Virtual Reality Based
High Fidelity Excavator Training Simulator

Submitted By

Ammar Shafqat

F19604014

Zakria Zaheer

F19604010

Supervised By

Dr. Awais Yasin

Co-Supervised By

Engr. Rida Batool

Department of Computer Engineering

National University of Technology (NUTECH)

Islamabad, Pakistan 2023

Virtual Reality Based
High Fidelity Excavator Training Simulator

By

Ammar Shafqat

F19604014

Zakria Zaheer

F19604010

A Project Report Submitted to the Department of Computer Engineering in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Engineering

Department of Computer Engineering

National University of Technology (NUTECH)

Islamabad, Pakistan 2023

BS CEN	**Ammar Shafqat**	**Zakria Zaheer**	**2023**

CERIFICATE OF APPROVAL

It is certified that the project titled "Virtual Reality Based High-Fidelity Excavator Training Simulator" was carried out by Ammar Shafqat, Reg. No. F19604014 and Zakria Zaheer Reg. No. F19604010. under the supervision of Dr. Awais Yasin, National University of Technology (NUTECH), Islamabad, is fully adequate in scope and quality, as a capstone project for the degree of BS in Computer Engineering.

Supervisor: ----------

Dr. Awais Yasin

Associate Professor

Dept. of Computer Engineering

National University of Technology (NUTECH), Islamabad

HOD: ----------

Dr. Kamran Javed

Associate Professor

Dept. of Computer Engineering

National University of Technology (NUTECH), Islamabad

ACKNOWLEDGMENT

We would like to express our special thanks and gratitude to our supervisor Dr. Awais Yasin as well as our co-supervisor Engr. Rida Batool who gave us this opportunity to do a wonderful project on the topic of “Virtual Reality Based High Fidelity Excavator Training Simulator”, Additionally, this endeavor would not have been possible without the generous support from Sir Afran, who helped us a lot in finalizing this project within the limited time frame.

DEDICATION

Our most profound thanks goes to our parents for their continuous prayers and unconditional love. Without their spiritual support, we would not possibly have achieved whatever we have and would not have overcome the tough time of our lives.

ABSTRACT

The project is aimed at the development of a VR-Based low-cost and effective training system of a most widely used earth moving machinery i.e. *An Excavator*. The project replicates the original driving as well as excavator-specific controls of a real excavator of Caterpillar Company through steering wheel, accelerator pedal, clutch, break and 2 x joy sticks integrated and interfaced in real-time with virtual model developed in PC using Unity-3D Software. It is a green solution meeting critical training needs with multidimensional societal and training impacts for low-cost, eco-friendly and safe training. The project will be meeting domestic needs especially for training man power heading to construction industries in Gulf.

Keywords: Virtual Reality, Training Simulator, Excavator, Construction machinery

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

A virtual reality simulator is a combination of computer software and physical devices designed to replicate the response of physical objects based on a user's interactions, with the output displayed through a visually immersive experience and experiencing the movement on the motion base. Various interpretations of this definition exist, but the core essence of a VR simulator lies in its simulation capabilities. Hence, it should adhere to the principles of simulation.

Simulation is defined as “the process of creating a model of a real or theoretical physical system, executing the model on a digital computer, and analyzing the execution output”. Nevertheless, traditional computer simulation lacks real-time user interaction to modify model variables and visualize the results in an interactive manner. Refining the previous definition, a VR simulator can be described as a system of computer programs and interfacing devices that allow users to interact in real-time, calculating physical object responses based on their representations, and presenting the outcomes through three-dimensional graphical objects. It's worth noting that VR simulators may incorporate advanced sensory interfaces such as audio or haptic devices to enhance human perception.

Since their inception in various industries like military, aerospace, automotive, and naval, virtual reality (VR) simulators have served not only as tools for cost reduction in product design and training but also for real-time decision-making. They have become valuable assets for device operation, manufacturing, and process evaluation. Their popularity stems from their capability to provide users with high-quality graphical 3D information on machine operations in an intuitive and real-time manner and with motion base providing users with a heightened sense of immersion and realism by incorporating physical movements and sensations. Another significant advantage is the representation of not just the physical machine but also its interaction with the environment in mathematical or computational form.

1.1 VR Simulator Architecture and Pipeline

While a definitive consensus is yet to be reached regarding the essential components and interactions within a VR simulator system, various functional models and pipelines have been proposed as potential prototypes for further development. These key elements are summarized in the diagram below:

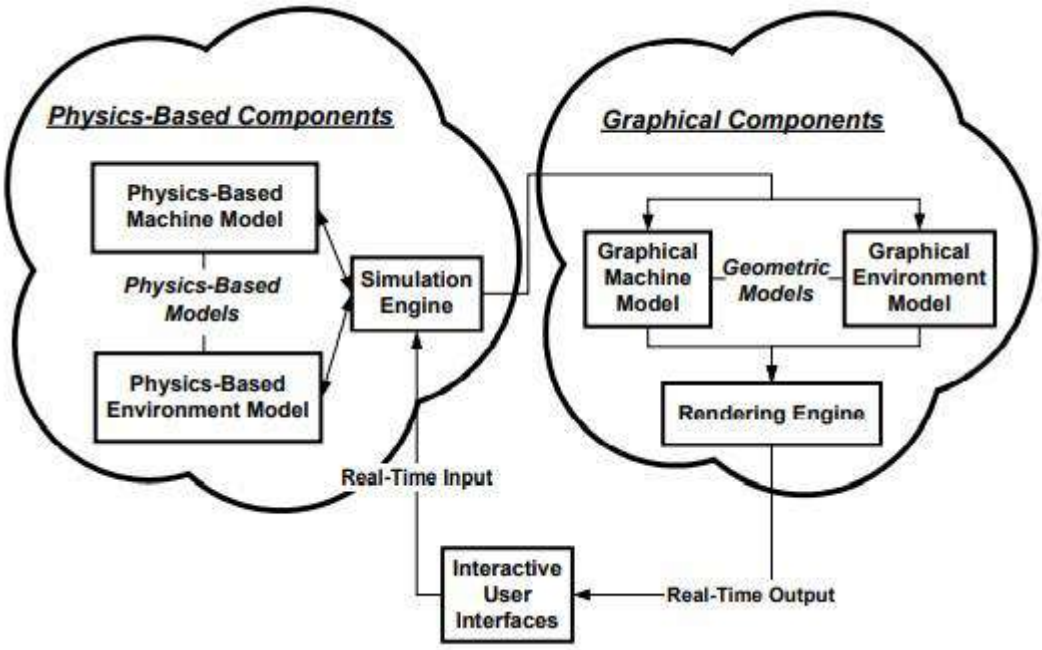

Figure 1.1 Architecture and Pipeline of VR Simulator

The architecture of a VR simulator primarily comprises four main components: physics-based elements, graphical elements, motion base and interactive user interfaces. The physics-based components encompass a simulation engine along with two physics-based models, which represent the machine's physics and the surrounding environment. Meanwhile, the graphical components consist of sub-parts, including graphical machine models and graphical environment models, whereas motion base provides with realistic sensation, all integrated with a built-in rendering engine.

1.1.1 Advantages of VR Training Simulation

- **Immersive and Engaging Learning Experiences**

VR-based simulations strive to replicate reality closely, stimulating learners' intellect and engaging them emotionally. By combining characters, stories, scenarios, sounds, and visuals, VR simulators create a rich learning environment that captivates learners' attention. The interactive nature of VR allows learners to move and interact within the virtual environment, enhancing engagement and fostering effective learning experiences. This heightened sense of presence and interactivity leads to better

knowledge retention and promotes critical thinking skills, preparing learners to apply their knowledge confidently in real-world situations.

- **Safe and Risk-Free Training Environment**

VR training simulations offer a secure space to practice tasks that involve excessive risk, such as electrical wiring repair or dealing with hazardous waste. In a virtual setting, trainees can interact with virtual scenes and hazards without exposing themselves to actual dangers. This capability makes VR simulator training an ideal tool for training personnel in high-risk, potentially dangerous situations without putting them in harm's way. The risk-free environment empowers learners to make mistakes, learn from them, and build their expertise through repeated practice, ultimately enhancing their competence and confidence in handling real-life challenges.

- **Highly Effective Skills**

VR simulators have proven to be highly effective for up skilling and developing unique skills. Research findings indicate that VR beginners complete their training significantly faster than classroom learners and e-learners alike. Moreover, individuals who use VR simulators exhibit a remarkable increase in confidence when applying their newly acquired knowledge after training. The immersive nature of VR fosters a profound emotional connection and engagement among learners, making VR novices notably more emotionally linked to the content compared to their classroom counterparts. Apart from these research-backed advantages, VR training simulations provide a realistic "real-world" environment, enabling employees to practice and implement concepts, even in hazardous and complex situations. The ability to monitor data and track progress keeps personnel motivated and focused on continuous improvement. Overall, VR training simulators serve as a powerful and efficient tool for enhancing skills and knowledge.

1.2 Overview

We proudly introduce our state-of-the-art VR-based high-fidelity excavator training simulator. Through accurate integration of driving and excavator controls (joy-sticks) with simulation software, a VR headset, and a motion base, we offer a truly authentic and cost-effective training solution for this extensively used construction machinery. The simulator encompasses diverse training scenarios, simulating various environments, weather conditions, and day-night situations, providing trainees with a comprehensive and adaptable learning experience.

Key Features:

- Realistic and Immersive Experience
- Accurate Excavator Controls
- Diverse Training Scenarios

Benefits:

- High-Fidelity Training
- Safe and Cost-Effective Training
- Skill Refinement and Confidence Building

Figure 1.2 Excavator Training Simulator

Figure 1.2.2 Excavator Training Simulator

1.3 Problem Statement

The construction industry poses significant challenges for workers, especially when it comes to operating heavy equipment like excavators. Workers must learn to control these

machines manually, which can be a major and complex task. Excavators play a crucial role in various industrial operations, including excavation, material handling, and grading. However, mastering the operation of excavators requires operators to manipulate a multitude of controls (joysticks, pedals, and switches) in a non-intuitive manner, leading to a lengthy and costly training process.

Existing training methods for excavator operators often lack the realism and practicality needed to effectively prepare them for real-world scenarios. As a result, there is a growing demand for a more efficient and immersive training solution that bridges the gap between theory and practical application.

To address these challenges and meet the specialized needs of construction workers globally, we have undertaken the development of a VR-based high-fidelity excavator training simulator. This simulator aims to provide trainees with a realistic and authentic experience by accurately integrating driving and excavator controls with simulation software, a VR headset, and a motion base. By creating diverse training scenarios in various environments, weather conditions, and day-night situations, we aim to offer a comprehensive and adaptable learning environment for mastering excavator operation.

The primary objective of this project is to revolutionize excavator training, significantly reducing the training period and associated costs, while enhancing skill development and operator safety. The virtual reality environment of the simulator allows users to gain hands-on experience in a risk-free setting, promoting confidence and efficiency in handling these heavy machines.

Through this project, we aspire to contribute to the construction industry's workforce by equipping operators worldwide with advanced training tools that foster expertise, productivity, and safety in excavator operations.

1.3.1 Lack of Operators' Performance

Operators in the construction industry spend a significant portion of their workday operating heavy machinery like excavators, with the primary goal of accomplishing tasks efficiently and safely. However, past accident reports show that the human factor has a significant impact on accidents. A recent study on earthmoving equipment revealed that operators misinterpreted hazards in more than 46% of cases. Furthermore, analysis of wheel loader accidents shows that accidents involving mining and non-mining workers being hit, run over, or crushed by wheel loaders account for almost 41% of all wheel loader-related fatalities. These statistics emphasize the urgent need to enhance operator awareness and skills to ensure the safety and well-being of operators, other personnel, and the surrounding environment. Improving operators' performance can play a crucial role in mitigating accidents and improving overall operational safety in the construction industry.

1.4 Specification of VR-based High Fidelity Excavator Training simulator

The VR-Based High-Fidelity Excavator Training Simulator project offers a cost-effective and efficient solution for training individuals in heavy construction machinery operation. Leveraging VR-based training simulators, construction companies can provide comprehensive training without the need for physical training sites or risking any damage to expensive machinery. The simulator ensures a safe and realistic learning experience within a virtual environment.

The project is divided into two main parts: software and hardware.

Software Specifications:

Unity 3D: The simulator is developed using the Unity 3D software, allowing the creation of immersive 3D environments with backgrounds, terrains, meshes, objects, and models.

Blender: Utilizing blender enhances the creation and customization of detailed 3D models, adding to the realism of the simulation.

Logitech SDK: The Logitech software development kit facilitates seamless integration of Logitech hardware components with the simulation.

Hardware Specifications:

Logitech 3D Joysticks: The simulator uses Logitech 3D joysticks to replicate excavator controls accurately, ensuring a realistic and intuitive learning experience.

Logitech Steering Wheel: The integration of a Logitech steering wheel enhances the simulation of vehicle operation, providing a comprehensive training platform.

Logitech Pedal Set: The Logitech pedal set enables trainees to control acceleration and braking, further enhancing the authenticity of the training experience.

Motion Base: The motion base adds an unparalleled level of realism to the simulator, replicating the movements and vibrations of the excavator, giving trainees a true-to-life feel while operating the machinery.

The seamless integration of software and hardware components allows for a cohesive and effective VR-based training simulator. Trainees can practice and refine their excavator operating skills within a risk-free virtual environment, boosting operator confidence and safety.

Overall, the VR-Based High-Fidelity Excavator Training Simulator serves as a cutting-edge tool, revolutionizing excavator training and ensuring well-trained operators worldwide.

1.5 Aims and Objectives

Aim:

The primary aim of this project is to address the safety hazards and operational challenges associated with onsite excavator operations in the construction and other industries. The

project seeks to provide a cost-effective and efficient solution through the development of a VR-based high-fidelity excavator training simulator.

Objectives:

Safety Enhancement: Develop a realistic and immersive VR training simulator that allows operators to practice excavator operations in a safe virtual environment. By offering hands-on training without real-world risks, the simulator aims to minimize accidents and safety hazards.

Cost Reduction: By simulating excavator operations, the project aims to reduce fuel consumption, maintenance costs, and repair problems associated with real machinery training. Providing a cost-effective training alternative, the simulator will help industries optimize their resources.

Quick and Efficient Training: The simulator intends to enable new workers to learn excavator operations quickly and efficiently. By providing a realistic training experience, trainees can gain practical skills that can be directly applied to real-world scenarios.

Educational Collaboration: The project aims to facilitate Construction Technology and Training Institute (CTTI) by providing them access to the simulator. CTTI can utilize the simulator to enhance the training of their students and workers, equipping them with valuable hands-on experience.

The VR-Based High-Fidelity Excavator Training Simulator project strives to revolutionize excavator training by ensuring operator safety, reducing operational costs, and offering efficient skill development. By collaborating with CTTI, the project aims to extend its benefits to the construction industry and contribute to the overall growth and safety of the workforce.

1.6 Application

The VR-Based High-Fidelity Excavator Training Simulator holds significant potential for a wide range of applications, playing a crucial role in enhancing safety, efficiency, and skill development across various industries.

Construction Industry Integration: The simulator will be introduced to different industrial companies and construction industries, revolutionizing their approach to excavator training. By incorporating the simulator into their training programs, these industries can eliminate hazardous on-site practices, ensuring workers can operate in a safer environment.

Cost and Resource Savings: The adoption of the simulator results in substantial cost savings for both workers and companies. Reduced fuel consumption, minimized maintenance and repair costs, and decreased downtime translate into improved operational efficiency and economic benefits for the industries.

Training and Skill Enhancement: Students and workers in construction-related fields can utilize the simulator to practice operating excavators and gain better hands-on experience. By offering a realistic training platform, the simulator empowers trainees to refine their skills, resulting in a competent and skilled workforce.

CTTI: The Construction Technology and Training Institute (CTTI) will be the inaugural site for the simulator. By integrating the simulator into their training curriculum, CTTI can equip their students and workers with cutting-edge training, preparing them for real-world challenges in the construction industry.

Overall, the application of the VR-Based High-Fidelity Excavator Training Simulator extends to diverse sectors, fostering safer work environments, cost-effectiveness, and skill enhancement. By promoting efficient and realistic training, the simulator contributes to the growth and success of the workforce while elevating industry standards in excavator operations.

1.7 Strategy

1.7.1 Capstone Phase I

Weeks	Discussed Topics
Week 1	Finding Team members
Week 2	Project Selection
Week 3	Finalizing working on proposal
Week 4	Proposal defense
Week 5	Installation of software and choosing the best one to work on
Week 6	Studying of research papers on excavator simulators
Week 7	Study of simulation of behavior of excavator ,bulk material and their interaction
Week 8	Physical configuration of structure of excavator simulator
Week 9	Software architecture of structure of excavator simulator
Week 10	Working on SketchUp software to learn how we can make our excavator module
Week 11	We run the different samples in Unigine with dongle
Week 12	Gain understanding of coding with samples
Week 13	Preparation of Presentation
Week 14	Seminar-2 (Design Progress)

Figure 1.3 Weekly schedule phase I

1.7.2 Capstone Phase II

As we have some parts left in software like excavation stones, hence applying physics there was done in the second half of the project. Also learning about the hardware

and then selection of hardware and buying them was included in this part. After this, we plug in all the hardware with the software by using Logitech gaming SDK and interfaced it by scripting in C-sharp.

Weeks	Discussed Topics
Week 1	Started working on with the help of tutorials of Unity 3D
Week 2	Working on excavators parts, tried to build them
Week 3	Now we imported the parts and build them all parts according to the real excavator
Week 4	build all parts together according to the real excavator
Week 5	Making terrain ,working on collision with all rigid bodies
Week 6	Making terrain ,working on collision with all rigid bodies
Week 7	Making terrain ,working on collision with all rigid bodies
Week 8	Generating a prototype that works with keyboard the same function we wanted
Week 9	Generating a prototype that works with keyboard the same function we wanted
Week 10	Generating a prototype that works with keyboard the same function we wanted
Week 11	Starting working on hardware and shifting the prototype to hardware modules
Week 12	Starting working on hardware and shifting the prototype to hardware modules
Week 13	Finals presentation
Week 14	Open house

Figure 1.4 Weekly schedule phase II

1.7.3 Gantt Chart

Key Milestones	Months								
	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul
Literature Review	Projects Selection/ Preliminary Meetings								
Design Algorithm			Progress Seminar - I	Progress Seminar - II					
Hardware Interfacing					Assessment- I				
Project Testing						Progress Seminar - III			
Project Completion							Progress Seminar - IV		
Final Documentation								Final Presentation	Assessment- II

Figure 1.5 Gantt Chart

1.7.4 Timeline graphs

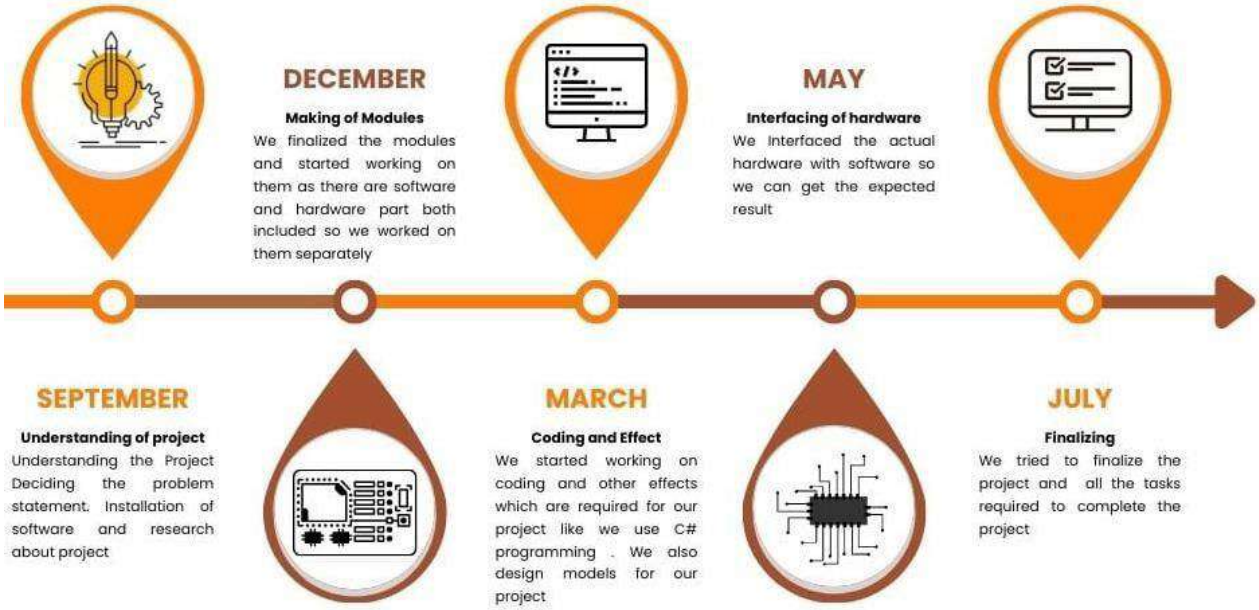

Figure 1.6 Timeline graph

1.8 Report Organization

This report is organized into six chapters.

- Chapter 1 describes a typical VR simulator system in terms of its important components and the parts of these components and the project introduction and specification with the project planning.
- The Literature review is proposed in Chapter 2 to provide basic information on the previous excavator training simulator research and used technologies.
- Project design is described in Chapter 3 to provide all development procedures, methodology, and detail about simulation modeling.
- Tools and Techniques are explained in Chapter 4 to provide all information about hardware and software technical specifications.
- All project results are included in Chapter 5 including all hardware and software outcomes as required with comparing to other excavators.
- In Chapter 6 we concluded our project and provides all the references from where we took all our help during the project.

Chapter 2

LITERATURE REVIEW

Looked through various research papers on the project and got some insights that were very helpful in completing the project.

2.1 Related Work

There are many Excavators Simulation has been introduced few of them are as follows:

2.1.1 VR Simulator Using Leap Motion

A leap motion controller and a digital headset are used as devices in this excavator simulation system. The Digital Truth headset used with this device is the Oculus Rift. Head movements of the person wearing the Oculus Rift are sent to the digital camera. Users can freely turn their heads to see inside the gadget's digital world. In addition to the digital excavator, a digital challenge website is also available. Leap Motion Controller, on the other hand, is used as a device for digitally managing gadgets. Gadgets are controlled by real finger movements. Consumer hand gestures are very important in controlling the simulator. The operator of this machine is his digital joystick and the hand holding the lever. While the digital hand is colliding with the digital joystick or lever, the person must perform the clutch gesture. After the grab gesture is maintained, the person wants to move on. Grab and slide the digital joystick or lever and the excavator will go through the joystick or lever function. Feedback from consumers is necessary to get real results from the developed system. These feedbacks are evaluated as individual decision-making stage and consumer satisfaction.[2]

2.1.2 A 3d Physics-Based Hydraulic Excavator

Real-time simulation techniques from multibody system dynamics allowed the development of a realistic but computationally efficient 3D physics-based model of a complex machine like a hydraulic excavator. The resulting simulator, which runs in a standard PC, can reproduce almost all the maneuvers performed by real excavators. The project has finished the software development stage, and now is starting the human evaluation stage: the simulator will be tested by experienced operators, and their findings will be used to adjust the mathematical model of the excavator (contact forces and other data not supplied by the manufacturer) to increase the realism of the simulator.[3]

2.2 Limitations and Bottlenecks

Leap motion controller and Oculus Rift remain difficult for excavator operators to master. New controls that are more precise and easier to learn are needed to develop excavator controls to achieve better results.

2.3 Summary

The first technology is still in development and is an excavator simulator that can be controlled by a Leap Motion Controller. A real hand appears as his 3D hand model, moving in real-time with the movement of the real hand. These 3D hand models take control of the system by grabbing the virtual joysticks and levers of the virtual excavator. New controls that are more precise and easier to learn are needed to develop excavator controls to achieve better results [2]. The second technology was used to build a pair of PC-based Show Digital Truth devices for excavator simulators. Enrich your scenes with meaningful detail with real-time terrain distortion and GPU-based texture blending. By inserting a photo into a dedicated monitor, the operator has an uninterrupted and immersive digital environment. Drilling force can be used to determine the most useful course of work and reverse mechanics can follow the mannequin [3].

Chapter 3

DESIGN AND IMPLEMENTATION

A hydraulic excavator is a construction dredge primarily powered by a hydraulic system. The excavator's hydraulic system transfers the hydraulic flow generated by the hydraulic pump to the working area through the hydraulic fluid. The external representation refers to some structural parts closely related to the internal hydraulic system. The external representation should include properties such as relative rod position, rod dimensions (length and width), rod stretch, and rod weight. Important assets identified in the internal and external representations are used to calculate the effective physical response of the machine. The motion calculation begins with the calculation of piston displacement given hydraulic flow and piston cross-sectional area, with a motion time segment formulated as follows:

Piston Displacement = (flow rate / ram cross-sectional area) × segmental time duration

Once this calculation is performed for the ram (cylinder) corresponding to the maneuver, the spatial information of the rod can be calculated through a series of geometric analyses. [4]

3.1 Design and Development

The project is divided into 2 parts; hardware and software. The software part consists of creating a simulator by using Unity 3D software where we created backgrounds, terrains, meshes, objects, and models in the 3D environment. On the other hand, hardware parts include joysticks, steering wheel, pedals, Oculus Rift S VR headset which are interfaced with the simulation software through USB interfaces and Motion Base which is interfaced through Arduino with simulation software through USB interface.

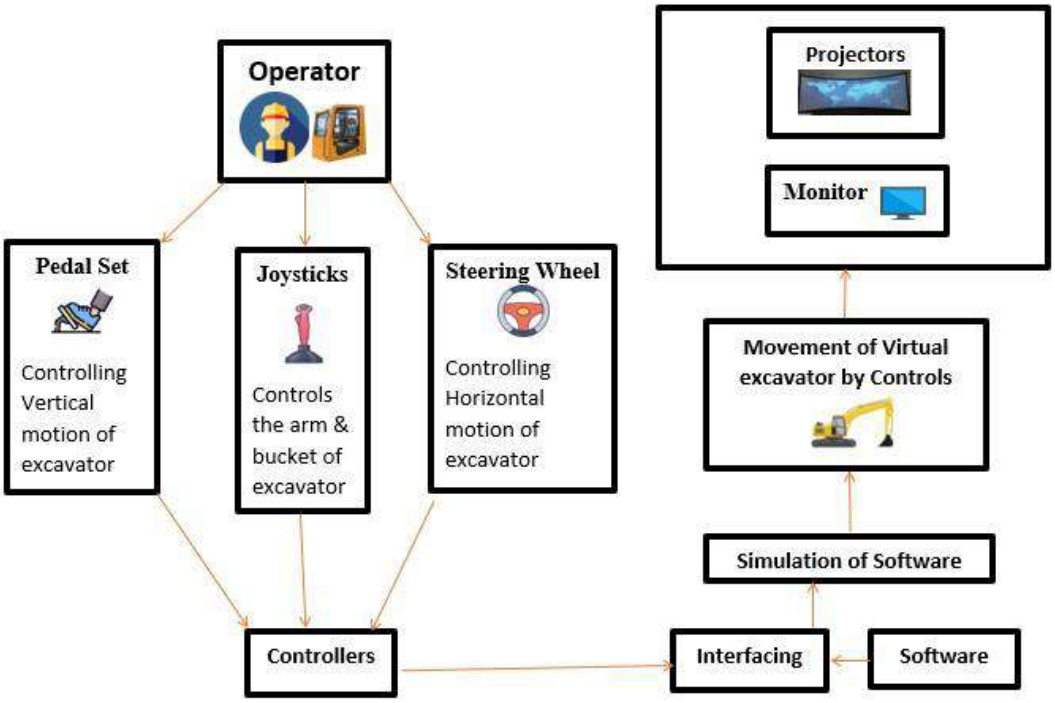

Figure 3.1 Block diagram

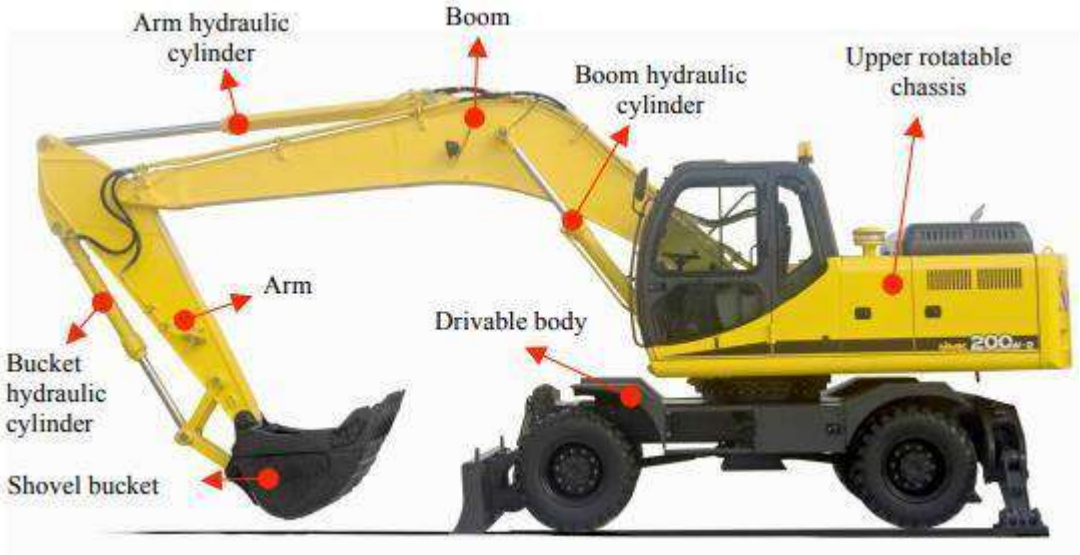

Figure 3.2 General View of Excavator

3.2 Methodology procedure

- Firstly, we have tried three different software (i.e., Unigine, Unity 3D, and Unreal Engine) and the best one we have selected is Unity 3D because of the easy availability of its tutorials and its vast assets store.
- Unity3D is a powerful cross-platform 3D engine and easy-to-use development environment. Simple enough for beginners and powerful enough for experts. Unity should be of interest to anyone looking to easily create 3D games and applications for mobile, desktop, web, and consoles. By using the software Unity 3D, we developed the simulator by creating the background/terrain. Using the excavator asset, we did scripting and set the collision for the objects. Added different objects like trees and stones. Then we picked the stones using the arm of the excavator and applied excavator physics there.
- Hardware parts include the Steering wheel, pedal set, joysticks VR headset and motion base. Furthermore we plan to arrange the three projectors and combine them to display. Also, we have thought of using a cabin to give our simulator a proper shape so that it looks like an original excavator.
- We have to interface software with the hardware. Make sure that they must align, and software controls must be linked with the hardware control. Timing is an important part hence there should be no delay so that it works precisely and accurately. For this interference, we changed the scripting and amend the codes, and did scripting again.
- For the motion base to achieve precise alignment between the motion base and the virtual excavator's movements in the simulation, multiple trials were conducted. The

main goal was to synchronize the motion base with the excavator's positioning accurately, especially during changes in terrain, such as steep inclines or declines.

- Through careful observation and analysis of the simulated excavator's movements, essential parameters were extracted to control the motion base. The collected data allowed for adjustments to the motion base's control system to ensure its movements closely mirrored the virtual excavator's actions.
- By fine-tuning various parameters and feedback mechanisms, a seamless correlation was achieved between the virtual excavator's maneuvers and the motion base's responses. The ultimate aim was to enhance the overall realism and immersion of the simulation, providing users with a more authentic experience for training, testing, and other simulation purposes.
- To seamlessly integrate the Oculus Rift S VR headset with the simulation, several technical considerations come into play. One crucial aspect involves meticulously calibrating the floor level within the Unity 3D environment to ensure an optimal virtual reality experience.
- Precise calibration of the floor level is essential to ensure that the user's virtual movements align seamlessly with their real-world physical movements, the user's real-world floor level is established as a reference point. This reference point serves as the foundation for positioning the virtual ground plane within the Unity 3D environment.
- In the end, we have to compile everything including software and hardware so that if we want to start the project the hardware and software both should start at the same time. Our Simulator is ready to work it can pick objects like stones from the ground and place is in different places and get to know how the excavator works.

3.3 Hardware Specifications

3.3.1 Steering Wheel

It is from the company Logitech. It is used to control the directions and movements of the excavator towards the horizontal X axis.

Figure 3.3 Steering wheel

3.3.2 Pedal Set

It is from the company Logitech. It is used to control the race and reverse of the excavator towards the vertical Y axis.

Figure 3.4 Peddles

Figure 3.5 Connection of Pedals and Steering Wheel

3.3.3 Joysticks

The Extreme 3D Pro Joystick from Logitech. It features a handgrip-designed twist grip, 12 programmable buttons, an 8-way hat switch, a rapid-fire trigger, and a weighted base. Once set up, Logitech's gaming software can be used to configure button assignments and macros to assist.

- Twist Rudder Control
- 12 Buttons
- 8-Way Switch
- Rapid Trigger
- Stable, Weighted Base

It is used to control the movements of the excavator in every direction. Also, it helps the excavator to move the bucket, stick, boom, and swing.

Figure 3.6 Excavator joystick

3.3.4 VR Headset

The VR Headset Rift S from Oculus. Designed to deliver an immersive and realistic virtual experience. Its specification features Crystal-Clear Visuals, wide field view, precise tracking, and spatial audio.

Display:

- Display Type: Fast-switch LCD
- Resolution: 2560 x 1440 pixels (1280 x 1440 per eye)
- Refresh Rate: 80 Hz

Field of View:

- 155 degrees

Tracking:

- Inside-out Tracking System
- 6 degrees of freedom (6DoF) for head and hand tracking
- Oculus Insight technology

Audio:

- Built-in stereo speakers
- 3.5mm audio jack for external headphones

Weight:

- Approximately 563 grams

Dimensions:

- Headset (folded): 10.94 x 6.30 x 7.87 inches (H x W x D)

Figure 2 Figure 3.6 VR Headset

3.3.5 Motion Base

The motion base is a critical component of our simulator, providing a dynamic and immersive experience for users. Our team undertook a meticulous approach in its creation, considering various dimensions and engineering aspects to ensure optimal performance. The motion base frame, carefully crafted from durable metal, forms the robust foundation of this system.

Hardware Components:

- **LBT2 Motor Driver:** The LBT2 motor drivers, renowned for their precision and reliability, play a key role in controlling the motion actuators of our simulator. These drivers facilitate seamless communication between the software and hardware, enabling precise and synchronized movements.
- **Windshield Wiper Motors:** Selected for their power and efficiency, the windshield wiper motors serve as actuators to generate the motion effects. Our thorough testing and assessment confirmed their suitability for the demanding requirements of the simulator, ensuring smooth and realistic motions.
- **12v 20 Amp DC Power Supply:** To deliver consistent and reliable power to the motion base, a robust 12V 20 Amp DC power supply was integrated. This power supply unit meets the high-power demands of the motors and ensures the stability of the entire system during operation.
- **Arduino:** The Arduino microcontroller acts as the brain of the motion base, orchestrating the precise control and synchronization of the various components. Its versatility and programmability empower us to tailor the motion profiles according to the specific scenarios encountered during the excavator training simulations.

3.4 Software Specifications

3.4.1 Design of Excavator Arm

As excavator is heavy machinery it has different parts including bucket, stick, boom, and swing. The bucket is the part in which the material is picked up or dig from the ground. It has sharp ends that help in picking. Then comes stick this is directly attached to the bucket and help the bucket to move more easily covering more angle and distance. Similarly, boom is used to help the arm excavator to move and cover more angle and distance to make excavator more useable. The last part is the swing which help the excavator to move as it help the whole body of excavator to move the body is quite heavy hence it take time to move. In our excavator lower portion is attached to wheels on which the heavy parts are attached.

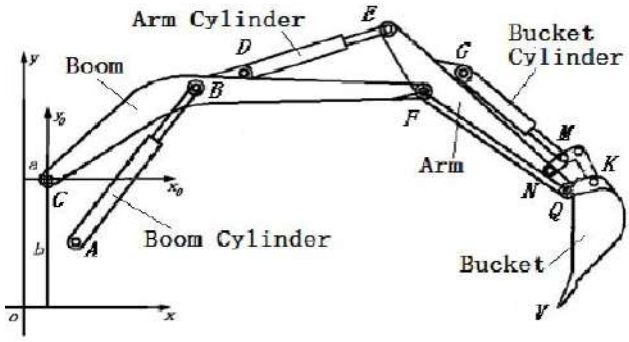

Figure 3.7 Arm backhoe of excavator

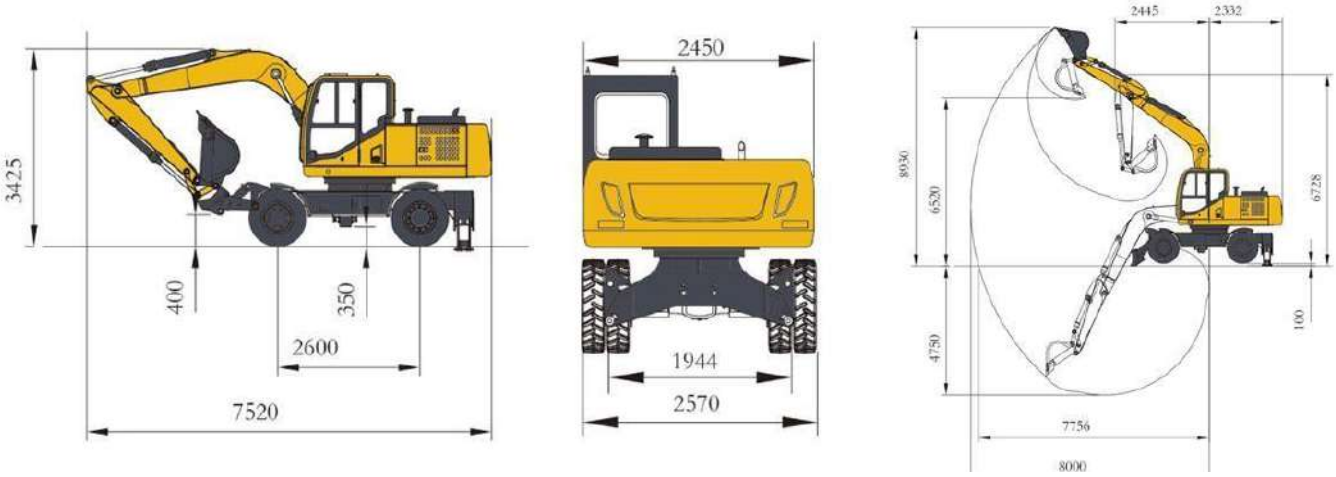

Figure 3.8 Dynamics model of excavator

3.4.1.1 Calculation of Pin Diameter

F = 10,350N

r = 392mm

L = 316mm

P = 15.7N/mm2

A = 66838mm2

Shear stress = 41N/mm2

Bending stress = 85N/mm2

Let, d = diameter of pin

Torque, T = F × r = 10300×392 = 4027400N-mm,

Load, W = P×A = 15.7×66838 = 1048656.9N

Now,

Max. bending moment = M = W*L/4 = (1048656.9 × 316)/4 = 82581724Nmm

Since, the pin is subjected to a suddenly applied load.

Considering,

Km = 1.5

Kt = 2

We know that, equivalent twisting moment Te = □(Kt × T)2 + (Km × M)2

= □ (2 × 4027300)2 + (1.5 × 82581724)2

Te = 124134176.5 N-mm

But,

Te = π/16 × d3 × shear stress 124134176.5 = π/16 × d3 × 42

Therefore,

d = 246.9mm

we know that, an equivalent bending moment

Me = 1/2[Km × M + Te] = ½[1.5 × 82581723 + 124134176.5]

Me = 124003380.5N-mm

But,

Me = π/32 × d3 × bending stress 124003380.5 = π/32 × d3 × 84

Therefore, d = 246.86mm

Taking the larger of the two values,

we have the diameter of pin = d = 246.9mm

3.4.1.2 Calculation for the total amount of material the bucket can lift

Since,

we have considered a light duty construction work. So, calculate a soil surface.

Density of soil is 1463kg/m3

For the existing model,

Volume of bucket V= 0.022m3

Total weight of soil,

W= Density × Volume = 1463 × 0.022 = 32.186 Kg

The calculated volume the of bucket,

V= 0.028m3

Total weight of soil, W= Density × Volume = 1463 × 0.028 = 40.964 Kg

Self-weight of bucket = 17 Kg

Now,

Total load acting on bucket = Self weight of bucket + Total weight of soil
=17+40.964 = 57.964Kg = 57.964×9.81=568.62N

By using this load, the dynamic analysis will be done [4].

3.4.3 Design of Simulator

This session explains the Unity interface, menu items, the usage of Assets, and developing Scenes:

- Downloading, installing, and activating Unity, the usage of the Hub, and managing to control your licenses.

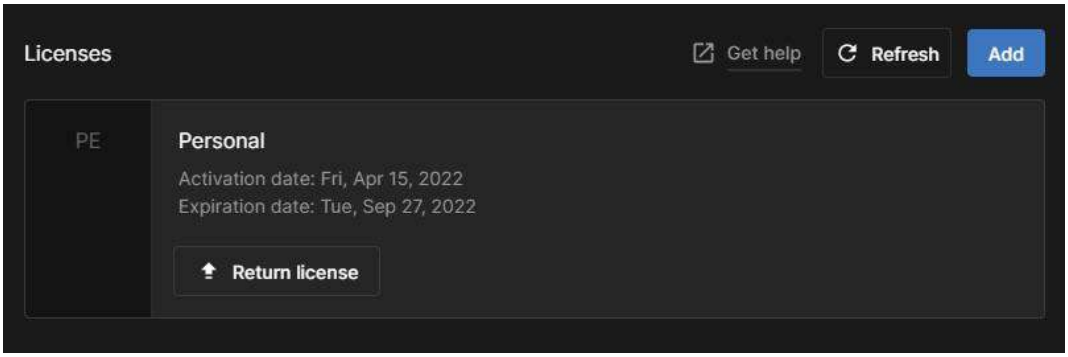

Figure 3.9 License manager

- A “package” is a container that holds our all types of features or assets such as:
  - Editor tools and libraries
  - Runtime tools and libraries
  - Asset collections
  - Project templates to share common project

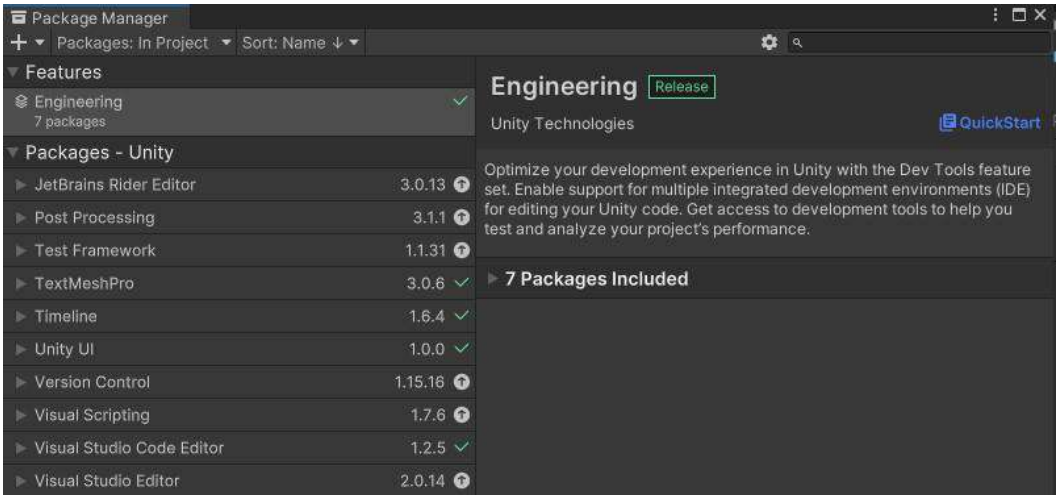

Figure 3.10 Package manager

- Excavator terrain meshes contain data for working, a statistic that describe a shape. Unity uses meshes in the following ways:

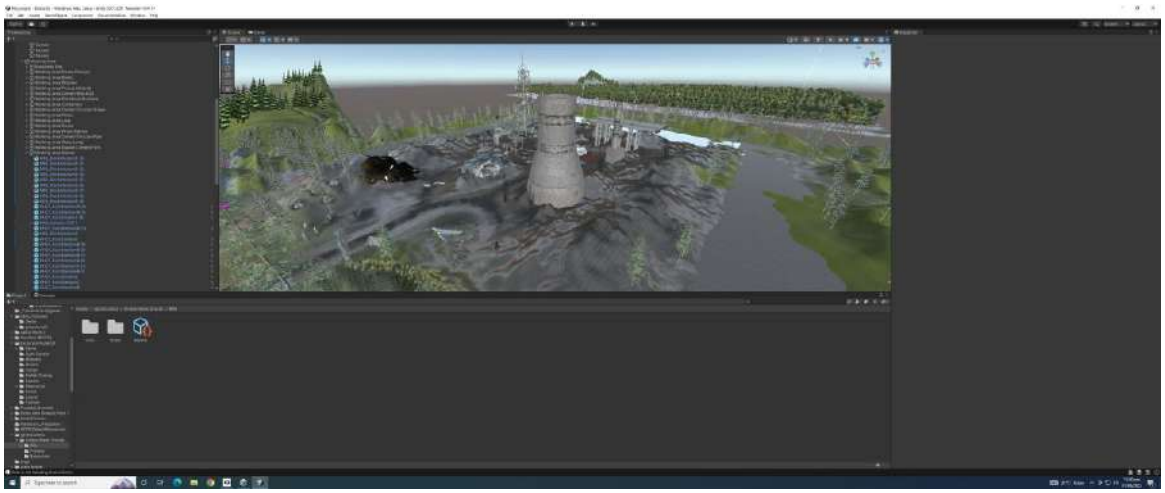

Figure 3.11 Meshes

- When using meshes in C# scripts, the Mesh category provides scripting access to an object's mesh geometry like for steering wheel the code given bellow.

```
if (mScriptCont.excavatorOn_Bool == true)
{
    m_Horizontal = Input.GetAxis("Horizontal");
    m_Vertical = Input.GetAxis("Vertical");
    Debug.Log(m_Vertical);
    if (LogitechSteeringWheel.PadelY < 0)
    {
        m_Vertical = 1;
    }
    if (LogitechSteeringWheel.PadelZ < 0)
    {
        m_Vertical = -1;
    }
```

Figure 3.12 C# Script

- Textures are just generic bitmap pictures used on mesh surfaces. we can assume it's a photograph of a texture, printed on a sheet of rubber that is stretched and fixed to the mesh as shown in Fig. 3.12 samples are given which we used on our terrain and assets.

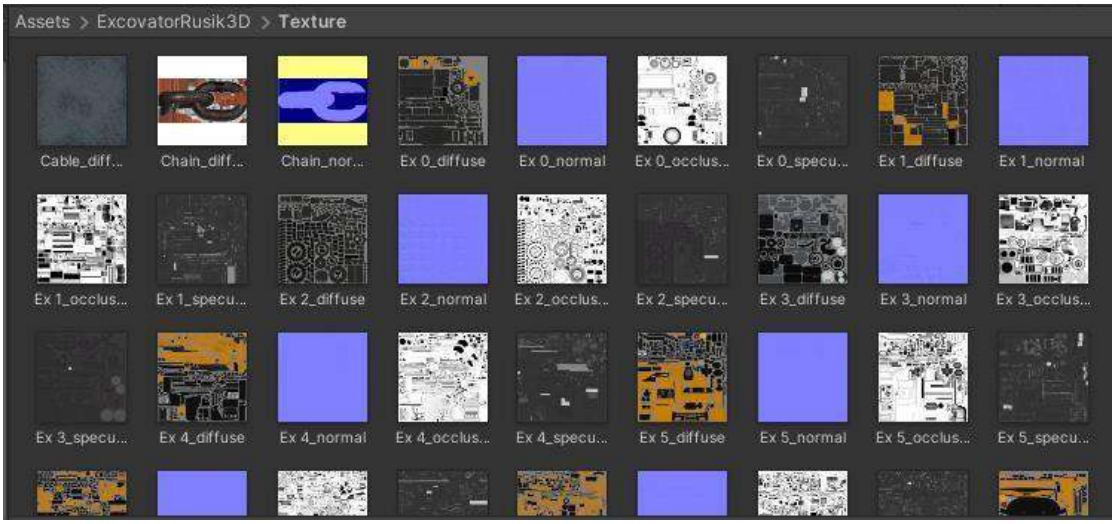

Figure 3.13 Textures

- In Excavator Simulation, inflexible bodies enable physics-based behavior such as motion, gravity, and collisions which are implemented on the all the rigid bodies as shown in Fig. 3.13 and Colliders are used to set up a collision between all these GameObjects to combine them all as shown in figure Fig. 3.14.

Figure 3.14 Rigid bodies

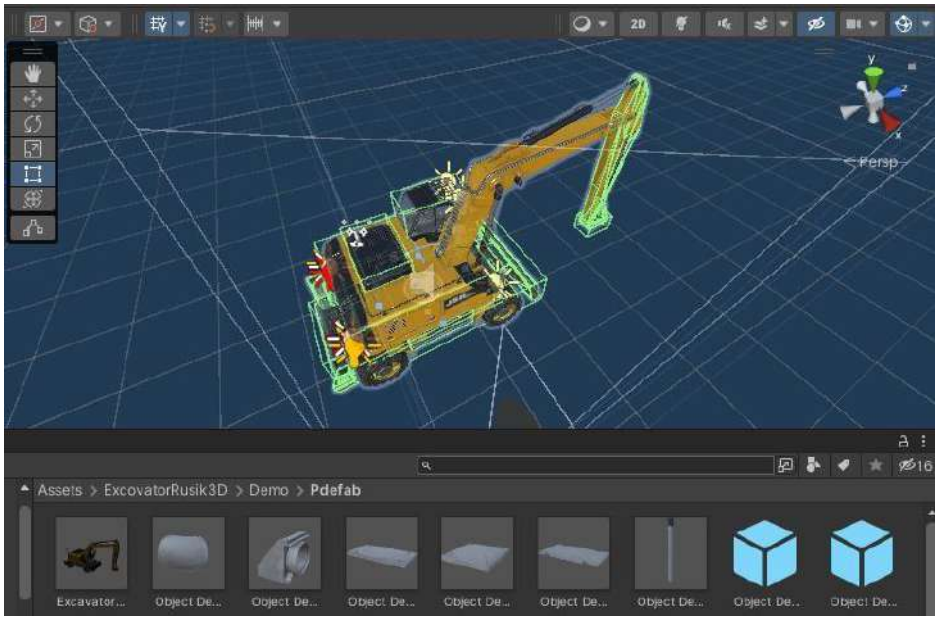

Figure 3.15 Collision

3.5 Motion Base

3.5.1 Introduction:

Motion simulators are specialized devices or systems designed to recreate the sense of movement and motion experienced in real-world scenarios or virtual environments. They play a crucial role in various industries, including aviation, automotive, entertainment, and training, by providing users with an immersive and realistic experience. These simulators are equipped with mechanisms that allow them to move along different axes, known as degrees of freedom (DOF), which define their capabilities and the complexity of motion they can generate.

3.5.2 Degrees of Freedom (DOF):

Degrees of freedom refer to the number of independent axes along which the simulator can move. A higher number of DOF enables the simulator to simulate more complex and realistic motions, enhancing the user experience. Here are the common types of motion simulators based on DOF:

1. **1-DOF Motion Simulator:** A 1-DOF motion simulator provides motion along a single axis. It typically allows back and forth or up and down movements, making it the simplest form of a motion simulator. These simulators are commonly found in basic amusement park rides, where the motion is limited to a single direction.
2. **2-DOF Motion Simulator:** A 2-DOF motion simulator incorporates two independent axes of motion. This type of simulator can offer pitch (forward and backward) and roll (side-to-side) movements. It is commonly used in arcade games and some flight simulators, providing users with a more engaging experience compared to 1-DOF systems.
3. **3-DOF Motion Simulator:** A 3-DOF motion simulator further enhances the user experience by adding an additional axis of movement. In addition to pitch and

roll, it also allows for yaw (rotation around a vertical axis). These simulators are widely used in advanced flight simulators, driving simulators, and virtual reality setups, offering users a more realistic sensation of movement.

4. **6-DOF Motion Simulator:** A 6-DOF motion simulator represents the most sophisticated type commonly used in professional flight training and aerospace research. With six independent axes of movement, including pitch, roll, yaw, surge (forward and backward), sway (side-to-side), and heave (up and down), these simulators provide a highly immersive and authentic experience.
5. **9-DOF Motion Simulator:** Some specialized applications require even more freedom of movement. A 9-DOF motion simulator goes beyond the six-axis system by incorporating rotational movement along the three perpendicular axes (X, Y, Z). This level of complexity is typically used in advanced research and development environments.

The use of motion simulators is widespread in various industries to provide users with realistic and immersive experiences of movement and sensation.

The degree of freedom used in a motion simulator is an important factor that determines how realistic and complex the simulated motions are. Why?

A variety of simulators, ranging from basic 1-DOF systems to complex 6-DOAF and 9-DO F systems, are available for various applications and training requirements .The motion simulator chosen will be influenced by the project or application's specific requirements, budget, and desired degree of realism.

To demonstrate our ingenuity and creativity, we created a 2-Dof Budget Simulator to demonstrate the ability of our project team to deliver exceptional results within their limited budget.

Our team took on the challenge and created a cost-effective yet highly sophisticated solution that not only meets project specifications but also demonstrates our commitment to excellence in all aspects. The 2-Dof Budget Simulator was created with innovative ideas in mind, taking into account the value and efficiency of each component before attempting to create a product that could compete directly with its production. We worked in tandem to develop the most efficient version, leveraging existing technologies and methodologies while also preserving the best possible quality of the product .Our design process was centered on making the simulator user-friendly and intuitive, while also providing seamless navigation and interaction. We adhered to best practices in the industry to ensure reliability and accuracy, providing stakeholders with accurate budget estimates and insightful analysis.

The budgetary challenges were undoubtedly challenging, but they also motivated us to explore new opportunities.

3.5.2 Hardware Components and its Working:

Below is an explanation of the components used in our 2-DOF (Degree of Freedom) simulator:

1. Metal Frame:

The solid metal frame is the foundation of our 2-DOF simulator.
By providing a strong foundation, it ensures that the components are supported and remai ns stable enough to allow users to participate in simulation activities without harm**.**
With careful consideration to its geometry, the frame provides a level of stability that is n ecessary for achieving believable and immersive simulation experiences.

2.U-Joint:

A Universal Joint, also known as Universal Joint, is utilized to transfer motion from one r otating shaft to another at different angles.
Our simulator's allows for precise articulation of the two degrees of freedom, making it p ossible to replicate real-world movements. This is particularly useful in this context.
By using this versatile component, the simulator can mimic intricate movements with pre cision, resulting in an authentic and engaging simulation experience.

3. WindShield Wiper Motor:

The Windshield Wiper Motor is a critical component of our 2-DOF simulator, acting as an actuator to generate the required motion for various dynamic scena rios.With its strong and stable performance, it is the perfect fit for providing smooth and lifelike movements.
The motor's controlled rotation, along with our advanced programming, creates a sense of immersion in the simulated environment, which enhances the simulation realism.

4. IBT-2 Motor Driver:

The integration of an IBT-2 Motor Driver as an interface between the Arduino controller and the Windshield Wiper Motor marks a significant advancement in our 2-DOF simulator's capabilities. This dedicated component serves as a crucial intermediary,

empowering us to finely control the motor's speed and direction with unparalleled accuracy.

With the IBT-2 Motor Driver in place, we gain precise command over the Windshield Wiper Motor, allowing us to orchestrate a diverse array of simulator motions. Users can now experience a wide range of sensations, from gentle turns to more dynamic and thrilling movements, all intricately synchronized with the simulation's environment.

The IBT-2 Motor Driver's ability to manage speed and direction ensures that each motion is executed smoothly and seamlessly, enhancing the overall realism of the simulation. Its responsive control enables us to create immersive scenarios that closely mirror real-life experiences, making the simulation more captivating and engaging for users.

Additionally, the inclusion of this specialized driver contributes to the simulator's safety and reliability. With such precise control, we can carefully regulate motion parameters and prevent any sudden, jarring movements that might compromise user comfort or safety.

In summary, the integration of the IBT-2 Motor Driver as an interface between the Arduino controller and the Windshield Wiper Motor elevates our 2-DOF simulator to new heights. By harnessing the driver's precision control capabilities, we can offer users an enhanced range of sensations, ensuring that their simulation experience feels authentic, exhilarating, and safe. This advancement represents a significant stride forward in delivering a truly immersive and captivating simulation environment.

5. 12V 20 amp DC Power Supply:

The motor of the simulator requires a consistent power supply that can be powered.

A 12V 20Amp DC Power Supply is the preferred choice for powering the Windshield Wiper Motor and other electronic components. This device requires no additional equipment.

With its robust performance in both high-pressure environments and large current capacity, it can handle the demanding simulation workloads that come with our simulator.

6. Arduino:

Our 2-DOF simulator's central component is the Arduino microcontroller, which functions as the brain by analyzing data and carrying out control algorithms.

By utilizing this versatile and programmable board, we can incorporate user inputs into the simulation, sensor feedback, and motion calculations to control the simulator's movements.

Our users can benefit from our user-friendly simulation models, which are crafted using the Arduino code and have unique features.

3.5 Summary

This chapter contains all the mediatory initial steps and information needed to work on the project. This information related to software and its working is explained also, hardware modules their connectivity steps are explained in detail that how can we connect this hardware with each other and the PC with the inbuild software's Logitech SDKs.

Chapter 4

TOOLS AND TECHNIQUES

4.1 Technical Specifications of Hardware

Figure 4.1 Simulator Controls

Following hardware had been used for developing VR Based Excavator Training Simulator:

- Logitech 3D joysticks
- Logitech Steering Wheel
- Logitech pedal set

4.1.1 Logitech Steering Wheel

We are controlling the steering wheel in the horizontal direction. Which only can turn left and right.

4.1.2 Logitech Pedal Set

The Pedal Set consists of three pedals that control the vertical movements of the excavators. From the leftmost pedal, we are controlling the speed/ race of the excavator, which helps our excavator to move only in the forward direction. From the middle pedal, we are controlling the brakes or reverse of the excavator, which helps our excavator to move only in the backward direction.

4.1.3 Logitech Joysticks

A joystick is an input device that can be used for controlling the movement of the cursor or a printer in a computer device. Joysticks have eight movements we can say it has four degrees of freedom(joints).

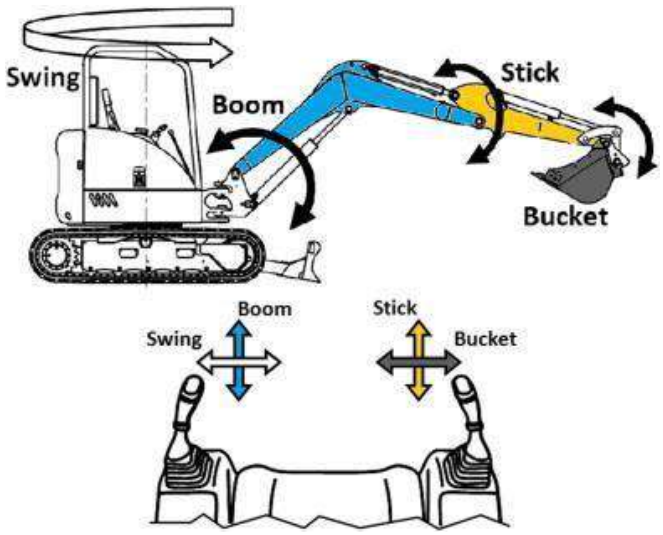

Figure 4.2 Joystick Movements

4.1.3.1 Right Joystick

If we move the right joystick in the left and right direction it will move the bucket of the excavator in the software. If we move the right joystick in the up and down direction it will move the stick of the excavator in the software.

4.1.3.2 Left Joystick

If we move the left joystick in the up and down direction it will move the boom of the excavator in the software. If we move the left joystick in the left and right direction it will move the swing of the excavator in the software.

4.2 Calibration of Hardware

2 calibration software have been used for the testing of controls and for assigning the function of buttons or movements that have to be performed through scripting

4.2.1 Steering Wheel SDK Demo Program

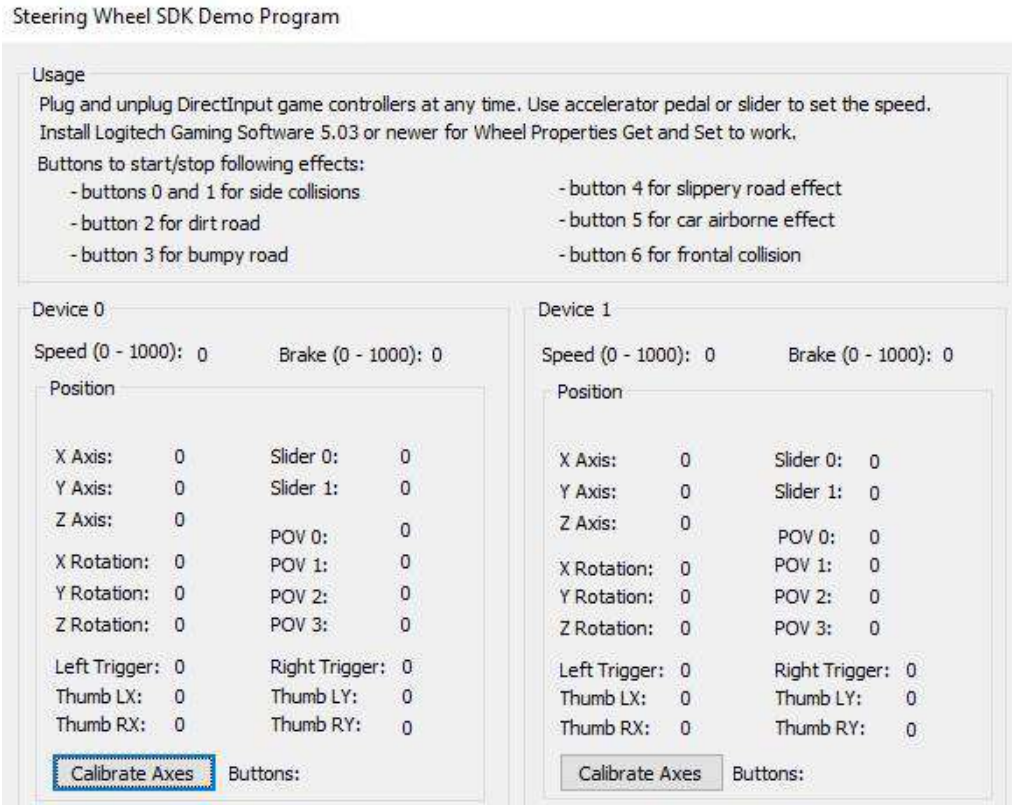

Figure 4.3 Steering wheel SDK Demo

4.2.2 Logitech SDK

The Unity 3Dpackage enables control of Logitech Gaming products. The Logitech Gaming SDK is divided into five categories:

- **Logitech|G Arx Control App SDK**

Easily create a second screen experience for the PC Game through an applet running on Logitech|G Arx Control app.

- **Logitech|g LED Backlighting**

Access backlighting features of Logitech|G devices, now including also per-key color backlighting on featured devices.

- **Logitech|G G-key SDK**

Receive notifications on G-keys events coming from Logitech|G featured mice, headsets, and keyboards.

- **Logitech|G LCD SDK**
- **Logitech|G Steering Wheel SDK**

Control Logitech Gaming controllers, such as steering wheels, flight controllers or gamepads.

Figure 4.4 Logitech Joystick SDK

4.3 Software Specifications

Software named “Unity 3D” on this software we developed terrain and made changes accordingly. Then we imported the asset of the excavator and did its coding of movements. The collision was also done between different objects. Like:

4.3.1 Camera Shifting

Using 3 different camera settings for driving

- **Excavator the main view**

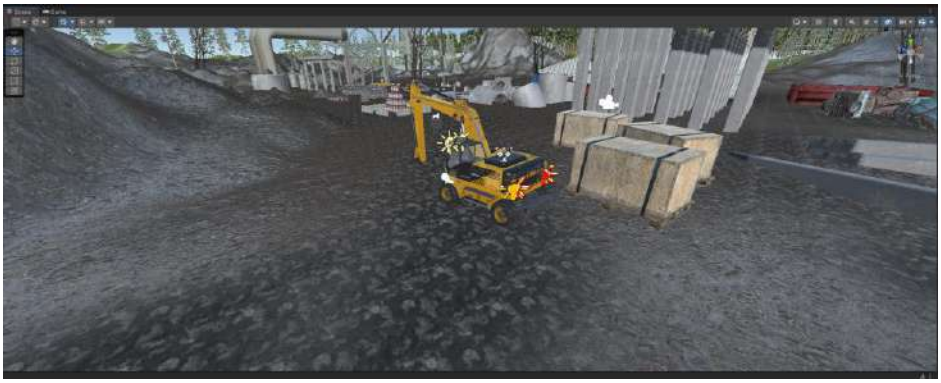

Figure 4.5 Simulator Main View

- **Zoom view**

Figure 4.6 Simulator Zoom View

- **Driver view**

Figure 4.7 Simulator Driver View

4.3.2 Excavator Controls

- **Engine**

Pressing the Y button on the keyboard enables all the controls by starting the engine of the excavator. When the power button is pressed it starts the engine and we can control these attributes shown in figure 4.8.

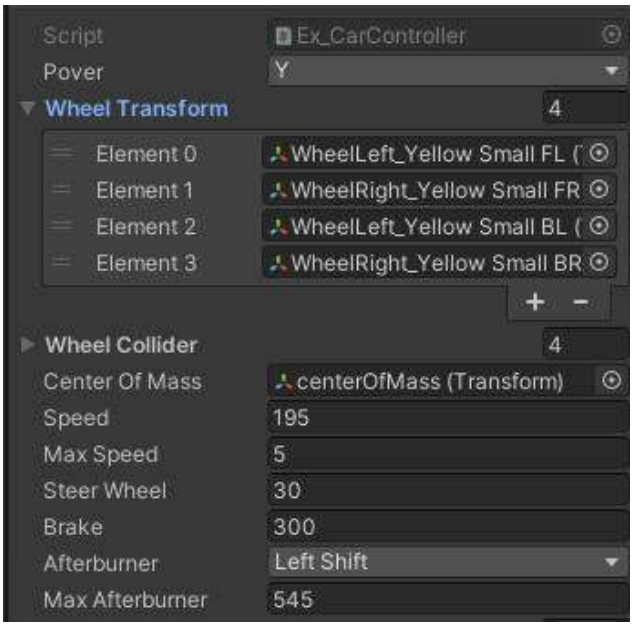

Figure 4.8 Engine Power Settings

- **Sounds**

When the engine starts, I produce the real excavator sound of the Engine, indicators, start, stop, and hydraulic forces.

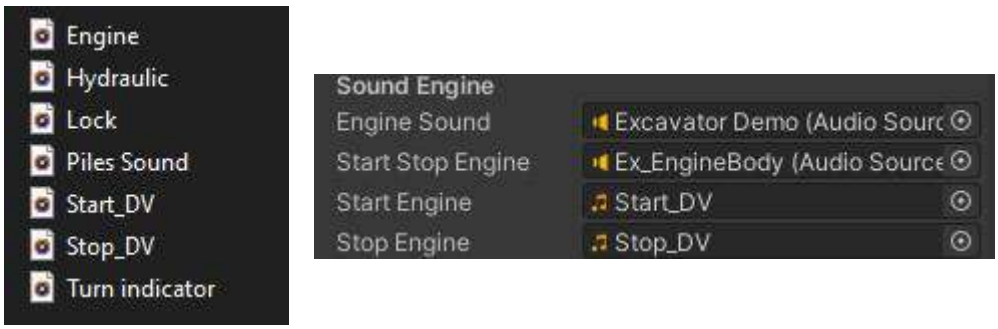

Figure 4.9 Excavator Sounds

- **Smoke**

We have added the smoke to the excavator. When the engine starts the excavator starts producing smoke also, we can control the particle limits, playback speed, playback time, and speed range.

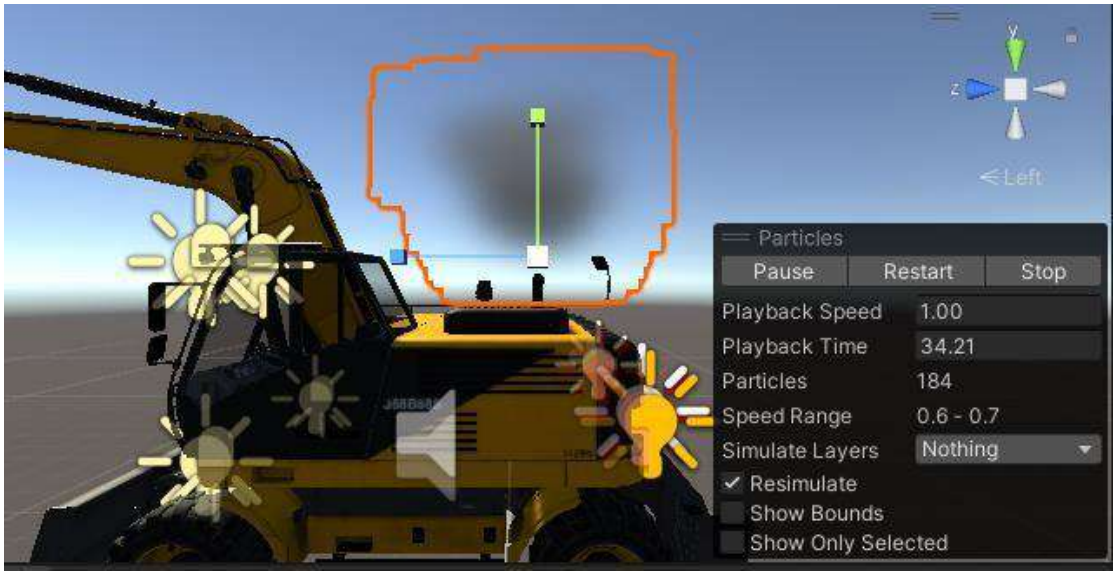

Figure 4.10 Excavator Engine Smoke

- **Indicators and Headlights**

Headlights and Indicators were added which can be controlled through a keyboard with buttons like Q for the left indicator and E for the right indicator

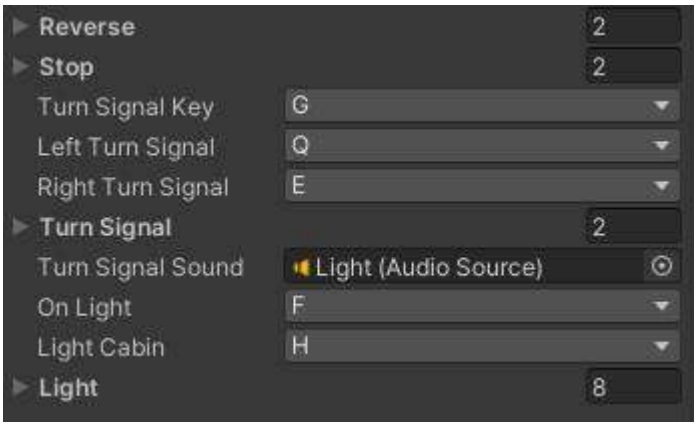

Figure 4.11 Indicator and headlight Settings

Figure 4.12 Excavator Cabin Light

Figure 4.13 Excavator Cabin Front View

Figure 4.14 Excavator Indicators

4.3.3 Excavators Arm Controls

4.3.3.1 Excavators Arm

The excavator's arm controls the four Degree of Freedom (DOF) means four joints or 8 movements:

4 DOF	8 Movement
Boom	Boom Up
	Boom Down
Swing	Swing Left
	Swing Right
Stick	Stick Up
	Stick Down
Bucket	Bucket Open
	Bucket Close

- **Swing**

Figure 4.15 Arm Swing

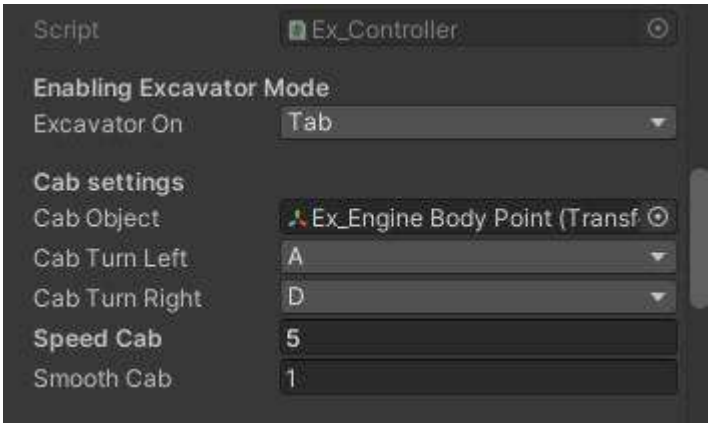

Figure 4.16 Enabling Excavator Cab Settings

- **Boom**

Figure 4.17 Arm Boom

- **Stick**

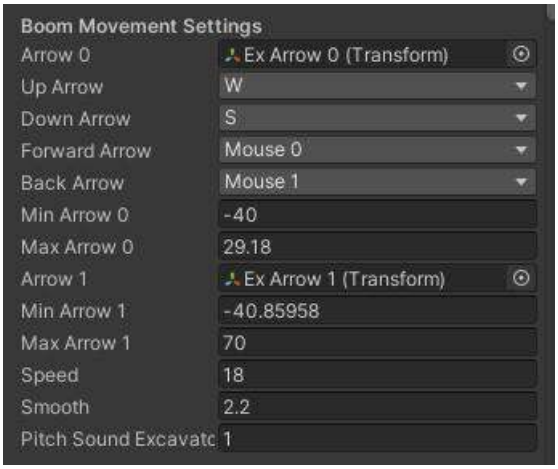

Figure 4.18 Boom and Stick Movements Settings

Figure 4.19 Arm Stick

- **Bucket**

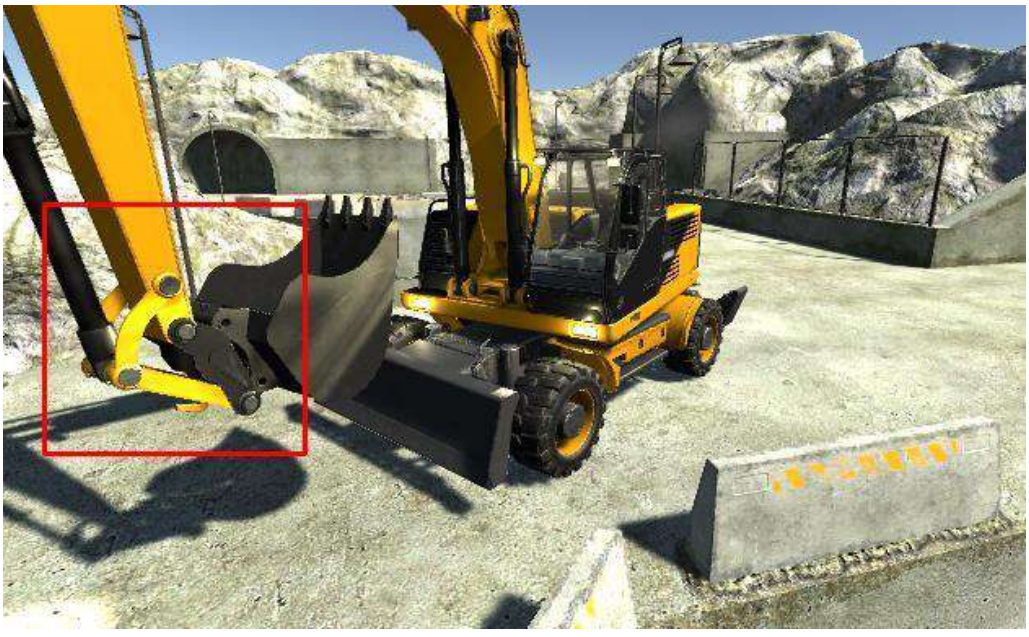

Figure 4.20 Arm Bucket

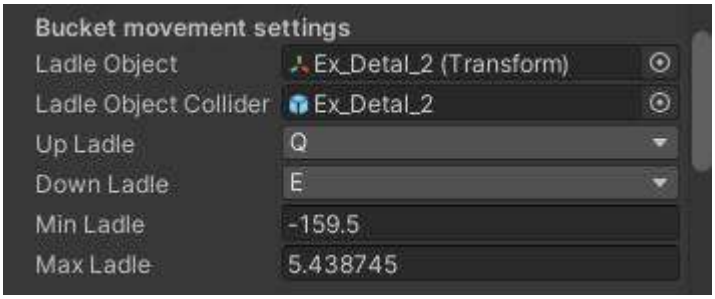

Figure 4.21 Bucket Movements Settings

4.3.3.2 Excavation

In this part we wanted our excavator to pick the stones from the ground which can be placed or dropped in any other place by controlling all 4 joints' degrees of freedom.

Figure 4.22 Stone Excavation

Figure 4.23 Stone rigid body and Colliders

4.3.4 Integrating Oculus Rift S with Unity:

1. **Set up Oculus Rift S and Software:**

   Begin by ensuring that you have the Oculus Rift S headset and controllers properly set up and configured on your PC. This includes installing the Oculus software, which you can download from the official Oculus website.

2. **Install Oculus Integration for Unity:**

   Open your Unity project and navigate to the "Window" menu. From there, access the "Package Manager" and click on the "+" button to add a package from a Git URL. Enter the URL for the Oculus Integration package, which is available on GitHub. This will initiate the download and installation process.

3. **Enable VR Support in Unity:**

   To enable VR support in Unity, go to "Edit" > "Project Settings" > "Player." Under "XR Settings," make sure to check the "Virtual Reality Supported" option. Then, click on the "+" button and add "Oculus" as a Virtual Reality SDK. Prioritize Oculus at the top of the list to make it the default VR platform for your project.

4. **Set Up Camera Rig:**

   In your Unity scene hierarchy, remove the Main Camera if it exists. Instead, use the "OVRCameraRig" prefab from the Oculus Integration package. This prefab contains camera and controller components optimized for Oculus devices.

5. **Implement VR Interactions:**

For interactive elements in your VR experience, you can use the "OVRGrabbable" component. Attach this component to objects you want users to interact with and grab in the virtual environment. For hand tracking and interactions, utilize the "OVRHand" components for both left and right controllers.

6. **Test in VR:**

   Before proceeding further, ensure your Oculus Rift S is connected to your PC. Make sure your Unity project is configured correctly for VR, with Oculus selected as the active VR SDK. Click on the "Play" button in Unity to test your project in VR mode. Once you wear your Oculus Rift S headset, your Unity project will be displayed in VR, and you can interact with the virtual world.

7. **Build and Deploy:**

   Once your VR application is ready for deployment, access "File" > "Build Settings" in Unity. Select the desired platform, such as Windows PC. Click on "Build and Run" to generate the executable file and automatically launch it on your Oculus Rift S headset.

4.3.4 Hardware and Software Connection:

To establish a SerialPort between Unity and Arduino.The SerialPort.IO library requires us to establish a serial connection between the two devices.
SerialPort.IO library facilitates communication with the serial port on the computer runni ng Unity, which permits data transfer to and from the connected Arduino board.

The procedure for setting up communication is as follows:

- **Install SerialPort.IO Library**
- **Setting Up the Arduino**

The process of setting up the Arduino board involves connecting it to your computer via a USB cable.

Upload a sketch to the Arduino IDE, which can read data from Unity using the serial port and perform necessary actions (e.g.).

4.4 Summary

All software and hardware implementation explained in this chapter where we have clearly explained the function of all joints, smoke, sound, effects, engine, headlights, and indicators to control them also mentioned the assignment of variables of the controls in the figures.

Chapter 5

CONCLUSIONS

In this chapter, we explained all our outcomes and results as our goal to achieve at the end of the project.

5.1 Results and Discussions

A description of the results of the project with hardware and software working.

5.1.1 Software Results

- **Excavator Model**

The output of combining all the small parts of excavators to build up a whole working excavator with all accurate joints.

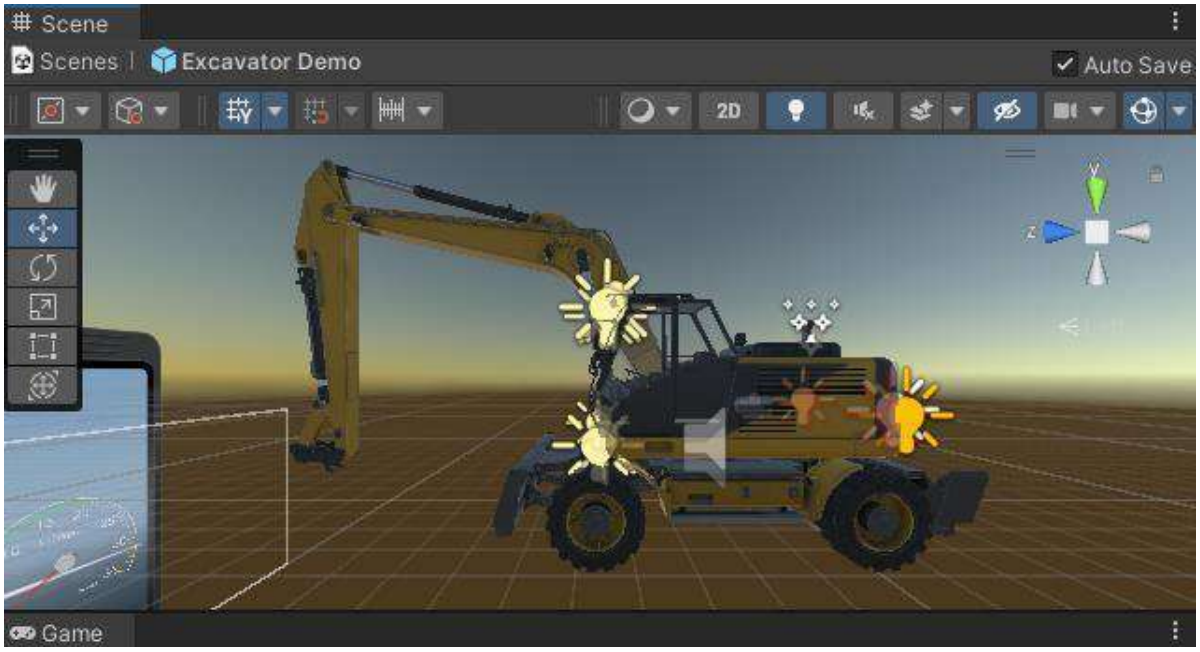

Figure 5.1 Excavator Model

- **Wheel Colliders**

Applied successfully wheel colliders to our vehicle and terrain so that the vehicle can drive on the ground when interacting.

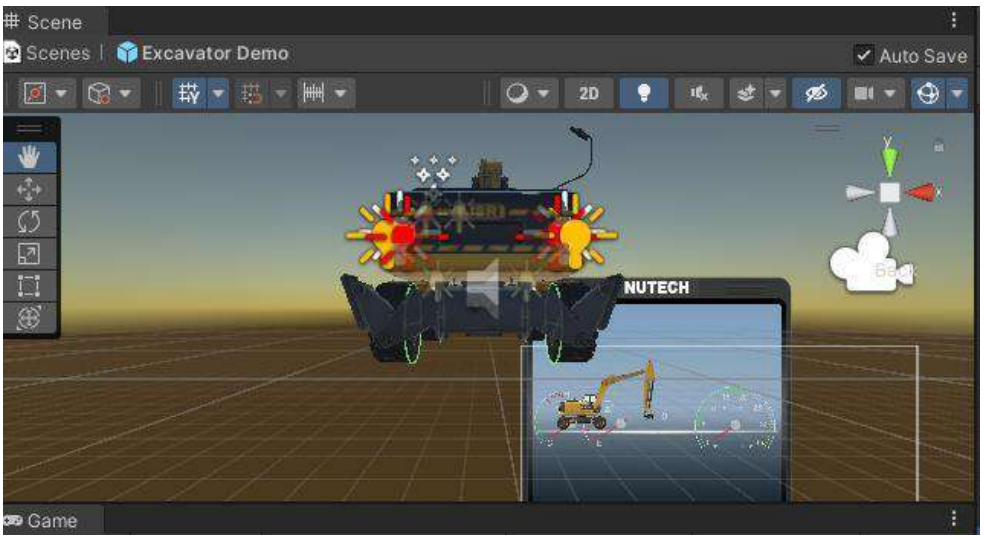

Figure 5.2 Wheel Colliders

- **Indicator and Headlights**

Added indicators lights and headlights which are currently controlled by a keyboard to turn on a headlight press ‘h’ and for indicators press ‘q’ for the left indicator and ‘e’ for the right indicator.

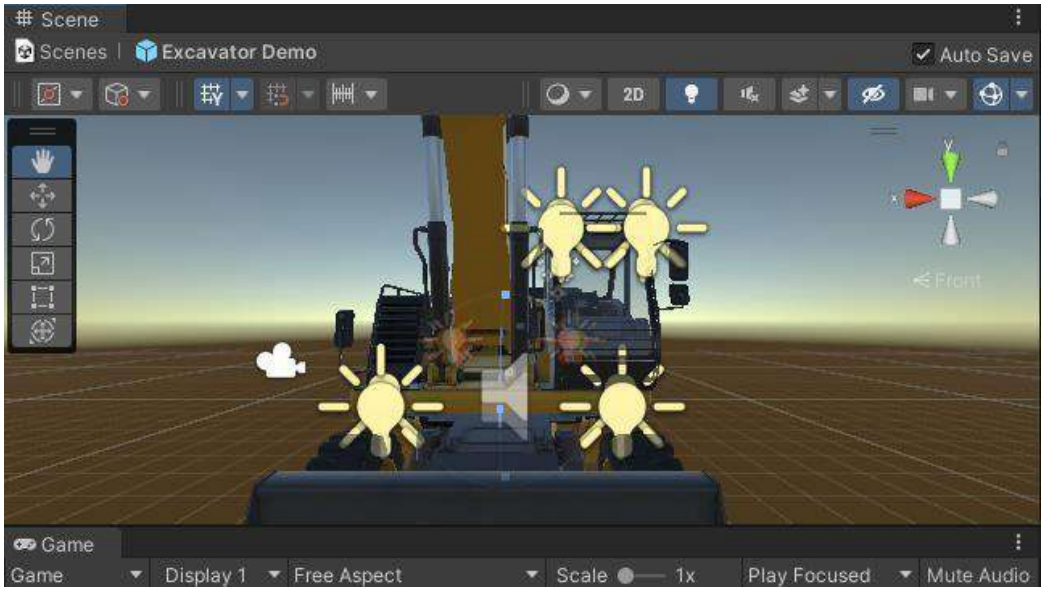

Figure 5.3 Front Indicator and headlights

Figure 5.4 Back Indicator and brake lights

- **Center of Gravity**

Added center of gravity to the vehicle to drive smoothly while driving or picking stone, its code is attached in car control’s part.

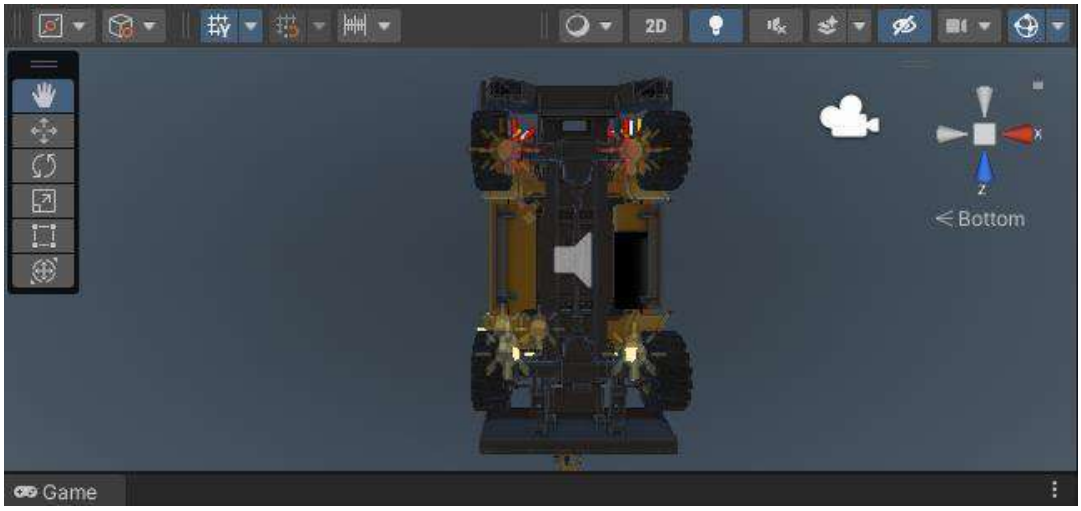

Figure 5.5 Center of Gravity

- **Terrain Model**

The colliders also applied on meshed terrain so the vehicle can drive on it and add excavator's scenes to it

Figure 5.6 Terrain Model

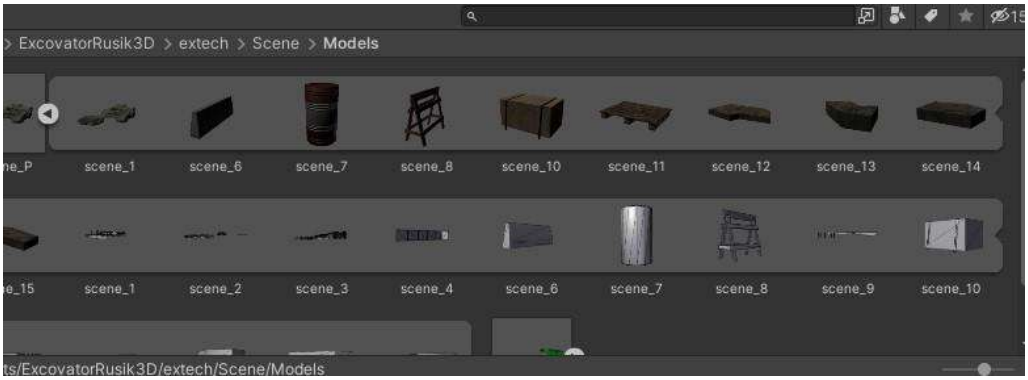

Figure 5.7 Terrain Scene Model

- **Excavator Canvas**

Currently working on the dashboard where we can see fuel level and arm motions angle for better accuracy.

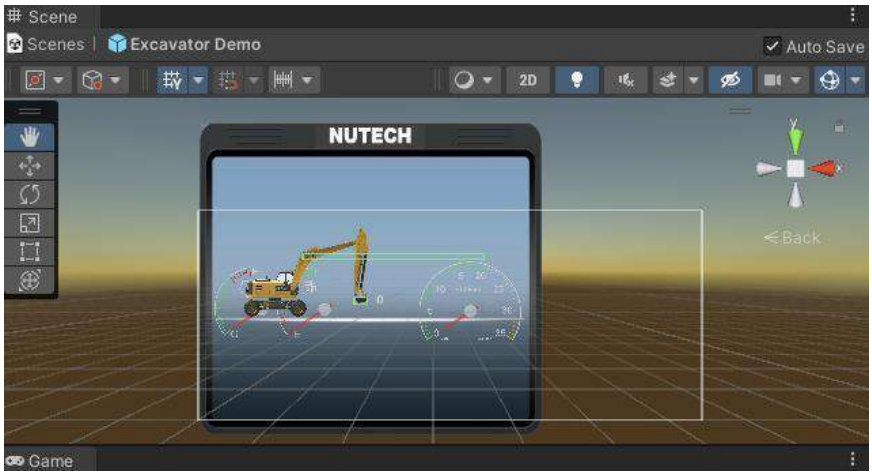

Figure 5.8 Excavator canvas model

5.1.2 Hardware results

In hardware, we are used

Logitech 3D joysticks

Logitech Steering Wheel

Logitech pedal set

Motion Base

Oculus VR Headset

Figure 5.9 Hardware Output

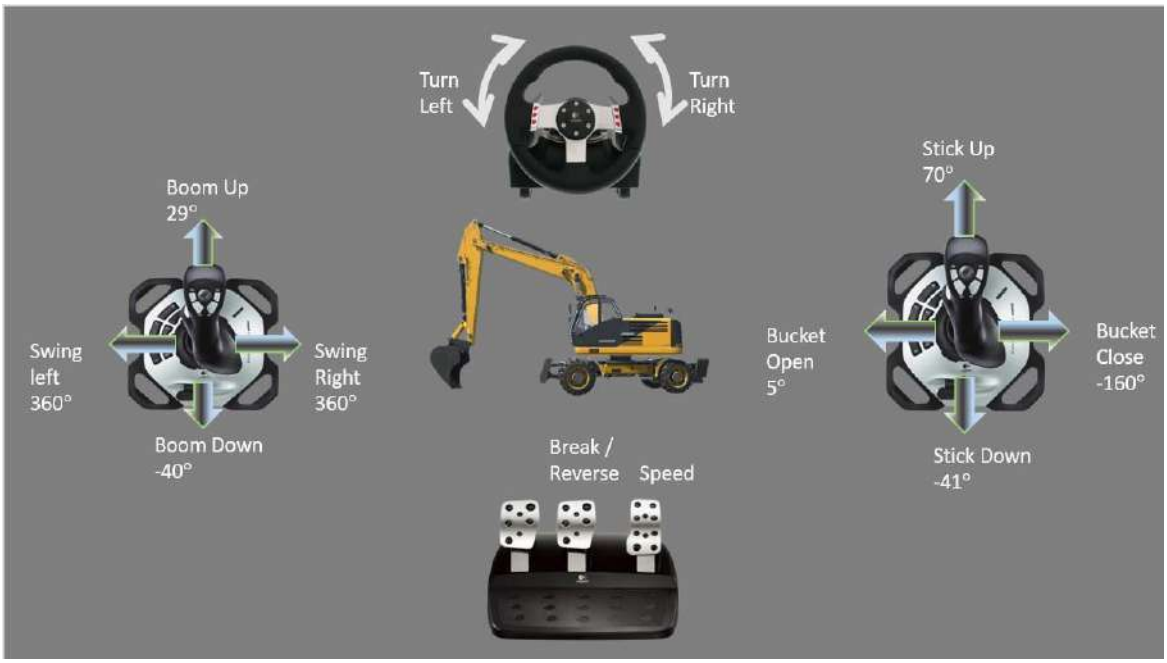

Figure 5.10 Simulator Controls and Range of Angles

- **Steering Wheel**

We are controlling the steering wheel in the horizontal direction.

- **Pedal Set**

From the leftmost pedal, we are controlling the speed/ race of the excavator, which helps our excavator to move only in the forward direction. From the middle pedal, we are controlling the brakes or reverse of the excavator, which helps our excavator to move only in the backward direction.

- **Right Joystick**

If we move the right joystick in the left and right direction it will move the bucket of the excavator in the software. The bucket will open up to 5 degrees when the joystick is moved towards the left. The bucket will close up to -160 degrees when the joystick is moved towards the right.

Right Joystick Joints	**Angles**
Stick Up	70□
Stick Down	-41□
Bucket Open	5□

Bucket Close	-160□

Figure 5.11 Right Joystick Angles

If we move the right joystick in the up and down direction it will move the stick of the excavator in the software. The stick will lift to 70 degrees when the joystick is moved up. The stick will down, up to -41 degrees when the joystick is moved down.

- **Left Joystick**

If we move the left joystick in the up and down direction it will move the boom of the excavator in the software. The boom will lift to 29 degrees when the joystick is moved up. The boom will down, up to -40 degrees when the joystick is moved down.

If we move the left joystick in the left and right direction it will move the swing of the excavator in the software. The swing will move the whole 360 degrees and along this, the upper part of the excavator including its cabin will also move up to 360.

Left Joystick Joints	**Angles**
Boom Up	29□
Boom Down	-40□
Swing Left	360□
Swing Right	360□

Figure 5.12 Left Joystick Angles

- **VR Headset**

  In the VR headset, the left joystick's up and down movements controls the excavator's boom. Leaning the joystick upwards will elevate the boom to a 29-degree, while leaning it downwards will lower the boom to -40 degrees.

  Additionally, the left joystick's left and right movement measures the swing of the excavator. This swing offers a complete 360-degree rotation, and as the upper part of the excavator, including its cabin, moves along, it will also have a range of motion up to 360 degrees.

  These user-friendly controls allow for an immersive and authentic construction site experience, enabling users to operate the virtual excavator intuitively and realistically as if they were sitting inside the actual cabin.

- **Motion Base**

  With the VR headset and simulation from the software the motion base gives the movement. When the position in the excavator changes it tilts the motion base seat accordingly giving the immersive feel to the operator.

5.2 Limitations

Limitations may include the different controls for different excavators. It may be possible the excavators of the different companies have different controls that are developed in the software for simulation that may affect the working. As this is a wheeled excavator it may perform unevenly on terrain.

5.2.1 Using Leap Motion Controller

This study uses an excavator simulator under development that can be controlled using the Leap Motion Controller. A real hand appears as his 3D hand model, moving in real-time with the movement of the real hand. These 3D hand models take control of the system by grabbing the virtual joysticks and levers of the virtual excavator [6].

The investigation is required to obtain user acceptance and controller satisfaction results. Survey respondents for this system are 25 excavator operators. After research, no statement has an average score to achieve a value that agrees and satisfies. The average score of the user awareness survey results is 3.44 points for "the camera moves according to the movement of the head". The highest average score in the user satisfaction survey is "I am satisfied with the joystick function that moves the excavator", with an average score of 3.16 points. New controls that are more precise and easier to learn are needed to develop excavator controls to achieve better results [5].

5.3 Recommendations

One can work and convert limitations into future work which we were not able to do due to a shortage of time.

Also, they can change the terrain and make it according to the background they want because the asset part will be almost the same but if they want to add something else like different types of buckets, etc., they can go for it too.

5.4 Summary

The main objective of our project was to achieve the working simulation of excavator controlling with actual hardware of any company (JSR company). Explained all the outcomes of software and hardware with one-by-one module screenshots are also attached.

CONCLUSION

In this Project, a VR based excavator training simulator has been developed is presented for developing a new type of construction heavy equipment training Earthmoving site Trainee-controlled Excavator simulators based on the integration of the actual site data and a multi-agent system in an immersive VR environment. The presented Working model and discussed project are good indications of the feasibility of combining actual spatial models in a training VR environment. Nevertheless, the proposed simulator is expected to offer the following advantages:

- Representing scenarios based on the model of actual complex construction sites through which the workers can familiarize themselves with the site.
- Providing scenarios with several pieces of equipment operated by several trainees simultaneously.
- Capturing the realistic behavior of the surrounding equipment, where trainees can be exposed to safety-specific education [7].

There are some limitations in the presented Project:

First, due to the lack of relevant actual site data about equipment movement. Second, the necessary parameters for feedback on safety and productivity performances are not incorporated in the presented prototype.

Finally, the case study was only tested for a single trainee. Based on the above limitations, the future work of this research will pursue the following:

- Collecting data of equipment tracking from actual sites that can be simulated in parallel.
- Improving the prototype to provide necessary feedback on safety and productivity performances.
- Conducting a comprehensive case study with multiple users [9].

REFERENCES

[1] Boyanovsky, H., Imagining the future for hydraulic excavators, SAE OHE 100:Future Look, pp. 77.

[2] Zubko, N., Heavy rotation - a new generation of hydraulic excavators: better, stronger, faster, Utility Contractor, pp. 19–23, 2007.

[3] Roth, M., 2010, Earthmoving's new frontier, Rental Equipment Register, Available:http://rermag.com/trends_analysis/interviews/rer-interviewsearthmoving-equipment-manufacturers-20100401/index2.html,9 February, 2011.

[4] Engel, S., Alda, W., and Krzysztof, B., Real-time computer simulator of hydraulic excavator, Proceedings of the 7th Conference on Computer Methods and Systems, Krakow, Poland, pp. 357-362, 2009.

[5] Fukaya, K., and Umezaki, S., 2002, "Development of Excavator Simulator Using Virtual Reality Technology," Virtual Reality Society of Japan Annual Conference.

[6] Akyeampong J , Udoka S and Park E 2012 A Hydraulic Excavator Augmented Reality Simulator for Operator Training Proceedings of the 2012 International Conference on Industrial Engineering and Operations Management Istanbul, Turkey, July 3 – 6

[7] Haddock K 2007 The Earthmover Encyclopedia Motorbooks-MBI Publishing Company: Minneapolis, Minnesota, USA

[8] Nainggolan F L , Siregar B and Fahmi F 2016 Anatomy learning system on human skeleton using Leap Motion Controller In Computer and Information Sciences (ICCOINS), 2016 3rd International Conference on (pp 465-470) IEEE

[9] Nowicki M , Pilarczyk O , Wąsikowski J and Zjawin K 2014 Gesture Recognition Library for Leap Motion controller Poznan University of Technology: Poznań

[10] Tao N , DingXuan Z , Yamada H and Shui N 2008 A low-cost solution for excavator simulation with realistic visual effect In Robotics, Automation and Mechatronics, 2008 IEEE Conference on (pp 889-894) IEEE

APPENDICES

APPENDIX A

Logitech|G Steering Wheel SDK axis's

```
public struct DIJOYSTATE2ENGINES
   {       public int lX;                  /* x-axis position              */
public int lY;                  /* y-axis position              */
public int lZ;                  /* z-axis position              */        public
```

```
int lRx;                /* x-axis rotation            */      public int
lRy;                /* y-axis rotation            */      public int lRz;
/* z-axis rotation            */
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 2)]
public int[] rglSlider;            /* extra axes positions       */
[MarshalAs(UnmanagedType.ByValArray, SizeConst = 4)]
    public uint[] rgdwPOV;                  /* POV directions             */
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 128)]
    public byte[] rgbButtons;               /* 128 buttons                */
public int lVX;                 /* x-axis velocity            */      public
int lVY;                /* y-axis velocity            */      public int lVZ;
/* z-axis velocity            */      public int lVRx;                /* x-axis
angular velocity     */      public int lVRy;                /* y-axis
angular velocity     */      public int lVRz;                /* z-axis
angular velocity     */
[MarshalAs(UnmanagedType.ByValArray, SizeConst = 2)]
    public int[] rglVSlider;           /* extra axes velocities      */
public int lAX;                 /* x-axis acceleration        */
public int lAY;                 /* y-axis acceleration        */
public int lAZ;                 /* z-axis acceleration        */      public
int lARx;               /* x-axis angular acceleration  */      public
int lARy;               /* y-axis angular acceleration  */      public
int lARz;               /* z-axis angular acceleration  */
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 2)]
    public int[] rglASlider;           /* extra axes accelerations   */
public int lFX;                 /* x-axis force               */      public
int lFY;                /* y-axis force               */      public int lFZ;
/* z-axis force               */      public int lFRx;                /* x-
axis torque            */      public int lFRy;                /* y-axis
torque             */      public int lFRz;                /* z-axis torque
*/
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 2)]
```

APPENDIX B Toggle

Camera Code

```
using System.Collections; using
System.Collections.Generic;
using UnityEngine;

public class Ex_Camera : MonoBehaviour
{
```

```
   //   public KeyCode menyHover =
KeyCode.L;     public KeyCode _camera =
KeyCode.P;     public Transform targetCam;
[HideInInspector]     public float distance = 6f;
private float horizontal = 0f;     private float h =
0f;     private float vertical = 0f;     private float v
= 0f;     public float speed = 2;     public float
smooth = 0.5f;     private float yMinLi = 0f;
private float yMaxLi = 90f;     public float
minDistanceX = 8.5f;     public float
maxDistanceX = 50f;     private int _cameraNext
= 1;     public Transform pointCameraPanorama;
public Transform pointCameraCabin;     public
Transform pointCameraArrow;     public
Transform pointLookCameraArrow;     public
Ex_Controller mScript;


   void Start()
   {
      Cursor.lockState = CursorLockMode.Locked;
      Cursor.visible = false;
   }
   void LateUpdate()
   {
      if (_cameraNext == 1)
      {
            h += Input.GetAxis("Mouse X") * speed;               horizontal =
Mathf.Lerp(horizontal, h, Time.deltaTime * speed / smooth);               v -=
Input.GetAxis("Mouse Y") * speed;               vertical = Mathf.Lerp(vertical, v,
Time.deltaTime * speed / smooth);               vertical = Mathf.Clamp(vertical,
yMinLi, yMaxLi);
            Quaternion rotation = Quaternion.Euler(vertical, horizontal, 0);
            Vector3 position = rotation * new Vector3(0f, 0f, -distance) +
targetCam.position;               transform.rotation = rotation;               transform.position =
position;
      }
      //      if (Input.GetKeyDown(menyHover))
      //      {
      //         Cursor.lockState = CursorLockMode.None;
      //         Cursor.visible = true;
      //         _speed = 0;
      //      }
```

```
        //      else if (Input.GetKeyUp(menyHover))
        //      {
        //          Cursor.lockState = CursorLockMode.Locked;
        //          Cursor.visible = false;
        //          _speed = speed;
        //       }
        CameraArrow();
        ToggleCamera();
        ZoomCamera();
        CameraCab();
    }
    public void ToggleCamera()
    {
        if (Input.GetKeyDown(_camera))
        {
            _cameraNext++;
if (_cameraNext > 3)
            {
                _cameraNext = 1;
            }

            if (_cameraNext == 1)
            {
                targetCam = pointCameraPanorama;
distance = 8;
            }
            if (_cameraNext == 2)
            {
                targetCam = pointCameraCabin;
distance = 0;
            }
            if (_cameraNext == 3)
            {
                targetCam = pointCameraArrow;
distance = 0;
            }
        }
    }
    public void CameraCab()
    {
        if (_cameraNext == 2)
        {
```

```
            transform.position = pointCameraCabin.position;
            Quaternion ar = Quaternion.LookRotation(pointLookCameraArrow.position - transform.position, pointLookCameraArrow.up);                transform.rotation = Quaternion.Slerp(transform.rotation, ar, Time.deltaTime * 3);
        }
    }
    public void CameraArrow()
    {
        if (_cameraNext == 3)
        {
            transform.position = pointCameraArrow.position;
            Quaternion ar = Quaternion.LookRotation(pointLookCameraArrow.position - transform.position, pointLookCameraArrow.up);                transform.rotation = Quaternion.Slerp(transform.rotation, ar, Time.deltaTime * 5);
        }
    }
    public void ZoomCamera()
    {
        if (Input.GetAxis("Mouse ScrollWheel") > 0 && _cameraNext == 1)
        {
            --distance;
        }
        else if (Input.GetAxis("Mouse ScrollWheel") < 0 && _cameraNext == 1)
        {
            ++distance;
        }
    }
}
```

APPENDIX C Excavator Movements

Controllers Code

```
  void Start()
   {       smoke.Stop();       rig =
gameObject.GetComponent<Rigidbody>();
rig.centerOfMass = centerOfMass.localPosition;
engineSound.pitch = pitchSound;
      _speed = speed;       mScriptCB =
gameObject.GetComponent<Ex_ConnectionBody>();       mScriptDis
= gameObject.GetComponent<Ex_Dasplay>();
      //Create point steer
      GameObject pLeft = new GameObject("_PointSteerWheelLeft");
      _pointSteerWheelLeft = pLeft.transform;
      _pointSteerWheelLeft.SetParent(wheelTransform[0]);
      _pointSteerWheelLeft.localPosition = new Vector3(-0.5f,0,0);
      GameObject pRight = new GameObject("_PointSteerWheelRight");
      _pointSteerWheelRight = pRight.transform;
      _pointSteerWheelRight.SetParent(wheelTransform[1]);
      _pointSteerWheelRight.localPosition = new Vector3(0.5f, 0, 0);
      GameObject sLeft = new GameObject("_SteerLeft");
```

```
        _steerLeft = sLeft.transform;
        _steerLeft.SetParent(transform);
        _steerLeft.localPosition = new Vector3(-0.8214498f, -0.3650666f, 1.322304f);
        GameObject sRight = new GameObject("_SteerRight");
        _steerRight = sRight.transform;
        _steerRight.SetParent(transform);
        _steerRight.localPosition = new Vector3(0.82019f, -0.3650669f, 1.322304f);
        _steerLeft.LookAt(_pointSteerWheelLeft.position, _steerLeft.up);
    _steerRight.LookAt(_pointSteerWheelRight.position, _steerRight.up);
    steerLeft.SetParent(_steerLeft);        steerRight.SetParent(_steerRight);


        GameObject sL_A = new GameObject("SL_A");
        _Point_srL_A = sL_A.transform;
        _Point_srL_A.SetParent(steerLeft);
        _Point_srL_A.localPosition = new Vector3(0.0818f, 0.02069998f, 0.1979f);
        GameObject sR_A = new GameObject("SR_A");
        _Point_srR_A = sR_A.transform;
        _Point_srR_A.SetParent(steerRight);
        _Point_srR_A.localPosition = new Vector3(-0.0815f, 0.0207f, 0.201f);
        GameObject sL_B = new GameObject("SL_B");
        _Point_srL_B = sL_B.transform;
        _Point_srL_B.SetParent(steeringRack);
        _Point_srL_B.localPosition = steeringRackLeft.localPosition;
        GameObject sR_B = new GameObject("SR_B");
        _Point_srR_B = sR_B.transform;
        _Point_srR_B.SetParent(steeringRack);
        _Point_srR_B.localPosition = steeringRackRight.localPosition;
        _Point_srL_B.LookAt(_Point_srL_A.position, _Point_srL_B.up);
    _Point_srR_B.LookAt(_Point_srR_A.position, _Point_srR_B.up);
    steeringRackLeft.SetParent(_Point_srL_B);        steeringRackRight.SetParent(_Point_srR_B);
        _startSteerRack =          new        Vector3(startSteerRack,        steeringRack.localPosition.y,
steeringRack.localPosition.z);

        _minSteerRack  =          new        Vector3(minSteerRack,        steeringRack.localPosition.y,
steeringRack.localPosition.z);

        _maxSteerRack      =      new        Vector3(maxSteerRack,        steeringRack.localPosition.y,
steeringRack.localPosition.z);        for (int i = 0; i <
    mScriptDis.panelIcon.Length; i++)
        {
           mScriptDis.panelIcon[i].enabled = false;
        }
        mScriptDis.distanceText.enabled = false;
    mScriptDis.panelDisplay.enabled = false;
      }
```

```
        void Update()
        {
            if(Input.GetKeyDown(pover) && pover_Bool == true)
            {
                pover_Bool = false;
                if(mScriptDis.fuelGauge == true)
                {
                    mScriptDis._flue = 98.34f;
                }
                mScriptDis.panelDisplay.enabled = true;
        mScriptDis._speedArrowDisplay = 2;
        StartCoroutine("SpeedArrowDisplay");
        startStopEngine.PlayOneShot(startEngine, 1);
                if (startStopEngine.isPlaying)
                {
                    engineSound.Play();
                }
                smoke.Play();           if
        (gameObject.GetComponentInChildren<Ex_ConnectionsPiles>() != null)
                {
                    mScriptCont.mScriptPiles.displayTop_winC.enabled = true;
        mScriptCont.mScriptPiles.textDistance.enabled = true;
        mScriptCont.mScriptPiles.textMetrS.enabled = true;
        mScriptCont.mScriptPiles.pilesImage.enabled = true;
        mScriptCont.mScriptPiles.textAnglePiles.enabled = true;
                }
                if (gameObject.GetComponentInChildren<Ex_ConnectionsTritronic>() != null)
                {
                    gameObject.GetComponentInChildren<Ex_ConnectionsTritronic>().tritronicIcon.enabled =
true;

gameObject.GetComponentInChildren<Ex_ConnectionsTritronic>().displayTop_winD.enabled = true;

gameObject.GetComponentInChildren<Ex_ConnectionsTritronic>().textAngleTritronicA.enabled = true;

gameObject.GetComponentInChildren<Ex_ConnectionsTritronic>().textAngleTritronicB.enabled = true;
                }
            }
            else if(Input.GetKeyDown(pover) && pover_Bool == false)
            {
                pover_Bool = true;
```

```
            mScriptDis.panelDisplay.enabled = false;
        if (mScriptDis.fuelGauge == true)
            {
               mScriptDis._flue = -4f;
            }
            mScriptDis.floatArrowDisplay_1 = 4.56f;
        engineSound.Stop();                                 if
        (!engineSound.isPlaying)
            {
               startStopEngine.PlayOneShot(stopEngine, 1);
            }
            mScriptDis.displayTop_winA.fillAmount = 0;
            mScriptDis.displayTop_winB.fillAmount = 0;
        mScriptDis.displayDown_winA.fillAmount = 0;
        mScriptDis.distanceText.enabled = false;
        mScriptDis.textArrow1.enabled = false;
        mScriptDis.textArrow2.enabled = false;
        mScriptDis.textArrowLadle.enabled = false;
        mScriptDis.textBody.enabled = false;            for (int i = 0;
        i < mScriptDis.panelIcon.Length; i++)
            {
               mScriptDis.panelIcon[i].enabled = false;
            }
            mScriptDis.distanceText.enabled = false;            if
        (gameObject.GetComponentInChildren<Ex_ConnectionsPiles>() != null)
            {
               mScriptCont.mScriptPiles.displayTop_winC.enabled = false;
        mScriptCont.mScriptPiles.textDistance.enabled = false;
        mScriptCont.mScriptPiles.textMetrS.enabled = false;
        mScriptCont.mScriptPiles.pilesImage.enabled = false;
        mScriptCont.mScriptPiles.textAnglePiles.enabled = false;
            }
            if (gameObject.GetComponentInChildren<Ex_ConnectionsTritronic>() != null)
            {
               gameObject.GetComponentInChildren<Ex_ConnectionsTritronic>().tritronicIcon.enabled =
false;

gameObject.GetComponentInChildren<Ex_ConnectionsTritronic>().displayTop_winD.enabled = false;

gameObject.GetComponentInChildren<Ex_ConnectionsTritronic>().textAngleTritronicA.enabled = false;
```

```
gameObject.GetComponentInChildren<Ex_ConnectionsTritronic>().textAngleTritronicB.enabled = false;
                }
                smoke.Stop();
                if (mScriptCB.forBodiIcon != null)
                {
                    mScriptCB.forBodiIcon.enabled = false;
                }
                if (mScriptCB.backBodyIcon != null)
                {
                    mScriptCB.backBodyIcon.enabled = false;
                }
            }
            if (pover_Bool == false)
            {
                if (mScriptDis._flue > -4)
                {
                    if (mScriptCont.excavatorOn_Bool == true)
                    {
                        m_Horizontal = Input.GetAxis("Horizontal");
        m_Vertical = Input.GetAxis("Vertical");
                    }
                    Motor();
                    UpdateWheelPoses();
                    SteerWheel();
                    Light();
                    SoundEngine();
                    AnimationSteer();
                    CardanAnimation();
                }
                if (mScriptDis.displayTop_winA.fillAmount != 1)
                {
                    mScriptDis.displayTop_winA.fillAmount += Time.deltaTime;
        mScriptDis.displayTop_winB.fillAmount += Time.deltaTime;
        mScriptDis.displayDown_winA.fillAmount += Time.deltaTime;
                }
                if (mScriptDis.displayTop_winA.fillAmount == 1)
                {
                    for (int i = 0; i < mScriptDis.panelIcon.Length; i++)
                    {
                        mScriptDis.panelIcon[i].enabled = true;
                    }
```

```
            mScriptDis.distanceText.enabled = true;
mScriptDis.textArrow1.enabled = true;
mScriptDis.textArrow2.enabled = true;
mScriptDis.textArrowLadle.enabled = true;
mScriptDis.textBody.enabled = true;                if
(mScriptCB.forBodiIcon != null)
            {
               mScriptCB.forBodiIcon.enabled = true;
            }
            if (mScriptCB.backBodyIcon != null)
            {
               mScriptCB.backBodyIcon.enabled = true;
            }
         }
      }
      Brake();
   }
   void FixedUpdate()
   {
      checkMoving = rig.transform.InverseTransformDirection(rig.velocity);
checkMovint_Int = (Mathf.RoundToInt(checkMoving.z));
      //  Speedometer
      Ex_UISpeed.ShowSpeed(rig.velocity.magnitude, 0, settingSpeed);
   }
   void LateUpdate()
   {
      _steerLeft.LookAt(_pointSteerWheelLeft.position, _steerLeft.up);
      _steerRight.LookAt(_pointSteerWheelRight.position, _steerRight.up);
      _Point_srL_B.LookAt(_Point_srL_A.position, _Point_srL_B.up);
      _Point_srR_B.LookAt(_Point_srR_A.position, _Point_srR_B.up);
   }
   IEnumerator SpeedArrowDisplay()
   {
      yield return new WaitForSeconds(1.6f);
mScriptDis._speedArrowDisplay = mScriptDis.speedArrowDisplay;
   }
   IEnumerator TurnSignal()
   {        while
(true)
      {
         yield return new WaitForSeconds(0.3f);
if (intLight == 1)
```

```
        {
            turnSignal[0].enabled = true;
            mScriptDis.turnSignalLeft.color = new Color32(0,255,63,255);
        }           if
(intLight == 2)
        {
            turnSignal[1].enabled = true;
            mScriptDis.turnSignalRight.color = new Color32(0, 255, 63, 255);
        }           if
(intLight == 3)
        {
            turnSignal[0].enabled = true;               turnSignal[1].enabled =
true;               mScriptDis.turnSignalLeft.color = new Color32(0, 255, 63,
255);               mScriptDis.turnSignalRight.color = new Color32(0, 255, 63,
255);
        }
        yield return new WaitForSeconds(0.3f);
if (intLight == 1)
        {
            turnSignal[0].enabled = false;
mScriptDis.turnSignalLeft.color = new Color32(255, 255, 255, 25);
        }           if
(intLight == 2)
        {
            turnSignal[1].enabled = false;
mScriptDis.turnSignalRight.color = new Color32(255, 255, 255, 25);
        }           if
(intLight == 3)
        {
            turnSignal[0].enabled = false;              turnSignal[1].enabled =
false;              mScriptDis.turnSignalLeft.color = new Color32(255, 255, 255,
25);             mScriptDis.turnSignalRight.color = new Color32(255, 255, 255,
25);
        }
    }
  }
  public void Motor()
  {
    if (Input.GetKeyDown(afterburner))
    {
      _speed = maxAfterburner;
    }
```

```
        else if (Input.GetKeyUp(afterburner))
        {
            _speed = speed;
        }
        _wheelCollider[0].motorTorque = m_Vertical * _speed;        _wheelCollider[1].motorTorque =
m_Vertical * _speed;
        _wheelCollider[2].motorTorque = m_Vertical * _speed;
_wheelCollider[3].motorTorque = m_Vertical * _speed;        rig.velocity
= Vector3.ClampMagnitude(rig.velocity, maxSpeed);
    }
    private void UpdateWheel(WheelCollider wcol,Transform wtran)
    {
        Vector3 _pos = wtran.position;
Quaternion _quat = wtran.rotation;
wcol.GetWorldPose(out _pos, out _quat);
wtran.transform.position = _pos;
wtran.transform.rotation = _quat;
    }
    public void UpdateWheelPoses()
    {
        UpdateWheel(_wheelCollider[0], wheelTransform[0]);
        UpdateWheel(_wheelCollider[1], wheelTransform[1]);
        UpdateWheel(_wheelCollider[2], wheelTransform[2]);
        UpdateWheel(_wheelCollider[3], wheelTransform[3]);
    }
    public void SteerWheel()
    {
        _wheelCollider[0].steerAngle = m_Horizontal * steerWheel;
        _wheelCollider[1].steerAngle = m_Horizontal * steerWheel;
    }
    public void Brake()
    {
        if (checkMovint_Int > 0 && m_Vertical < 0)
        {
            for (int i = 0; i < _wheelCollider.Length; i++)
            {
                _wheelCollider[i].brakeTorque = (_brake) * (Mathf.Abs(m_Vertical));
            }
            _stop[0].enabled = true;
            _stop[1].enabled = true;
        }
        else if (checkMovint_Int < 0 && m_Vertical > 0)
```

```
        {
            for (int i = 0; i < _wheelCollider.Length; i++)
            {
                _wheelCollider[i].brakeTorque = (_brake) * (Mathf.Abs(m_Vertical));
            }
            _stop[0].enabled = true;
            _stop[1].enabled = true;
        }
        else if (m_Vertical == 0)
        {
            for (int i = 0; i < _wheelCollider.Length; i++)
            {
                _wheelCollider[i].brakeTorque = (_brake) * (Mathf.Abs(0.4f));
            }
            _stop[0].enabled = false;
            _stop[1].enabled = false;
        }
else
        {
            for (int i = 0; i < _wheelCollider.Length; i++)
            {
                _wheelCollider[i].brakeTorque = 0;
            }
            _stop[0].enabled = false;
            _stop[1].enabled = false;
        }
        if(mScriptCont.excavatorOn_Bool == false)
        {
            for (int i = 0; i < _wheelCollider.Length; i++)
            {
                _wheelCollider[i].brakeTorque = 500;
            }
        }
    }
    public void Light()
    {
        if(checkMovint_Int < 0)
        {
            reverse[0].enabled = true;
reverse[1].enabled = true;
        }
else        {
```

```
reverse[0].
enabled =
false;
reverse[1].
enabled =
false;
        }
        if(Input.GetKeyDown(onLight) && onLight_Bool == true)
        {
            _light[0].enabled = true;
            _light[1].enabled = true;
            _light[2].enabled = true;            _light[3].enabled =
true;            mScriptDis.light.color = new Color32(0, 255, 63,
255);            onLight_Bool = false;
        }
        else if(Input.GetKeyDown(onLight) && onLight_Bool == false)
        {
            _light[0].enabled = false;
            _light[1].enabled = false;
            _light[2].enabled = false;            _light[3].enabled =
false;            mScriptDis.light.color = new Color32(255, 255,
255, 25);            onLight_Bool = true;
        }
        if(Input.GetKeyDown(lightCabin)&& lightCabin_Bool == true)
        {
            _light[4].enabled = true;
            _light[5].enabled = true;
            _light[6].enabled = true;            _light[7].enabled = true;
mScriptDis.lightCabin.color = new Color32(255,221,0,255);
lightCabin_Bool = false;
        }
        else if (Input.GetKeyDown(lightCabin) && lightCabin_Bool == false)
        {
            _light[4].enabled = false;
            _light[5].enabled = false;
            _light[6].enabled = false;
_light[7].enabled = false;
            mScriptDis.lightCabin.color = new Color32(255, 255, 255, 25);
lightCabin_Bool = true;
        }
        if (mScriptCont.excavatorOn_Bool == true)
        {
```

```
        //Turn Signal           if (Input.GetKeyDown(turnSignalKey) &&
turnSignal_Bool == true)
        {
            turnSignal_Bool = false;
if (leftTurnSignal_Bool == false)
            {
                leftTurnSignal_Bool = true;
            }
            if (rightTurnSignal_Bool == false)
            {
                rightTurnSignal_Bool = true;
            }
intLight = 3;
        }
        else if (Input.GetKeyDown(turnSignalKey) && turnSignal_Bool == false)
        {               intLight = 0;               turnSignal[0].enabled = false;
turnSignal[1].enabled = false;              mScriptDis.turnSignalLeft.color =
new Color32(255, 255, 255, 25);             mScriptDis.turnSignalRight.color =
new Color32(255, 255, 255, 25);             turnSignal_Bool = true;
        }
        //Left Turn Signal          if (Input.GetKeyDown(leftTurnSignal) &&
leftTurnSignal_Bool == true)
        {
            leftTurnSignal_Bool = false;
if (turnSignal_Bool == false)
            {
                turnSignal_Bool = true;
turnSignal[0].enabled = false;
turnSignal[1].enabled = false;
                mScriptDis.turnSignalLeft.color = new Color32(255, 255, 255, 25);
mScriptDis.turnSignalRight.color = new Color32(255, 255, 255, 25);
            }
            if (rightTurnSignal_Bool == false)
            {
                rightTurnSignal_Bool = true;
turnSignal[1].enabled = false;
mScriptDis.turnSignalRight.color = new Color32(255, 255,
255, 25);
            }
intLight = 1;
        }
        else if (Input.GetKeyDown(leftTurnSignal) && leftTurnSignal_Bool == false)
```

```
        {
            leftTurnSignal_Bool = true;                turnSignal[0].enabled =
false;                mScriptDis.turnSignalLeft.color = new Color32(255, 255,
255, 25);                intLight = 0;
        }
        //Right Turn Signal            if (Input.GetKeyDown(rightTurnSignal) &&
rightTurnSignal_Bool == true)
        {
            rightTurnSignal_Bool = false;
if (turnSignal_Bool == false)
            {
                turnSignal_Bool = true;                    turnSignal[0].enabled = false;
turnSignal[1].enabled = false;                    mScriptDis.turnSignalLeft.color =
new Color32(255, 255, 255, 25);                    mScriptDis.turnSignalRight.color =
new Color32(255, 255, 255, 25);
            }
            if (leftTurnSignal_Bool == false)
            {
                leftTurnSignal_Bool = true;                    turnSignal[0].enabled =
false;                    mScriptDis.turnSignalLeft.color = new Color32(255, 255,
255, 25);
            }
intLight = 2;
        }
        else if (Input.GetKeyDown(rightTurnSignal) && rightTurnSignal_Bool == false)
        {
            rightTurnSignal_Bool = true;
turnSignal[1].enabled = false;
            mScriptDis.turnSignalRight.color = new Color32(255, 255, 255, 25);
intLight = 0;
        }
        if (intLight != 0)
        {
            if (checkLight == true)
            {
                StartCoroutine("TurnSignal");
turnSignalSound.Play();                    checkLight
= false;
            }
}
else
        {
```

```
                StopCoroutine("TurnSignal");
        turnSignalSound.Stop();              checkLight
        = true;
                }
            }
        }
        private void SoundEngine()
        {
            engineRPM = Mathf.Clamp((((Mathf.Abs((_wheelCollider[0].rpm + _wheelCollider[3].rpm)) *
gearShiftRate) + minEngineRPM)) / (float)(currentGear + 1), minEngineRPM, maxEngineRPM);
engineSound.pitch          =          Mathf.Lerp(engineSound.pitch,       Mathf.Lerp(pitchSound,
            maxPitch, (engineRPM - minEngineRPM / 1.82f) / (maxEngineRPM + minEngineRPM)),
Time.deltaTime * smoohtPitch);       if(engineSound.pitch < 0.80f)
            {
                var s = smoke.GetComponent<ParticleSystem>().main;
        s.startColor = new Color(255, 255, 255, 255);
            }
            else if(engineSound.pitch > 0.88f)
            {
                var s = smoke.GetComponent<ParticleSystem>().main;
        s.startColor = colorSmoke;
            }
        }
        public void AnimationSteer()
        {
            if (Input.GetKey(KeyCode.A))
            {
                steeringRack.localPosition       =       Vector3.MoveTowards(steeringRack.localPosition,
_minSteerRack, speedSteerRack * Time.deltaTime);
            }
            else if (Input.GetKey(KeyCode.D))
            {
                steeringRack.localPosition       =       Vector3.MoveTowards(steeringRack.localPosition,
_maxSteerRack, speedSteerRack * Time.deltaTime);
            }
        else
                steeringRack.localPosition       =       Vector3.MoveTowards(steeringRack.localPosition,
_startSteerRack, speedSteerRack * Time.deltaTime * 1);
        }
        public void CardanAnimation()
        {
            if (checkMovint_Int > 0)
```

```
        {
            floatCardan += speedCardan * 10 * Time.deltaTime;
        }
        else if(checkMovint_Int < 0)
        {
            floatCardan -= speedCardan * 10 * Time.deltaTime;
        }
        var _car_A = Quaternion.AngleAxis(floatCardan, Vector3.forward);
        cardanFor_A.localRotation = Quaternion.Lerp(cardanFor_A.localRotation, _car_A,
Time.deltaTime * speedCardan);
        cardanBack_A.localRotation = Quaternion.Lerp(cardanBack_A.localRotation, _car_A,
Time.deltaTime * speedCardan);
        cardanCenter.localRotation = Quaternion.Lerp(cardanCenter.localRotation, _car_A,
Time.deltaTime * speedCardan);}}
```

APPENDIX D

Excavator Arm Controllers

```
 void
Start()
   {
      mScriptHook = gameObject.GetComponent<Ex_Hook>();
mScriptD = gameObject.GetComponent<Ex_Dasplay>();
      _pitchEx = pitchSoundExcavator;
      //Slowdown effect
      _speed = speed;
      _pushUp = 0.5f;
      //Piston A
      GameObject p_0A = new GameObject("_PistonArrow0_A");
```

```
        _pistonArrow0_A = p_0A.transform;
        _pistonArrow0_A.SetParent(cabObject);
        _pistonArrow0_A.localPosition = new Vector3(0.1375265f, 0.2870061f, 1.087389f);
        GameObject p_0B = new GameObject("_PistonArrow0_B");
        _pistonArrow0_B = p_0B.transform;
        _pistonArrow0_B.SetParent(arrow0);
        _pistonArrow0_B.localPosition = pistonArrow0_B.localPosition;
        _pistonArrow0_A.LookAt(_pistonArrow0_B.position, _pistonArrow0_A.up);
_pistonArrow0_B.LookAt(_pistonArrow0_A.position, _pistonArrow0_B.up);
pistonArrow0_A.SetParent(_pistonArrow0_A);
pistonArrow0_B.SetParent(_pistonArrow0_B);
        //Piston B
        GameObject p_1A = new GameObject("_PistonArrow1_A");
        _pistonArrow1_A = p_1A.transform;
        _pistonArrow1_A.SetParent(arrow1);
        _pistonArrow1_A.localPosition = pistonArrow1_A.localPosition;
        GameObject p_1B = new GameObject("_PistonArrow1_B");
        _pistonArrow1_B = p_1B.transform;
        _pistonArrow1_B.SetParent(leverArm_A);
        _pistonArrow1_B.localPosition = new Vector3(0.0006445069f, 0.3558334f, 0.2512825f);
        //Piston C
        GameObject p_2A = new GameObject("_PistonArrow2_A");
        _pistonArrow2_A = p_2A.transform;
        _pistonArrow2_A.SetParent(arrow0);
        _pistonArrow2_A.localPosition = pistonArrow2_A.localPosition;
        GameObject p_2B = new GameObject("_PistonArrow0_B");
        _pistonArrow2_B = p_2B.transform;
        _pistonArrow2_B.SetParent(arrow1);
        _pistonArrow2_B.localPosition = pistonArrow2_B.localPosition;
        _pistonArrow2_A.LookAt(_pistonArrow2_B.position, _pistonArrow2_A.up);
_pistonArrow2_B.LookAt(_pistonArrow2_A.position, _pistonArrow2_B.up);
pistonArrow2_A.SetParent(_pistonArrow2_A);
pistonArrow2_B.SetParent(_pistonArrow2_B);
        //Detali
        GameObject l_A = new GameObject("_LeverArm_A");
        _leverArm_A = l_A.transform;
        _leverArm_A.SetParent(arrow1);
        _leverArm_A.localPosition = leverArm_A.localPosition;
        GameObject l_B = new GameObject("_LeverArm_B");
        _leverArm_B = l_B.transform;
        _leverArm_B.SetParent(leverArm_B);
        _leverArm_B.localPosition = new Vector3(0.0006433718f, 0.5327493f, -0.2626385f);
```

```
        GameObject l_C = new GameObject("_LeverArm_C");
        _leverArm_C = l_C.transform;
        _leverArm_C.SetParent(ladleObject);
        _leverArm_C.localPosition = leverArm_B.localPosition;
        GameObject l_D = new GameObject("_LeverArm_D");
        _leverArm_D = l_D.transform;
        _leverArm_D.SetParent(leverArm_A);
        _leverArm_D.localPosition = new Vector3(0.0006445069f, 0.3558334f, 0.2512825f);
        _leverArm_A.LookAt(_leverArm_B.position, _leverArm_A.up);
_leverArm_C.LookAt(_leverArm_D.position, _leverArm_C.up);
leverArm_A.SetParent(_leverArm_A);
leverArm_B.SetParent(_leverArm_C);
        _pistonArrow1_A.LookAt(_pistonArrow1_B.position, _pistonArrow1_A.up);
_pistonArrow1_B.LookAt(_pistonArrow1_A.position, _pistonArrow1_B.up);
pistonArrow1_A.SetParent(_pistonArrow1_A);
pistonArrow1_B.SetParent(_pistonArrow1_B);
    }
    void LateUpdate()
    {
        _pistonArrow0_A.LookAt(_pistonArrow0_B.position, _pistonArrow0_A.up);
        _pistonArrow0_B.LookAt(_pistonArrow0_A.position, _pistonArrow0_B.up);
        _leverArm_A.LookAt(_leverArm_B.position, _leverArm_A.up);
        _leverArm_C.LookAt(_leverArm_D.position, _leverArm_C.up);
_pistonArrow1_A.LookAt(_pistonArrow1_B.position, _pistonArrow1_A.up);
_pistonArrow1_B.LookAt(_pistonArrow1_A.position, _pistonArrow1_B.up);
        _pistonArrow2_A.LookAt(_pistonArrow2_B.position, _pistonArrow2_A.up);
        _pistonArrow2_B.LookAt(_pistonArrow2_A.position, _pistonArrow2_B.up);
    }
    void Update()
    {
        if (mScript.pover_Bool == false)
        {
            if (mScriptD._flue > -4)
            {
                //Turn on excavator mode Turn off CarController                if
(Input.GetKeyDown(excavatorOn) && excavatorOn_Bool == true)
                {
                    excavatorOn_Bool = false;
//Slowdown effect
                    _pushUp += myAngle_B;
                }
                else    if    (Input.GetKeyDown(excavatorOn)    &&    excavatorOn_Bool    ==    false    &&
```

```
blockArrowIfPiles == true)
                {
                   excavatorOn_Bool = true;
                   //Slowdown effect
                   _pushUp = 0.5f;
                   _pitchEx = pitchSoundExcavator;
                }
                CabTurn();
                AnimationArrow();
                Ladle();
                PushUp();
                CheckDistance();
                if (ladleObjectCollider.GetComponent<MeshCollider>().enabled == false)
                {
                   timeEnambledCollider -= Time.deltaTime;
      if (timeEnambledCollider < 0)
                   {
                      if (arrow1.GetComponent<MeshCollider>().enabled == false)
                      {
                         arrow1.GetComponent<MeshCollider>().enabled = true;
      leverArm_B.GetComponent<MeshCollider>().enabled = true;                    }
                      ladleObjectCollider.GetComponent<MeshCollider>().enabled = true;
      timeEnambledCollider = 1;
                   }
                }
             }
             PanelInfo();
          }
       }
       IEnumerator ConnectedIcon()
       {       while
      (true)
          {
             yield return new WaitForSeconds(0.3f);
      pointConnectionIcon_B.rectTransform.position = new Vector3(59.8f, 287.75f, 0);          yield
      return new WaitForSeconds(0.3f);          pointConnectionIcon_B.rectTransform.position =
      new Vector3(63.8f, 287.75f, 0);
          }
       }
       public void CabTurn()
       {
```

```
        if (excavatorOn_Bool == false && blockArrowIfPiles == true)
        {
            if (Input.GetKey(cabTurnLeft))
            {
                floatTurnCab -= Time.deltaTime * speedCab;
    SoundPitchDump();              mScriptD.displayArrowLeft.enabled
    = true;
            }
            else if (Input.GetKeyUp(cabTurnLeft))
            {
                mScriptD.displayArrowLeft.enabled = false;
            }
            if (Input.GetKey(cabTurnRight))
            {
                floatTurnCab += Time.deltaTime * speedCab;
    SoundPitchDump();
                mScriptD.displayArrowRight.enabled = true;
            }
            else if (Input.GetKeyUp(cabTurnRight))
            {
                mScriptD.displayArrowRight.enabled = false;
            }
        }
        var cab = Quaternion.AngleAxis(floatTurnCab, Vector3.up);      cabObject.localRotation =
    Quaternion.Lerp(cabObject.localRotation, cab, Time.deltaTime *
speedCab / smoothCab);
    }
    public void SoundPitchDump()
    {       if (mScript !=
    null)
        {
            mScript.engineSound.pitch      =      Mathf.Lerp(mScript.engineSound.pitch,
      _pitchEx, Time.deltaTime * 10);
        }
    }
    public void AnimationArrow()
    {
        if (blockArrowIfPiles == true)
        {
            if (excavatorOn_Bool == false)
            {
                blockArrowDisplay = Mathf.RoundToInt(myAngle_A);
```

```
            //Arrow0            if
        (Input.GetKey(upArrow))
            {
               _speed = speed;
               _pitchEx = pitchSoundExcavator;
        floatArrow0 += _speed * Time.deltaTime;
        mScriptD.floatExArrow1 += _speed * Time.deltaTime;
               SoundPitchDump();
            }
            else if (Input.GetKey(downArrow))
            {
               floatArrow0    -=    _speed    *    Time.deltaTime;
        mScriptD.floatExArrow1    -=    _speed    *    Time.deltaTime;
        checkDownKey_A = 1;
               SoundPitchDump();
            }
            if (Input.GetKeyUp(downArrow))
            {
               checkDownKey_A = 0;
               _speed = speed;
            }
            floatArrow0 = Mathf.Clamp(floatArrow0, minArrow0, maxArrow0);          var _arrow0
= Quaternion.AngleAxis(floatArrow0,  Vector3.left);                    arrow0.localRotation =
Quaternion.Lerp(arrow0.localRotation, _arrow0, Time.deltaTime * _speed / smooth);
            //Arrow1            if
        (Input.GetKey(forwardArrow))
            {
               floatArrow1 -= _speed * Time.deltaTime;
        if(mScriptD.distanceText.text != "0,1m"){
        mScriptD.floatExArrow2 -= _speed * Time.deltaTime;
               }
               checkDownKey_B = 1;
               SoundPitchDump();
            }
            else if (Input.GetKey(backArrow))
            {
               _pitchEx = pitchSoundExcavator;
               _speed = speed;            floatArrow1 += _speed *
        Time.deltaTime;            mScriptD.floatExArrow2 += _speed *
        Time.deltaTime;
               SoundPitchDump();
            }
```

```
                if (Input.GetKeyUp(forwardArrow))
                {
                    checkDownKey_B = 0;


                }
                floatArrow1 = Mathf.Clamp(floatArrow1, minArrow1, maxArrow1);
        var _arrow1 = Quaternion.AngleAxis(floatArrow1, Vector3.left);
                arrow1.localRotation = Quaternion.Lerp(arrow1.localRotation, _arrow1, Time.deltaTime *
_speed / smooth);
            }
        }
    }
    public void Ladle()
    {
        if (mScriptPiles == null)
        {
            if (excavatorOn_Bool == false)
            {               if
        (mScriptT == null)
                {
                    if (Input.GetKey(upLadle))
                    {
                        floatLadle += speed * Time.deltaTime;
                        SoundPitchDump();
                    }
                    else if (Input.GetKey(downLadle))
                    {
                        floatLadle -= speed * Time.deltaTime;
                        SoundPitchDump();
                    }
                }
                //Key matching. If the tritronic is connected to the arrow, the keys will be blocked. "Q" "E"
        else if (mScriptT != null)
                {
                    if (Input.GetKey(upLadle) && Input.GetKey(mScriptT.pressed) == false)
                    {
                        floatLadle += speed * Time.deltaTime;
                        SoundPitchDump();
                    }
                    else if (Input.GetKey(downLadle) && Input.GetKey(mScriptT.pressed) == false)
                    {
                        floatLadle -= speed * Time.deltaTime;
```

```
                    SoundPitchDump();
                }
            }
            floatLadle = Mathf.Clamp(floatLadle, minLadle, maxLadle);
    var _Ladle = Quaternion.AngleAxis(floatLadle, Vector3.left);
            ladleObject.localRotation = Quaternion.Lerp(ladleObject.localRotation, _Ladle,
Time.deltaTime * speed / smooth);
        }
      }
    }
    void OnTriggerEnter(Collider _col)
    {
      if (mScriptHook.hangTheHook_Bool == true) // Blocks hardware search
      {
        if (excavatorOn_Bool == false)
        {
          if (blockConnection == true) //Blocks hardware search. Deny for connection
          {
            if (_col.tag == tagEquipment)
            {                    attach_Bool = true;
    if (_col.GetComponent<Ex_Connections>() != null)
                {
                  _col.GetComponent<Ex_Connections>().mScript =
GetComponent<Ex_Controller>();
                  _col.GetComponent<Ex_Connections>().checkPoint = 1;
    blockTritronic_Bool = false;
                }
                if (_col.GetComponent<Ex_ConnectionsTritronic>() != null)
                {
                  _col.GetComponent<Ex_ConnectionsTritronic>().mScript =
GetComponent<Ex_Controller>();
                }
              if (pointConnection0.GetComponentInChildren<Ex_ConnectionsTritronic>() != null)
              {
                if (_col.GetComponent<Ex_ConnectionsPiles>() == null)
                {
```

```
                    _col.GetComponent<Ex_Connections>().mScriptTritronic =
pointConnection0.GetComponentInChildren<Ex_ConnectionsTritronic>();
                    _col.GetComponent<Ex_Connections>().checkPoint = 2;
    blockImageConnection = true;
                }
                else blockImageConnection = false;
            }
            if (_col.GetComponent<Ex_ConnectionsPiles>() != null)
            {
                if (gameObject.GetComponentInChildren<Ex_ConnectionsTritronic>() == null)
                {
                    _col.GetComponent<Ex_ConnectionsPiles>().mScript =
GetComponent<Ex_Controller>();
                }
            }
     // Blocks UI if "Tritronium" and "Piles"
    if (blockImageConnection == true)
            {
                pointConnectionIcon_A.enabled = true;
    pointConnectionIcon_B.enabled = true;
                StartCoroutine("ConnectedIcon");
            }
        }
        }
        else if (blockConnection == false &&
gameObject.GetComponentInChildren<Ex_Tritronic>() != null && blockConnectionTritronic == true)
        {       attach_Bool = true;
    pointConnectionIcon_A.enabled = true;
    pointConnectionIcon_B.enabled = true;
    StartCoroutine("ConnectedIcon");        if
    (_col.GetComponent<Ex_Connections>() != null)
            {
                _col.GetComponent<Ex_Connections>().mScript =
GetComponent<Ex_Controller>();
            }
        }
    }
  }
    void OnTriggerExit(Collider _col)
    {   attach_Bool =
    false;
```

```
        pointConnectionIcon_A.enabled = false;
pointConnectionIcon_B.enabled = false;
StopCoroutine("ConnectedIcon");         blockTritronic_Bool
= true;
    }
    public void PushUp()
    {
        if (excavatorDeceleration == true)
        {
            if (excavatorOn_Bool == true)
            {
                Ray ray = new Ray(transform.position, transform.TransformDirection(-Vector3.up) * 3);
RaycastHit _hit;                if (Physics.Raycast(ray, out _hit))
                {
                    myAngle_B = Vector3.Angle(_hit.normal, transform.up);
                }
            }
            else if (excavatorOn_Bool == false)
            {
                Ray ray = new Ray(transform.position, transform.TransformDirection(-Vector3.up) * 3);
RaycastHit _hit;                if (Physics.Raycast(ray, out _hit))
                {
                    myAngle_A = Vector3.Angle(_hit.normal, transform.up);
                }
                if (myAngle_A > _pushUp)
                {
                    _speed = 2.5f;
if (_pitchEx_Bool == true)
                    {
                        _pitchEx += 0.2f;
                        _pitchEx_Bool = false;
                    }
                }
                if(myAngle_A == 0)
                {
                    _speed = speed;
                    _pitchEx = pitchSoundExcavator;
                    _pitchEx_Bool = true;
                }
            }
        }
    }
```

```
    //Checking the distance from the ground to the boom
    public void CheckDistance()
    {
        if (excavatorOn_Bool == false)
        {
            Debug.DrawRay(ladleObject.position, transform.TransformDirection(-Vector3.up) * 90,
Color.red);
            Ray rayDis = new Ray(ladleObject.position, transform.TransformDirection(-Vector3.up) * 90);
    RaycastHit hitDis;          if (Physics.Raycast(rayDis, out hitDis))
            {
                checkDistance = hitDis.distance;
            }
        }
    }
    public void PanelInfo()
    {
        if (blockArrowIfPiles == true)
        {
            mScriptD.textBody.text=
(Mathf.RoundToInt(Mathf.Abs(cabObject.localEulerAngles.y)).ToString());
    mScriptD.textArrow1.text = (Mathf.RoundToInt(floatArrow0).ToString());
    mScriptD.textArrow2.text = (Mathf.RoundToInt(floatArrow1).ToString());
        }
    else
        {
            mScriptD.textBody.text = "Locked";
    mScriptD.textArrow1.text = "Locked";
    mScriptD.textArrow2.text = "Locked";
        }
        if (blockArrowLadle == true)
        {
            mScriptD.textArrowLadle.text = (Mathf.RoundToInt(floatLadle).ToString());
        }
    else
        {
            mScriptD.textArrowLadle.text = "Locked";
        }
    }
```

APPENDIX E

UNITY ARDUINO COMMUNICATION SCRIPT

```
using System.Collections;
using System.IO.Ports;
using UnityEngine;

public class ArduinoCommunication : MonoBehaviour
{
    // Define the COM port of your Arduino (check Arduino IDE > Tools > Port)
    public string comPort = "COM3"; // Update with your specific COM port

    private SerialPort serialPort;

    void Start()
    {
        // Create a new SerialPort object
        serialPort = new SerialPort(comPort, 9600);
        // Open the serial port
        serialPort.Open();
    }

    public void SendCharacterInput(float verticalInput)
    {
        // Convert float input to a character
        char characterInput = (verticalInput > 12) ? 'w' : 's';
        // Send the character input to Arduino
        serialPort.Write(characterInput.ToString());
    }
```

```
    void OnApplicationQuit()
    {
        // Close the serial port when the application is closed
        serialPort.Close();
    }
}
```

APPENDIX E

Arduino Uno Code for Motor Control

```
/*Motor Right
 * R_PWM  : Backward
 * L_PWM  : Forward
 //Motor Left
 * R_PWM : Backward
 * L_PWM : Forward
 */
const int L_PWM_Right = 9;        // Left PWM of Right Side Motor
const int R_PWM_Right = 10;       // Right PWM of Right Side Motor
const int L_PWM_Left = 5;         // Left PWM of Left Side Motor
const int R_PWM_Left = 6;        // Right PWM of Left Side Motor
int pos=0;
int upcheck,downcheck=1;
int cond=0;
void setup() {
 //Pin Mode Declaration of Right Side Motor
  pinMode(L_PWM_Right, OUTPUT);
  pinMode(R_PWM_Right, OUTPUT);
 //Pin Mode Declaration of Left Side Motor
```

```
  pinMode(L_PWM_Left, OUTPUT);
  pinMode(R_PWM_Left, OUTPUT);
  Serial.begin(9600);
}
//a=down   w=up
void loop()
{
if (Serial.read()=='a' && downcheck==1)
{
  if(cond==2)
  {
    down();
    delay(400);
    stop();
  }
down();
delay(400);
stop();
  downcheck=0;
  upcheck=1;
  cond=1;
}
else if(Serial.read()=='w' && upcheck==1)
{
   if(cond==1)
  {
    up();
    delay(400);
    stop();
  }
 up();
```

```
 delay(400);
 stop();
  upcheck=0;
  downcheck=1;
cond=2;
}
else
{
  stop();
}
  //Getting Right Counter Value With the help of Movement of Rotary Encoder Mode
}
void stop()
{
   analogWrite(L_PWM_Right, 0);  // Set the PWM value to maximum (full speed)
   analogWrite(R_PWM_Right, 0);
   analogWrite(L_PWM_Left, 0);  // Set the PWM value to maximum (full speed)
   analogWrite(R_PWM_Left, 0);
}
void up()
{
   analogWrite(L_PWM_Right, 0);  // Set the PWM value to maximum (full speed)
   analogWrite(R_PWM_Right, 70);
   analogWrite(L_PWM_Left,0);  // Set the PWM value to maximum (full speed)
   analogWrite(R_PWM_Left, 70);
}
 void down()
 {
     analogWrite(L_PWM_Right, 70);  // Set the PWM value to maximum (full speed)
     analogWrite(R_PWM_Right, 0);
     analogWrite(L_PWM_Left, 70);  // Set the PWM value to maximum (full speed)
```

```
    analogWrite(R_PWM_Left, 0);
 }
 void leftup()
 {
      analogWrite(L_PWM_Right, 0);  // Set the PWM value to maximum (full speed)
  analogWrite(R_PWM_Right, 70);
    analogWrite(L_PWM_Left, 0);  // Set the PWM value to maximum (full speed)
  analogWrite(R_PWM_Left, 0);

 }
void leftdown()
{
      analogWrite(L_PWM_Right, 220);  // Set the PWM value to maximum (full speed)
  analogWrite(R_PWM_Right, 0);
    analogWrite(L_PWM_Left, 220);  // Set the PWM value to maximum (full speed)
  analogWrite(R_PWM_Left, 0);
}
void rightdown()
{
      analogWrite(L_PWM_Right, 220);  // Set the PWM value to maximum (full speed)
  analogWrite(R_PWM_Right, 0);
    analogWrite(L_PWM_Left, 220);  // Set the PWM value to maximum (full speed)
  analogWrite(R_PWM_Left, 0);
}
void rightup()
{
      analogWrite(L_PWM_Right, 220);  // Set the PWM value to maximum (full speed)
  analogWrite(R_PWM_Right, 0);
    analogWrite(L_PWM_Left, 220);  // Set the PWM value to maximum (full speed)
  analogWrite(R_PWM_Left, 0);
}
```

CONTACT INFORMATION

1) Ammar Shafqat

Email ID: ammarshafqatf19@nutech.edu.pk

2) Zakria Zaheer

Email ID: muhammadzakria19@nutech.edu.pk