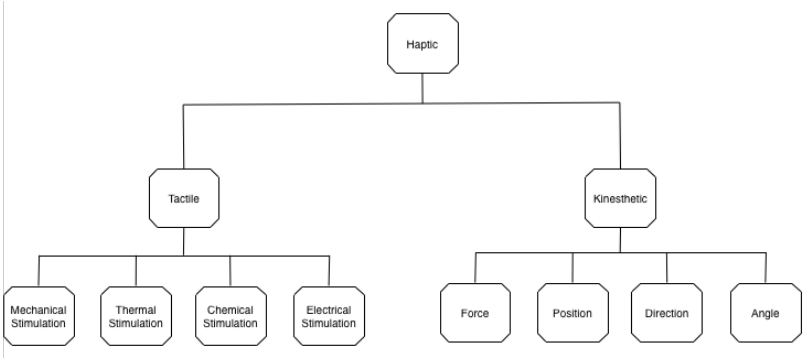


Figure 2. Haptic feedback categories (by ISO [5]).

2.2 Types of Haptic Feedback

When we use our hands to explore the world around us, we receive two types of feedback, ***kinesthetic*** and ***tactile*** [2]. As the hand reaches for an object and adjusts its shape to grasp, a unique set of data points describing joint angle, muscle length and tension is generated. This information is collected by the proprioceptors. The brain processes this kinesthetic information to provide a sense of the object's gross size and shape, as well as its position relative to the hand, arm and body. This is known as kinesthetic feedback.

In *tactile* feedback on the other hand the data received from the receptors describes the type of contact the skin makes. When the fingers touch an object, contact is made between the finger pads and the object's surface. Each finger pad is a complex sensory structure containing a number of different type of receptors both in the skin and in the underlying tissue. The data received from these receptors contains information about heat, pressure, vibration, texture or pain in the skin area contact occurred. This data helps the brain

understand subtle tactile details about the object. As the fingers explore, they sense the smoothness of the surface and the hardness of the object as force is applied.

The type of feedback the implemented haptic system aims to provide is tactile feedback. There is a number of tactile feedback devices that are applied for this task. Each one of them produces a specific tactile respond. From vibrotactile feedback to thermal, electrotactile or skin pressure, the technologies used base their operating principles on the feedback they want to achieve. This thesis explores two different approaches to tactile feedback provision. The first one is provision by skin pressure and the second through vibrotactile stimulation. Next are presented hardware components usually used to provide these types of tactile feedback.

2.3 Linear Solenoid

A way to interact with the skin in the form of pressure is by using **linear solenoids (**Figure 3**)**. Linear Solenoids are electromagnetic devices that convert electric energy directly into linear mechanical motion. Solenoid actuators require high current and provide the strongest haptic responses. They usually consist of a coil and a moveable iron core called the armature. When electrical current flows through the coil it generates a magnetic field, and the direction of this magnetic field is determined by the direction of the current flow within the coil. The strength of this magnetic field can be increased or decreased by either controlling the amount of current flowing through the coil or by changing the number of turns or loops that the coil has. The armature inside the coil is attracted towards the center of the coil by the magnetic field within the coils body, which in turn compresses a small spring attached to one end of the armature. The force and speed of the armature's movement is determined by the strength of the magnetic field generated within the coil. As the armature retracts its other end can make contact with any surface attached on it. By disabling the current inside the coil the electromagnetic field ceases to exist and the armature with the help of the spring goes back to its initial position.

One of the main disadvantages of solenoids and especially the linear solenoid is that they are "inductive devices" made from coils of wire. The solenoid coil converts some of the electrical energy used to operate them into heat due to the resistance of the wire. The longer the time that the power is applied to a solenoid coil, the hotter the coil will become. Also as the coil heats up, its electrical resistance also changes allowing more current to flow increasing its temperature. With a continuous voltage input applied to the coil, the solenoids coil does not have the opportunity to cool down because the input power is always on. In order to reduce this self-generated heating effect, it is necessary to reduce either the amount of time the coil is energized or reduce the amount of current flowing through it.

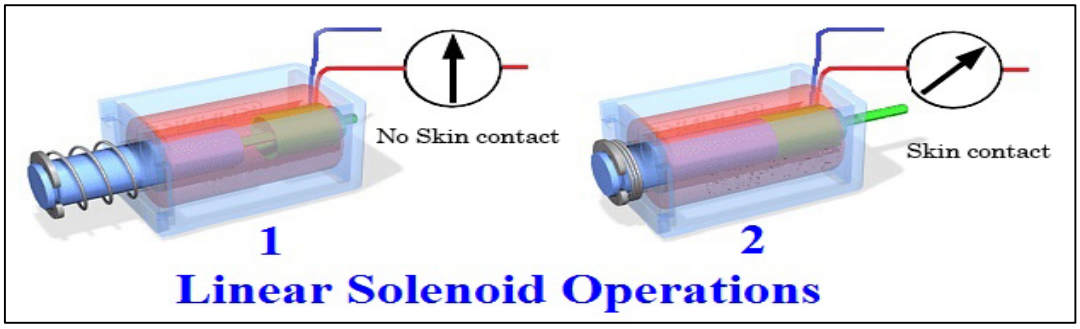


Figure 3. Linear solenoid.

2.4 Eccentric Rotating Mass Motor

Vibrotactile stimulation is the most widely used method of tactile provision in modern haptic devices. It is usually achieved using electric motors. An electric motor is an electrical machine that converts electrical energy into mechanical energy. Electric motors are used to produce linear or rotary force called torque and should be distinguished from linear solenoids that convert electricity into motion but do not generate usable mechanical powers. The most common type of electric motor used for vibrations in haptic technologies is the **eccentric rotating mass motor** (ERM) (Figure 4).

The ERM motor is similar to a regular DC motor. It uses the magnetic field of a direct electrical current to move an object in a circle. The ERM moves a small weighted object called the rotating mass that is off-center from the point of rotation. Due to the uneven centripetal force produced by the rotation of the mass, the entire motor will move back and forth to produce a vibration from side to side producing this way a lateral vibration. The intensity of vibration produced by an ERM motor will change according to the supplied voltage at its terminal.

There are two aspects of vibrations that are usually taken into consideration. The vibration frequency and vibration strength. Both the vibration frequency and the vibration strength can be found by the following equations (Precision microdrivers [3]).

Vibration Frequency (HZ) $= \frac{RPM}{60}$
HZ = Cycles per second
RPM = Revolutions per minute

Amplitude (Centripetal Force)= $m \times r \times \omega^2$
m= Mass of eccentric mass (kg)
r= Distance from the motor shaft to the center of the eccentric mass (m)
ω= Angular Velocity ($rads^{-1}$)

One of the drawback of ERM motors is that since an ERM motor must move an eccentric mass using the current it is provided, the frequency and amplitude of vibration cannot be modified independently.

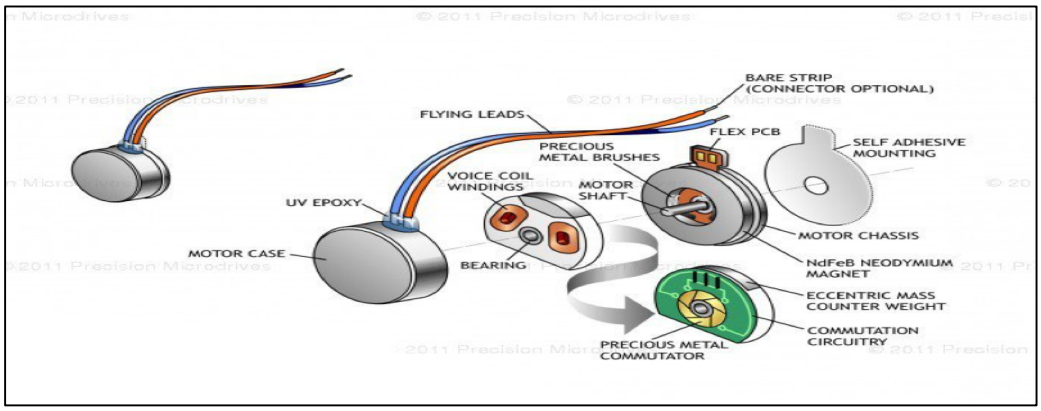


Figure 4. ERM Coin Vibrator

2.5 Linear Resonant Actuator

A more modern approach to vibrotactile provision would be with the use of a **linear resonant actuator** (LRA) (Figure 5) instead of an ERM motor. Unlike an ERM motor, *a* linear resonant actuator uses a voice coil instead of a DC motor [4]. A voice coil takes an AC input and produces a corresponding vibration with a frequency and amplitude corresponding to the electrical signal it is provided. Although LRAs also use magnetic fields and electrical current to create force, the small voice coil remains stationary inside the device. Instead of moving, the voice coil presses against a magnetic mass attached to a spring. By driving the magnetic mass up and down against the spring, the LRA as a whole will be displaced and produce a vibration. This works much like a speaker producing sound.

Instead of directly transferring the force produced by the voice coil to the skin, the device optimizes for power consumption by taking advantage of the resonant frequency of the spring. As the voice coil pushes the magnetic mass against the spring at the spring's resonant frequency, the device can produce a vibration of higher amplitude more efficiently. Since the voice coil is driven by an AC the frequency and amplitude may be independently modified, unlike the ERM motor that couples the two properties of the resulting vibration.

Since the device must be controlled with alternating current, the necessary circuit to drive the actuator is significantly more complex than a circuit used to drive an ERM motor with direct current. In spite of the increased complexity, the devices have several unique advantages. LRA's will typically consume less power to produce a vibration than an ERM motor, and their performance characteristics allow for significantly shorter start-stop times in typical applications. In addition, LRA's don't produce as much noise because they do not have a spinning mass inside of them.

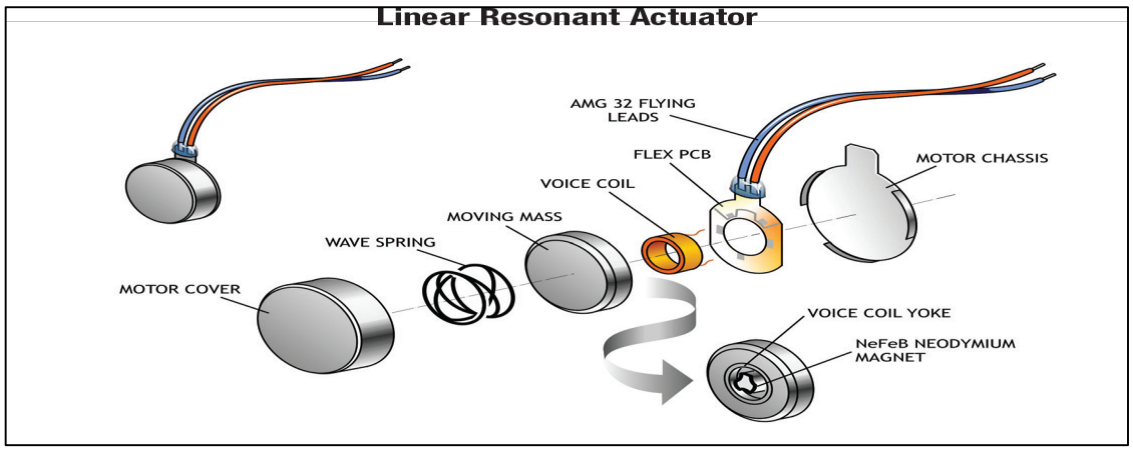


Figure 5. LRA Coin Vibrator.

2.6 Hand Tracking

In order haptic feedback to be supplied, the computing system with which the user interacts must be able to track the position and gestures of the user's hand. The capture of these hand actions can be performed by hand tracking devices. In the field of haptic technologies hand tracking and more specifically finger tracking is a technique that is employed to know the consecutive position of the hand and fingers of the user and represent them in 3D. In addition to that, these techniques are used as a tool of the computer, acting as an external device similar to keyboard and a mouse.

Throughout the years the field of haptics and touchless interaction has offered different ways to capture the hand and fingers movements. Each method has its own set of characteristics, advantages and disadvantages. Next are presented 3 different hardware components able to offer hand tracking capabilities.

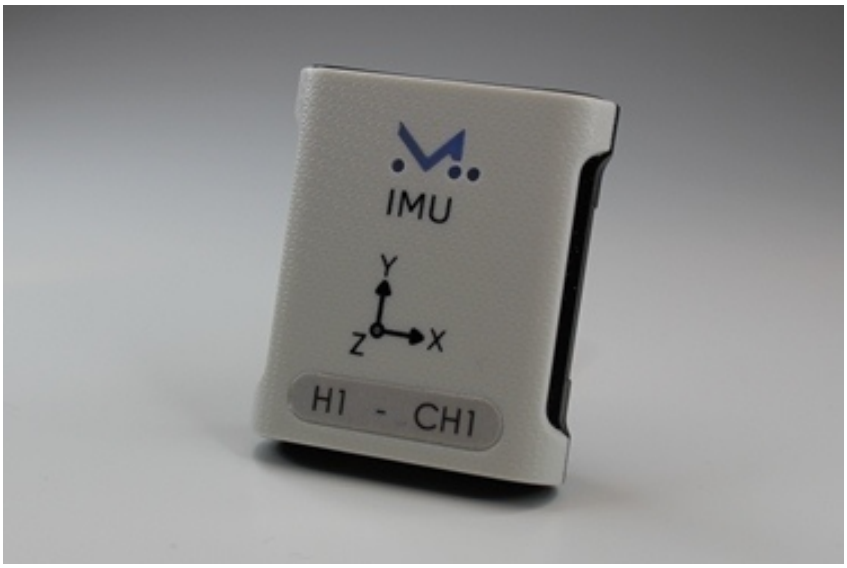


Figure 6. Inertial Measurement Unit.

A first option with which body part movements can by tracked is with the use of **inertial measurement units** *(IMU) (*Figure 6*)*. Inertial measurement units are electronic devices that are able to report a body's specific force, angular rate and the magnetic field surrounding the body [5]. They can achieve this by using a combination of accelerometers, gyroscopes and magnetometers. Accelerometers can detect linear acceleration, gyroscopes rotational rates and magnetometers the magnetic field. An example of hand inertial motion capture system is the "Synertial Mocap Gloves" [6] that use tiny IMU based sensors located on each finger segment. The number of these tiny IMU sensors can vary between 7,13 and 16 although for a more precise capture they have to be used at least 16. Because the inertial sensors are capturing movements in all 3 directions, flexion, extensions and abduction can be captured for all fingers and thumb.

Figure 7. Microsoft Kinect.

Microsoft Kinect (Figure 7**)** is also an alternative worth mentioning. Kinect is another motion capturing device most famously known as complementary hardware in Microsoft's X-box gaming consoles. There is a trio of hardware innovation powering the Kinect sensor. The device features an RGB camera, depth sensor and multi-array microphone which provide 3D motion capture, facial recognition, depth perception as well as voice recognition capabilities. A final and essential component is the device's software which can perceive and configure the space around the user. Then it detects and tracks 48 points on each user's body, mapping them to a digital reproduction of that player's body shape and skeletal structure including facial details.

Figure 8. Leap Motion.

Besides the first hardware components already presented above, a very common alternative for hand tracking is the **Leap Motion Controller (**Figure 8***). This device is the hand tracking device utilized by the haptic system***.** Its purpose was to capture the user's hand and finger movement's and re-enact them in Unity's scene. The leap motion was chosen due to its ease of use, ability to recognize different aspects of input, provision of a range of realistic low-poly hand models and provision of an API that helped in the development process

This controller is a type of touchless interaction device that can detect a user's hands, fingers, and finger-like objects (tools). From a hardware perspective, the device consists of two cameras and three infrared LEDs. These track infrared light with a wavelength of 850 nanometers, which is outside the visible light spectrum. The field of view has an effective range of 150° with a roughly 8 cubic feet of interactive 3D space (Figure 9). The Leap is designed so that it sits in front of the user's computer screen. Interaction is done by making gestures with the hands, 22 fingers, or finger-like objects such as a pen or pencil. Although the fingers and hands can be tracked accurately, they must be positioned on top of the Leap so that your inner palms should always be facing downwards towards the Leap. This is because the Leap will stop detecting the hands and fingers if the hands are tilted due to the fingers no longer being in its vision. According to the Leap Motion Developer website [7], the Leap can recognize three aspects of hand input.

- The first aspect is the ability to recognize hands, fingers, and fingerlike tools and provide software interfaces to get information on each of these input types.
- The second aspect is the recognition of gestures, such as circles, key taps, and screen taps.
- The third is the recognition of motions of the hands, fingers, and finger-like tools such as scaling, translation, and rotation.

The device sends the image data it receives from the cameras to the computer via a USB cable. The Leap Motion Service is the software that processes these image data that are streamed. After compensating for background objects (such as heads) and ambient

environmental lighting, the images are analyzed to reconstruct a 3D representation of what the device sees.

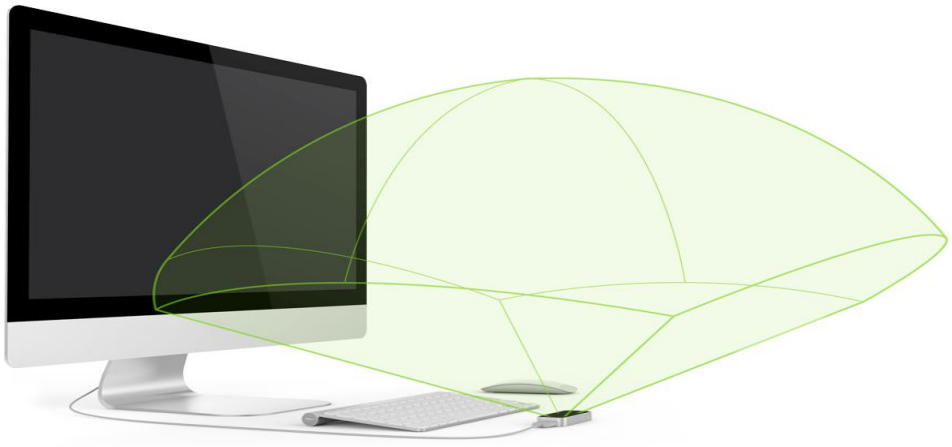


Figure 9. Leap Motion's field of view.

2.7 Related Work

There is a plethora of work and research on the field of haptics. It is worth presenting and reviewing recent work in order to have a better understanding in the state of art of haptic technologies.

Vibol Yem and Hiroyuki Kajimoto recently presented their work titled "Wearable Tactile Device using Mechanical and Electrical Stimulation for Fingertip Interaction with Virtual World" [8]. In this project they created a finger glove which was able to provide electrical and mechanical tactile feedback when interacting with a virtual object. The device was able to provide a four mode stimulation. With the use of a dc motor it was able to provide mechanical stimulation in the form of skin deformation and high-frequency vibration (Figure 10). While with the use of an array of electrodes the device provided electrical stimulation in the form of pressure and low-frequency vibration. Moreover, they designed specific algorithms in order to evaluate the quality of the touch simulation they offered.

Finally, they conducted an evaluation test. The experimental results showed that depending on the mode of stimulation different type of tactile sensation were affected including macro roughness, fine roughness, hardness and friction. An intention in developing a future algorithm to control the intensities of these four tactile dimensions was stated.

Figure 10. FinGar a finger glove able to provide electrical and mechanical feedback

Another work in this field was a study named "Designing a Vibrotactile Head-Mounted Display for Spatial Awareness in 3D Spaces" (V. Adriel et al., 2017 [9]). They built a vibrotactile headband with the help of LRA electric motors which was able to identify a virtual object's place in relation to the user in that given space (Figure 11). The stimuli were created by a varying frequency level up to 175Hz. The maximum frequency indicates that the subject is facing the target in its actual elevation. As the subject pitches his head far from the target, the frequency is reduced following a given modulating function. The modulating functions were selected going from a more continuous variation (Linear Growth), passing by a more discrete one (Stair Growth), until a steep frequency variation (Quadratic Growth)

An experiment between these three modalities was conducted to assess the effectiveness of the device. After testing three different functions for frequency modulation, a Quadratic function was shown to support a more accurate, precise and fast selection of targets. The more steep variation of the growth function of that modality allowed the subjects to better detect the position of the virtual objects. The Quadratic function also presented less mental demand than both Linear and Stair conditions, which tended to reduce the mean workload of the task. It was expected for the Linear condition to present less perceptually distinct levels, but also the Stair condition presented some difficulty to be performed. As the Stair condition had less variable levels, the frequency dwells more on each level while the subject moves the head. That slow variation affected the subject's performance and experience.

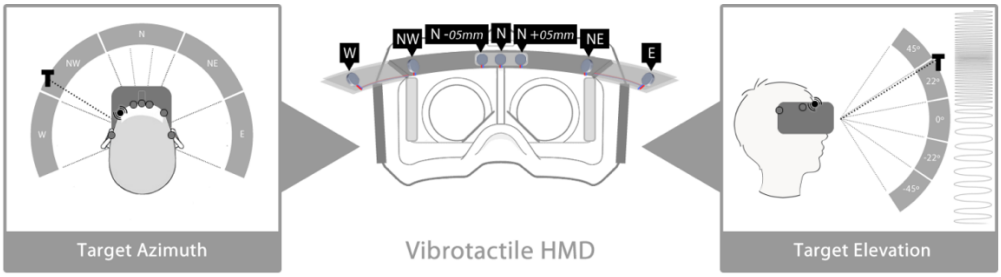


Figure 11. The vibrotactile band embedded in an HMD.

The final work presented is the investigation of the effect of vibrotactile feedback delivered to one's feet in an immersive virtual environment in the study titled "Exploring the Effect of Vibrotactile Feedback through the Floor on Social Presence in an Immersive Virtual Environment" (L. Myunhgo et al., 2017 [10]). In this study participants observed a virtual environment where a virtual human (VH) walked toward the participants and paced back and forth within their social space (*Figure 12*). They compared three conditions as follows: participants in the "Sound" condition heard the footsteps of the VH; participants in the "Vibration" condition experienced the vibration of the footsteps along with the sounds; while participants in the "Mute" condition were not exposed to sound nor vibrotactile feedback. They found that the participants in the "Vibration" condition felt a higher social presence with the VH compared to those who did not feel the vibration. The participants in the "Vibration" condition also exhibited greater avoidance behavior while facing the VH and when the VH invaded their personal space.

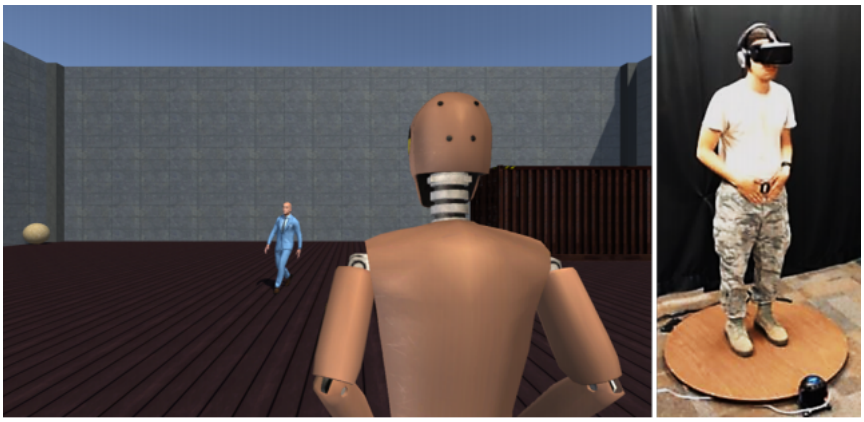


Figure 12. Participant observes the presence of a virtual human through different stimuli

The related work on this section combined concepts and principles of haptic technologies already presented. LRAs, DC motors, 3D printing and hand tracking with IMUs were all elements these studies consisted of. This thesis has also implemented a wearable haptic system which was brought to life through 3D printing, has embedded on it electric motors and provides tactile feedback with the use of vibrations.

Chapter Three

3D Graphics

3 3D Graphics

Since the system as well as the demo application are built the Unity game engine, certain basic theory about 3D graphics must be presented. 3D computer graphics (in contrast to 2D computer graphics) are graphics that utilize a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering 2D images. 3D computer graphics creation falls into three basic phases. The process of forming a computer model of an object's shape known as 3d modelling, the placement and movement of objects within a scene as well as the computer calculations that, based on light placement, surface types, and other qualities, generate the image. This image generating process is called the rendering process.

3.1 Three Dimensional Modeling

Figure 13. Teapot Model.

The model describes the process of forming the shape of an object. A 3D model is the mathematical representation of any three-dimensional object (Figure 13). This mathematical representation consists of arrays of data that represent a collection of points in 3D space, connected by various geometric entities such as triangles, lines and curved surfaces. The two most common sources of 3D models are those that an artist or engineer originates on the computer with some kind of 3D modeling tool, and models scanned into a computer from real-world objects. Models can also be produced procedurally or via physical simulation. Basically, a 3D model is formed from points called vertices or vertexes that define the shape and form polygons. A polygon is an area formed from at least three vertexes (a triangle). The overall integrity of the model and its suitability to use in animation depend on the structure of the polygons.

In addition, their surfaces may be further defined with texture mapping. Texture mapping is the process in which an image is applied or else mapped to the surface of a shape or polygon. It can define high frequency detail, surface texture or color information on the 3D model it will be applied.

There are two main categories in which 3d models can be divided. These are solid models and Shell or boundary models. **Solid models** define the volume of the object they represent. They are mostly used for engineering and medical simulations, and are characterized by their emphasis on physical fidelity. This physical fidelity gives them the ability to be consistent with the physical behavior of real objects. **Shell or boundary** models on the other hand represent the surface or boundary of the object and not its volume. Almost all visual models used in games and film are shell models since are simpler and easier to be processed and rendered.

Models can also be categorized in the way they are represented. The three most popular ways to represent a model include polygonal modeling, curve modeling and a more modern technique called digital sculpting.

In **polygonal modeling** 3D models are built as textured polygonal meshes. Polygonal meshes are formed from points in 3D space, called vertices that are connected by line segments. Polygonal modeling is the representation most 3d models today use due to the fact polygonal models are flexible and computers can render them very fast. However, polygons are planar and can only approximate curved surfaces using many polygons.

In **curve modelling** surfaces are defined by curves, which are influenced by weighted control points. The curve follows but does not necessarily interpolate the points. Increasing the weight for a point will pull the curve closer to that point.

Still a fairly new method of modeling, **3D sculpting** has become very popular in the few years it has been around. It uses software that offers tools to push, pull, smooth, grab, pinch or manipulate a digital object as if it were made of a real-life substance such as clay. A benefit of mesh-based programs is that they support sculpting at multiple resolutions on a single model. Areas of the model that are finely detailed can have very small polygons while other areas can have larger polygons. In many mesh-based programs, the mesh can be edited at different levels of detail, and the changes at one level will propagate to higher and lower levels of model detail. A limitation of mesh-based sculpting is the fixed topology of the mesh; the specific arrangement of the polygons can limit the ways in which detail can be added or manipulated.

Modeling can be performed by means of a dedicated program. There is a plethora of 3D modelling programs. Depending on the type of the 3D object, experience of the user and field of science the 3d model will be used at, different programs can be selected. Programs most commonly used for 3D modelling are Solidworks for creating solid models in engineering and manufacturing, Mudbox for 3d sculpting high quality meshes and SketchUp for creating drawing applications including architectural, interior design and landscape architecture.

Last but not least, 3D models can also be printed into real life objects through a process called ***3D* printing** *(*Figure 14*)*. 3D printing is a form of additive manufacturing technology where a three dimensional object is created by laying down or build from successive layers of material. This object can be almost any shape or geometry. Nowadays, 3D printing process is being used in manufacturing, medical, industry and sociocultural sectors which facilitate 3D printing to become successful commercial technology.

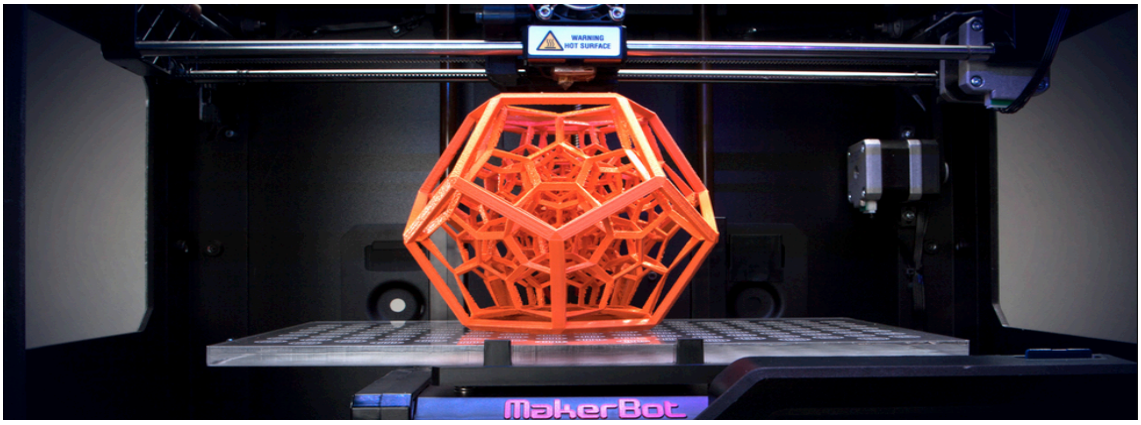


Figure 14. MakerBot 3D printing an object.

3.2 Layout and Animation

Before rendering the created 3D objects into an image, they have to be laid out in a scene. This defines spatial relationships between objects, including location and size. Laying out the 3D objects properly within a scene is necessary in order to stage every shot, plot the action and create proper animations. Animation refers to the temporal description of an object including its movement and deformation over time.

When animating a 3D object, the object to be animated will be taken into a software and processes like key framing and tweening will be applied to it. **Key framing** defines the start and ending points which fundamentally creates an outline of the most important movements. **Tweening** is the process that after creating all the important frames of action the computer fills all the "in- between frames" achieving this way a smoother animation. In humanoid 3d characters most of the different poses in each frame are created by rigging the character with a virtual skeleton and moving that skeleton appropriately (Figure 15).

Figure 15. Different frames of a running animation.

Creating a different pose for each frame can be a very time consuming task. To raise production times and the quality of animations a newer method called motion capture is being used by AAA animation studios. Motion capture makes use of live action footage. When a computer animation is driven by motion capture, a real performer acts out the scene as if they were the character to be animated. The performer's motion is recorded to a computer using video cameras and markers and that performance is then applied to the animated character. (Figure 16)

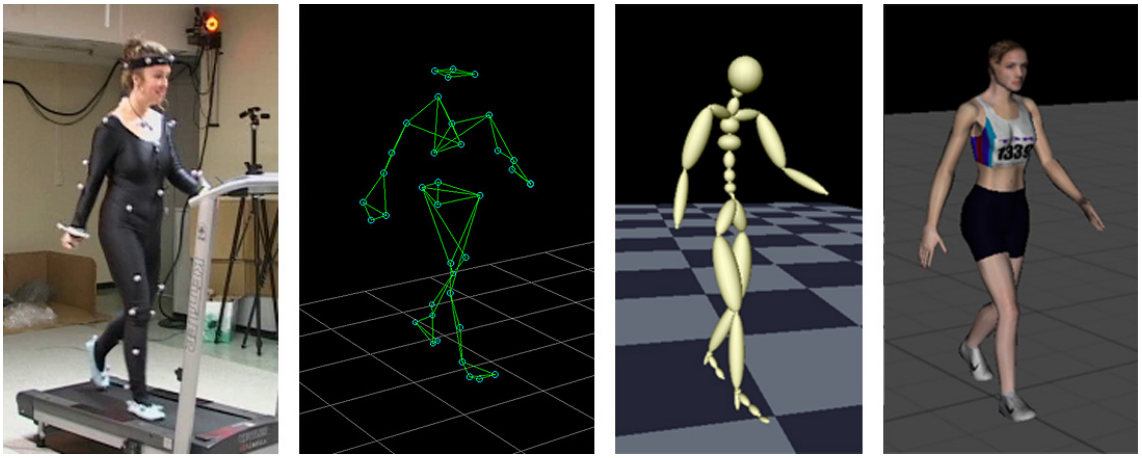


Figure 16. Full body motion capture for a walking animation.

3.3 3D Rendering

The final step after the 3D objects are created, animated and laid out in the scene is to create the actual 2D image through a process called rendering. The process of rendering aims to add the simulation of lighting, shadows, atmosphere, colour, texture and optical effects. These three dimensional qualities can either be realistic creating this way a photorealistic 2D image or non-realistic which results in a stylized final 2D image. The process of rendering usually requires a serious amount of processing power, memory and

time. Several different, and often specialized, rendering methods have been developed. In general, different methods are better suited for either non-real time rendering, or real-time rendering.

In real time rendering all the calculations are and displayed in real time, at rates of approximately 20 to 120 frames per second. The goal is to achieve an as high as possible degree of photorealism at an acceptable minimum rendering speed, usually 24 frames per second as that is the minimum the human eye needs to see to successfully create the illusion of movement. Because these calculations happen so fast this is the basic method employed in games, interactive worlds and virtual reality.

Animations for non-interactive media, such as feature films and video, are rendered much more slowly. Non-real time rendering enables the leveraging of limited processing power in order to obtain higher image quality. Rendering times for individual frames may vary from a few seconds to several days for complex scenes. Rendered frames are stored on a hard disk then can be transferred to other media such as motion picture film or optical disk. These frames are then displayed sequentially at high frame rates, typically 24, 25, or 30 frames per second, to achieve the illusion of movement.

3.4 Game Engines

A game engine is essentially a software development kit that contains the source code and tooling necessary to create video game and simulations, letting the developer script in logic, levels, and characters. The games created in a game engine can be for a variety of platforms including consoles, mobile devices and personal computers. The core functionality typically provided by a game engine includes a rendering engine for 2D or 3D graphics, a physics engine or collision detection, sound, scripting, animation, artificial intelligence, networking, streaming, memory management, threading, localization support, scene graph, and may include video support for cinematics. The process of game development is often economized, in large part, by reusing or adapting the same game engine to create different games, or to make it easier to "port" games to multiple platforms.

The above mentioned tools a game engine offers are generally provided in an integrated development environment to enable simplified, rapid development of games and applications in a data driven manner. This helps developers to focus on game ideas, level design and gameplay logic instead of virtual world physics, rendering process and other low level technologies. Moreover, game engines provide platform abstraction, allowing the same game to be run on various platforms including game consoles and personal computers with few, if any, changes made to the game source code.

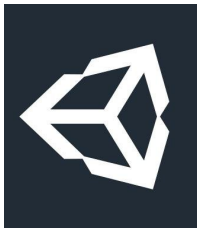
The next section presents a number of popular game engines used both in educational and commercial sectors.

The Unreal Engine is developed by Epic Games and was first release in 1998 [11]. Although primarily developed for first-person shooters, it has been successfully used in a variety of other genres, including stealth, MMORPGS and other RPGs. With its code written in C++, the Unreal Engine features a high degree of portability and is a tool used by many game developers today. Its blueprint system in the newer version makes scripting almost none-existence as it gives the developers the ability to create game logic by simply connecting lines and blocks of commands. Finally, it has its own version of an asset store providing its users with content both premium and free to add into their projects.

CryEngine is another widely used game engine developed by Crytek [12]. The first version of CryEngine was release on 2002 and as with most game engines it is still in development. It offers high end graphics featuring advanced shader and lightning systems. Its scripting languages include C++ and Lua and in its arsenal of tools include a large amount of advanced visual features, audio/physics systems and character and animation systems.

Frostbite engine developed by EA DICE it's a fairly new game engine as its first iteration

appeared on 2008 [13]. It was first used to create first-person shooter games but has since been expanded to include various other genres. Frostbite's newest iteration introduces new features such as new weathering systems, physically based rendering, real-time compositing and support for various development techniques. The game engine has had several upgrades including improved tessellation technology. It also features Destruction 4.0, which enhances the in-game destruction over its predecessors. The engine developers have not ruled out the feasibility of releasing mod tools for the Frostbite engine. Unfortunately, Frostbite engine is exclusively used by Electronic arts as its in-house engine.



Unity is a cross-platform game engine developed by Unity Technologies, which is primarily used to develop video games and simulations for computers, consoles and mobile devices [14]. Unity is marketed to be a multiple purpose engine, and as a result supports both 2D and 3D graphics, drag and drop functionality and scripting through its 3 custom languages C# and Javascript. It also provides cross platform publishing, millions of ready-made assets in the Asset Store and an online community. For individual developers and studios, Unity's environment reduces the

time and cost allowing its users to develop advance interactive games. It provides flexibility to deploy projects on multi-platforms like iOS, Windows and Android.

3.5 Unity Architecture

The initial step starting this project was to select the software which will host the 3D objects, render them, determine any forces between them as well as have the ability to detect any collision the user's virtual hand makes with those objects. A software suite able to offer these features including a variety of other is a game engine**. The game engine that was selected was Unity game engine**. Unity as previously stated is a game engine able to prove high-end graphics, a large variety of usable tools, great support for platforms and devices, without compromising usability and efficiency. Moreover, Unity's community support, ease of use and compatibility with most of the other components of the system made it an easy choice compared to other alternative game engines. It is important to present some core concepts and important components of the engine.

3.5.1 Scene

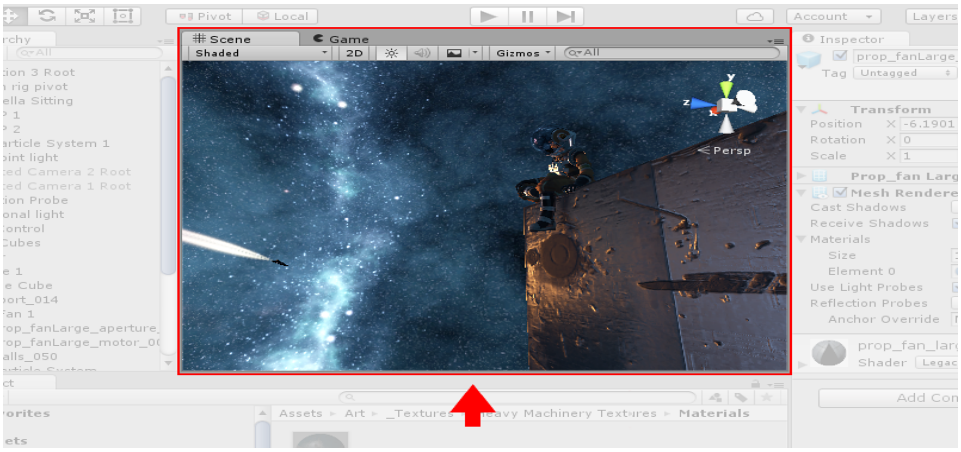


Figure 17. The scene view inside the unity game engine.

When creating a new project in Unity the main point of attention is the scene [Figure 17]. In the scene view the developer can have a preview of the virtual world having the ability to select, manipulate and modify 3D objects. An application or game can have many scenes, each one with each own game objects.

Inside each scene with the help of 3D Objects, levels can be created which may contain menus, sounds, cameras and even whole virtual worlds. The combination of Scene contents and parameters can be saved as a scene file and loaded when that scene file is opened.

3.5.2 Game Objects and Unity Components

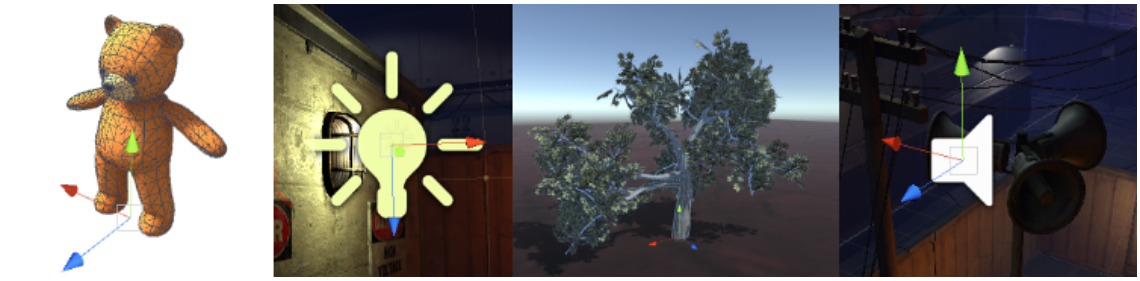


Figure 18. Different Game objects.

The "GameObject" is a vital element of the unity engine. Every object in a scene is a "GameObject" which is basically a container (Figure 18). This container can have in it a combination of different components including scripts, audio, physics and more. These components are the elements of Unity that give a "GameObject" meaning. Next the most widely used of them will be reviewed

Transform component

This component is created by default when a game object is first created and cannot be removed as the GameObject wouldn't have a location in the world. All game objects include a Transform component dictating where the GameObject is located, and how it is rotated and scaled. The location is defined by a x, y and z Cartesian coordinate system.

These parameters are initialized by hand and/or can modified in runtime by a script component in order to make objects move, rotate and scale inside the scene. It is important to note that Unity considers the Z axis as forward/backward in space, Y axis as up/down and finally X axis as left/right.

Mesh component.

3D Meshes are the main graphic object primitive of Unity. In order for a 3D object to be rendered and displayed unity uses a set of components called mesh renderer and mesh filter. These components work in collaboration in order for a mesh to be able to be rendered by Unity. The purpose of the mesh filter is to pass a mesh to the mesh renderer. The mesh renderer renders the geometry it receives from the mesh filter at the position defined by the geometry's transform component. Different parameters can be given to the mesh renderer in order to change how the light behaves, control the existence of shadows as well as a variety of other rendering options. If the mesh renderer is absent from a 3D

object's list of embedded components, that 3D object will be present on the scene but it will not be able to be drawn. Mesh filter and mesh renderer components are created automatically when a mesh asset is imported.

Material and Shader components.

Materials and shaders are crucial components that are categorized in the asset component group. There is a close relationship between materials and shaders. Materials are used in conjunction with mesh renderers and other rendering components used in Unity. They play an essential part in defining how the object is displayed. The properties that a materials inspector displays are determined by the shader that the material uses. A shader is a specialized kind of graphical program that determines how texture and lighting information are combined to generate the pixels of the rendered object onscreen. In other words, it tells the graphics hardware how to render surfaces. The user can select which shader each material will use. Specifically, a material defines which texture and color to use for rendering, whereas the shader defines the method to render an object.

Physics component.

Unity has NVIDIA PhysX physics engine built-in. This allows for unique emergent behavior and has many useful features. To put an object under physics control, a rigid body is added to it (Figure 19). A rigid body component will result in the object being affected by gravity, and having the ability to collide with other objects embedded also with the rigid body component. In addition, objects can be moved around directly by adding forces to it through a unity script component. A rigid body has also the option to be a kinematic rigid body. Kinematic rigid bodies are not affected by force, gravity or collisions. They are driven explicitly by setting the position and rotation of the Transform or animating them, yet they can interact with other non-Kinematic Rigid bodies. Unity's rigid body physics components was used in the project in order to detect collisions between objects and the user's hand.

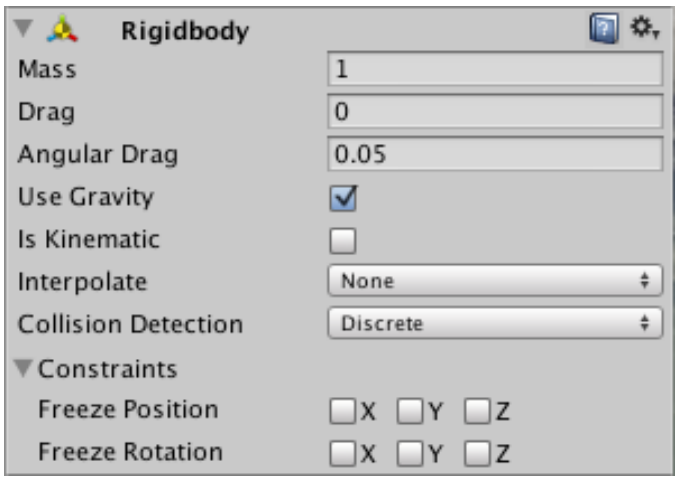


Figure 19. Rigid body component.

Scripting component

Scripting is what gives an application or game life defining its behavior and features. Scripts can trigger effects, create effects, define behavior of objects and a number of other things. Unity scripting system supports 2 languages C# and Javascript. Each scripting component can control the behavior of the object it is attached to.

Much of the power of Unity is in its rich scripting language, C #. It can be used to handle user input, manipulate objects in the scene, detect collisions, spawn new GameObjects and cast directional rays around the scene to help with game logic. These scripts can be written and edited in MonoDevelop, unity's integrated development environment (IDE) or in any similar IDE. An IDE is usually consisted of a text editor with additional features for debugging, auto-complete and other project management tasks.

A script makes its connection with the internal working of Unity by implementing a class which derives from the built in class called MonoBehavior. A class can be thought as a kind of blueprint for creating a new component type that can be attached to Gameobjects. Each time a script component is attached to a GameObject it creates a new instance of the object defined by the blueprint. The name of the class is taken from the name the developer supplied when the file was created. The class name and the file name must be the same to enable the script component to be attached to a GameObject.

The main things to note, however, are the two functions defined inside the class (Figure 20).

Update function is called on every frame. Update is the most commonly used function to implement any kind of game behavior. This function runs on a loop. Inside this loop usually a developer checks for changes on an event status, triggers new events and controls the behavior of a game mechanic.

Start function in the very first frame the script is enabled. Start is called exactly once in the lifetime of the script and it is usually used to initialize values or run processes enabling necessary mechanics before the update function is called.

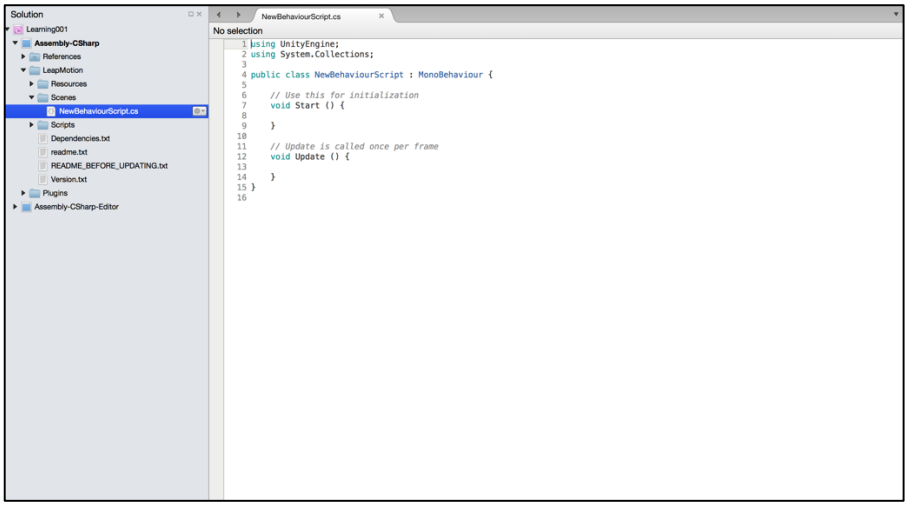


Figure 20. Unity's scripting interface

A script in Unity is not like the traditional idea of a program where the code runs continuously in a loop until it completes its task. Instead, Unity passes control to a script intermittently by calling certain functions that are declared within it. Once a function has finished executing, control is passed back to Unity. These functions are known as event functions since they are activated by Unity in response to events that occur during gameplay. Unity uses a naming scheme to identify which function to call for a particular event. Such function is the Update function (called before a frame updates occurs) and the start function (called just before the object's first frame update). Beside these event functions unity includes a number of others including other regular update events, initialization events, GUI events and physics events.

Besides these functions already provided by Unity the developer can create his own functions in order to control or determine the behavior of a GameObject, change the properties of a component or altering the overall state of the application. In order for these custom functions to be executed, they have to be called inside a Unity event function, like Update. The basic notion of the Unity scripting is that the scripts are components that can control the GameObject simply by being embedded in them and programmed. Finally, is important to state that the scripts besides accessing the objects in which they are embedded in they can also access other objects in the scene.

Finally, unity provides a console window for code debug including errors, warnings and other messages (Figure 21). The toolbar of the console window has a number of options that affect how messages are displayed.

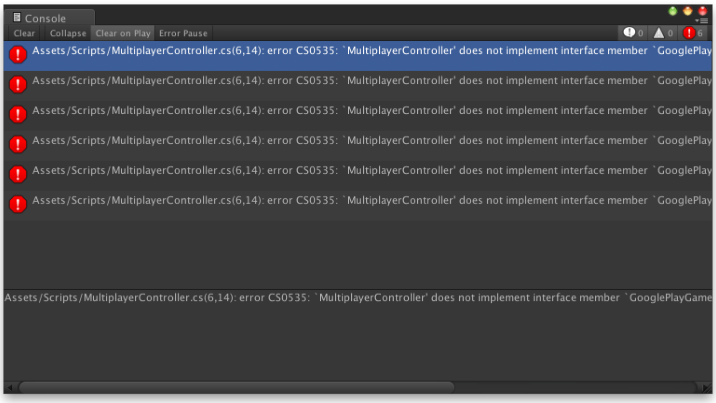


Figure 21. Console Window.

The Clear button removes any messages generated from your code but retains compiler errors. The console can be arranged to be cleared automatically whenever the game is run, by enabling the Clear On Play option.

The way in which messages are shown and updated in the console can be changed. The Collapse option shows only the first instance of an error message that keeps recurring. This is very useful for runtime errors, such as null references, that are sometimes generated identically on each frame update. The Error Pause option will cause playback to be paused whenever Debug.LogError is called from a script (but note that Debug.Log will not pause in this way). This can be handy when you want to freeze playback at a specific point in execution and inspect the scene.

Finally, there are two options for viewing additional information about errors. The Open Player Log and Open Editor Log items on the console tab menu access Unity's log files which record details that may not be shown in the console.

Audio component.

Figure 22. Audio sources and listener.

Sound is an integral part of every application or game. A game would be incomplete without some kind of audio, be it background music or sound effects. Unity's audio system is flexible and powerful. It can import most standard audio file formats and has sophisticated features for playing sounds in 3D space, optionally with effects like echo and filtering applied. Unity can also record audio from any available microphone on a user's machine for use during gameplay or for storage and transmission.

In real life, sounds are emitted by objects and heard by listeners (Figure 22). The way a sound is perceived depends on a number of factors. A listener can tell roughly which direction a sound is coming from and may also get some sense of its distance from its loudness and quality. A fast-moving sound source (like a falling bomb or a passing police car) will change in pitch as it moves as a result of the Doppler Effect. Also, the surroundings will affect the way sound is reflected, so a voice inside a cave will have an echo but the same voice in the open air will not.

To simulate the effects of position, Unity requires sounds to originate from Audio Sources attached to objects. The sounds emitted are then picked up by an Audio Listener attached to another object, most often the main camera. Unity can then simulate the effects of a source's distance and position from the listener object and play them to the user accordingly. The relative speed of the source and listener objects can also be used to simulate the Doppler Effect for added realism.

Chapter Four

Glove Design and Implementation

4 Glove Design and Implementation

In this Chapter the haptic system's implementation phases are presented. The implementation process consisted of four phases. Initially the hardware devices responsible for supplying the user with haptic sensations were chosen. Then the design and creation process of the system's glove is explained. In the third phase, the circuit was created and the microcontroller was programmed to drive the electric motors. Finally, the last section of this chapter focuses on how Unity is able to detect collisions and notify the hardware about changes on collision status.

4.1 Type of Haptic Feedback

There are several approaches to creating haptic systems. Although they may look drastically different, they all have two important things in common. Software to determine the forces that result when a user's virtual identity interacts with an object and a device through which those forces can be applied to the user. As already mentioned in Chapter 3 the software used to determine these forces is unity game engine. This chapter focuses on depth in the way the system utilizes this engine to detect collisions and notify the hardware about each collision. Moreover, the hardware programming and circuit needed to drive the haptic feedback devices are also a major focus point. In addition, due to the fact the system will be in form of a glove, the design and creation phases of the glove are also included in this chapter. However, before diving into the design and implementation of the haptic glove it is essential to define its traits and characteristics based on the technological background presented on chapter two.

A vital element to take into account when building a haptic system is the type of haptic feedback the system will provide. Since kinesthetic feedback is out of the scope of this thesis, two different types of tactile feedback provision were examined. Deforming the skin in the form of pressure and simulating the sensation of touch with the help of vibrations were two types of tactile feedback this thesis took into consideration (Figure 23). The pressure of the skin was examined with the help of linear solenoids. However, due to the fact that those small solenoids were heated easily when they were enabled for long periods of time, the sense of pressure they provided was relative weak and their shape reduced how compact the system was resulted in rejecting this implementation in the first stages of the development process. Thus, **vibrotactile stimulation of the skin was chosen as the feedback of this haptic system**. As will be presented in the next section of this chapter this was achieved with the help of coin shaped electric motors. Their type, quantity and placement are key factors that will be addressed.

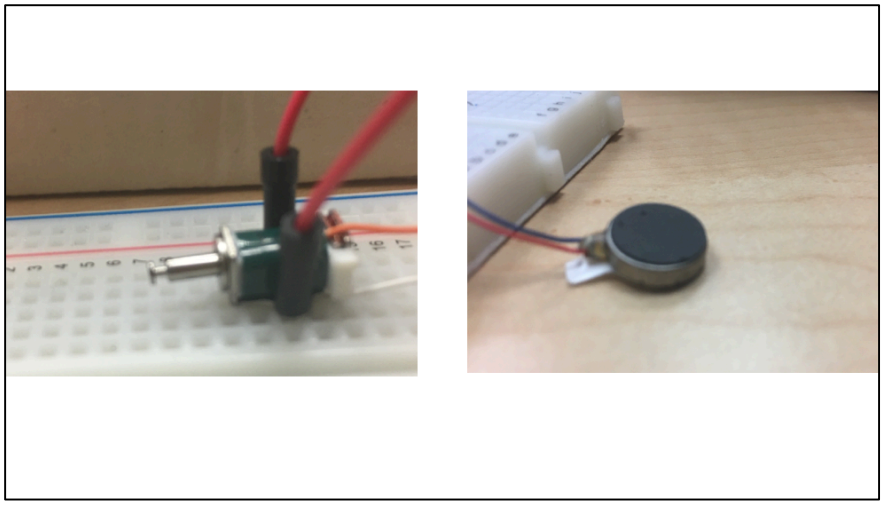


Figure 23. Components used for testing skin deformation and vibrotactile stimulation

4.2 Vibrotactile Feedback with Electric Motors

Figure 24. Places of motor position.

Since this haptic glove system aims to provide feedback in the form of vibrations a selection had to be made in terms of the appropriate hardware for this task. Two different devices and approaches able to provide vibrations have already been introduced. Each with its own sets of characteristics, advantages and disadvantages. Both eccentric rotating mass motors and linear resonant actuators are capable of providing vibrations with different frequencies and intensities. However before determining the one to be used it is important to take into account the number of these hardware devices as well as their placement on the user's hand. In order to provide the user a more realistic feedback with many different points of tactile simulation ten different vibration devices were used, two on each finger, one on the tip of the finger and another on the bottom of the finger (Figure 24). With this placement and number of devices in mind a selection about which device has the best traits can be made.

Using a resonant linear actuator seems ideal for this task. RLA consumes less power than an ERM motor which makes it a more power friendly choice. The power consumption of both hardware devices were put to the test by Texas Instruments [15]. In their study they examined the power consumed by these devices on a 1200 mAh smartphone battery. After running a typical use case scenario of phone calls, messages and browsing the power consumption results showed that an LRA consumed half the power of an ERM motor and was inherently more efficient (Figure 25). It must be noticed that the LRA in contrary to the ERM motor must be operated within a narrow frequency range in order to optimize its power consumption. This ideal frequency of the spring is known as the resonant frequency.

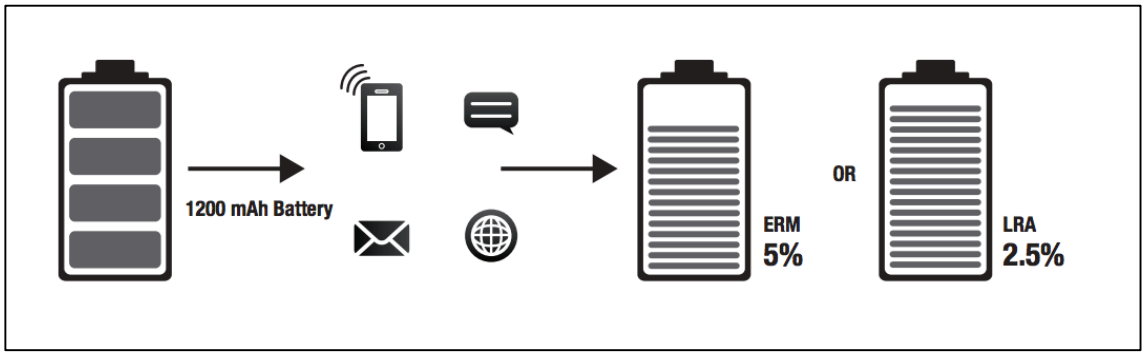


Figure 25. Power consumption of an ERM motor and LRA in a mobile device.

In addition, LRA's amplitude and frequency are independent of each other. The ability to separate frequency from amplitude allows the input to have a more complex waveform than with an ERM. Thus, a richer pattern of tactile sensations can be created. In applications such as haptic devices high response times are of great importance. The response time of an LRA is superior of an ERM motor as the typical start time for an LRA is approximately 5-10ms, a fraction of the time required to produce a vibration with an ERM motor. This incredible speed results from the immediate movement of the magnetic mass as current is applied to the voice coil inside of the device. However, despite this high start time the stop time of an LRA can be significantly longer than an ERM. An LRA can take up to 300ms to stop vibrating due to the continued storage of kinetic energy in the internal spring during operation. Thankfully, an active braking mechanism can also be used for an LRA by performing a 180-degree phase shift of the AC signal provided to the actuator, the vibration can be stopped very quickly within approximately 10ms by producing a force opposite to the oscillation of the spring. Another positive feature worth mentioning is their ability to have longer life span as in comparison to ERM motors they don't have internal brushes susceptible to wear.

However, despite these positive features there are also other factors that must be taken into consideration. The total budget and complexity of the circuit driving these vibrating devices are important factors when faced with the decision of selecting the hardware and components. As already stated LRA works with an Alternative Current(AC) instead of Direct Current (DC). This can result in more complicated circuits than those for an ERM

motor that works with DC current. Moreover, the process of detecting the proper resonant frequency and performing active braking increase this complexity. In addition, the small budget available for the creation of this system creates important limitations. LRAs are significantly more expensive and less available than ERM motors. Depending on the model a single LRA can be 5 times more expensive than an ERM motor. In this haptic glove system there had to be at least ten vibrating devices which made the choice of selecting LRA's out of the budget's limitation. Thus the **vibrating devices chosen were ten ERM motors. (**Figure 26)

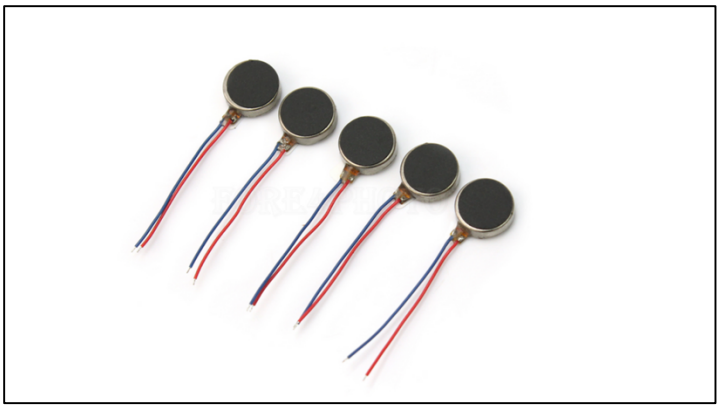


Figure 26. The vibrating devices chosen. Ten ERM coin motors.

4.3 Glove creation

Given the fact the haptic feedback system is in the form of a wearable glove a glove design was also implemented. Before starting the design process, the glove's material, aesthetics and overall look were elements taken into consideration. The aim was to build a stable, solid glove with a futuristic look. The implemented design was a heavily modified version of Bryan Cera's Glove One [16]. The modifications as well as new parts creation were achieved with the help of a solid modelling program.

4.3.1 Designing the 3D solid model

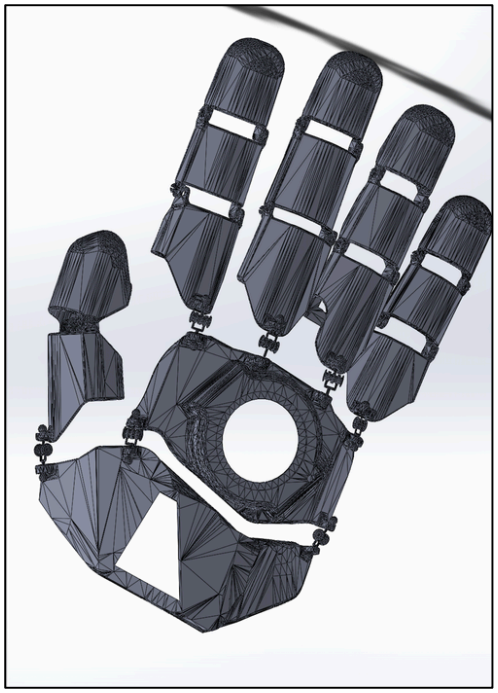


Figure 27. The CAD model of the glove.

The solid modelling software suite selected was Solidworks. As mentioned before in Chapter 3, Solidworks is a solid modelling software widely used in the manufacturing and engineering fields. It offers powerful features to create models and assemblies as well as editing capabilities. Initially the base glove was loaded on the program as a solid body. This was done on the options section of the panel appearing when importing the model. Because the glove consisted of 16 different parts each part was separated in order for the editing process to be faster and more organized.

The editing process included changing dimensions, creating new geometry on specific parts as well as splitting, cutting, moving and filling existing geometry. The main editing processes were the following:

- **Creating new geometries**. The first step was to create a 2d sketch (Sketch->shape). Each sketch consisted of geometry such as points, lines, arcs and more. After the sketch was created on a selected plane, dimensions were added to define the size and location of the geometry (fix -> fully defined). Finally, it was extruded in order to have volume and depth. This was how the motor base on each fingertip was created (Figure 29).

- **Editing pre-existing geometries**. Editing specific parts of a pre-existing geometry was the majority of operations done on this editing process. This was achieved by selecting the surfaces of the object that had to be modified. After that these entities were converted to a sketch (select-> sketch -> convert entities). Finally, operations like extrude (Figure 28), split and cut were applied on these surfaces resulting in an altered mesh.

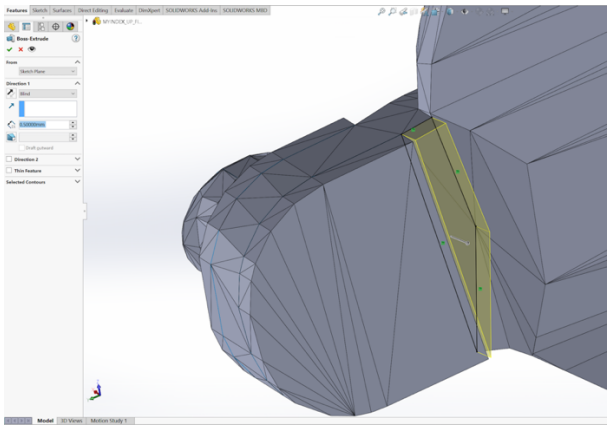


Figure 28. Extruding an existing area.

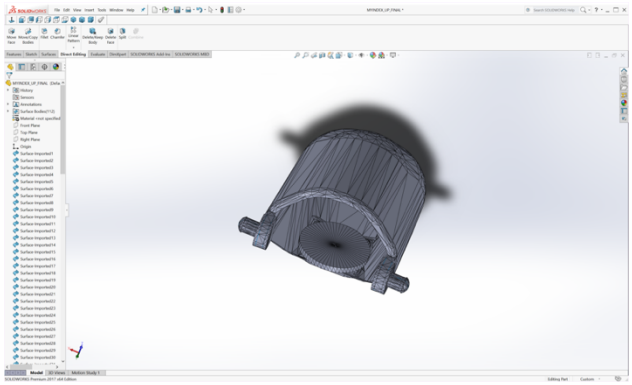


Figure 29. Creating the motor base for the tip finger.

4.3.2 Printing the model

When the CAD model of the glove (Figure 27) was ready it was brought to life through the process of 3D printing. The 3D printer used was “Anet A8” by Anet (Figure 30) and the material which the printer used for building the glove was PLA. There are two main materials often used in the 3D printing process, ABS and PLA [17]. Both are thermoplastics which become malleable when heated. Although the printing process of both materials is similar there are some key differences worth mentioning.

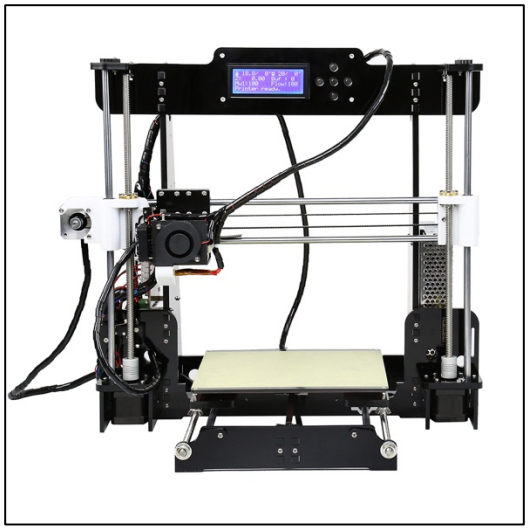


Figure 30. Anet A8 3D printer.

PLA (PolyLactic Acid) is made from renewable raw materials such as corn starch or sugarcane. Aside from 3D printing, it is typically used for packaging material, plastic wrap, plastic cups and plastic water bottles. It is considered to be more ecologically friendly than ABS. PLA is more brittle and has a higher surface hardness. It is more prone to break when bent. Treating them with acetone for improving surface smoothness is not possible. Overall, PLA is better suited for 3D printing beginners and is widely used in 3D printing for household items, gadgets, and toys. It is also better suited when flexibility is not the major requirement as it is more prone than ABS to break under pressure. Finally, it is biocompatible with the human body and can be used for objects that are worn on the skin.

ABS (Acrylonitrile-Butadiene Styrene) is an oil-based plastic. It is a tough material that can be used to create robust plastic objects for everyday use, for example in cars, electrical equipment and more. It is better suited for mechanical parts and for objects that need to be weatherproof. Moreover, ABS parts are more flexible than PLA parts and tend to bend rather than break when under pressure. They can also be treated with acetone to get a smooth and shiny surface or to weld two objects together. ABS is not biodegradable but can be easily recycled. ABS is better suited for objects that need to withstand rough usage, hot environments, that need to be weather-proof, that may be dropped or have to be bendable. It can be used for parts that are subject to mechanical stress, for interlocking parts or pin-joints.

4.4 Driving the motors

The ERM motors have to be powered when a virtual hand collision occurs and turned off when that collision is no more. This task can be easily achieved with the help of a microcontroller. A microcontroller is a self-contained system with peripherals, memory and a processor on a single integrated circuit. Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, toys and other embedded systems. The role of the microcontroller in the current system is to get the information about which parts of the user's virtual finger collide and enable the vibrating motors that are attached on the corresponding parts of the physical hand. It must also turn the electrical current off when that virtual collision has ceased to exist. The hardware selected for this task was the **Arduino UNO REV3** board (Figure 31).

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board its microcontroller and a piece of software, or Integrated Development Environment (IDE) that runs on the developer's computer, used to write and upload computer code to the physical board. Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware called a programmer in order to load new code onto the board. The board can be loaded with a new code via a USB cable. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to program it. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package.

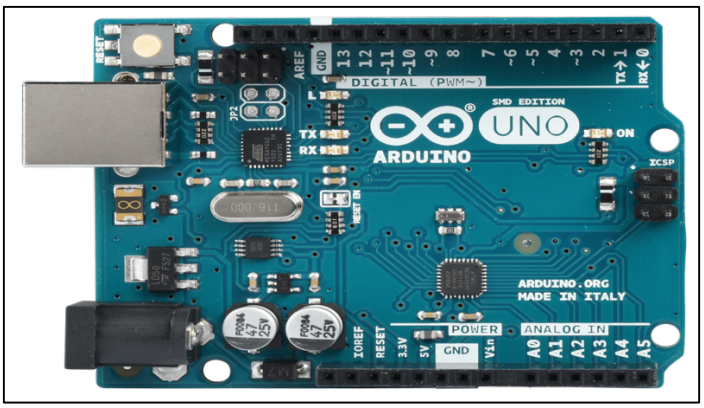


Figure 31. The microcontroller of the system. The UNO REV

Although Arduino is capable of powering on and off the motors, it has to be aware when changes in collision status are made and on which motor. As already mentioned before the collisions are detected by the software suite (Unity). Unity whenever detects a change in collision status it notifies Arduino. The communication of these two technologies happens though a set of different characters and numbers. Different characters correspond

in different collision status of the motors. The characters received from the serial port are handled by the microcontroller with the help of a script.

A minimal Arduino C/C++ sketch, consist of two functions

- **setup()**: This function is called once when a sketch starts after power-up or reset. It is used to initialize variables, input and output pin modes, and other libraries needed in the sketch

- **loop()**: After setup() has been called, function loop() is executed repeatedly in the main program. It controls the board until the board is powered off or is reset

The Arduino board interacts with hardware components through its PINS. The UNO REV 3 model used in this thesis had 13 different pins. Before programming the microcontroller to take an action on a selected pin, that pin must first be configured with the *pinMode()* function. This function configures a pin as an input or an output. It receives two parameters as an input, the number of the pin to be configured and the constant INPUT or OUTPUT depending on the way that pin will be used. When configured as an input, a pin can detect the state of a sensor like a pushbutton. As an output, it can drive an actuator like a dc motor. Typically, a pin will be configured in the setup function before the main loop of the program.

With the help of the digitalwrite() function a voltage value can be applied to a pin. The input parameters are the pin's number and the keyword HIGH or LOW. Writing HIGH supplies the actuator with 5 V while writing LOW stops the supply of voltage.

There is also a possibility to write a pseudo-analog voltage value with the function analogWrite(). This function uses pulse width modulation in order to produce voltages with values between the HIGH and the LOW.

4.4.1 The Circuit

Unfortunately driving actuators like electric motors presents a limitation due to their need for a higher current than the microcontroller can supply. The maximum dc current the Arduino board can supply is 40 mA while the working current range of the electric motors is from 70mA to 0,5 A. Moreover, the microcontroller lacks the ability to provide voltages higher than 5 Volt. These limitations can be surpassed with the use of a transistor. A transistor is a semi-conductive device used to amplify or switch electronic signals and electrical power. There are two types of transistors, which have slight differences in how they are used in a circuit. These transistors are known as the Bipolar junction transistor (BJT) (Figure 32) and field effect transistor (FET) (Figure 33).

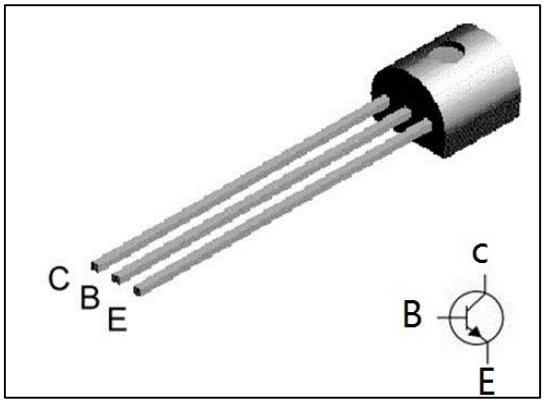


Figure 32. Bipolar junction transistor

A bipolar junction transistor has terminals labelled base, collector and emitter. The current applied to the base terminal can control or switch a larger current between the terminals collector and emitter. Bipolar junction transistors are current controlled requiring a biasing current to the base terminal in order an amplified current to start flowing between the other two terminals. This amplification is by a factor of β or H_{fe}. This is also known as the current gain or current amplification of the transistor. Since the working current of the electric motors is at least 70 mA, the supplied current to the gate terminal must result to an amplified current that is enough to drive the motors. Depending on how the current flows in them bipolar junction transistors can be classified as npn or pnp.

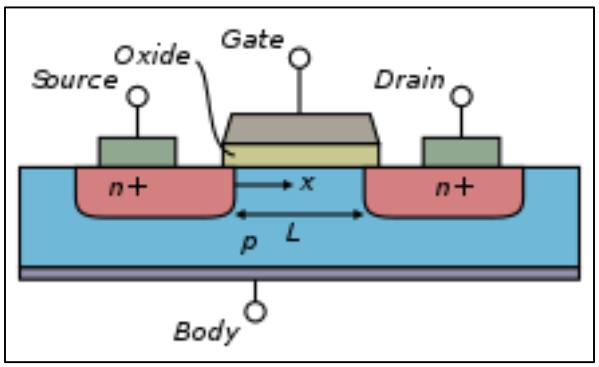


Figure 33. FET transistor

Field-effect transistor also known as FET have also three terminals labeled gate, source, drain. A voltage at the gate can control a current between source and drain. Field effect transistors can be made much smaller than an equivalent BJT transistor and along with their low power consumption and power dissipation makes them ideal for use in integrated circuits.

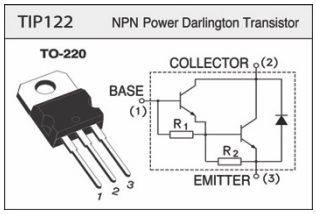


Figure 34.The TIP 122 Transistor

The type of the transistors used in the implemented circuit was a TIP120 (Figure 34). This transistor is a NPN Darlington pair transistor. A Darlington transistor uses a pair of bipolar junction transistors in order to achieve a higher current amplification. The current amplification of the TIP120 is on a factor of 1000. So the 40ma applied to the base from the microcontroller's pin will result in $40\times10^{-3}\times10^{3}=40$ A flowing through the electric motors. In order to reduce this amount of current a resistor was placed between the PIN and base of the transistor. To select the resistors Ohm's law was used $I=\frac{V}{R}$. The voltage drop between base-emitter in a Darlington transistor is 1.5V (0.7 per BJT) so the voltage hitting the resistor is 5-1.5 =3.5 for a 5V supply. According to Ohm's law by increasing the resistance value the current applied to the base is decreased.

The haptic glove system is built to be able to supply different intensity levels of voltage. As stated above the microcontroller is able to provide a pseudo-analog voltage value by using a method called Pulse Width Modulation (PWM). Unfortunately, not all the PINS of the board have this feature. There are 10 different vibrating motors and only 6 Pins able to provide PWM. Therefore, only the upper motors of the fingers were given the ability to change their voltage intensity levels while the lower motors work with a steady 3,3 Volt.

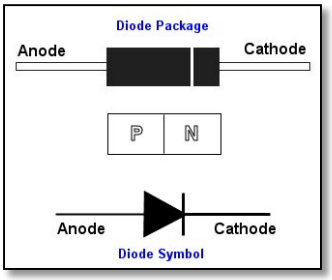


Figure 35.Diode

Electric Motors work through a process called induction. When electric charge flows through a wire, a magnetic field is created. Coiling the wire or increasing the amount of current makes the electric field stronger. In a DC motor such as the ERM coin motors, a coiled wire surrounds the motor's shaft. The generated magnetic field is pulled and repulsed by magnets inside the motor's body. When a motor stops, there is the potential for a small amount of current to be generated as the shaft continues spinning. A diode placed in a parallel with the motor leads will keep any generated electricity from kicking

back and damage the Arduino. A diode is a two-terminal electronic component that allows electric current to pass from one direction, while blocking it in the opposite (Figure 35). Thus current can flow to the motors but cannot return and damage the Arduino when the motors are turned off fast.

In conclusion, every coin motor is powered by a simple circuit (controller by the current of the Arduino) consisting of a transistor, a diode and a resistance at the base of the transistor. The way in which these components are connected is presented on (Figure 36). Since there were 10 different coin motors, each motor had its own circuit.

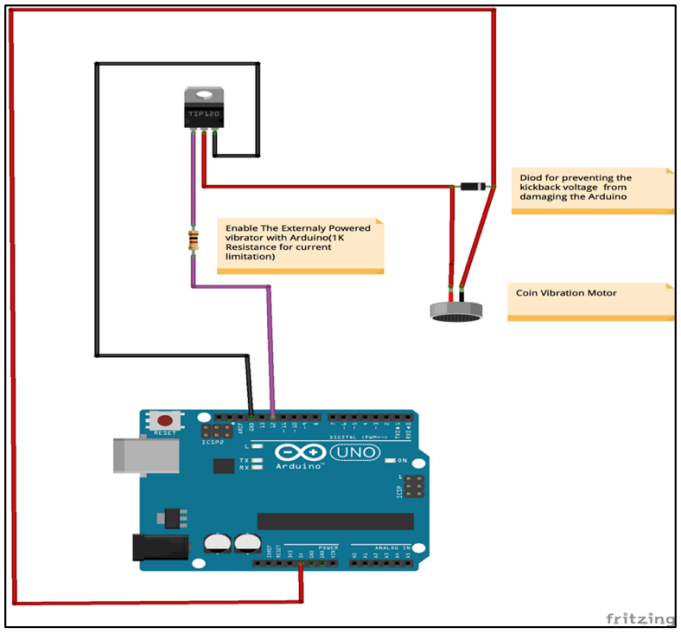


Figure 36.Circuit for an ERM Coin vibrator

4.4.2 Programming the Arduino

After surpassing the limitations presented with the help of the implemented circuit, the Arduino was programmed to enable or disable coin vibrators depending on the characters it received from unity. Since the 10 coin motors need to be powered by the microcontroller the PINS responsible for this task must be configures as outputs and initialized at a low value.

```
//thump
 pinMode(TU, OUTPUT);
 pinMode(TD, OUTPUT);
 //index
 pinMode(IU, OUTPUT);
 pinMode(ID, OUTPUT);

//middle
pinMode(MU, OUTPUT);
pinMode(MD, OUTPUT);

//ring
pinMode(RU, OUTPUT);
pinMode(RD, OUTPUT);

//pinky
pinMode(PU, OUTPUT);
pinMode(PD, OUTPUT);

//Initialize analogs
 analogWrite(TU, 0); //0 TO 255
 analogWrite(IU, 0); //0 TO 255
 analogWrite(MU, 0); //0 TO 255
 analogWrite(RU, 0); //0 TO 255
 analogWrite(PU, 0); //0 TO 255
//initialize digitals
 digitalWrite(TD, LOW);
 digitalWrite(ID, LOW);
 digitalWrite(MD, LOW);
 digitalWrite(RD, LOW);
 digitalWrite(PD, LOW);
```

Figure 37.Code Sample - Initialize Arduino's Pin.

Opening a serial port to read the data from unity is achieved with the help of the Serial commands. The method Serial.begin(9600) opens the serial port and sets the data rate at 9600 bits per second. The data is read with the help of Serial.ReadBytesUntil(termination character, buffer, length). This function reads characters from the serial port and saves them into an array. The function terminates if the termination character is detected, the determined length has been read or it times out. The buffer is declared as an array called myCol. Since in every loop only one character was read and checked the third parameter was put as one. In addition, the character to terminate the searching process was put as the ASCII value (10) of the termination character although that character is not written in the stream of data. Finally, the incoming characters were checked with the help of IF statements whether they matched any of the motor enabling/disabling characters. The characters were compared with the help of a C/C++ function called strcmp. This function compares two characters and returns zero if the two characters are the same. There are three different voltage values for the upper motors.

```
int lf = 10;
Serial.readBytesUntil(lf, myCol, 1);


//for the thump UP
  if(strcmp(myCol,"a")==0){
   analogWrite(TU, 100);
 }

 else if(strcmp(myCol,"l")==0){
  analogWrite(TU, 0);
  }
 else if(strcmp(myCol,"g")==0){
  analogWrite(TU, 255);
  }
```

Figure 38.Code Sample – Three Voltage Values of the upper thump.

With the help of digitalwrite function a HIGH OR LOW voltage value was written on the PINS. The PINS connected to the lower part motors who were not able to provide PWM were powered or switch off with the help of this command. However, since the system aims to showcase a haptic glove with different intensities, there are characters that result to an output of different voltage intensities. To write different voltages other than the HIGH voltage the function analogwrite was used. The upper motors powered by PINS able to provide PWM used this command. This function takes as an input the number of the PIN to power and the value of the PWM's duty cycle 0(OFF) – 255(Always on, HIGH voltage). In order to understand these values, the process of pulse width modulation must be explained.

Pulse Width Modulation, or PWM, is a technique for getting Analog results with digital means [18]. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying Analog values, the pulse with must be changed or modulated. If this pattern is repeated fast enough the result is a signal similar to a signal with a steady voltage of an in-between value of LOW and HIGH.

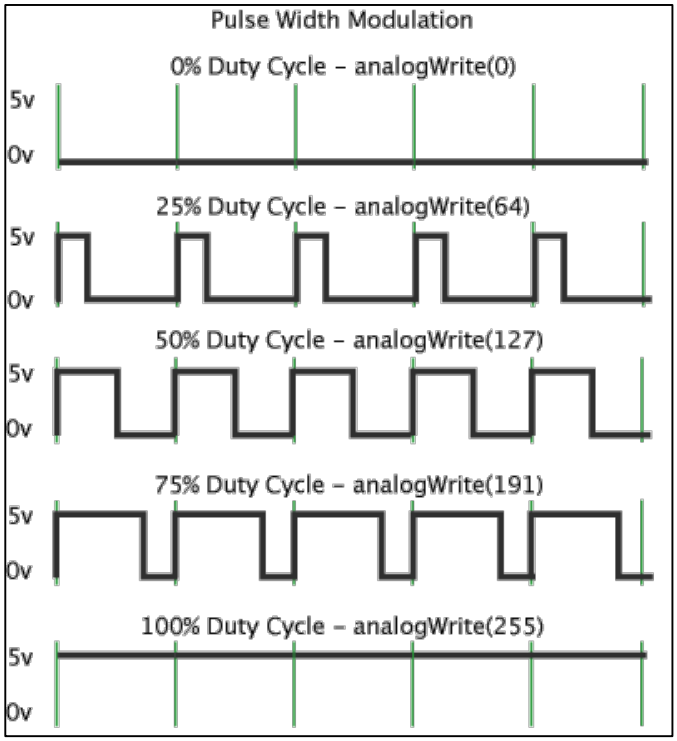


Figure 39.PWM – Different duty cycles.

4.5 Collision Detection

The Leap Motion was the device responsible for the detection of hand and finger movements as well as their 3D representation on Unity's scene. The leap motion controller offers a hand module compatible with unity which contains a small collection of example Leap Hand prefabs, including new low polygon rigged mesh hands, improved scripts for driving rigged meshes, libraries and scenes that demonstrate the leap motion's functionality. Starting the project, the first step was to import the hand module and place it on unity's scene as a game object (Figure 40).

Both the LeapHandController and Handmodels objects are set as a child of the main camera. This way the virtual hands appear constantly in front of the user's field of view even if the camera moves.

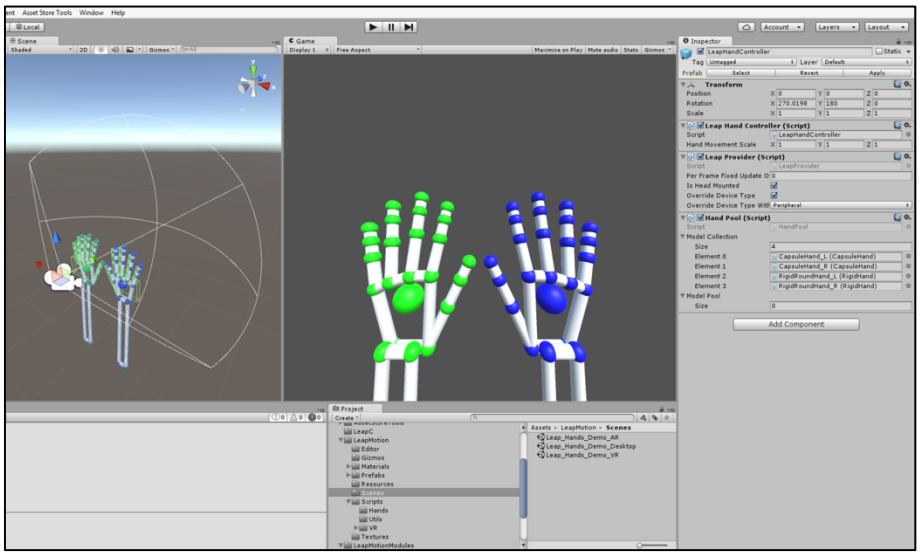


Figure 40. Leap Motion's Hand models placed on the scene.

4.5.1 Hand model Hierarchy

The structure of the handModel object resembles a hierarchy with different levels representing parts of the physical hand (Figure 41). On the top level of the hierarchy and the father entity of all the objects in the hierarchy is the handModel object which consists of a set of hand objects that offer graphic representation (Capsule Hands) and physics simulation (Rigid Hands). On a lower level of the hierarchy each rigid hand is comprised of seven different game objects, the five fingers, the palm as well as the forearm. On the last level of this hierarchy exist the bone game objects. Each finger is made of three different bones corresponding on the upper, middle and lower part of the finger (Figure 42).

Figure 41. Leap's hand hierarchy.

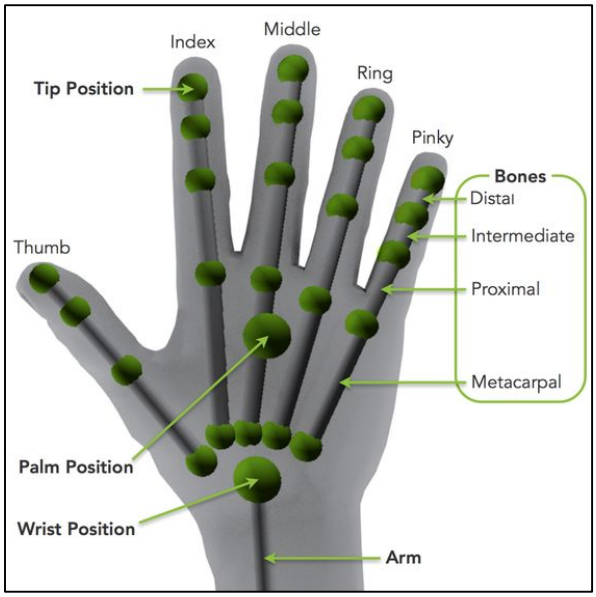


Figure 42. Leap motion's hand bones.

Due to the fact each bone in the lower levels of the hierarchy is a unity game object, components can be embedded on them. There were three unity components inserted in these bone game objects, a collider component, a physics component and a script component. The physics component inserted was a rigid body. Adding a Rigid body component to an object will put its motion under the control of Unity's physics engine. A Rigid body object will be pulled downward by gravity and will react to collisions with incoming objects if the right Collider component is also present. The rigid body component offers functions that are able to know if the collider/rigid body is being touched or no by another rigid body/collider. OncollisionEnter is called when two rigid bodies collide whereas OnCollisionExit is called when that collision stops.

4.5.2 Scripting in Unity

The scripting process started by creating a main communication script and inserting it in the top hierarchy of the handModel. The scripting was implemented with C# one of unity's two programming languages. The purpose of this script was to open a connection with the Arduino which controls the motors and notify it about the collisions. The communication happens via the serial port. Because the script uses the class *SerialPort* the Api compatibility level had to be changed to .NET 2.0 so the necessary libraries are included.

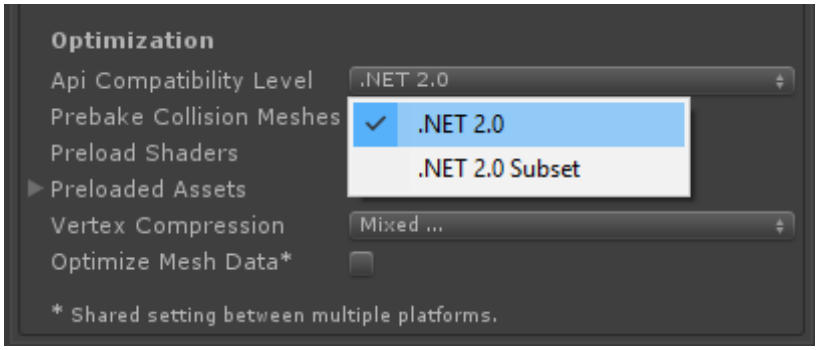


Figure 43. Selecting the .NET 2.0 Api

As already mentioned the communication of these two technologies happens though a set of different characters and numbers. Before opening the communication, a serial port object was created. The constructor of the object takes as input the name of the port and the data rate, which in this project was at 9600 bits per second. The Serial port object offered a set of methods including opening or closing the port, checking its status, writing, reading and setting a timeout.

```
10      public static SerialPort sp = new SerialPort("COM3", 9600);
11
12      // Use this for initialization
13      void Start () {
14          OpenConnection();
15      }
16
17      // Update is called once per frame
18      void Update () {
19
20      }
21
22      public void OpenConnection()
23      {
24
25          if (sp != null)
26          {
27
28              if (sp.IsOpen)
29              {
30                  sp.Close();
31                  print("Closing port,bacause it was already open!");
32              }
33              else
34              {
35                  sp.Open(); //Opens the connection
36                  sp.ReadTimeout = 16;
37                  print("Port Opened!");
38              }
```

Figure 44. Code Sample - Opening the serial port

After the serial port was opened and the connection established, the next step was to communicate through that port by writing in the stream a set of different characters. Each of these characters has a unique meaning for the microcontroller. Their purpose is to specify which motors should be turn on, turn off and at what intensity. On the next section of this chapter a more detailed explanation will be presented about how the Arduino handles these characters. A set of different functions was created each one writing a character which corresponded on the region it was associated with as well as its functionality and intensity. A sample code of how these characters are written is presented below.

```
97      // 2     ------------------------------- INDEX -----------------------------------
98      public static void IndexUp()
99      {
100         Debug.Log("Collision index UP!");
101         sp.Write("c");
102     }
103
104     public static void IndexUpNo()
105     {
106         Debug.Log("Index Up no more!");
107         sp.Write("n");
108     }
```

Figure 45.Characters corresponding to the upper index's behavior

The first function writes to the stream the character 'c' which is associated with a collision of the upper part of the index finger. The character 'n' is associated with the lack of collision of that finger part.

However, opening a communication port and having a means of writing in the port is not enough. The characters must be written in the stream when there is a change in a collision status. This was achieved by making use of unity's collision detection capabilities through its rigid body component. As mentioned above each bone area has embedded in it a collider, a rigid body as well as a script. The purpose of the scripts on the bones was to call the functions that write characters on the serial port. When a collision occurred the bone area that collided called through its script the function that wrote on the serial port the character associated with it. The call of these functions happen inside Unity's OnColissionEnter and OnColissionexit.

```
6       void OnCollisionEnter(Collision col)
7       {
8           MyHandsControli.IndexDown();
9       }
10
11
12      //When the colission with the upper middle finger is no more
13      void OnCollisionExit(Collision col)
14      {
15          MyHandsControli.IndexDownNo();
16      }
17
```

Figure 46. Changing collision status writes character on the serial port

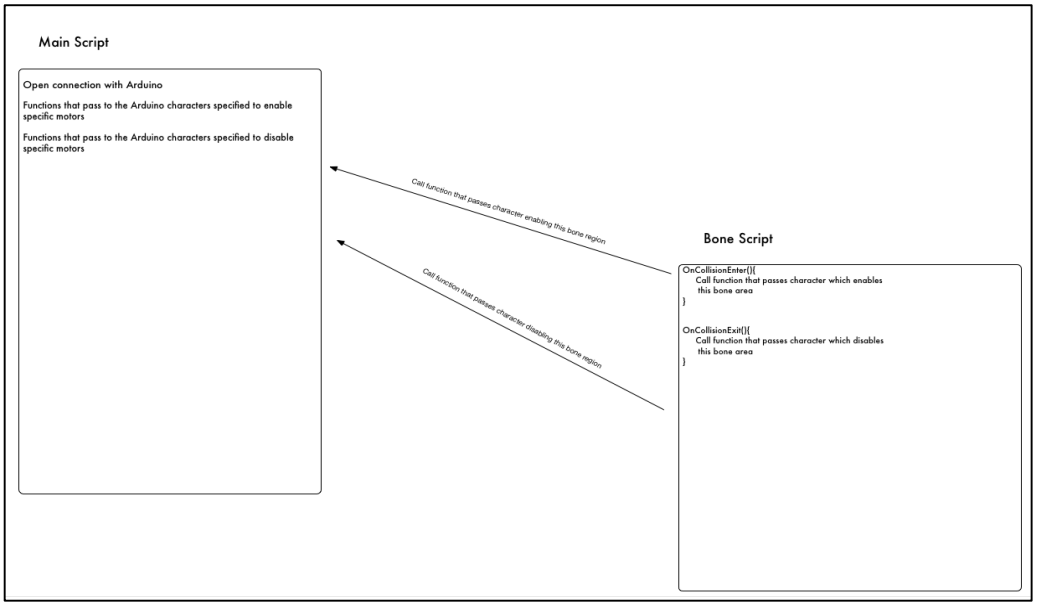


Figure 47. The bone script calls write functions from the main script

4.6 System Blueprint

Different technologies hardware and devices have to work in conjunction for the system to be able to provide vibrotactile feedback. Unity is responsible for 3D graphic rendering and collision detection, leap motion for hand tracking, electric motors for tactile provision and the Arduino for the control of these motors. Moreover, the 3d printed glove serves as a component on which the motors will be attached to. (Figure 48)

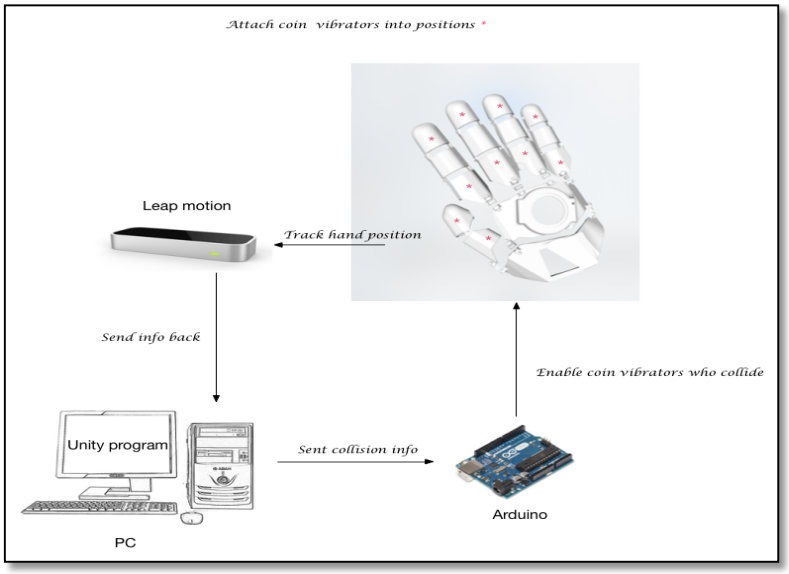


Figure 48.System Blueprint.

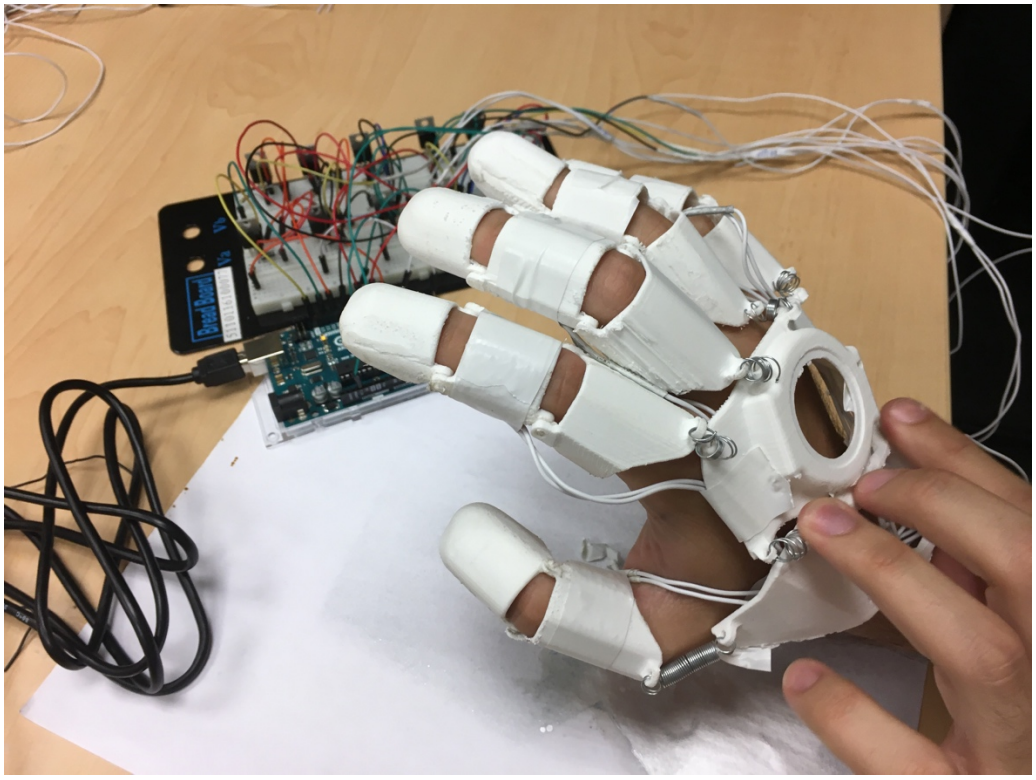


Figure 49. The implemented haptic glove

Chapter Six

3D Demo application

5 3D Demo application

Figure 50.Game's Main Menu

A 3D game application was also developed (Figure 50). The purpose if this application was to showcase the functionalities of the haptic glove. The game was a whack-a-mole clone game. The goal of the game is to hit with your hand as many rabbits as possible in a short amount of time (30 seconds). Each rabbit jumps from the hole randomly and is hittable while on the air. The game was developed in unity game engine and takes advantage of both the leap motion's controller technology and the haptic glove's capabilities. In addition, different bunnies offer different vibrating intensities to the hand. This section will present the stages of the development process.

5.1 Environment and 3D models

The cottage farm on which the scene takes place is created with the help of unity's terrain editor and 3D assets from the asset store. The first step to create the level was to create a unity terrain. This unity game object adds a large, flat plane to the scene. The Terrain's Inspector window provides a number of tools a developer can use to create detailed landscape features. These tools are based around the concept of painting detail and with the exception of the tree placement tool, all have the same options for brushes, brush size and opacity. The tools can be used to set the height of the terrain and also add coloration, plants and other objects.

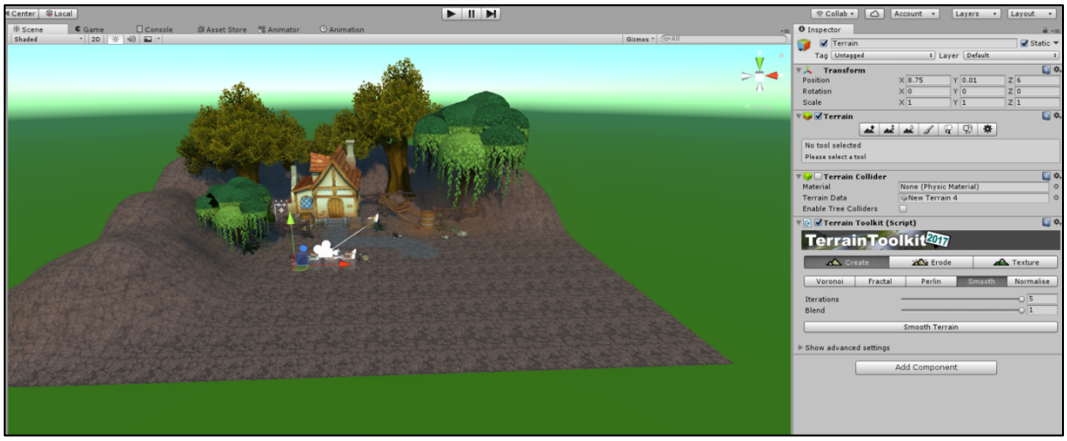


Figure 51.The terrain with game objects on it

Most of the 3D models imported to the scene's terrain were from the Unity store including the house, fence, trees, skybox, barrels, carts and bunnies (Figure 51). Their position in the terrain and size was changed through the Transform component attached on them. Their position was layout in front of the camera's field of view in order to create the desired scenery.

The scene included 7 different rabbits jumping out of position. The game includes 3 different types of rabbits. A red small rabbit gives a lower intensity of haptic feedback, a blue bigger rabbit labeled "boss" a higher and finally a yellow rabbit that jumps on the background. The way to interact and hit the yellow rabbit is through a hand gesture (making the hand into a gun and shooting). This was implemented with the help of the leap Motion's API. Each rabbit contributed to the final score a different amount of points (the red rabbit offered 10 points, the blue 30 points and the yellow 50 points). The rabbit models imported in this project were included in the "Level one monster pack" created by PI Entertainment Limited. The package was downloaded from the asset store and imported to the project. The models were textured, rigged and fully animated. On each rabbit game object, a collider, rigid body and a script was attached. With the use of the script the animations played in accordance to the player's action. However, before focusing on this game's scripting process, a reference on basic concepts of unity's animation is required.

5.2 Animations

There are two important key elements in regards to animation creation in Unity. Animation clips and animation controllers are the two building blocks of animation (Figure 41). They work together in order different frames of action to be created and controlled.

Animation clips are the smallest building blocks of animation in Unity. They represent an isolated piece of motion such as a rabbit ear movement, a body's change in position, a foot's rotation and so on. Animation clips can be manipulated and combined in various ways to produce lively end results. The animations are created by keyframing the different piece's position throughout the timeline. Although Unity offers tools for simple animations, creating more sophisticated sets of animations requires the use of external applications such as Maya, Blender, 3DS Max etc.

The **animator controller** has references to the animation clips used within it, and manages the various animation states between them using a State Machine. The alternation between the states of the state Machine is achieved with the help of animation parameters. Animation Parameters are variables that are defined within an Animator Controller that can be accessed and assigned values from scripts. This is how a script can control or affect the flow of the state machine. There can be four basic types of animation parameters, an int (an integer), float (a number with a fractional part), Bool (true or false value) and a Trigger (a boolean parameter that is reset by the controller when consumed by a transition). In order to assign animation to a 3D object the animator controller is included in an Animator component and that component in the 3D model.

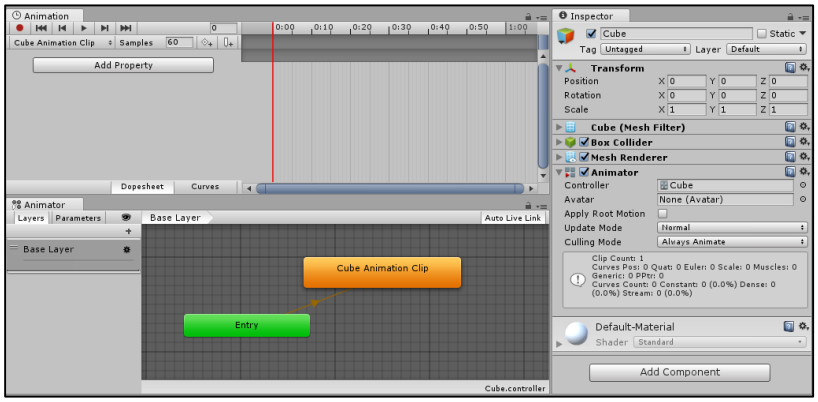


FIGURE 42. Animation Clip and Animation Controller panels.

All the 3D models from the monster pack are pre-animated. The rabbit model includes attack, damage, die, idle and move animations. This game made use of three of these animations. An animator controller was used to alternate between an idle, a move and a kill (the hit of the rabbit) animation (Figure 52). When a rabbit is randomly enabled to jump, the animation status changes from idle to move. In this state the jumping animation is executed and from this point of time the rabbit can either return on its idle animation on the ground or get hit by the player enter the rabbit-die animation and then eventually end up to the idle animation as well. The state animation to a move animation changes with the help of the move parameter while the entrance to the rabbit-die state is achieved with the animation parameter hit.

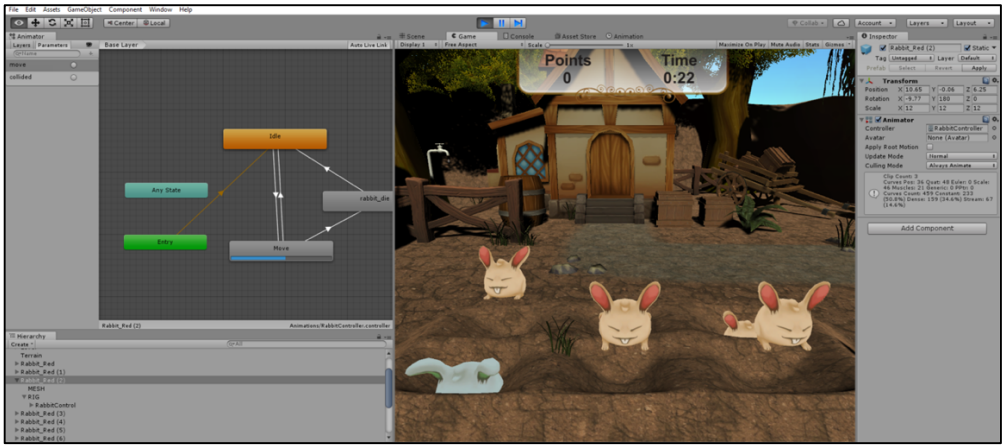


Figure 52. Rabbit Animator.

An animation clip and animation controller was also used for the points appearing when hitting a rabbit (Figure 53). Initially, an animation clip of the text movement was created with the use of the Animation panel. The animation consisted of the text being enabled in the first frame, gradually elevating and lastly being disabled on the last frame. The opacity of the text color was also changed between frames. The transition between the idle (text disable) and up (the text is animated as described above) states was achieved with the help of the hit parameter. A text was placed above each rabid waiting to be enabled when that rabbit would be hit. Each text displayed a different number of points depending on the type of the rabbit. The text objects were an element of a canvas which is the basic component of Unity's user interface (UI) system.

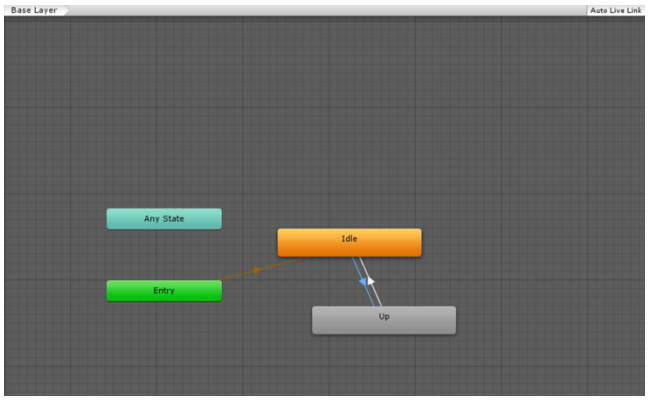


Figure 53.State machine of the text's animation controller

Figure 54.Point text being animated.

5.3 Graphical User Interface

Unity's UI system offers tools for designing user interfaces. As stated above the basic element of a graphical user interface in unity is a canvas. The **Canvas** is the area that all UI elements should be inside. The Canvas is a Game Object with a Canvas component on it, and all UI elements must be children of such a Canvas. By creating an UI element, a canvas is automatically created if there isn't any. The graphical user interface element included on a canvas can be an image, button, text, slider, scrollbar etc. The game consisted of four different canvases.

- A Head-Up Display (HUD) canvas used for basic information such as the time left and total points

- The main menu canvas. The main menu had also sub menus labeled, how to play and Settings. The "how to play" submenu offered a brief description about the goal of the game while with the settings submenu the player was able to adjust the volume's strength.

- The end canvas for when the game is over. The game over menu provided the player with the final score and the abilities to either play again or exit the application

- A canvas was also needed for the text appearing when a rabbit is hit.

The GUI elements included on the canvases were photos, text, buttons and a slider. The images used as stylized borders were created with the use of Photoshop. (Figure 55)

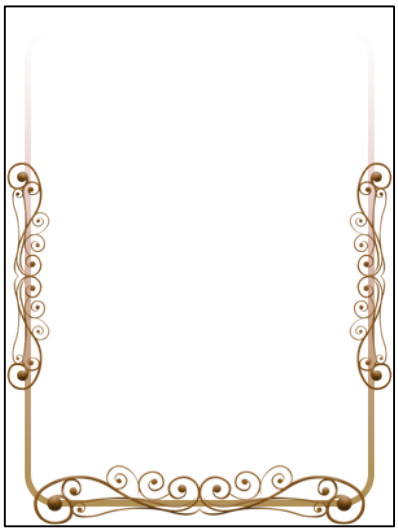


Figure 55.Menu border image

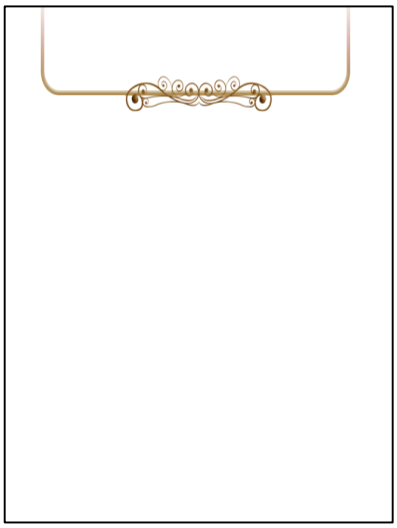


Figure 57. HUD border image

5.4 Game scripting

The gameplay of this game application (Figure 56) was based on simple concepts. There is a time countdown, animations triggering randomly, score count as well as other script related functionalities. The game consisted of three important scripts *Gamecontroller.cs* responsible for the basic functionalities of the game, *CollisionCheck.cs* responsible for updating game points, enable the text points animation as well as the rabbit hit animations and *MenuStuff.cs* containing functions for menu functionalities such as quitting the game and make transitions between different submenus.

Figure 56. Gameplay

The game's main Loop takes place inside the Gamecontroller.css update function. In this function rabbit animations are triggered through a random function (possibility to be triggered 50%) on specific time intervals, the points are updated and the time is counted down until it reaches zero.

```
50      // Update is called once per frame
51      void Update () {
52
53          if (!gameEnded)
54          {
55              CountGameTime();
56              CountRoundTime();
57
58              if(roundTimer == fixedRoundTime)
59              {
60                  ActivateRandomRabbits();
61                  ActivateExtraRabbit();
62              }
63              pointsText.text = GUIPoints();
64              timeText.text = GUITime();
65          }
66          else if(!gameStopped)
67          {
68              finalPoints.text = points.ToString();
69              canvasEnd.SetActive(true);
70              gameStopped = true;
71          }
72      }
73
```

Figure 57. Code sample - Game's main loop

```
rabbits[i].transform.parent.transform.parent.GetComponent<Animator>().SetTrigger("move");
```

Figure 58. Code Sample - Triggering a rabbit animation

In this script's start function the basic values of the game are being initialized. CollisionCheck.cs script takes advantage of the rigid body's OncollisionEnter function in order to trigger rabbit hit animations, text animations and increase the score if the virtual hand collides with the rabbits.

```
30      // Use this for initialization
31      void Start () {
32          InitGame();
33      }
34
35      public void InitGame()
36      {
37          gameEnded = false;
38          gameStopped = false;
39          points = 0;
40          roundTimer = fixedRoundTime;
41          ActivateRandomRabbits();
42          canvasMenu.SetActive(false);
43          canvasEnd.SetActive(false);
44          gameTimer = gameTime;
45          canvasHUD.SetActive(true);
46          pointsText.text = GUIPoints();
47          timeText.text = GUITime();
48      }
```

Figure 59. Code sample - Game values initialized

In addition, the big rabbit which offers a higher intensity haptic feedback is differentiated from the other rabbits by adding a “boss” tag to it. When the handcontroller script detects a collision with a 3D object containing a boss tag writes to the stream characters corresponding to a higher intensity vibration.

Finally, with the help of the Leap motion’s API a gesture based interaction with the yellow bunny is achieved. The way in which the yellow rabbit is hit is by extending the index finger and pointing to the bunny. The FingerDirectionDetector.css script provided a way to be aware if the finger was pointing on the rabbit object. The ExtendedFingerDetector.css script was able to be aware if the finger was extended. By using the DetectorLogicGate.css the hit function of the yellow rabbit (animation and score count) was used when both detector conditions were true. This script acted as an AND gate between these finger detections. All of the scripts were embedded on the Game controller 3D object.

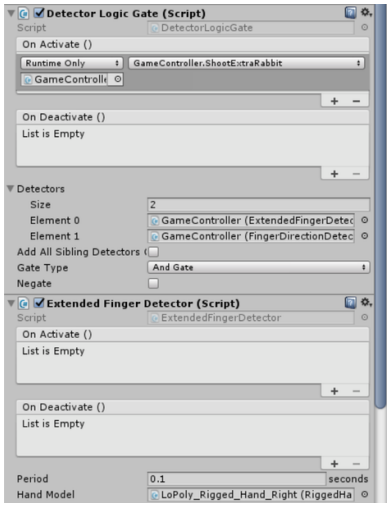


Figure 60. When AND gate returns true the hit function of the yellow rabbit is enabled

5.5 Audio

The audio of the game is added with the help of the audio source component. An Audio Source is attached to a Game Object for playing back sounds in a 3D environment. In order for the sounds to be played an Audio listener is also required. In this application the audio listener was attached to the camera as it represented the user. Each rabbit had an audio source playing sound whenever a rabbit was hit. Moreover the gamecontrol game object had an audio source which was enabled all the time. The audio played by that audio source was the background music of the game. Audio clips are played using PLAY, PAUSE, STOP. The volume can also be adjusted (like on the menu options) using the volume property.

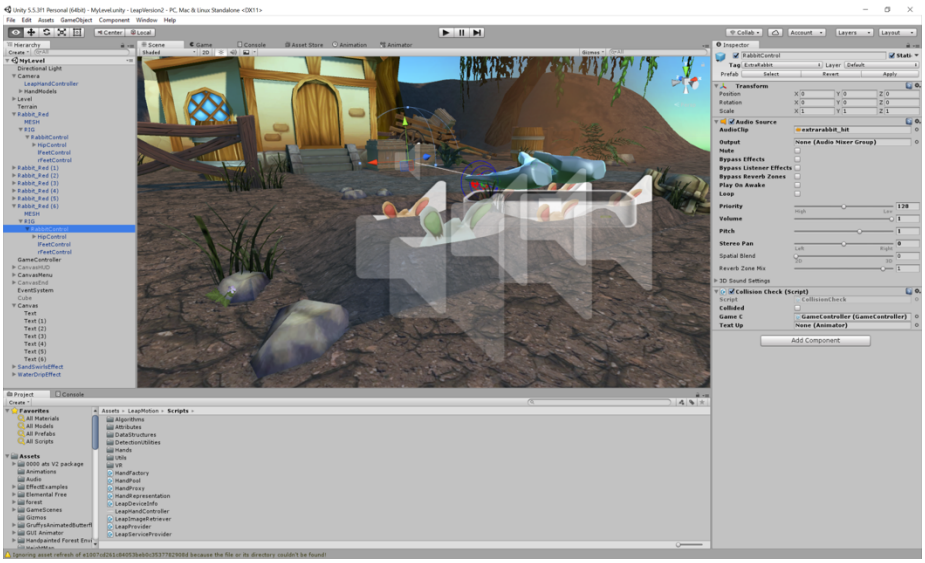


Figure 61. Audio sources for each rabbit

5.6 Particle Systems

Finally, two particle systems were imported in order to add a more realistic feel to the cottage scene. The water dripping from the faucet and the sand flying around were both particle systems. **Particles** are small, simple images or meshes that are displayed and moved in great numbers by a particle system. Each particle represents a small portion of a fluid or amorphous entity and the effect of all the particles together creates the impression of the complete entity. Using a smoke cloud as an example, each particle would have a small smoke texture resembling a tiny cloud in its own right. When many of these mini-clouds are arranged together in an area of the scene, the overall effect is of a larger, volume-filling cloud.

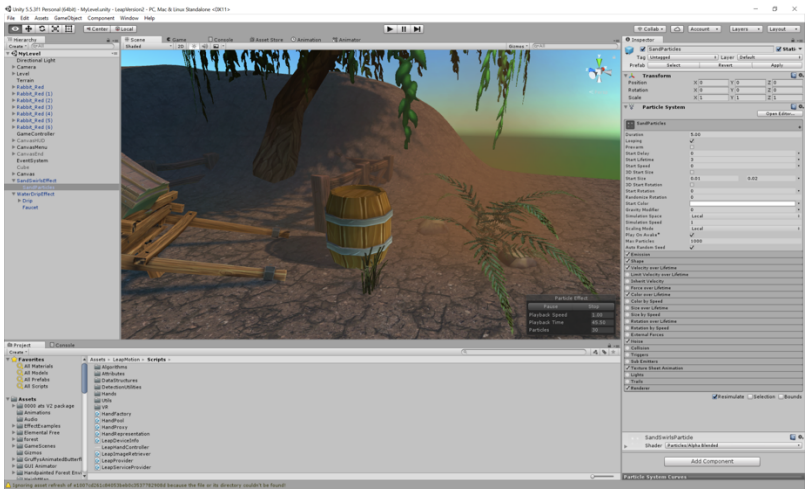


Figure 62. Sand particle system

Chapter Six

Conclusion and Future Work

6 Conclusion and Future Work

This thesis presented a way of tactile feedback provision using ERM vibration motors. The system designed was in the form of a wearable glove and offered features such as multiple points of tactile simulation, different vibration intensities and the ability to enable the motors independently. By presenting a way to create a haptic system and discussing important aspects of haptics there is hope current readers and future developers can be in position to expand on this work by adding more types of tactile feedback (thermal, skin pressure) and even add kinesthetic feedback to the system in order to control muscle and joints.

6.1 Technical Challenges

The biggest challenge starting this work was to engineer a way the hand can feel haptic sensations based on interactions with virtual objects. A study had to be made about the different types of feedback the hand can feel, what technologies are available and how they can work in conjunction to stimulate the skin. Although the individual technologies were based on simple principles combining them required a deep technical knowledge of them.

In addition, creating the cad model of the glove proved much more time consuming than initially planned. The glove consisted of 16 different components who had to go through extensive amount of editing before being able to be 3D printed. The whole process creating the glove and 3D printing took about a month, a considerable amount of time in a time constricted project.

Besides these technical related challenges, the initial budget defined in a certain degree the development process. Selecting different hardware components such as prototyping breadboard, microcontroller, motors, cables, resistors, transistors has to be made in accordance to the initial budget. Compromises had to be made in order for the budget not to be surpassed and the system to achieve its initial objectives in the best way possible.

The availability of each component was a factor also taken into consideration as the project was developed with a specific time schedule in mind and waiting for components for long periods of time to be acquired was not a feasible option.

6.2 System Limitations

Although the system achieved its initial goal of haptic feedback provision limitations occurred due to some of the above challenges or unexpected factors. A first limitation of the system is that due to the fact it uses ERM motors the response time is not as fast as it could have been with LRAs. On very fast collisions with virtual object because the ERM motor takes some milliseconds to be fully on the vibration is either weaker or almost no

detectable. In addition, because the glove is designed in specific measurements different hand sizes may have issues wearing the glove. Finally, the number of motors lead to an excessive amount of cables which made the whole system more fragile and less compact.

6.3 Future Work

Despite some limitations there are still opportunities for further enchantments. Upgrading the motors, adding different types of haptic feedback and improving the glove's design are all feasible options. A recommendation of future work includes

- Replacing the ERM motors with LRAs. *Linear resonant actuators* are able to provide a richer pattern of vibrations since their frequency and intensity values are independent from each other. In addition, LRAs have a higher response time which makes the tactile sensations they offer feel more realistic.

- Creating a more sophisticated way of providing different vibration intensities. Instead of passing a single character every time a collision occurs by passing a number and a character, the number corresponding to the velocity of the hand and the character to the bone of collision. By changing the duty cycle of the pulse width modulation according to the hand's velocity the haptic feedback received can have a more believable sense of touch. The velocity of the hand can be tracked with the help of the Unity's rigid body component.

- A redesign of the glove system. Using a printed circuit board instead of a prototyping breadboard as well as making the glove wireless can increase how compact the system is. Moreover, adding light and sound effect will enhance the futuristic look the glove is trying to achieve.

- Adding kinesthetic feedback through hardware components such as servo motors. Kinesthetic feedback will enable the control of joints and fingers.

- Adding temperature feedback with the use of thermoelectric devices.

In conclusion, building or expanding on a prototype system requires a developer able to engineer new ways technologies and frameworks work together and have an understanding of the basic principles defining the technologies used. Defining the goals, traits and characteristics of the system is a major first step any new design needs to take into consideration.

References

7 References

[1] Homesciencetools, "Sense of Touch," 2014.

[2] W. HARRIS, "How Haptic Technology Works".

[3] P. microdrivers, "UNDERSTANDING ERM VIBRATION MOTOR CHARACTERISTICS".

[4] P. microdrivers, "LINEAR RESONANT ACTUATORS - LRAS".

[5] Wikipedia, "inertial measurement unit".

[6] Synertial, "About Synertial gloves".

[7] L. Motion, "Leap Motion," [Online]. Available: https://www.leapmotion.com.

[8] V. Yem and H. Kajimoto, "Wearable Tactile Device using Mechanical and Electrical Stimulation for Fingertip Interaction with Virtual World," *Virtual Reality (VR), 2017 IEEE,* 18 March 2017.

[9] V. A. d. J. Oliveira, L. Brayda and L. Nedel, "Designing a Vibrotactile Head-Mounted Display for Spatial Awareness in 3D Spaces," *IEEE Transactions on Visualization and Computer Graphics,* vol. 23, no. 4, 23 January 2017.

[10] M. Lee, G. Bruder and G. F. Welch, "Exploring the Effect of Vibrotactile Feedback through the Floor on Social Presence in an Immersive Virtual Environment," *Virtual Reality (VR), 2017 IEEE,* 18 March 2017.

[11] Wikipedia, "Unreal Engine".

[12] Wikipedia, "Cry Engine".

[13] Wikipedia, "Frostbite".

[14] Wikipedia, "Unity".

[15] T. Instruments, "www.ti.com," 2013. [Online]. Available: http://www.ti.com/lit/ml/sszb151/sszb151.pdf.

[16] B. Cera, "Glove One," 2012.

[17] L. Chilson, "The Difference Between ABS and PLA for 3D Printing".

[18] T. Hirzel, "Pulse width modulation".

[19] ISO, "Ergonomics of human-system interaction — Part 910: Framework for tactile and haptic interaction," 2011.

[20] Wikipedia, "Haptics".