Productivity and Vocational Training Service

Misr International Computer & AI

Department of Artificial Intelligence and Embedded Systems

Arduino VR Lab

Team Members:

Firas Ashraf Mohey Eldein

Hassan Hussein Desouki

Mark Ramzi Ibrahim Sobhi

Seif Eldein Hazem Hassan

Supervision:

Dr. Michael Nassif Michael

Eng. Hassan Talal Hetta

2024/2025

This project is submitted in fulfillment of the requirements for obtaining an industrial apprenticeship diploma in artificial intelligence and embedded systems.

Productivity and Vocational Training Service

Misr International Computer & AI

Department of Artificial Intelligence and Embedded Systems

Arduino VR Lab

Team Members:

Firas Ashraf Mohey Eldein

Hassan Hussein Desouki

Mark Ramzi Ibrahim Sobhi

Seif Eldein Hazem Hassan

Supervision:

Dr. Michael Nassif Michael

Eng. Hassan Talal Hetta

2024/2025

This project is submitted in fulfillment of the requirements for obtaining an industrial apprenticeship diploma in artificial intelligence and embedded systems.

Table of Contents

Abstract

The **Arduino VR Lab** project is designed to create an advanced **virtual electronics laboratory** that leverages **Arduino platforms** to provide an interactive and educational environment. This simulator offers users a **practical and immersive learning experience** by allowing them to design, build, and connect **electronic circuits** through an intuitive digital interface, eliminating the need for physical components.

One of the key objectives of this project is to enable users to interact with and control **various sensors and electronic actuators**, experiment with **Arduino programming modules**, and explore the fundamental principles of **electronic system operations** in a **safe and controlled virtual environment**. By integrating real-time circuit simulation and interactive programming features, the **Arduino VR Lab** enhances hands-on learning and enables users to test and debug their electronic designs efficiently.

This project serves as a significant advancement in **engineering education and technical training**, bridging the gap between **theoretical knowledge and practical application**. It fosters a deeper understanding of **electronics, programming, and embedded systems**, making it a valuable tool for students, researchers, and professionals looking to develop their **skills in circuit design, microcontroller programming, and system integration**.

By incorporating **virtual reality (VR) and interactive simulation technologies**, the **Arduino VR Lab** represents a step forward in **modern educational methodologies**, promoting **engagement, innovation, and accessibility** in electronics learning. Ultimately, this project contributes to the evolution of **digital learning tools**, providing a cost-effective and scalable solution for enhancing **technical education and professional development** in the field of **electronics and embedded systems**.

Acknowledgment

First and foremost, all praise and thanks be to Almighty God, whose guidance and blessings have enabled us to successfully complete this project.

We extend our deepest gratitude and appreciation to **Dr. Michael Nassif Michael** for his invaluable guidance, continuous support, and insightful advice throughout this journey. His expertise, patience, and encouragement played a crucial role in shaping this project into its final form.

We would also like to express our sincere thanks to **Eng. Hassan Talal Hetta** for his unwavering assistance, technical insights, and constructive feedback. His dedication and willingness to share his knowledge significantly contributed to the quality and success of this work.

Furthermore, we are immensely grateful to our professors, colleagues, and everyone who has offered their support, whether through mentorship, valuable discussions, or motivation. Their encouragement has been a driving force behind our perseverance.

Last but not least, we extend our heartfelt appreciation to our families and friends, whose constant encouragement, patience, and unwavering belief in us have been a source of strength throughout this endeavor.

May this work serve as a meaningful contribution to the field and inspire future advancements.

History of Virtual Reality and Embedded Systems

Introduction

The evolution of technology has shaped the way we interact with digital environments and physical systems. Two key fields that play a crucial role in modern technological advancements are Virtual Reality (VR) and Embedded Systems. This chapter explores the origins, development, and impact of these fields, highlighting their role in shaping projects like the Arduino VR Lab.

The Birth and Evolution of Virtual Reality

Virtual Reality, commonly known as VR, has a long history dating back to the mid-20th century. While VR is often associated with modern headsets and immersive simulations, its roots can be traced back to early mechanical devices and conceptual ideas.

Early Concepts (1800s – 1960s)

1838 – Stereoscopic Images: Sir Charles Wheatstone developed the concept of stereoscopic vision, demonstrating how the brain combines two images into a single 3D view (Wheatstone, 1838).

1929 – Link Trainer: Edwin Link created the first flight simulator to train pilots, laying the foundation for VR-based training (Mindell, 2002).

1962 – Sensorama: Morton Heilig designed a machine that provided an immersive experience using visuals, sound, vibration, and smell (Heilig, 1962).

1968 – The Sword of Damocles: Ivan Sutherland and his student Bob Sproull developed the first head-mounted display (HMD), marking the birth of modern VR (Sutherland, 1968).

Advancements in the Late 20th Century (1970s – 1990s)

1980s – NASA and VR Training: NASA used VR for astronaut training, advancing the technology for practical applications (Fisher et al., 1986).

1985 – VPL Research: Jaron Lanier founded VPL Research and coined the term "Virtual Reality," developing early VR gloves and headsets (Lanier, 1989).

1990s – Early Consumer VR: Companies like Sega and Nintendo attempted to introduce VR to gaming, though the technology was still in its infancy (Takeda, 1995).

Modern VR and Widespread Adoption (2000s – Present)

2010 – Oculus Rift: Palmer Luckey's prototype revolutionized VR, leading to advancements in affordable and high-quality headsets (Luckey, 2012).

2016 – Commercial Boom: Companies like HTC, Sony, and Microsoft released VR headsets, bringing immersive experiences to the masses (Steinicke, 2016).

2020s – VR in Education and Industry: VR is now widely used in education, medical training, engineering, and industrial simulations, making it an essential tool for modern learning experiences (Burdea & Coiffet, 2020).

Figure 1: Evolution of VR

The Evolution of Embedded Systems

Embedded systems are specialized computing units designed to perform dedicated functions within larger systems. They play a crucial role in automation, robotics, and IoT devices.

Early Developments (1950s – 1970s)

1950s – First Embedded System: The MIT Whirlwind computer introduced real-time computing concepts (Redmond & Smith, 1980).

1960s – Apollo Guidance Computer: NASA used an embedded system to navigate the Apollo missions to the moon (Mindell, 2008).

1971 – Intel 4004: The first commercially available microprocessor laid the foundation for modern embedded computing (Faggin, 1971).

Expansion and Commercial Use (1980s – 1990s)

1980s – Microcontrollers: The development of microcontrollers made embedded systems more accessible and efficient (Ganssle, 1992).

1990s – Rise of Consumer Electronics: Embedded systems became integral to household appliances, gaming consoles, and automotive applications (Wolf, 1994).

Modern Embedded Systems (2000s – Present)

2000s – Arduino and Open-Source Hardware: The launch of Arduino and Raspberry Pi revolutionized embedded system development (Banzi, 2005).

2010s – IoT and Smart Devices: Embedded systems became the backbone of smart home technologies and industrial automation (Evans, 2011).

2020s – AI Integration: Modern embedded systems integrate AI for real-time decision-making in robotics, healthcare, and autonomous vehicles (Goodfellow et al., 2016).

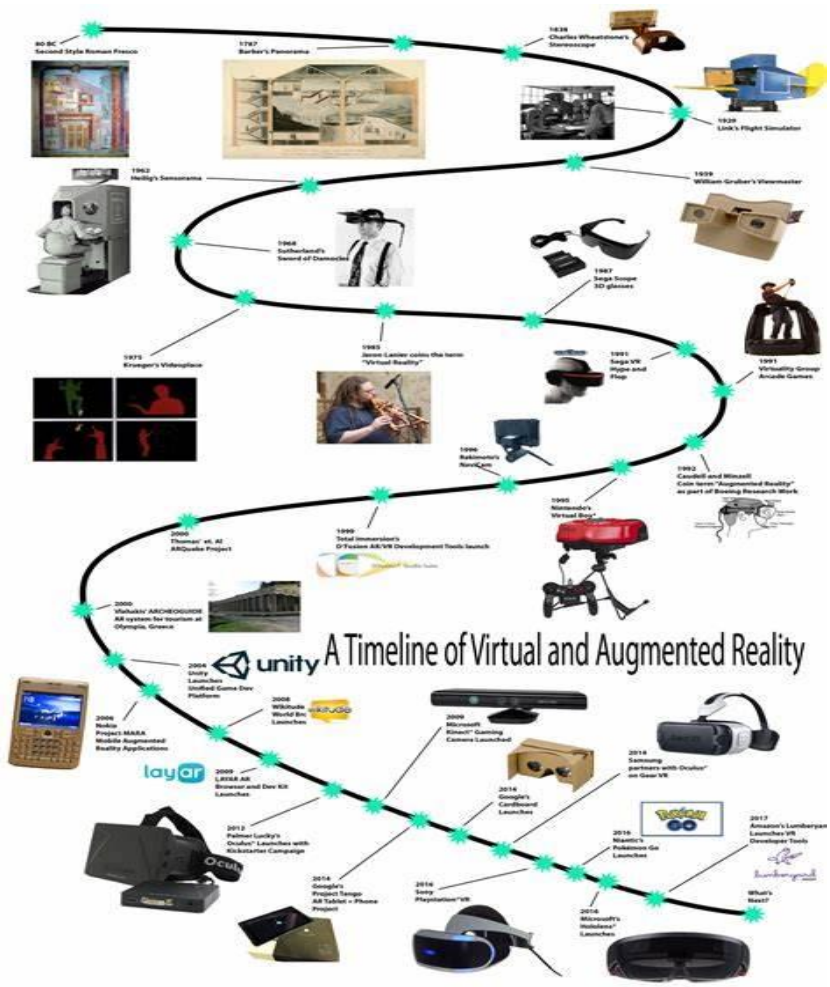

Figure 2: A Timeline of Virtual and Augmented Reality

Conclusion

The evolution of VR and embedded systems has paved the way for innovative projects like the Arduino VR Lab, where immersive simulations meet hands-on electronics learning. Understanding the historical context helps us appreciate the technological advancements that make such projects possible today. As VR and embedded systems continue to evolve, their applications in education and industry will only expand, making them fundamental pillars of future technological innovations.

Chapter 1

Introduction

1.1 Project idea:

Providing a virtual reality experience to simulate electronic parts and Arduino and apply them through VR or computer to achieve a less expensive educational experience for schools and academies.

1.2 project objectives:

1- Providing a safe and realistic learning environment: Learners can experience various scenarios and circuits electronic devices without risking damage to the actual components or exposure to any electrical hazards.

2- Enhancing programming and control skills: The Arduino VR Lab provides interactive programming tools that make it easier for users to write codes And experience its effects on electronic systems in real time.

3- Providing an interactive and inspiring experience: The Arduino VR Lab aims to make the learning process more interactive and exciting, which increases The Learners motivation and understanding of electronic and programming concepts.

4- Simulating complex electronic systems: Learners can build and test complex circuits and analyze their performance through Comprehensive interface, contributing to a deeper understanding of the real-world applications.

5- Desktop software availability: This virtual lab will be made available on desktop software, making it accessible and easy to use for educational institutions and individuals.

6- Providing an augmented reality experience for educational purposes.

7- Reducing the cost of tools will lead to the spread of educational academies in abundance.

1.3 Why this project:

Through this project, a door will be opened in Egypt and the Arab world to use augmented reality in educational applications, especially in simulating electronic parts that may be very expensive for scientific institutions.

Solve:

We solve the high cost of providing educational tools to provide an enhanced website experience.

Learn:

- C# programming language
- Blender 3D
- Animation
- Unity

Implement:

- Meta Quest 2

Economy:

It reduces the cost of establishing educational institutions and academies due to the availability of components that may be very expensive, but the cost of purchasing the software.

1.4 project Block Diagram:

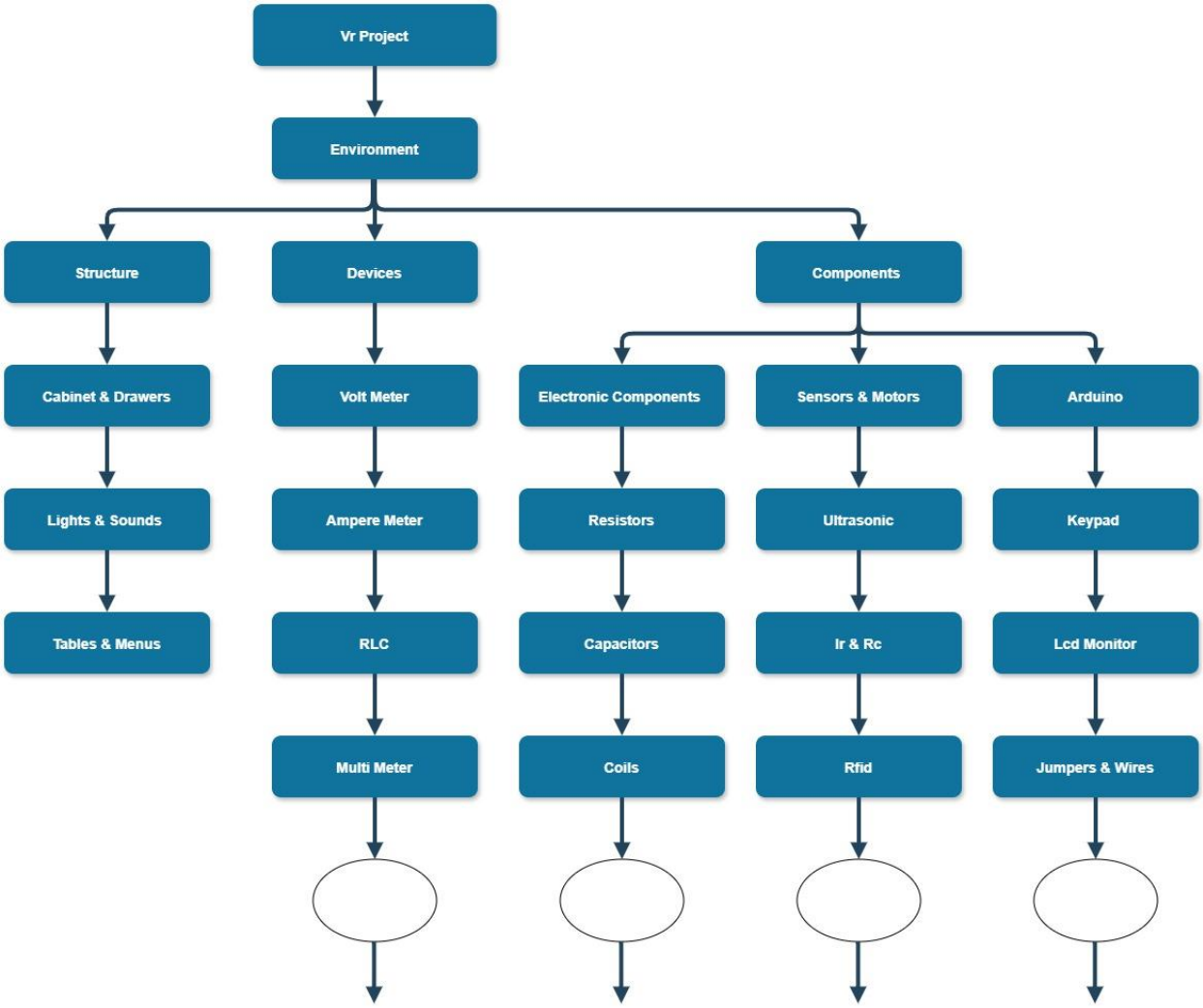

Figure 3: Block Diagram

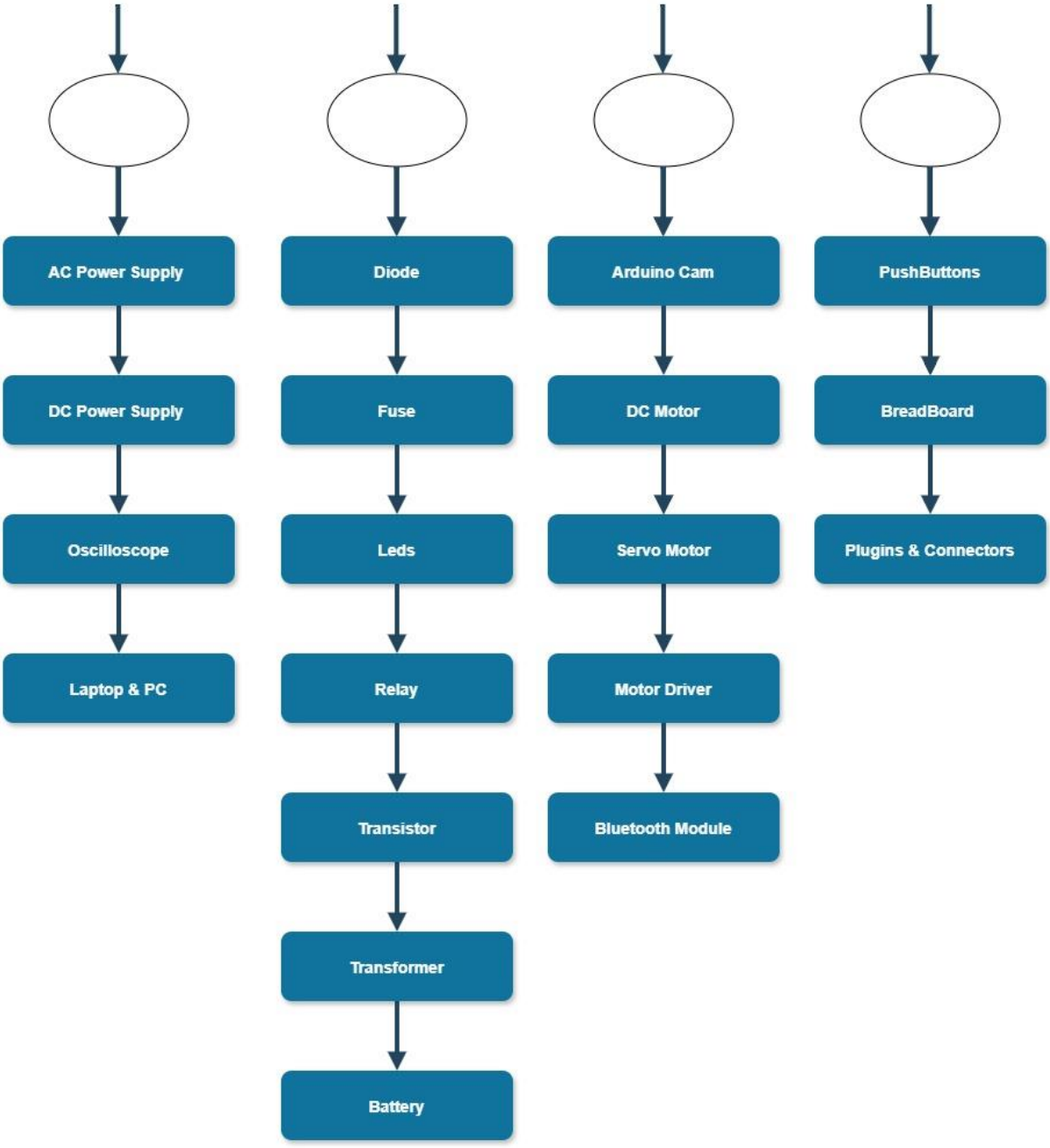

Figure 3: Block Diagram

1.5 project Flowchart:

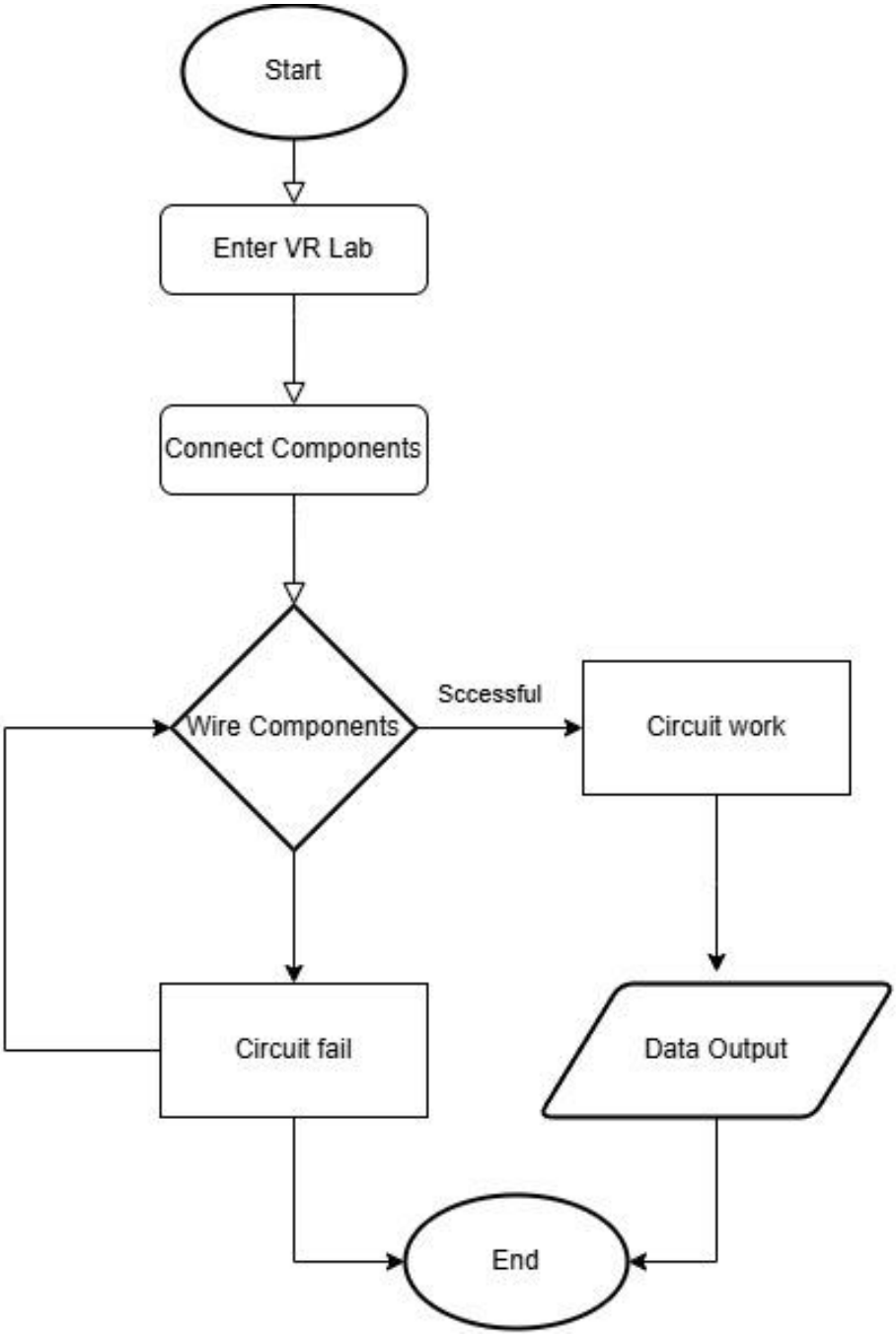

Figure 4: **Flowchart**

Chapter 2

scientific background

2.1 Introduction:

The Arduino VR Lab project integrates Augmented Reality (AR), Virtual Reality (VR), game development, and electronics education to create an interactive virtual electronics laboratory. This project leverages game engines, immersive technologies, and embedded systems to provide a realistic and engaging learning experience for students and engineers.

---

2. 2Virtual Reality (VR) and Augmented Reality (AR) in Education:

2.2.1 Virtual Reality (VR) in Learning

Virtual Reality (VR) is a computer-generated simulation that allows users to interact with a 3D environment using specialized hardware such as VR headsets and motion controllers. In education, VR enhances engagement, understanding, and retention by providing a hands-on experience.

Key benefits of VR in learning:

- Immersive Experience: Students can explore and interact with complex circuits in a fully 3D environment.
- Safe Learning: Eliminates the risks associated with handling real electronic components.
- Interactive Feedback: Users receive real-time feedback when building and testing circuits.
- Remote Accessibility: VR labs can be accessed from anywhere, enabling distance learning.

2.2.2 Augmented Reality (AR) in Learning:

Augmented Reality (AR) overlays digital content onto the real world, allowing users to visualize circuit connections and components in real-time. AR enhances learning by:

- Providing step-by-step guidance in assembling electronic circuits.
- Allowing users to interact with virtual components in a real-world setting.
- Making learning more engaging and interactive.

2.3 Game Development and Unity as a Development Tool

2.3.1 Unity Game Engine

The project utilizes Unity, a powerful game development engine, to create an interactive VR-based electronics lab. Unity is widely used in the VR industry due to its:

- Cross-platform support (Meta Quest 2, PC, mobile, AR/VR devices).
- Advanced physics and rendering engine for realistic simulations.
- VR and AR integration tools for creating immersive experiences.
- Extensive community support and asset store for rapid development.

Using Unity, we designed a virtual electronics lab where users can:

- Select electronic components (resistors, capacitors, microcontrollers, sensors, etc.).
- Assemble circuits using interactive drag-and-drop features.
- Test and debug circuits using virtual instruments like oscilloscopes and multimeters.
- Write and upload Arduino code to control the virtual circuits.

2.3.2 Game Development Principles in Education

To make learning engaging, the project applies game development principles, such as:

- Gamification: Adding rewards, challenges, and achievements to enhance motivation.
- Interactive Tutorials: Step-by-step learning with instant feedback.
- Simulation-Based Learning: Hands-on circuit building with guided experiments.

2.4 Meta Quest 2 as the VR Platform

2.4.1 Why Meta Quest 2?

The project is developed for Meta Quest 2, one of the leading standalone VR headsets. The choice of Meta Quest 2 is based on:

- Wireless and Standalone Capability: No need for external PCs or cables.
- Hand and Controller Tracking: Enables natural interaction with virtual components.
- High Performance: Provides a smooth VR experience with high frame rates.

- Large Developer Community: Supports Unity integration with extensive VR tools.

Figure 5: Meta Quest 2

2.4.2 Meta Quest 2 History

The Quest 2 is a standalone virtual reality headset developed by Reality Labs, a division of Meta Platforms. It was officially unveiled on September 16, 2020, and released on October 13, 2020, under the name Oculus Quest 2. As the successor to the original Oculus Quest, the Quest 2 featured significant improvements in performance, display resolution, and processing power while maintaining an affordable price point compared to other VR headsets.

Powered by the Qualcomm Snapdragon XR2 platform with 6GB of RAM, the Quest 2 delivered enhanced graphics, faster processing, and improved AI capabilities. It featured an LCD display with a per-eye resolution of 1832 × 1920, offering a sharper and more immersive visual experience than its predecessor. The headset also supported a refresh rate of up to 120Hz (through software updates), providing smoother motion and reducing motion sickness for users.

One of the key advantages of the Quest 2 was its standalone capability, meaning it did not require a PC or external sensors to operate. However, it also supported Oculus Link (later rebranded as Meta Quest Link)

In 2022, the Oculus Quest 2 was rebranded as the Meta Quest 2 as part of Meta's broader effort to phase out the Oculus brand following Facebook, Inc.'s rebranding as Meta. This change aligned with Meta's vision of expanding the metaverse and solidified its focus on immersive virtual reality experiences.

Despite being succeeded by the Quest 3 in 2023, the Quest 2 remained a popular and affordable entry point into VR, receiving software updates and continued support from Meta.

2.4.3 Meta Quests Specifications

	Developer	Release Date	Life Span	Inductory Price	Operating System	Storage	Display
Oculus Quest	Oculus VR	May 21, 2019	2019–2020	US$95 (64 GB)	Android 10	64 GB	PenTile OLED 1440 × 1600 per eye 72 Hz
Quest 2	Reality Labs	October 13, 2020	2020–2024	US$299 (64 GB) US$249 (128 GB) US$399 (256 GB)	Android 12.1	64 GB, 128 GB, 256 GB	RGB LCD 1832 x 1920 per eye 72 - 120 Hz
Meta Quest 3	Reality Labs	October 10, 2023	2023–present	US$499 (128 GB) US$649 (512 GB)	Meta Horizon OS	128 GB, 512 GB	2 × 2064 × 2208p RGB-stripe LCD+ panels, one per eye 90–120 Hz[
Meta Quest 3S	Reality Labs	October 15, 2024	2024–present	US$299 (128 GB) US$399 (256 GB)	Meta Horizon OS	128 GB, 256 GB	RGB LCD 1832x1920 per eye 90-120 Hz

2.5 Implementation in Meta Quest 2

The system is designed to:

- Recognize user interactions using hand tracking and controllers.
- Allow precise manipulation of virtual components.
- Provide real-time physics simulations for realistic circuit behavior.
- Support voice and text-based Arduino programming for easy coding within VR.

2.6 Electronics and Arduino Integration

2.6.1 Arduino as a Virtual Microcontroller

Arduino serves as the core embedded system for controlling sensors and actuators within the VR environment. Users can:

- Write Arduino code in a virtual interface.
- Simulate how the microcontroller interacts with other components.
- Test circuit responses before implementing them in real hardware.

2.6.2 Virtual Components and Sensors

The system includes a library of virtual electronic components, such as:

- Resistors, capacitors, and diodes.
- LEDs, motors, and displays.
- Sensors (temperature, light, motion, etc.).

2.7 Algorithms:

Algorithms Used in Unity for VR Development

When developing VR projects in Unity, several important algorithms are used to ensure smooth performance, immersive interactions, and realistic physics. Here are some of the key algorithms:

2.7.1 Spatial Mapping Algorithm

Purpose: Allows VR applications to recognize and interact with the physical world.
How it Works: Uses depth sensors and cameras to create a 3D mesh of the environment, enabling realistic collisions and interactions in VR.
Use in Project: Helps users interact with virtual components as if they were in a real lab.

2.7.2 Hand and Controller Tracking Algorithms

Purpose: Converts real-world hand or controller movements into virtual actions.
Techniques Used:

- Inverse Kinematics (IK): Calculates realistic hand and finger positions.
- Quaternion Interpolation (SLERP): Ensures smooth controller rotations and hand movements.
  Use in Project: Ensures precise manipulation of electronic components in the VR lab.

2.7.3 Collision Detection Algorithms

Purpose: Detects when virtual objects interact or collide in VR.
Common Methods:

- Bounding Box & Sphere Collision: Simple, efficient detection for objects.
- Raycasting: Determines where a user is pointing in the virtual environment.
- Mesh Colliders: Enables accurate collision between irregular objects.
  Use in Project: Ensures accurate circuit connections when placing and wiring electronic components.

2.7.4 Object Grabbing and Interaction Algorithms

Purpose: Allows users to grab, move, and manipulate virtual objects naturally.
Techniques Used:

- Spring Joints & Fixed Joints: Simulate real-world physics when picking up objects.
- Velocity-Based Grabbing: Ensures natural object movement when dropped.
  Use in Project: Lets users pick up components like resistors, LEDs, and wires in VR.

2.7.5 Rendering and Optimization Algorithms

Purpose: Ensures high-performance rendering without lag in VR.
Techniques Used:

- Occlusion Culling: Hides objects not in the user's field of view, improving performance.
- Level of Detail (LOD): Reduces polygon count for far-away objects.
- Foveated Rendering: Enhances details where the user is looking while reducing quality in peripheral vision.
  Use in Project: Ensures smooth and immersive VR experiences.

2.7.6 VR Interaction Algorithms

Purpose: Enables intuitive user interactions with virtual components.
Techniques Used:

- Physics-Based Interaction: Uses Unity's Rigidbody system for realistic object movement.
- Gaze-Based Interaction: Allows users to select components by looking at them.
- Gesture Recognition: Detects hand movements to trigger specific actions.
  Use in Project: Helps users place, rotate, and connect circuit components efficiently.

2.7.7 Physics Simulation Algorithms

Purpose: Ensures realistic simulation of circuit components and electrical properties.
Techniques Used:

- Finite State Machines (FSM): Controls interactive elements like switches and sensors.
- RigidBody Dynamics: Simulates component weight, movement, and gravity.
- Soft Body Physics: Improves realistic behavior of flexible wires and cables.
  Use in Project: Makes circuit assembly and movement more realistic.

2.8 Audio Spatialization Algorithm

Purpose: Ensures that sounds come from the correct virtual location.
Techniques Used:

- Head-Related Transfer Function (HRTF): Adjusts sound based on the user's head movement.
- Doppler Effect: Simulates realistic sound shifts as objects move.
  Use in Project: Provides realistic feedback sounds when users connect circuits or interact with components.

2.9 Artificial Intelligence (AI) Assistance Algorithms *(Future Development)*

Purpose: Provides real-time guidance and automated feedback.
Techniques Used:

- Machine Learning Models: Recognize common user mistakes.
- Natural Language Processing (NLP): Allows users to ask for help in VR using voice commands.
  Planned Use in Project: An AI assistant that can provide hints and explanations for electronic circuits.

Chapter 3

System Hardware

3.1 Hardware block Diagram:

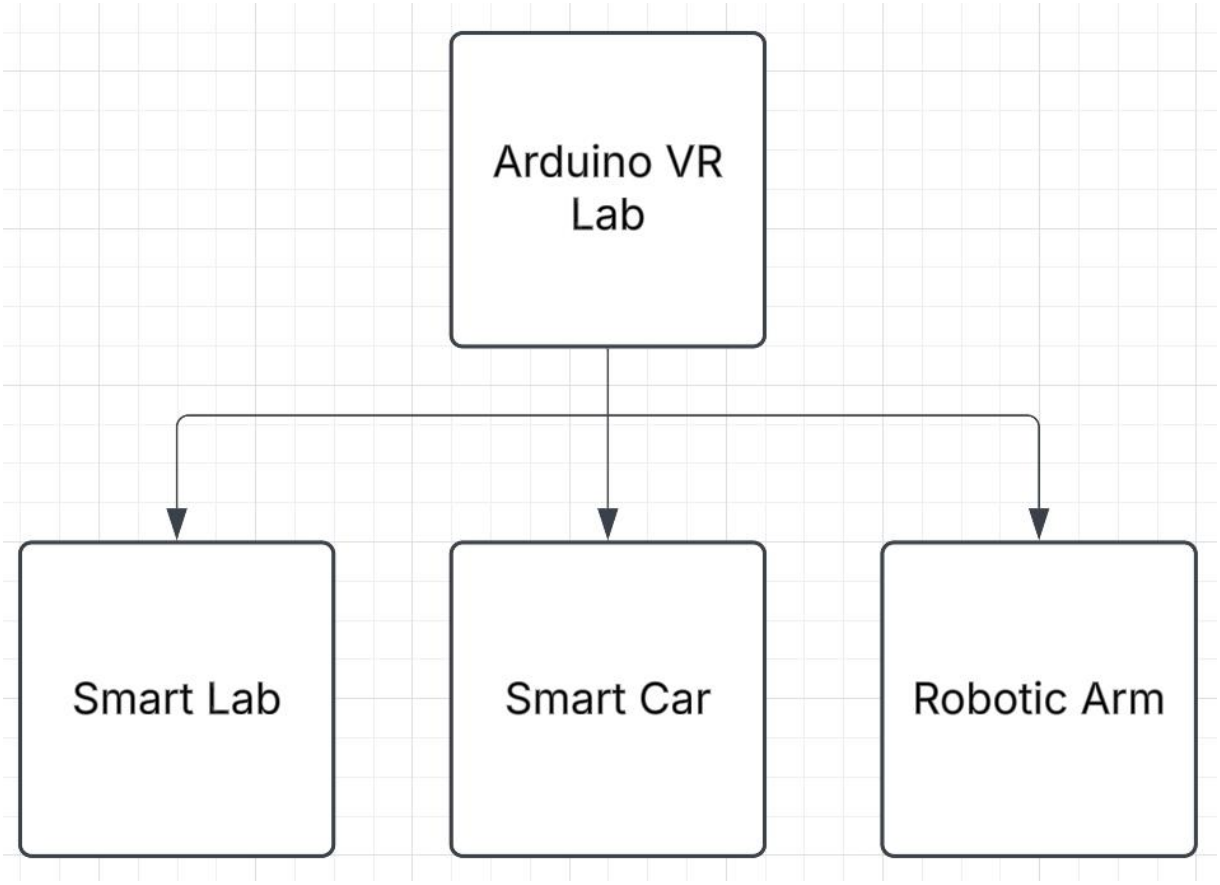

Figure 6: **Hardware block Diagram**

3.2 Controlling a Car from Inside Virtual Reality:

This section focuses on developing a system that allows users to control a car through a virtual reality (VR) environment. The system enables interaction with the car inside a simulated environment that mimics real-world conditions using motion sensors or advanced control tools such as VR headsets and controllers.

Main Components:

- Virtual Reality Environment: A simulated 3D model of the car and its surroundings.
- Control Interfaces: A virtual dashboard that lets the user steer, accelerate, and brake.
- Communication System: Wireless protocols like Wi-Fi or Bluetooth connect the VR environment to the real car.
- Real-time Interaction System: Motion sensors track user inputs and transfer them to the car for real-time execution.

Working Mechanism:

1. The user enters the VR environment using a headset and controllers.
2. The user interacts with the virtual dashboard to control the car.
3. Commands are sent from the VR environment to the actual car via a communication unit.
4. The system provides real-time feedback to enhance control accuracy.

3.3 Controlling a Robot Arm from Inside Virtual Reality:

This section aims to design and develop a system that allows users to control a robotic arm through VR. The simulation enables precise execution of movements using interactive interfaces.

Main Components:

- **VR Simulation: A realistic 3D model of the robotic arm that mirrors real movements.**
- **Control Systems: Hand motion sensors allow the user to control the arm using gestures or controllers.**
- **Communication Unit: Commands are transmitted via Wi-Fi, Bluetooth, or wired connections.**
- **Haptic Feedback System: Provides visual and tactile feedback to improve control accuracy.**

Working Mechanism:

1. **The user wears a VR headset and interacts with a virtual interface.**
2. **The virtual arm moves based on the user's hand gestures.**
3. **Commands are sent to the real robotic arm, executing the movements in real-time.**
4. **Feedback is provided through visual and haptic responses.**

3.4 Project Description:

The project focuses on developing a fully integrated system that allows users to control physical devices (a car and a robotic arm) from a VR environment. The system combines cutting-edge simulation and real-world interaction, enabling remote operations with high precision.

3.5 Complete System:

The fully integrated system consists of interconnected units that work together to create a seamless VR-based control experience.

System Components:

1. **VR Environment: Developed using game engines like Unity or Unreal Engine, featuring realistic 3D models.**
2. **Input Devices: VR headsets, motion controllers, and gloves with sensors.**
3. **Communication Network: Transfers data between VR and physical devices using advanced wireless communication.**
4. **Mechanical Units: The car and robotic arm, equipped with electronic control modules.**
5. **Feedback Mechanisms: Haptic and visual feedback systems enhance user interaction.**

Working Process:

1. **The user interacts with the VR system.**
2. **The system detects gestures and movements.**
3. **Commands are transmitted to physical devices.**
4. **Actions are executed in real-time, with immediate feedback provided.**

Key Features:

- **Accurate & Smooth Control: Enables precise operation of devices.**
- **Realistic Simulation: The VR environment enhances user experience.**
- **Remote Functionality: Allows control from distant locations.**
- **Training & Education: Useful for learning to drive or operate robotic systems.**

Chapter 4

AI Assistant for Interactive Learning in Arduino VR Lab

4.1 Introduction

In the Arduino VR Lab, an AI assistant plays a crucial role in guiding users through the process of understanding and connecting electronic circuits. The assistant enhances the learning experience by providing interactive tutorials, real-time feedback, and troubleshooting support within the virtual environment. By leveraging artificial intelligence, the system ensures that users, whether beginners or experienced electronics enthusiasts, can effectively grasp key concepts and build functional circuits with ease.

4.2 Role of the AI Assistant:

The AI assistant in our Unity VR project acts as a virtual mentor, offering:

Step-by-Step Guidance: The assistant provides detailed instructions on how to connect components such as resistors, LEDs, sensors, motors, and microcontrollers like the Arduino Uno.

Real-Time Error Detection: It monitors circuit connections and alerts users of incorrect wiring, potential short circuits, or missing components.

Voice and Text Interaction: Users can ask the AI assistant questions using voice commands or text input, allowing for a more natural learning process.

Simulation Assistance: The AI explains how each component functions, demonstrates electrical laws in real-time, and assists in running circuit simulations before real-world implementation.

Component Suggestions: Based on the user's project requirements, the assistant suggests suitable electronic components and explains their usage.

4.3 Understanding Circuit Connections in the Virtual Lab

1. Setting Up a Simple LED Circuit

One of the fundamental projects in electronics is lighting up an LED.

The AI assistant helps users connect the following components:

1. Arduino Uno
2. LED
3. Resistor (330Ω)
4. Breadboard
5. Jumper wires

Steps in the Virtual Lab:

1. Placing Components: The AI guides the user to place the Arduino and breadboard correctly in the virtual workspace.

2. Connecting the Resistor: The AI explains the importance of a resistor in limiting current and ensures proper placement.

3. Wiring the Circuit: The user connects the anode of the LED to a digital pin on the Arduino and the cathode to the ground via the resistor.

4. Uploading the Code: The assistant provides a pre-written script for blinking the LED and explains each line of the code.

5. Running the Simulation: Users can test their circuit in the VR environment before applying it in real life.

6. Controlling a Motor Using an L298N Driver

7. For more advanced projects, the AI assistant helps users control motors using an L298N motor driver.

The setup involves:

1. Arduino Uno
2. L298N motor driver module
3. DC motor
4. External power source (e.g., 12V battery)
5. Jumper wires
6. Guided Steps:
7. Explaining the L298N Module: The assistant details how the module functions and its pin configurations.
8. Connecting the Motor: It assists in wiring the motor to the output terminals of the L298N.
9. Interfacing with the Arduino: The AI explains the role of control pins and how to send signals from the Arduino.
10. Power Management: The assistant ensures the user understands voltage requirements and avoids circuit damage.
11. Code Implementation: Users receive a sample code to control motor speed and direction, with explanations for each function.

Chapter 5

Website and social media

5.1 Website:

The official website of the Arduino VR Lab serves as a comprehensive platform that provides users with detailed insights into the project, including:
Introduction to the Project – A clear overview of what Arduino VR Lab is, its objectives, and how it enhances electronics education.
How to Use the Platform – Step-by-step guides and tutorials to help users understand how to navigate and utilize the virtual electronics lab effectively.
Supported Platforms – Information about the Meta Quest 2 and other compatible VR hardware for running the lab.
Online Store – A dedicated e-commerce section where users can purchase:

- VR headsets (such as Meta Quest 2)
- VR accessories (controllers, sensors, and cables)
- Electronic components (Arduino boards, sensors, and prototyping kits)
  This ensures that users can easily access all essential tools and components without the hassle of searching for them separately.
  Community Forum – A dedicated space where users can connect, share experiences, ask questions, and exchange ideas about electronics and VR learning.
  Contact & Support – A direct communication channel allowing users to submit feedback, requests, or business proposals, fostering collaboration and continuous project improvement.

5.2 Social Media:

To engage with the community and keep users updated, Arduino VR Lab is active on various social media platforms:

YouTube Channel: Features tutorials, demos, and live sessions showcasing the functionalities of Arduino VR Lab. It also provides step-by-step projects for users to follow.

Facebook Page: Serves as a hub for news, updates, user discussions, and project showcases. Users can share their experiences, interact with the developers, and participate in polls and surveys.

Twitter/X: Provides real-time updates, announcements about new features, and discussions on VR technology and electronics education.

Instagram: Features visual content, including VR lab setups, user-generated content, and behind-the-scenes development updates.

Discord/Telegram Community: A place for real-time discussions, troubleshooting, and collaborations between developers and users.

By maintaining a strong online presence, the Arduino VR Lab ensures continuous user engagement, real-time support, and an interactive learning experience beyond the virtual lab itself.

Chapter 6

3D Modeling

6.1 What is Blender:

Blender is an open-source software used for creating 3D models and producing animated works. It is considered one of the most powerful free tools available for designers and developers, offering extensive capabilities in various fields such as game design, animated films, engineering visualization, visual effects, and virtual reality. It was first developed in 1995 by the company "Not a Number" and became open-source in 2002, which contributed to the growth of its user community and continuous development.

Figure 7: blender 4.2 logo

6.2 Features of Blender:

6.2.1 3D Modeling:

Provides advanced tools for creating and modifying geometric shapes with high precision, including sculpting and mesh editing.

6.2.2 Texturing and Materials:

Supports the creation of realistic materials using techniques like PBR (Physically Based Rendering

6.2.3 Animation:

Includes professional tools for animating characters and objects, such as rigging and motion simulation.

6.2.4 Visual Effects (VFX):

Supports adding effects like particles and physical simulations.

6.2.5 Integration with Game Engines:

Models can be exported to engines like Unity and Unreal Engine.

6.2.6 Virtual Reality (VR) Support:

Enables the creation of VR-compatible content

6.3 Using Blender in a Graduation Project:

In our graduation project, we used Blender to design 3D models of electrical components, aiming to use them in a virtual reality simulation environment. The main goal was to create models that accurately mimic reality, allowing for scientific experiments in a safe and effective virtual environment.

6.4 Work Stages:

6.4.1 Designing Electrical Components:

- 3D models of components like resistors and capacitors were created using Blender's precise modeling tools.

- Focus was placed on simulating actual dimensions to ensure the realism of the models.

6.4.2 Adding Textures and Materials:

- PBR techniques were used to create realistic materials reflecting the properties of components like metal and plastic.

- Fine details such as printed labels and specifications were added to the components.

6.4.3 Optimizing Lighting and Reflections:

- Actual lighting conditions were simulated to ensure accurate representation of the models in the VR environment.

6.4.4 Exporting to the VR Engine:

- Models were exported to Unity, ensuring they interacted correctly within the simulation.

6.5 Importance of Using Blender in the Project:

6.5.1 High Precision:

Components were designed with high accuracy to realistically simulate scientific experiments.

6.5.2 Flexibility:

Models could be customized to suit different needs.

6.5.3 Safety:

Provided a safe virtual environment for conducting experiments without the need for expensive or hazardous real components.

3D Models used in the project

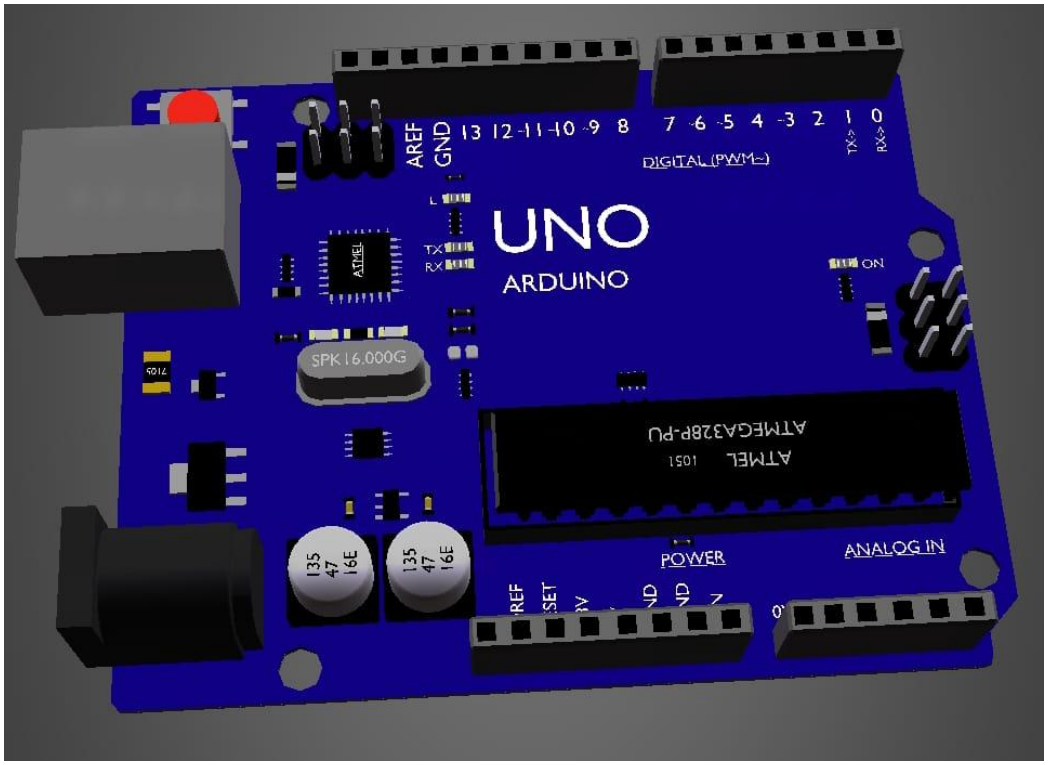

Figure 8: ARDUINO

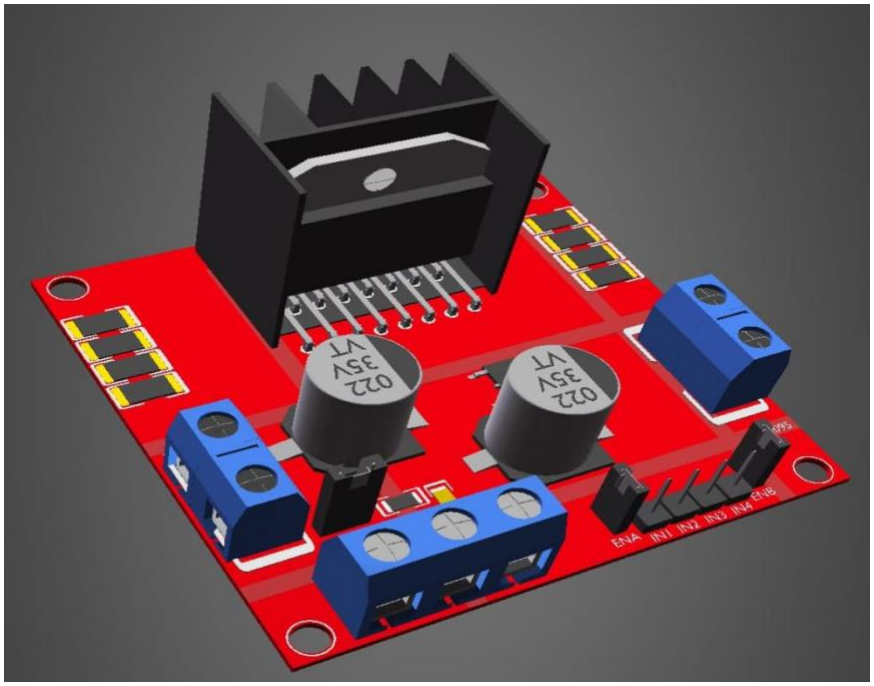

Figure 9: L298 Motor Driver Module

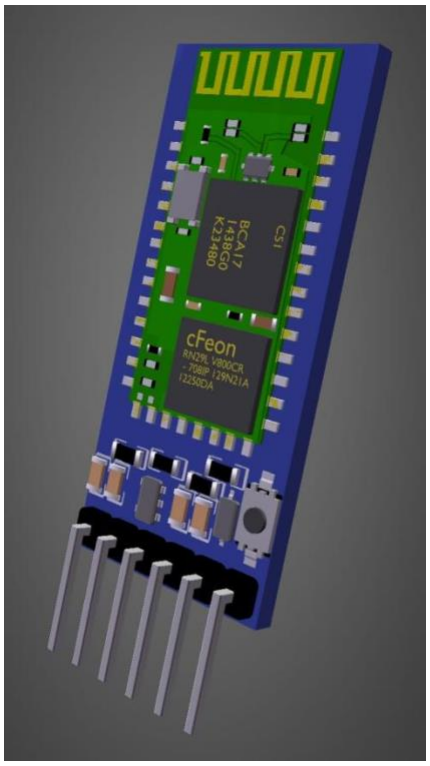

Figure 10: Bluetooth Module HC-05

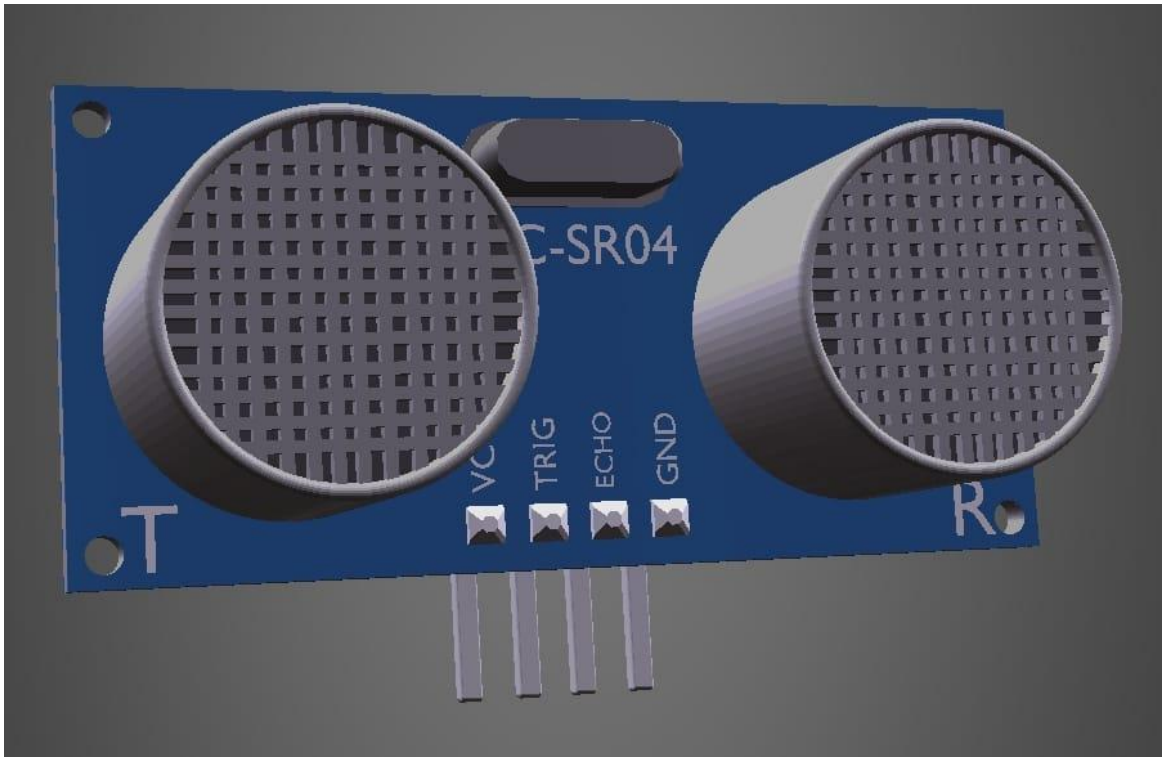

Figure 11: Ultrasonic Sensor (HC-SR04)

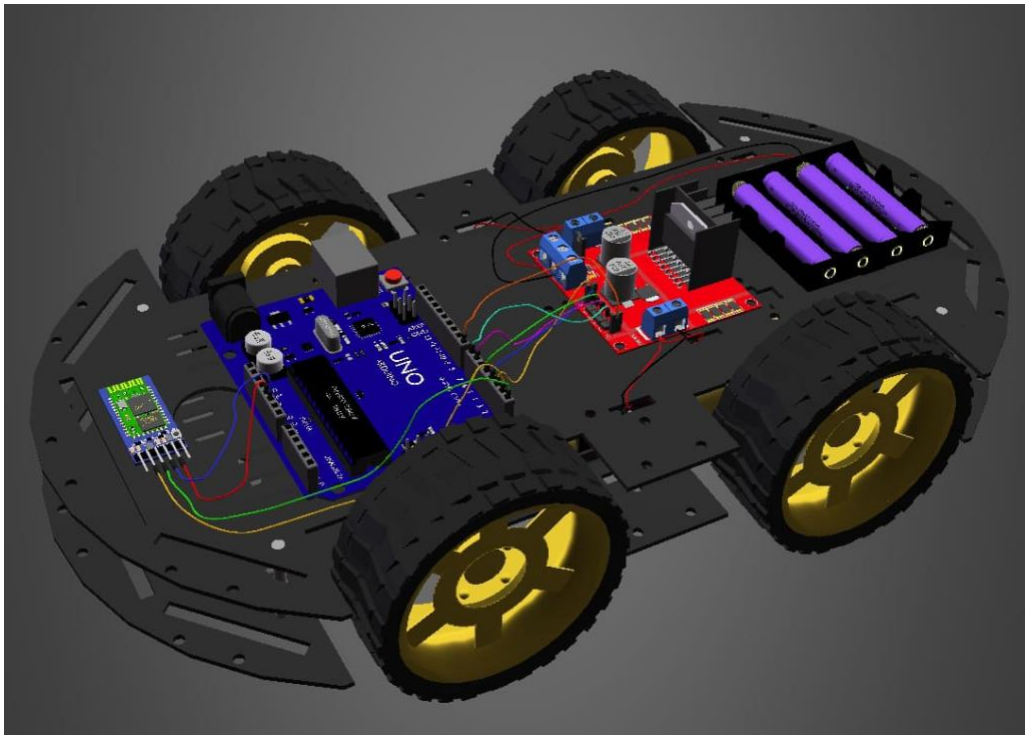

Figure 12: car robot

Figure 13: AC POWER11

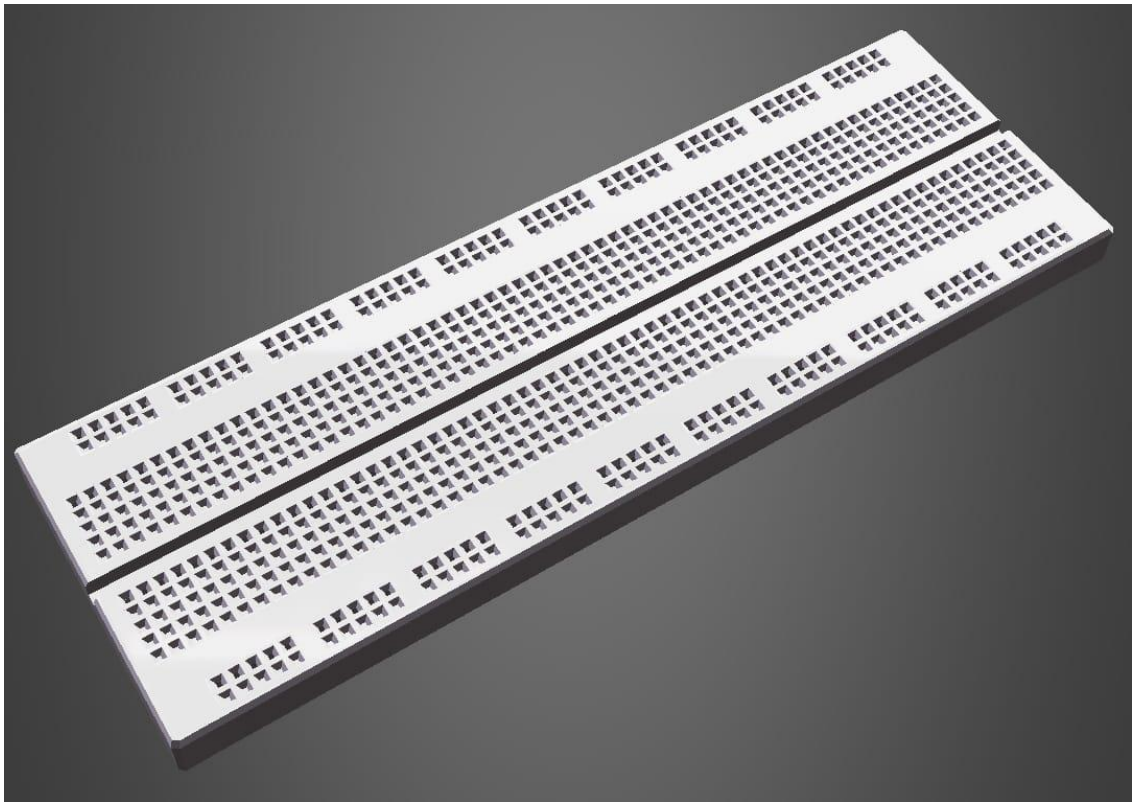

Figure 14: Breadboard

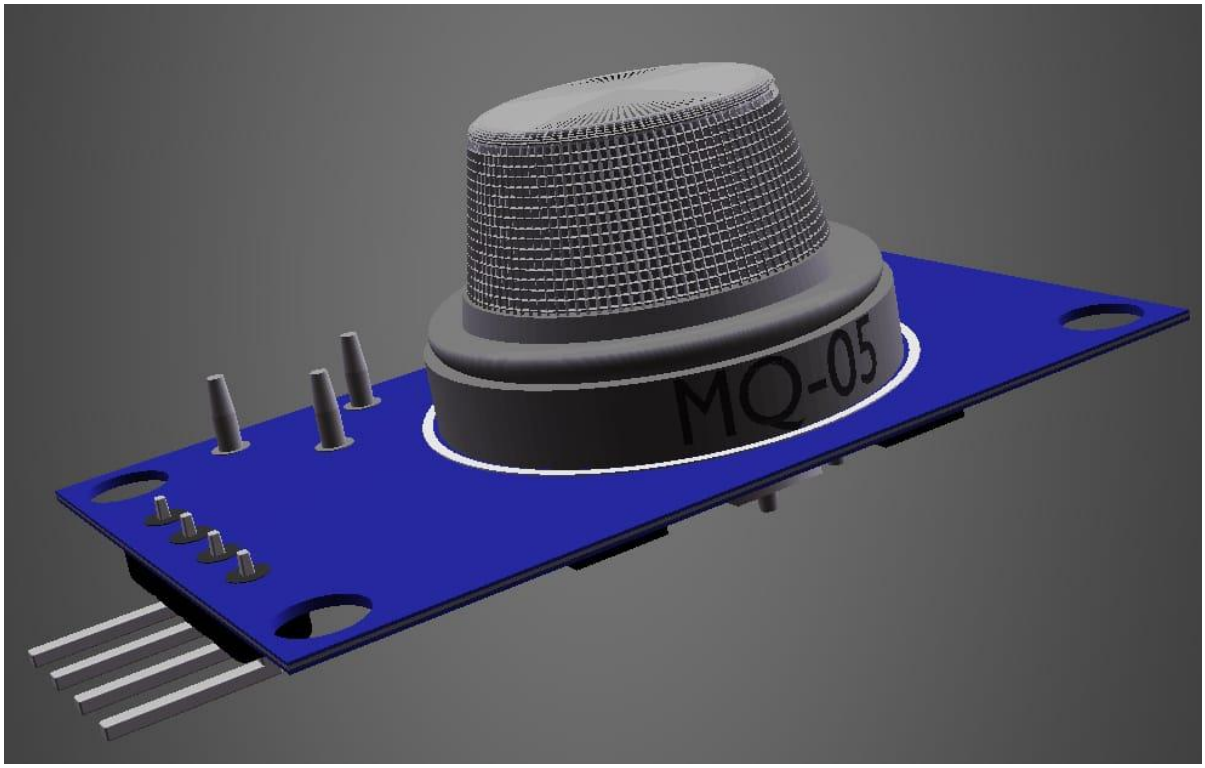

Figure 15: MQ-4 Natural Gas Methane Gas Sensor Module

Figure 16: AMPERE METER

Figure 17: OHM METER

Figure 18: VOLT METER

Figure 19: RFID - RC522

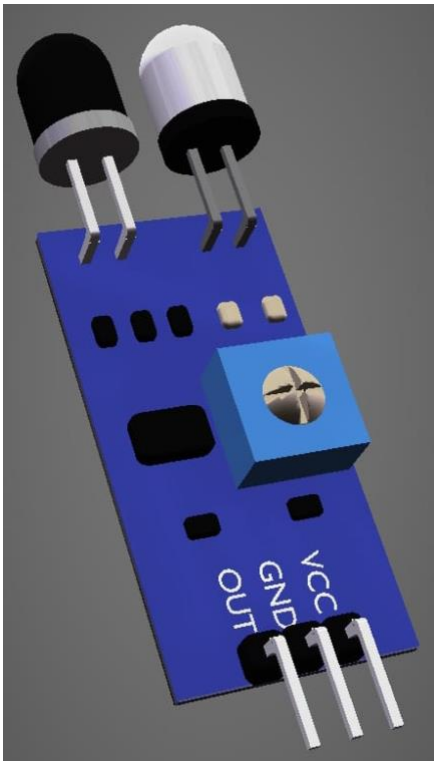

Figure 20: IR Sensor

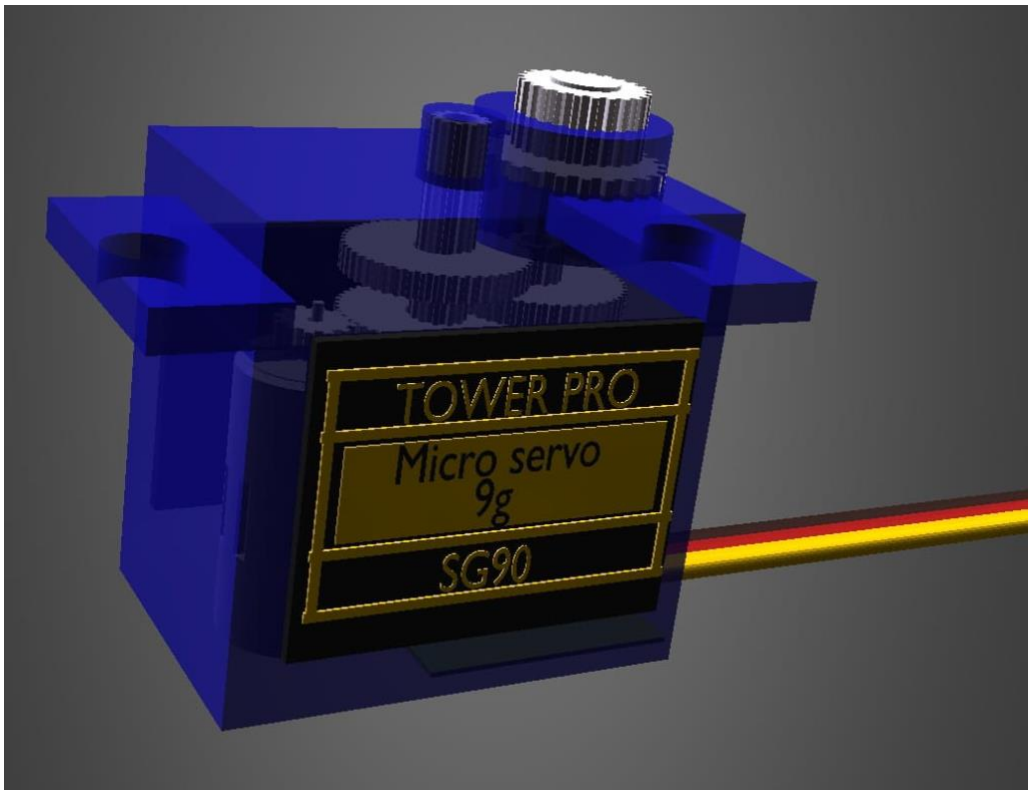

Figure 21: MICRO SERVO SG90

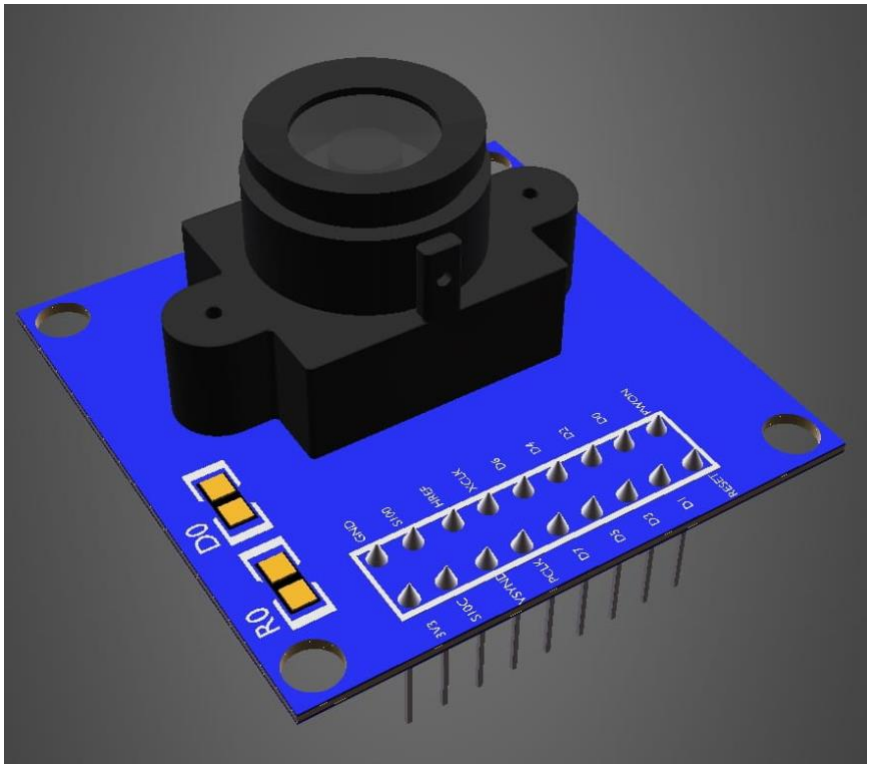

Figure 22: Camera Module (OV7670)

Figure 23: Battery 9V

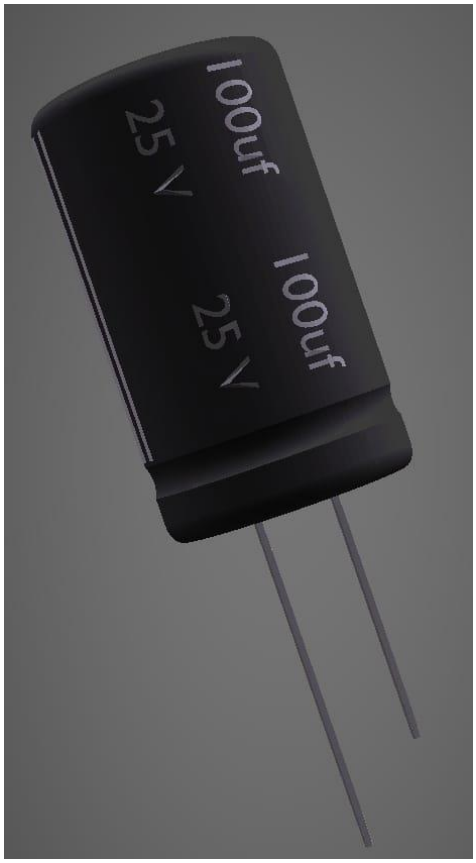

Figure 24: Capacitor

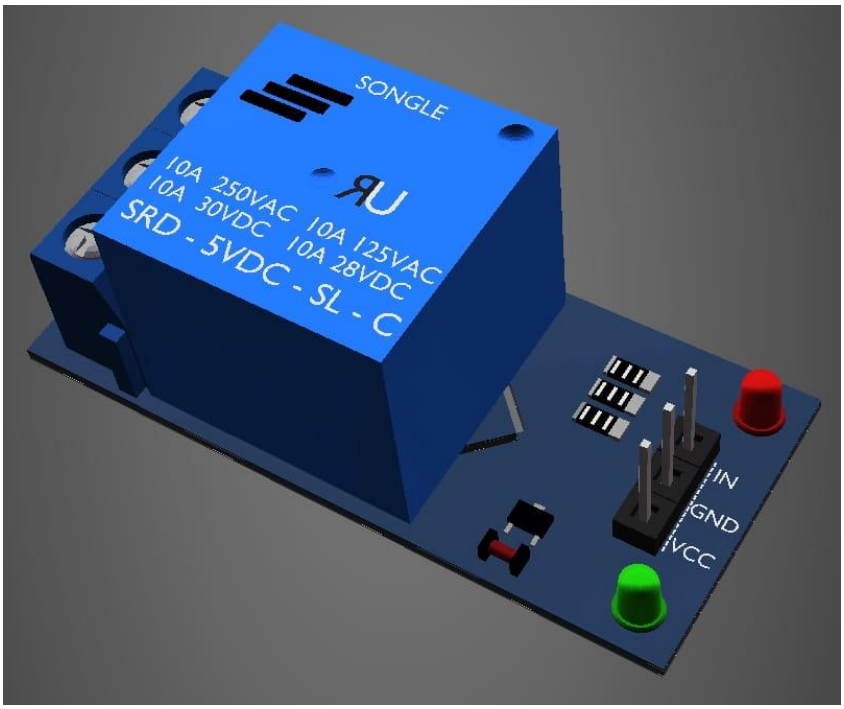

Figure 25: Relay Module

Figure 26: Laptop

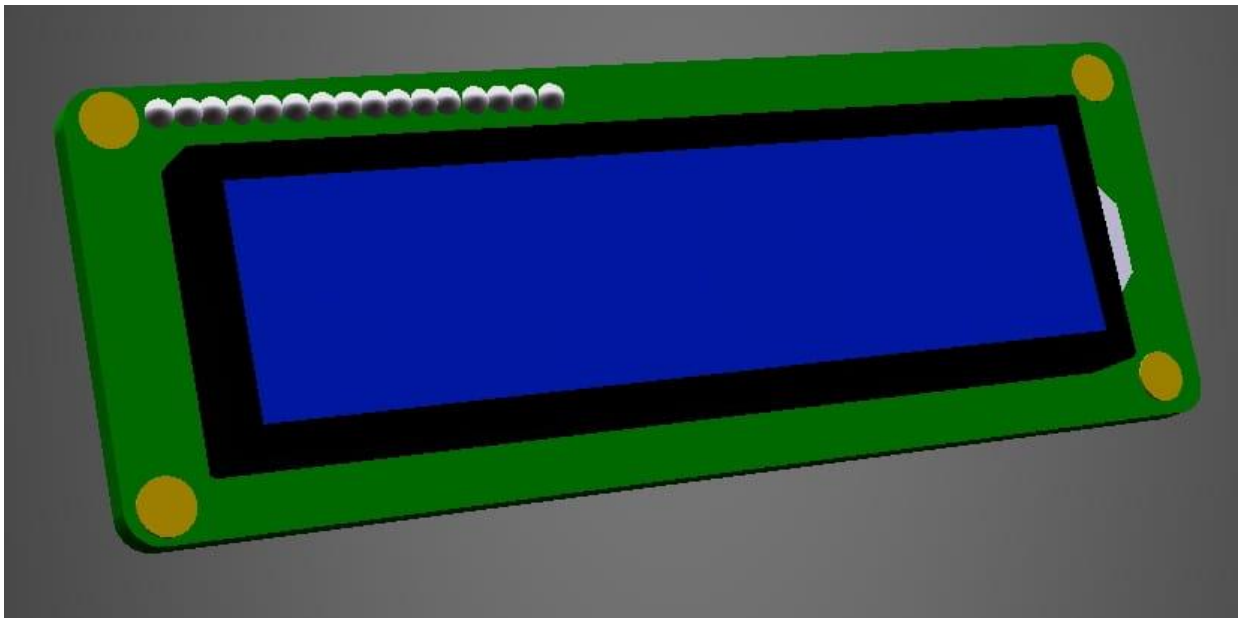

Figure 27: LCD 16*2

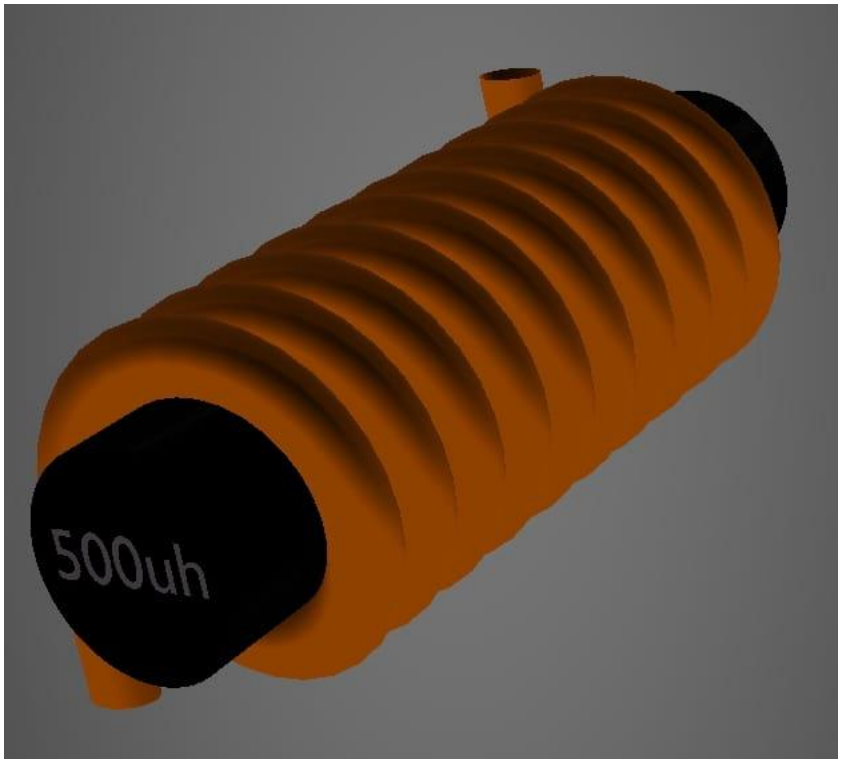

Figure 28: Coil

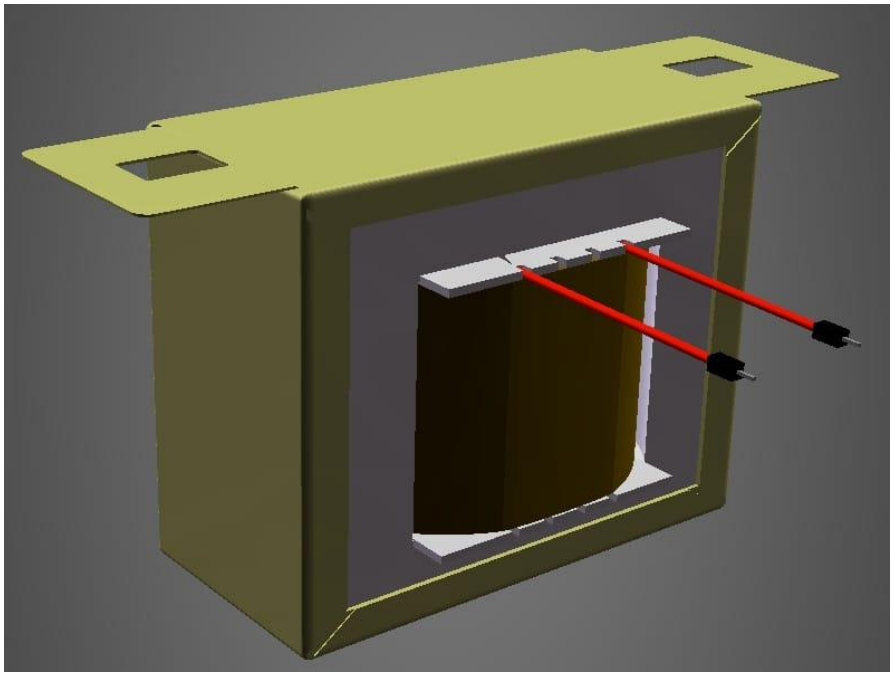

Figure 29: transformer

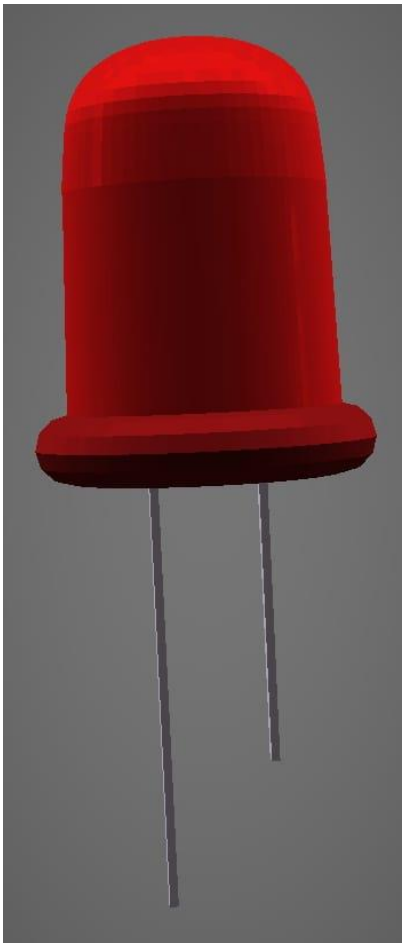

Figure 30: LED

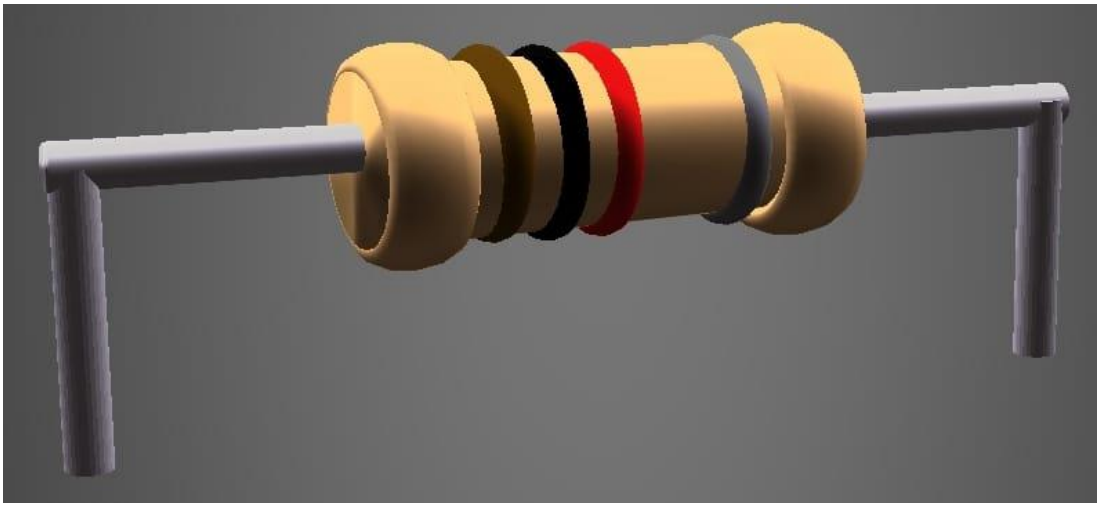

Figure 31: Resistor

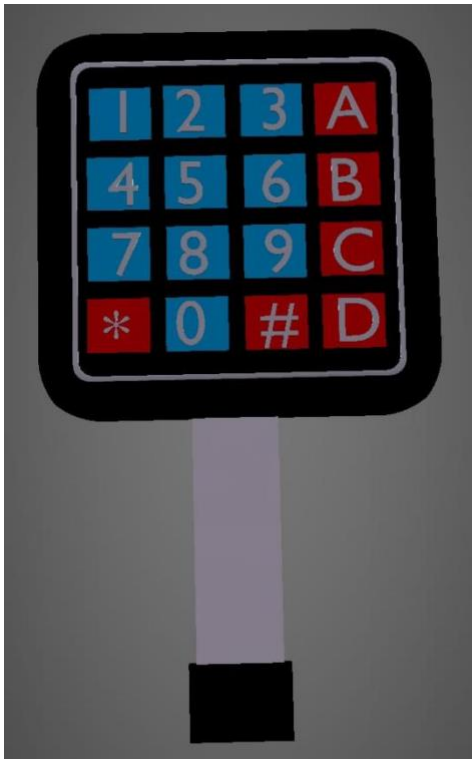

Figure 32: Keypad 4*4

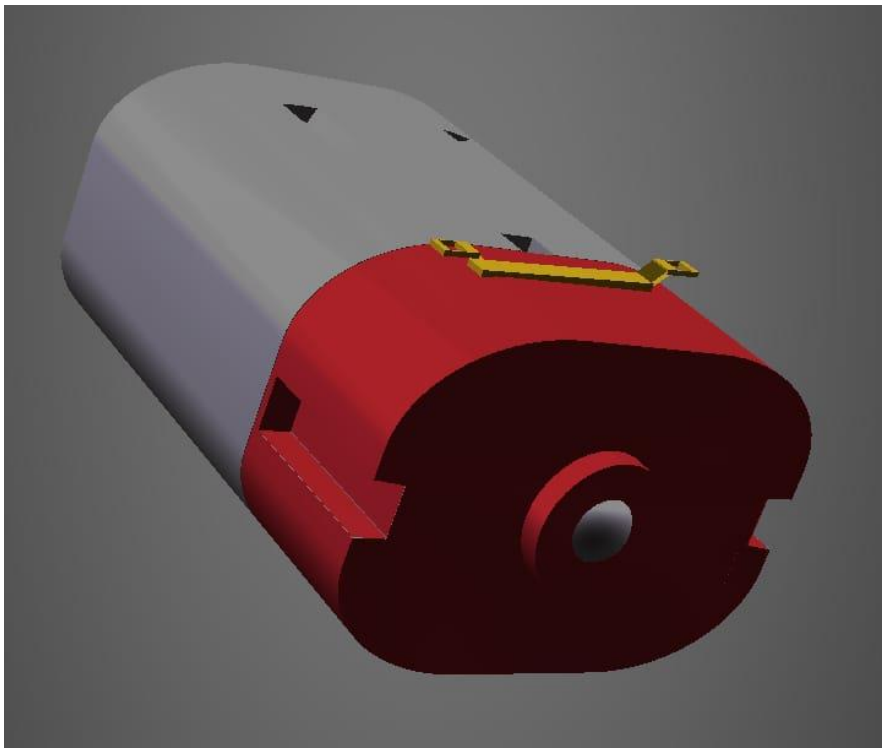

Figure 33: DC MOTOR

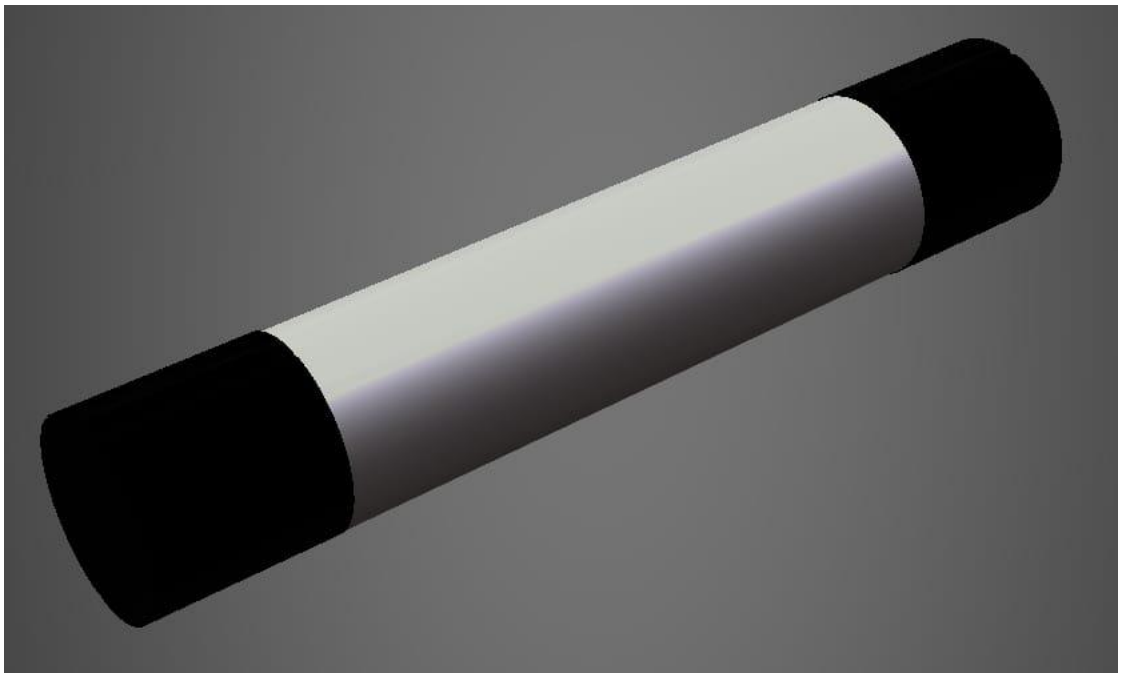

Figure 34: FUSE

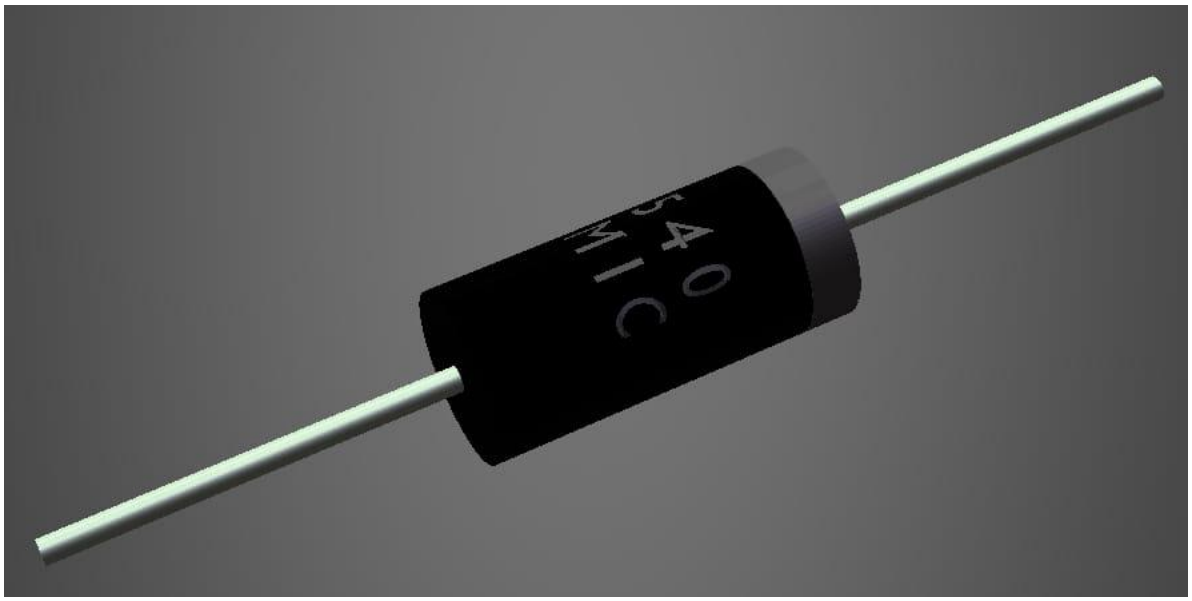

Figure 35: DIODE

Figure 36: 2N2222F

Figure 37: Potentiometer

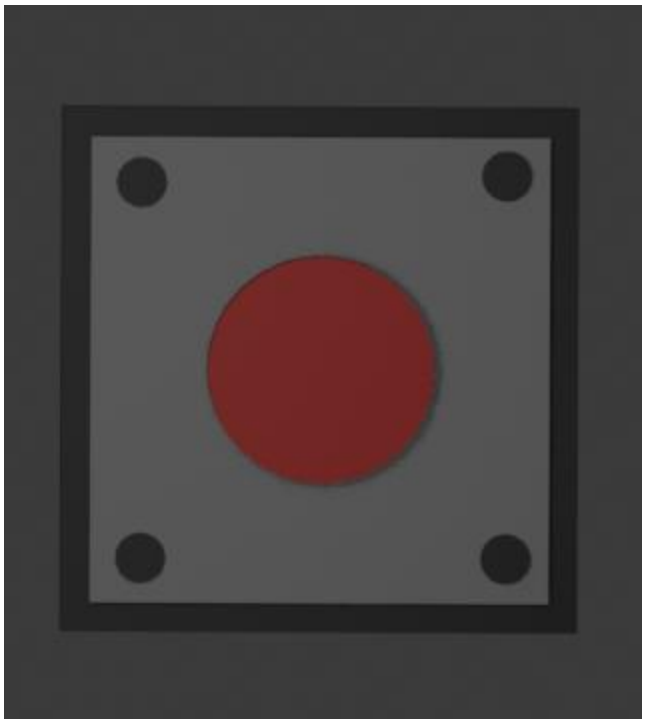

Figure 38: Button

Chapter 7

Result and future work

7.1 Find Results:

The **Arduino VR Lab** successfully provides an **interactive and immersive virtual environment** for learning electronics and programming with Arduino. Key results include:

- **Realistic Virtual Circuit Simulation:** Users can **build, test, and modify electronic circuits** within a VR environment, ensuring a practical and engaging learning experience.

- **User-Friendly Interaction:** The use of **Meta Quest 2** enables **natural hand-tracking and controller-based interactions**, providing a seamless and intuitive experience.

- **Enhanced Learning Outcomes:** Early feedback from users indicates **improved engagement and understanding** of electronic circuit concepts.

- **Gamification and Interactive Tutorials:** A **step-by-step guide** has been implemented, making the learning process **more structured and interactive**.

- **Comprehensive Virtual Electronics Lab:** A **fully equipped educational lab** has been integrated, bringing together all necessary **tools and electronic components** in one place.

- **Wide Range of Electronic Components:** The VR lab includes essential **electronic components**, such as:
    - **Resistors** of various values
    - **Inductors**
    - **Capacitors**
    - **Diodes and Transistors**
    - **Microcontrollers and Sensors**

- **Advanced Measurement Tools:** The lab also provides **virtual measurement devices**, including:
    - **Voltmeter** (Voltage measurement)
    - **Ammeter** (Current measurement)
    - **Ohmmeter** (Resistance measurement)

By integrating **all essential electronic components and tools in a single immersive environment**, the **Arduino VR Lab** enables users to experiment, practice, and enhance their knowledge **without physical limitations or risks**.

7.2 Photos:

Figure: 39 (Lab view)

Figure: 40 (Lab view)

Figure: 41 (Measuring tools)

7.3 Explanation:

The **Arduino VR Lab** is designed to provide an immersive and interactive learning experience for electronics and Arduino programming. By leveraging **VR technology**, users can build and test electronic circuits in a **virtual environment** without the need for physical components.

Key aspects of the project include:

- **Realistic Circuit Simulation:** Users can assemble and connect electronic components in a **virtual workspace**, experiencing real-time circuit behavior.
- **Interactive Learning Approach:** The VR lab features a **user-friendly interface** that supports **hand-tracking and controller-based interactions**, making it easy to manipulate components.
- **Comprehensive Educational Lab:** A fully equipped environment with essential tools, such as **resistors, capacitors, inductors, sensors, and measurement devices** like voltmeters and ammeters.
- **Gamification & Tutorials:** A structured, **step-by-step learning system** guides users through building circuits, programming Arduino, and troubleshooting common issues.

By integrating **real-world physics, virtual electronics, and intuitive interaction methods**, the **Arduino VR Lab** offers an innovative way to learn electronics and programming **safely and effectively**.

7.4 Conclusion:

The **Arduino VR Lab** represents a significant advancement in **educational technology**, offering students and hobbyists a **safe, cost-effective, and engaging** way to learn electronics. Through the use of **VR and interactive simulations**, users can:

- Gain **hands-on experience** in circuit design and Arduino programming.
- Improve **problem-solving skills** through **real-time feedback** and experimentation.
- Learn in a **risk-free environment** without worrying about component damage or incorrect wiring.
- Experience **a fully equipped electronics lab** in VR, eliminating the need for expensive hardware.

Moving forward, the project aims to **expand the range of components, enhance AI-driven guidance, and integrate real-time collaboration features**, allowing multiple users to work together in the **same VR environment**. The **Arduino VR Lab** is a step towards the future of **interactive and immersive engineering education**.

7.5 Future work:

The **Arduino VR Lab** continues to evolve, with the team actively refining its core features. Current efforts are centered on enhancing module compatibility, streamlining the user interface for seamless interaction, and expanding the library of supported components to offer greater flexibility for creators.

Future developments for the Arduino VR Lab will focus on integrating advanced simulation capabilities, enabling real-time circuit analysis, and improving interactive tutorials to support users of all skill levels. The team also plans to incorporate cloud-based project sharing, allowing collaboration between makers, educators, and students worldwide.

As the platform matures, the goal is to provide a seamless blend of virtual and physical prototyping, making electronics more accessible and engaging. Whether for education, research, or innovation, the Arduino VR Lab aspires to be a cutting-edge tool that redefines how users interact with and understand electronics in a digital space.

Looking ahead, the vision for Arduino VR Lab is to establish itself as a dynamic ecosystem where users can prototype, experiment, and innovate with electronics in a fully immersive virtual environment.

Some of the upcoming works:

Figure 42: Arduino Smart Lab (In dev.)

Figure 43: Inside the Arduino Smart Lab (In dev.)

Appendix

Wire Spawning Script:

```
using UnityEngine;

public class WireSpawner : MonoBehaviour
{
    [Header("Wire Prefab Setup")]
    [Tooltip("Assign the prefab that contains Object A, Object B, and the connecting wire.")]
    public GameObject wirePrefab;

    [Header("Spawn Location")]
    [Tooltip("Assign the Transform where the wire should be spawned.")]
    public Transform spawnPoint;

    /// <summary>
    /// Spawns the wire prefab at the specified spawn point.
    /// </summary>
    public void SpawnWire()
    {
        // Check if wirePrefab and spawnPoint are assigned
        if (wirePrefab == null)
        {
            Debug.LogError("WirePrefab is not assigned in the inspector!");
            return;
        }
        if (spawnPoint == null)
        {
            Debug.LogError("SpawnPoint is not assigned in the inspector!");
            return;
        }

        // Instantiate the wire prefab at the spawn point's position and rotation
        Instantiate(wirePrefab, spawnPoint.position, spawnPoint.rotation);
        Debug.Log("Wire spawned at " + spawnPoint.position);
    }
}
```

Explanation: This script allows the connection wires to spawn when a specific button is pressed.

Data Transmission Script:

```
using UnityEngine;

public class ValueProvider : MonoBehaviour
{
    [Header("Electrical Values")]
    [SerializeField] private float resistance;  // Resistance in ohms
    [SerializeField] private float voltage;     // Voltage in volts
    [SerializeField] private float current;     // Current in amperes
    [SerializeField] private float capacitance; // Capacitance in farads

    // Public properties to access the values
    public float Resistance => resistance;
    public float Voltage => voltage;
    public float Current => current;
    public float Capacitance => capacitance;

    // Debugging information
    private void OnValidate()
    {
        if (resistance < 0f)
        {
            Debug.LogWarning("Resistance cannot be negative. Resetting to 0.");
            resistance = 0f;
        }

        if (voltage < 0f)
        {
            Debug.LogWarning("Voltage cannot be negative. Resetting to 0.");
            voltage = 0f;
        }

        if (current < 0f)
        {
            Debug.LogWarning("Current cannot be negative. Resetting to 0.");
            current = 0f;
        }

        if (capacitance < 0f)
        {
            Debug.LogWarning("Capacitance cannot be negative. Resetting to 0.");
            capacitance = 0f;
        }
    }

    // Method to provide values to a ValueConsumer
    public void ProvideValuesTo(ValueConsumer consumer)
    {
        if (consumer != null)
        {
            consumer.AddValuesFromProvider(this);
            Debug.Log($"Provided values to ValueConsumer: Resistance={resistance}, Voltage={voltage},
Current={current}, Capacitance={capacitance}");
        }
        else
        {
            Debug.LogWarning("Target ValueConsumer is null. Cannot provide values.");
        }
    }
}
```

Explanation: This script allows the transmission of data values between electronics for example: when a resistor is connected to a battery the resistance value is transferred to the battery.

Connection Wires Script:

```
using UnityEngine;

namespace BNG
{
    /// <summary>
    /// This script scales and positions an object smoothly between two other objects (cubes) to act as
a wire or line.
    /// </summary>
    public class ScaleBetweenObjectsSmooth : MonoBehaviour
    {
        public Transform Object1;   // First object (e.g., Cube 1)
        public Transform Object2;   // Second object (e.g., Cube 2)

        public float smoothingFactor = 0.1f;  // Controls the smoothness of the movement

        private Vector3 velocity = Vector3.zero;

        void Update()
        {
            if (Object1 != null && Object2 != null)
            {
                // Smoothly move the object between the two positions
                Vector3 targetPosition = (Object1.position + Object2.position) / 2;
                transform.position = Vector3.SmoothDamp(transform.position, targetPosition, ref
velocity, smoothingFactor);

                // Calculate the distance between the two objects and update the scale (Z axis)
                float distance = Vector3.Distance(Object1.position, Object2.position);

                // Smoothly scale the object (adjusting Z scale only)
                Vector3 targetScale = new Vector3(transform.localScale.x, transform.localScale.y,
distance);
                transform.localScale = Vector3.Lerp(transform.localScale, targetScale,
smoothingFactor);

                // Optionally, make the object look at the second object to give it a wire-like
appearance
                transform.LookAt(Object2.position, transform.up);
            }
        }
    }
}
```

Explanation: The primary wiring script is responsible for establishing connections between various components, ensuring seamless communication and interaction.

Plugins Script:

```
using UnityEngine;

public class PlaceableObject : MonoBehaviour
{
    public string objectName; // Name or value for this object
    public MeshRenderer specificMeshRenderer; // MeshRenderer for this specific object

    private void Reset()
    {
        // Automatically find and assign the MeshRenderer if it exists
        if (specificMeshRenderer == null)
        {
            specificMeshRenderer = GetComponent<MeshRenderer>();
        }
    }

    /// <summary>
    /// Toggles visibility of the specific mesh based on a name match.
    /// </summary>
    /// <param name="targetName">The name to compare.</param>
    public void ToggleMeshBasedOnName(string targetName)
    {
        if (specificMeshRenderer != null)
        {
            // Enable the mesh if the names match, otherwise disable it
            specificMeshRenderer.enabled = objectName == targetName;
        }
    }
}
```

Explanation: This script ensures that the electronic components are installed on the plugins correctly.

Electrical Calculations and Data Reception Script:

Part 1:

```
using UnityEngine;
using TMPro;  // Import TextMeshPro namespace

public class ValueConsumer : MonoBehaviour
{
    // Internal variables for Resistance, Voltage, and Current
    [Header("Electrical Values")]
    [SerializeField] private float resistance;
    [SerializeField] private float voltage;
    [SerializeField] public float current;

    // UI Text references for displaying Voltage, Current, and Resistance
    [Header("UI Elements")]
    public TextMeshProUGUI voltageText;  // Reference to TMP UI Text to display voltage
    public TextMeshProUGUI currentText;  // Reference to TMP UI Text to display current
    public TextMeshProUGUI resistanceText; // Reference to TMP UI Text to display resistance

    // Event to notify when voltage or other values change
    public delegate void ValueChanged(float voltage, float current, float resistance);
    public event ValueChanged OnValueChanged;

    // Method to receive and add data from ValueProvider
    public void AddValuesFromProvider(ValueProvider provider)
    {
        if (provider == null)
        {
            Debug.LogWarning("Provided ValueProvider is null.");
            return;
        }

        // Add provider values to internal values
        resistance += provider.Resistance;
        voltage += provider.Voltage;
        current += provider.Current;


        // Log the added values
        Debug.Log($"Added values - Resistance: {resistance}, Voltage: {voltage}, Current: {current}");

        // Calculate current based on Ohm's Law (I = V / R)
        CalculateCurrent();

        // Update the UI
        UpdateDisplay();

        // Trigger the value change event (for external listeners)
        OnValueChanged?.Invoke(voltage, current, resistance);
    }
```

Part 2:

```
// Method to add values from another ValueConsumer
public void AddValuesFromConsumer(ValueConsumer consumer)
{
    if (consumer == null)
    {
        Debug.LogWarning("Provided ValueConsumer is null.");
        return;
    }

    // Add consumer values to this consumer's values
    resistance += consumer.resistance;
    voltage += consumer.voltage;
    current += consumer.current;

    // Log the added values
    Debug.Log($"Added values from another ValueConsumer - Resistance: {resistance}, Voltage: {voltage},
Current: {current}");

    // Calculate current based on Ohm's Law
    CalculateCurrent();

    // Update the UI
    UpdateDisplay();

    // Trigger the value change event
    OnValueChanged?.Invoke(voltage, current, resistance);
}

// Method to transfer values to another ValueConsumer
public void TransferValuesTo(ValueConsumer targetConsumer)
{
    if (targetConsumer != null)
    {
        targetConsumer.SetValues(resistance, voltage, current);
        Debug.Log("Values transferred to another ValueConsumer.");
    }
    else
    {
        Debug.LogWarning("Target ValueConsumer is null.");
    }
}

// Method to set values manually (e.g., when transferring)
public void SetValues(float resistanceValue, float voltageValue, float currentValue)
{
    resistance = resistanceValue;
    voltage = voltageValue;
    current = currentValue;

    // Log the values
    Debug.Log($"Values set - Resistance: {resistance}, Voltage: {voltage}, Current: {current}");

    // Update UI
    UpdateDisplay();

    // Trigger the value change event
    OnValueChanged?.Invoke(voltage, current, resistance);
}
```

Part 3:

```
// Method to clear all values
public void ClearValues()
{
    resistance = 0f;
    voltage = 0f;
    current = 0f;

    Debug.Log("Values cleared!");

    // Update UI to reflect cleared values
    UpdateDisplay();

    // Trigger value change event after clearing
    OnValueChanged?.Invoke(voltage, current, resistance);
}

// Method to calculate current using Ohm's Law: I = V / R
private void CalculateCurrent()
{
    if (resistance != 0f)
    {
        current = voltage / resistance;
        Debug.Log($"Calculated Current: {current}A");
    }
    else
    {
        Debug.LogWarning("Resistance is zero. Cannot calculate current.");
        current = 0f;  // Set current to 0 to avoid invalid calculations
    }
}
```

Part 4:

```
    // Method to update the display in the UI
    private void UpdateDisplay()
    {
        if (voltageText != null)
        {
            voltageText.text = $"Voltage: {voltage}V";
        }
        else
        {
            Debug.LogWarning("Voltage TextMeshProUGUI reference is missing!");
        }

        if (currentText != null)
        {
            currentText.text = $"Current: {current}A";
        }
        else
        {
            Debug.LogWarning("Current TextMeshProUGUI reference is missing!");
        }

        if (resistanceText != null)
        {
            resistanceText.text = $"Resistance: {resistance}Ω";
        }
        else
        {
            Debug.LogWarning("Resistance TextMeshProUGUI reference is missing!");
        }
    }

    // Method to update and recalculate data based on current values
    public void UpdateData()
    {
        // Recalculate current using Ohm's Law
        CalculateCurrent();

        // Update the UI
        UpdateDisplay();

        // Log the update action
        Debug.Log("Data updated and recalculated.");

        // Trigger the value change event
        OnValueChanged?.Invoke(voltage, current, resistance);
    }
}
```

Explanation: This script handles all electrical calculations and facilitates the sharing of values between connected components.

Led Script:

```
using UnityEngine;
using UnityEngine.UI;  // Import for UI elements (if you're using buttons)

public class LightControl : MonoBehaviour
{
    [Header("Point Light Settings")]
    public Light pointLight;  // Reference to the point light
    public Color newColor = Color.red;  // New color for the light

    // Start is called before the first frame update
    void Start()
    {
        if (pointLight == null)
        {
            Debug.LogWarning("Point light reference is missing!");
        }
    }

    // Method to change the light color on a button click or any other event
    public void ChangeLightColor()
    {
        if (pointLight != null)
        {
            pointLight.color = newColor;  // Change the color of the point light
            Debug.Log($"Light color changed to: {newColor}");
        }
    }
}
```

Explanation: The LED script manages the illumination of LEDs, ensuring they display the appropriate colors as required.

Joystick Script:

```
using UnityEngine;
using UnityEngine.Events;

public class JoystickAngleEvents : MonoBehaviour
{
    public Transform joystick; // Assign the joystick's Transform
    public UnityEvent onForward;
    public UnityEvent onBackward;
    public UnityEvent onLeft;
    public UnityEvent onRight;

    private void Update()
    {
        if (joystick == null)
        {
            Debug.LogError("Joystick Transform is not assigned!");
            return;
        }

        Vector2 joystickDirection = new Vector2(joystick.localPosition.x, joystick.localPosition.z);
        Debug.Log("Joystick Direction: " + joystickDirection);

        if (joystickDirection.magnitude > 0.1f) // Ignore small movements
        {
            float angle = Mathf.Atan2(joystickDirection.y, joystickDirection.x) * Mathf.Rad2Deg;
            angle = (angle + 360) % 360; // Normalize to 0-360
            Debug.Log("Joystick Angle: " + angle);

            if (angle >= 45 && angle < 135)
            {
                Debug.Log("FORWARD");
                onForward.Invoke();
            }
            else if (angle >= 225 && angle < 315)
            {
                Debug.Log("BACKWARD");
                onBackward.Invoke();
            }
            else if (angle >= 135 && angle < 225)
            {
                Debug.Log("LEFT");
                onLeft.Invoke();
            }
            else
            {
                Debug.Log("RIGHT");
                onRight.Invoke();
            }
        }
    }
}
```

Explanation: This script enables the joystick to control the RC car and its movement directions.

Fuse Script:

Part 1:

Arduino VR Lab

```
using UnityEngine;
using UnityEngine.Events;

public class Fuse : MonoBehaviour
{
    [Header("Fuse Settings")]
    [SerializeField] private float currentThreshold = 5f;  // Current threshold to trigger fuse disconnect
    [SerializeField] private ValueConsumer targetValueConsumer;  // The ValueConsumer whose current will be monitored
    [SerializeField] private GameObject fuseObject;  // The fuse object to show the fuse state (Optional)

    [Header("Events After Fuse Blow")]
    public UnityEvent fuseBlownEvents;  // List of events that will happen after the fuse is blown

    // Called to check if the fuse should disconnect based on the current
    private void Update()
    {
        if (targetValueConsumer == null)
        {
            Debug.LogWarning("Target ValueConsumer is not assigned.");
            return;
        }
```

Part 2:

```
        // Check if the current exceeds the threshold and trigger fuse disconnect logic
        if (targetValueConsumer.current > currentThreshold)
        {
            TriggerFuse();
        }
    }

    // Method to trigger fuse disconnection logic and invoke events
    private void TriggerFuse()
    {
        if (fuseObject != null)
        {
            fuseObject.SetActive(false);  // Disables the fuse object, representing fuse disconnection
            Debug.Log("Fuse disconnected due to excess current.");
        }
        else
        {
            Debug.LogWarning("Fuse object not assigned.");
        }

        // Trigger events after the fuse is blown
        fuseBlownEvents?.Invoke();
    }
```

Explanation: This script defines the core logic of the fuse.

Bluetooth Module Script:

```
using UnityEngine;
using TMPro;
using UnityEngine.UI;

public class BluetoothSender : MonoBehaviour
{
    public BluetoothManager bluetoothManager;
    public TMP_InputField inputField;
    public Button sendButton;
    public TMP_Text debugText;

    void Start()
    {
        if (bluetoothManager == null)
        {
            Debug.LogError("BluetoothManager is not assigned!");
            return;
        }

        if (sendButton != null)
        {
            sendButton.onClick.AddListener(SendMessage);
        }
    }

    void SendMessage()
    {
        if (inputField != null && !string.IsNullOrEmpty(inputField.text))
        {
            bluetoothManager.SendData(inputField.text);
            LogToUI("Sent: " + inputField.text);
        }
        else
        {
            LogToUI("No message to send!");
        }
    }

    void LogToUI(string message)
    {
        if (debugText != null)
        {
            debugText.text += "\n" + message;
        }
        Debug.Log(message);
    }
}
```

Explanation: This script establishes the connection between the VR system and the RC car.

Battery Script:

```
using UnityEngine;

public class Battery : MonoBehaviour
{
    public Transform PositiveTerminal;  // الطرف الموجب للبطارية
    public Transform NegativeTerminal;  // الطرف السالب للبطارية
    public float Voltage = 9f;  // قيمة الجهد الافتراضية للبطارية (يمكن تعديلها من المحرر)

    // يمكنك إضافة وظائف مثل التفاعل مع المكونات الأخرى أو حساب التوصيلات
}
```

Explanation: This script defines the core logic of the battery.

Capacitor Script:

```
using UnityEngine;

public class Capacitor : MonoBehaviour
{
    public Transform InputPoint;  // الطرف المدخل للمكثف
    public Transform OutputPoint; // الطرف المخرج للمكثف
    public float Capacitance = 100f; // قيمة السعة بالفاراد (مثال: 1 ميكرو فاراد)

    // يمكنك إضافة وظائف إضافية هنا للتفاعل مع الأسلاك أو حساب الجهد
}
```

Explanation: This script defines the core logic of the capacitor.

Diode Script:

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

public class Diode : MonoBehaviour
{
    public float forwardVoltageThreshold = 0.7f; // الجهد الأمامي المطلوب لتمرير التيار (مثلاً 0.7 فولت لدايود السيليكون)
    public float currentFlow = 0f; // مقدار التيار الذي يمر عبر الدايود (بالأمبير)

    // الجهد المطبق على الأنود والكاثود
    public float anodeVoltage = 0f;
    public float cathodeVoltage = 0f;

    // Start is called before the first frame update
    void Start()
    {
        // قيمة مبدئية للجهد
        anodeVoltage = 1f;  // جهد الأنود (فولت)
        cathodeVoltage = 0f;  // جهد الكاثود (فولت)
    }

    // Update is called once per frame
    void Update()
    {
        // محاكاة سلوك الدايود
        SimulateDiodeBehavior();
    }

    // محاكاة سلوك الدايود
    void SimulateDiodeBehavior()
    {
        // إذا كان الجهد الأمامي أكبر من الجهد العكسي وكان الجهد أكبر من العتبة
        if (anodeVoltage > cathodeVoltage + forwardVoltageThreshold)
        {
            // السماح بمرور التيار في الاتجاه الصحيح
            currentFlow = 1f;  // التيار يمر (بالأمبير)
            Debug.Log("Current flows through the diode in the forward direction.");
        }
        else if (cathodeVoltage > anodeVoltage)
        {
            // منع مرور التيار في الاتجاه العكسي
            currentFlow = 0f;  // لا تيار يمر
            Debug.Log("No current flows in the reverse direction.");
        }
        else
        {
            // في حالة الجهد غير كافي لتمرير التيار
            currentFlow = 0f;  // لا تيار يمر
            Debug.Log("No current flows. Voltage is too low.");
        }
    }
}
```

Explanation: This script defines the core functionality of the diode and its control over voltage and current direction.

Coil Script:

```
using UnityEngine;

public class Inductor : MonoBehaviour
{
    public Transform InputPoint;  // الطرف المدخل للملف
    public Transform OutputPoint; // الطرف المخرج للملف
    public float Inductance = 100f; // قيمة الحث بالهنري (مثال: 1 ملي هنري)

    // يمكنك إضافة وظائف إضافية هنا للتفاعل مع الأسلاك أو حساب التيار
}
```

Explanation: This script defines the core logic of the coil.

Resistor Script:

```
using UnityEngine;

public class Resistor : MonoBehaviour
{
    public Transform InputPoint;  // الطرف المدخل للمقاومة
    public Transform OutputPoint; // الطرف المخرج للمقاومة
    public float Resistance = 10f; // Resistance value in Ohms

    // يمكنك إضافة وظائف أخرى هنا مثل التفاعل مع الأسلاك أو حساب الجهد
}
```

Explanation: This script defines the core logic of the resistors.

NPN Transistor Script:

Arduino VR Lab

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

public class NPN : MonoBehaviour
{
      //كنقاط اتصال NPN أطراف الترانزستور
    public Transform collectorPoint;   //النقطة التي تمثل الجامع (C)
    public Transform basePoint;        //النقطة التي تمثل القاعدة (B)
    public Transform emitterPoint;     //النقطة التي تمثل المصدر (E)

    public bool isActive;              //حالة الترانزستور (يعمل أم لا)
    private float thresholdVoltage = 0.7f;  //الجهد اللازم لتشغيل الترانزستور

    // Start is called before the first frame update
    void Start()
    {
        isActive = false;  //يبدأ الترانزستور في وضع الإيقاف
    }

    // Update is called once per frame
    void Update()
    {
          //محاكاة تشغيل الترانزستور بناءً على الجهد عند القاعدة
        SimulateTransistor();
    }

    void SimulateTransistor()
    {
          //تحقق إذا تم توصيل القاعدة بجهد مناسب
        if (basePoint != null && emitterPoint != null)
        {
            float baseToEmitterVoltage = GetVoltage(basePoint, emitterPoint);

            if (baseToEmitterVoltage > thresholdVoltage)
            {
                isActive = true;  //الترانزستور يعمل
                Debug.Log("الترانزستور NPN يعمل: تدفق التيار من الجامع إلى المصدر.");
            }
            else
            {
                isActive = false;  //الترانزستور لا يعمل
                Debug.Log("الترانزستور NPN لا يعمل: لا يوجد تدفق للتيار.");
            }
        }
    }
```

Explanation: This script defines the core logic of the NPN transistor.

PNP Transistor Script:

Arduino VR Lab

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

public class PNP : MonoBehaviour
{
      //كنقاط اتصال PNP أطراف الترانزستور
    public Transform collectorPoint;   //النقطة التي تمثل الجامع (C)
    public Transform basePoint;        //النقطة التي تمثل القاعدة (B)
    public Transform emitterPoint;     //النقطة التي تمثل المصدر (E)

    public bool isActive;              //حالة الترانزستور (يعمل أم لا)
    private float thresholdVoltage = 0.7f;   //الجهد اللازم لتشغيل الترانزستور

    // Start is called before the first frame update
    void Start()
    {
        isActive = false;   //يبدأ الترانزستور في وضع الإيقاف
    }

    // Update is called once per frame
    void Update()
    {
         //محاكاة تشغيل الترانزستور بناءً على الجهد عند القاعدة
        SimulateTransistor();
    }

    void SimulateTransistor()
    {
         //تحقق إذا تم توصيل القاعدة بجهد مناسب
        if (basePoint != null && emitterPoint != null)
        {
            float baseToEmitterVoltage = GetVoltage(basePoint, emitterPoint);

            if (baseToEmitterVoltage < -thresholdVoltage)  //فرق الجهد عكسي مقارنة بـ NPN
            {
                isActive = true;   //الترانزستور يعمل
                Debug.Log("الترانزستور PNP يعمل: تدفق التيار من المصدر إلى الجامع.");
            }
            else
            {
                isActive = false;   //الترانزستور لا يعمل
                Debug.Log("الترانزستور PNP لا يعمل: لا يوجد تدفق للتيار.");
            }
        }
    }
```

Explanation: This script defines the core logic of the PNP transistor.

Potentiometer Script:

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

public class potentiometer_10K : MonoBehaviour
{
    public float maxResistance = 10000f; // (أوم 10,000) أقصى قيمة مقاومة
    public float minResistance = 0f; // (0 أوم) أقل قيمة مقاومة
    public Transform controlLever;  // الطرف الذي يمثل ذراع التحكم

    private float currentResistance;

    // Start is called before the first frame update
    void Start()
    {
        // تعيين المقاومة الأولية بناءً على زاوية الذراع
        UpdateResistance();
    }

    // Update is called once per frame
    void Update()
    {
        // تحديث المقاومة بناءً على زاوية الذراع
        UpdateResistance();
    }

    void UpdateResistance()
    {
        // Y نحصل على زاوية الذراع في المحور
        float angle = controlLever.rotation.eulerAngles.y;

        // نحول الزاوية إلى قيمة بين 0 و 1
        float normalizedAngle = angle / 360f;

        // نحسب المقاومة بناءً على الزاوية
        currentResistance = Mathf.Lerp(minResistance, maxResistance,
normalizedAngle);

        // طباعة قيمة المقاومة للمراجعة
        Debug.Log("Current Resistance: " + currentResistance);
    }
}
```

Explanation: This script defines the core logic of the potentiometer.

Transformer Script:

```
using System;
using UnityEngine;

public class TransformerSimulation : MonoBehaviour
{
    // المدخلات
    public float inputVoltageAC = 220f; // الجهد المتردد الداخل (فولت)
    public float turnsRatio = 220f / 12f; // نسبة عدد اللفات
    public float rectifierEfficiency = 0.9f; // كفاءة التوحيد (90%)
    public float smoothingCapacitance = 1000f; // سعة المكثف (ميكروفاراد)
    public float loadResistance = 10f; // المقاومة الحمل (أوم)

    void Start()
    {
        // 1. خفض الجهد باستخدام المحول
        float outputVoltageAC = inputVoltageAC / turnsRatio;
        Debug.Log($"Step Down Transformer Output (AC): {outputVoltageAC} V AC");

        // 2. توحيد الجهد باستخدام دائرة التوحيد
        float rectifiedVoltage = outputVoltageAC * rectifierEfficiency; // تقليل الجهد
بسبب كفاءة التوحيد
        Debug.Log($"Rectified Voltage (DC Approx): {rectifiedVoltage} V DC");

        // 3. تنعيم الجهد باستخدام المكثف
        float smoothedVoltage = CalculateSmoothedVoltage(rectifiedVoltage,
loadResistance, smoothingCapacitance);
        Debug.Log($"Smoothed Voltage (DC): {smoothedVoltage} V DC");
    }

    // حساب الجهد بعد التنعيم باستخدام معادلة تفريغ المكثف
    float CalculateSmoothedVoltage(float rectifiedVoltage, float resistance, float
capacitance)
    {
        float rippleVoltage = 1 / (capacitance * resistance); // الجهد المموج
        return rectifiedVoltage - rippleVoltage; // الجهد المتبقي بعد التنعيم
    }
}
```

Explanation: This script defines the core logic of the Transformer.

Blue Highlighted Hands Script:

Arduino VR Lab

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

namespace Autohand{
    public delegate void HandEvent(Hand hand);

    public class HandTriggerAreaEvents : MonoBehaviour{
        [Header("Trigger Events Settings")]
        [Tooltip("Whether or not first hand to enter should take ownership and be the only one to call events")]
        public bool oneHanded = true;
        public HandType handType = HandType.both;
        [Tooltip("Whether or not to call the release event if exiting while grab event activated")]
        public bool exitTriggerRelease = true;
        [Tooltip("Whether or not to call the release event if exiting while grab event activated")]
        public bool exitTriggerUnsqueeze = true;

        [Header("Events")]
        public UnityHandEvent HandEnter;
        public UnityHandEvent HandExit;
        public UnityHandEvent HandGrab;
        public UnityHandEvent HandRelease;
        public UnityHandEvent HandSqueeze;
        public UnityHandEvent HandUnsqueeze;

        //For Programmers <3
        public HandEvent HandEnterEvent;
        public HandEvent HandExitEvent;
        public HandEvent HandGrabEvent;
        public HandEvent HandReleaseEvent;
        public HandEvent HandSqueezeEvent;
        public HandEvent HandUnsqueezeEvent;

        List<Hand> hands;
        bool grabbing;
        bool squeezing;
```

Explanation: This script displays a highlighted blue hand when the VR hand approaches an intractable object, helping the user identify and interact with it more easily.

VR Hands Script:

Arduino VR Lab

```
using NaughtyAttributes;
using System;
using System.Collections;
using System.Collections.Generic;
#if UNITY_EDITOR
using UnityEditor;
#endif
using UnityEngine;
using UnityEngine.Serialization;
using UnityEngine.XR;

namespace Autohand {
    public enum FingerEnum {
        index,
        middle,
        ring,
        pinky,
        thumb
    }
Coroutine _grabRoutine;
        Coroutine grabRoutine {
            get { return _grabRoutine; }
            set {
                if(value != null && _grabRoutine != null) {
                    StopCoroutine(_grabRoutine);
                    if(holdingObj != null) {
                        holdingObj.body.linearVelocity = Vector3.zero;
                        holdingObj.body.angularVelocity = Vector3.zero;
                        holdingObj.beingGrabbed = false;
                    }
                    BreakGrabConnection();
                }
                _grabRoutine = value;
            }
        }
        Coroutine tryGrab;
        Coroutine highlightRoutine;
        float startGrabDist;
```

Explanation: This is the primary VR hands script, responsible for managing hand movements and controls.

References

- Ardiny, H. and Khanmirza, E., 2018, October. The role of AR and VR technologies in education developments: opportunities and challenges. In *2018 6th rsi international conference on robotics and mechatronics (icrom)* (pp. 482-487). IEEE.

- Bryson, S., 1995. Approaches to the successful design and implementation of VR applications. *Virtual reality applications*, (1), pp.3-15.

- Holly, M., Pirker, J., Resch, S., Brettschuh, S. and Gütl, C., 2021. Designing VR experiences–expectations for teaching and learning in VR. *Educational Technology & Society*, *24*(2), pp.107-119.

- Jerald, J., 2015. *The VR book: Human-centered design for virtual reality*. Morgan & Claypool.

- Kreylos, O., 2008, December. Environment-independent VR development. In *International Symposium on Visual Computing* (pp. 901-912). Berlin, Heidelberg: Springer Berlin Heidelberg.

- Seymour, N.E., 2008. VR to OR: a review of the evidence that virtual reality simulation improves operating room performance. *World journal of surgery*, *32*, pp.182-188.

- Huang, T.K., Yang, C.H., Hsieh, Y.H., Wang, J.C. and Hung, C.C., 2018. Augmented reality (AR) and virtual reality (VR) applied in dentistry. *The Kaohsiung journal of medical sciences*, *34*(4), pp.243-248.

- Bierbaum, A., Just, C., Hartling, P., Meinert, K., Baker, A. and Cruz-Neira, C., 2001, March. VR Juggler: A virtual platform for virtual reality application development. In *Proceedings IEEE virtual reality 2001* (pp. 89-96). IEEE.

- Brooks, F.P., 1999. What's real about virtual reality?. *IEEE Computer graphics and applications*, *19*(6), pp.16-27.

- Ferrell, B.R., Rhiner, M. and Ferrell, B.A., 1993. Development and implementation of a pain education program. *Cancer*, *72*(S11), pp.3426-3432.

- Giroux, H.A., 1978. Developing educational programs: Overcoming the hidden curriculum. *The Clearing House*, *52*(4), pp.148-151.

- Ramírez-Montoya, M.S., Andrade-Vargas, L., Rivera-Rogel, D. and Portuguez-Castro, M., 2021. Trends for the future of education programs for professional development. *Sustainability*, *13*(13), p.7244.

- Kremer, M., 2003. Randomized evaluations of educational programs in developing countries: Some lessons. *American Economic Review*, *93*(2), pp.102-106.

- Korthagen, F., Loughran, J. and Russell, T., 2006. Developing fundamental principles for teacher education programs and practices. *Teaching and teacher education*, *22*(8), pp.1020-1041.

- Banzi, M. (2005). Getting Started with Arduino. O'Reilly Media.

- Burdea, G., & Coiffet, P. (2020). Virtual Reality Technology. John Wiley & Sons.

- Evans, D. (2011). The Internet of Things: How the Next Evolution of the Internet is Changing Everything. Cisco.

- Faggin, F. (1971). The History of Intel 4004. Intel Corporation.

- Fisher, S., McGreevy, M., Humphries, J., & Robinett, W. (1986). Virtual Environment Display System. NASA Ames Research Center.

- Ganssle, J. (1992). The Art of Designing Embedded Systems. Elsevier.

- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

- Heilig, M. (1962). Sensorama Simulator Patent. US Patent No. 3,050,870.

- Lanier, J. (1989). Virtual Reality: The Visionary Perspective. VPL Research.

- Luckey, P. (2012). The Development of Oculus Rift. Oculus VR.
- Mindell, D. (2002). Between Human and Machine: Feedback, Control, and Computing before Cybernetics. Johns Hopkins University Press.

- Mindell, D. (2008). Digital Apollo: Human and Machine in Spaceflight. MIT Press.

- Redmond, K., & Smith, T. (1980). Project Whirlwind: The History of a Pioneer Computer. Digital Press.

- Steinicke, F. (2016). Being Really Virtual: Immersive Natives and the Future of Virtual Reality. Springer.

- Sutherland, I. (1968). A Head-Mounted Three-Dimensional Display. Proceedings of the AFIPS Fall Joint Computer Conference.

- Takeda, H. (1995). The History of VR in Gaming. SEGA Research & Development.

- Wheatstone, C. (1838). Contributions to the Physiology of Vision – Part the First: On Some Remarkable and Hitherto Unobserved Phenomena of Binocular Vision. Philosophical Transactions of the Royal Society.

- Wolf, W. (1994). Computers as Components: Principles of Embedded Computing System Design. Morgan Kaufmann.