

TABLE OF CONTENTS

| | |
|---|------|
| TABLE OF CONTENTS | i |
| ACKNOWLEDGEMENTS | v |
| LIST OF FIGURES | vi |
| LIST OF ABBREVIATIONS | viii |
| ABSTRACT | ix |
| সারসংক্ষেপ | x |
| CHAPTER ONE INTRODUCTION | 1 |
| 1.1 Introduction | 1 |
| 1.2 What is Robot | 1 |
| 1.3 Types of Robot | 2 |
| 1.3.1 Legged Robot | 2 |
| 1.3.2 Wheeled Robot | 2 |
| 1.3.3 Stationary or fixed robots | 2 |
| 1.4 Bomb Disposal Robot | 3 |
| 1.5 Project Objectives | 4 |
| 1.6 Organization of the Project | 4 |
| CHAPTER TWO LITERATURE REVIEW | 5 |
| 2.1 Review Summary | 5 |
| 2.2 Motivation | 6 |
| 2.3 Existing bomb disposal robot and Their Limitations | 6 |
| 2.3.1 Talon | 6 |
| 2.3.2 MK3 – CALIBER | 7 |
| 2.3.3 KNIGHT | 8 |
| 2.3.4 LT2-F Bulldog | 8 |
| 2.3.5 HD2-S Mastiff | 9 |

| | |
|------------------------------------|----|
| CHAPTER 03 DESIGN AND CONSTRUCTION | 10 |
| 3.1 Introduction | 10 |
| 3.2 Design | 10 |
| 3.3 Mechanical body | 10 |
| 3.4 Mechanical Components | 11 |
| 3.4.1 Acrylic sheets | 11 |
| 3.4.2 Mounting brackets | 12 |
| 3.4.3 End effector | 14 |
| 3.4.4 Wheels | 14 |
| 3.4.5 Nuts & Bolts | 15 |
| 3.5 Electromechanical Components | 15 |
| 3.5.1 High torque DC gear motor | 15 |
| 3.5.2 Servo motor | 16 |
| 3.6 Electrical Components | 17 |
| 3.6.1 Power source | 17 |
| 3.6.2 DC-DC buck converter | 18 |
| 3.6.3 Wires | 19 |
| 3.7 Embedded Components | 19 |
| 3.7.1 Raspberry Pi | 19 |
| 3.7.2 Arduino UNO | 20 |
| 3.7.3 Arduino Nano | 21 |
| 3.7.4 DC motor driver | 22 |
| 3.7.5 Servo PWM driver | 23 |
| 3.7.6 Control Pins | 24 |
| 3.7.6.1 SCL-I2C Clock Pin | 24 |
| 3.7.6.2 SDA-I2C Data Pin | 24 |
| 3.7.6.3 OE - Output enable | 24 |

| | | |
|--|--|----|
| 3.7.6 | Output Ports | 24 |
| 3.7.8 | ESP-32 Camera | 25 |
| 3.8 | Software Components | 26 |
| 3.8.1 | Raspbian OS | 26 |
| 3.8.2 | Python 3.9 | 27 |
| 3.8.3 | Arduino IDE | 27 |
| 3.8.4 | Pyserial | 27 |
| CHAPTER 04 SIMULATION & EXPERIMENTAL SETUP | | 28 |
| 4.1 | Introduction | 28 |
| 4.2 | Hardware and software implementation | 28 |
| 4.2.1 | Electronics Simulation | 28 |
| 4.2.2 | Mechanical model of the ARM | 30 |
| 4.2.3 | Control interface | 33 |
| 4.3 | Working flow | 34 |
| 4.4 | Robotic System Architecture | 35 |
| CHAPTER 05 RESULT & DISCUSSION | | 36 |
| 5.1 | Introduction | 36 |
| 5.2 | Final project outlook | 36 |
| 5.3 | Initialization & Testing | 37 |
| 5.3.1 | Connect to raspberry pi by SSH protocol | 37 |
| 5.3.2 | Setting up VNC server in raspberry pi | 37 |
| 5.3.3 | Connecting Raspberry pi with our laptop through VNC server | 38 |
| 5.3.4 | Open GUI interfaced from Raspberry pi | 39 |
| 5.3.5 | Open video streaming server for the esp-32 camera | 40 |
| 5.4 | Result & its Analysis | 40 |
| CHAPTER 06 CONCLUSION AND RECOMMENDATION | | 42 |
| 6.1 | Introduction | 42 |

| | | |
|-----|---------------------------|------|
| 6.2 | Conclusion | 42 |
| 6.3 | Future Work | 42 |
| | REFERENCES | 43 |
| | APPENDIX | A-1 |
| | APPENDIX A: DC MOTOR CODE | A-1 |
| | APPENDIX B: ARM CODE | A-3 |
| | APPENDIX C: ROVER PY | A-5 |
| | APPENDIX D: ARM PY | A-8 |
| | APPENDIX E: ESP32 CAMERA | A-14 |

ACKNOWLEDGEMENTS

All glory be to Almighty Allah, the most benevolent and merciful, who bestowed upon us the will to complete our project paper on time. Various individuals have helped finish this dissertation, both intellectually and emotionally, throughout the process. We owe them our heartfelt appreciation.

For this opportunity, we would like to express our deep and heartiest appreciation to the Military Institute of Science and Technology (MIST). We are also thankful to the faculty of the Department of Electrical, Electronics, and Communication Engineering (EECE) for allowing us the opportunity to complete the project. Commander K M Tanveer Anwar, (L), BN, has been our supervisor during our project. He has provided us required moral support and guidance regarding Robotic Engineering than anyone else. His professional and personal teachings would be an example to any student. We want to express our gratitude for his inspired and invaluable direction during this project's development. Today, his moral support has enabled us to complete our project. We are highly indebted to our supervisor for his benevolent care and affection. We also like to thank our supervisor for exposing us to the field of robotic applications.

Finally, we owe a debt of gratitude to our parents, who have always supported us and encouraged us to get this point.

LIST OF FIGURES

| | |
|--|----|
| Figure 1.1: Early bomb disposal robot controlled via cumbersome control cables | 3 |
| Figure 2.1: Talon bomb disposal robot | 7 |
| Figure 2.2: MK3 – CALIBER | 7 |
| Figure 2.3: KNIGHT | 8 |
| Figure 2.4: LT2-F Bulldog | 9 |
| Figure 2.5: HD2-S Mastiff | 9 |
| Figure 3.1: 3D design of our prototype | 11 |
| Figure 3.2: Acrylic sheets in pitch-black color | 12 |
| Figure 3.3: Multifunctional servo bracket | 13 |
| Figure 3.4: Long U bracket | 13 |
| Figure 3.5: Short L bracket | 13 |
| Figure 3.6: Mechanical gripper | 14 |
| Figure 3.7: CAD design of 85mm anti-skid wheel | 14 |
| Figure 3.9: CAD design of the DC motor used in our prototype | 16 |
| Figure 3.10: MG995 servo motor | 16 |
| Figure 3.11: Wiring diagram of MG995 | 17 |
| Figure 3.12: LiPo battery | 17 |
| Figure 3.13: LM2596 DC-DC Buck Converter | 18 |
| Figure 3.14: LM2596 DC-DC Buck Converter with a digital display and Voltmeter | 18 |
| Figure 3.15: Raspberry Pi 4 | 19 |
| Figure 3.16: Raspberry pi pin diagram | 20 |
| Figure 3.17: Pin diagram of UNO | 21 |
| Figure 3.18: Arduino Nano Pin diagram | 22 |
| Figure 3.19: H-bridge circuit | 22 |
| Figure 3.20: Dual L293D motor driver | 23 |
| Figure 3.21: Servo PWM driver | 24 |
| Figure 3.22: Servo driver with Raspberry pi | 25 |
| Figure 3.24: ESP-32 Camera | 26 |
| Figure 3.25: ESP-32 Camera (2) | 26 |
| Figure 4.1: Circuit diagram for controlling the rover in four-wheel-drive mode | 29 |
| Figure 4.2: Connecting PCA9685 with raspberry pi | 29 |
| Figure 4.3: Connecting servos with PCA9685 | 30 |

| | |
|--|----|
| Figure 4.4: Feature of 6 DOF arm | 31 |
| Figure 4.5: 3D model of our 6 DOF arm | 31 |
| Figure 4.6: The robot coordinate frame | 32 |
| Figure 4.7: DH parameters | 32 |
| Figure 4.8: Rover control interface | 33 |
| Figure 4.9: ARM control interface | 33 |
| Figure 4.10: The working flow of communication and power system | 34 |
| Figure 4.11: Experimental Flow Chart | 34 |
| Figure 4.12: Block diagram of Robotics System architecture | 35 |
| Figure 5.1: Final Project Outlook | 36 |
| Figure 5.2: Final Project Outlook in Solid Work Design | 36 |
| Figure 5.3: Connect to raspberry pi by SSH protocol | 37 |
| Figure 5.4: Setting up VNC server in raspberry pi | 38 |
| Figure 5.5: Connecting Raspberry pi with our laptop through VNC server | 38 |
| Figure 5.6: Open GUI interfaced from Raspberry pi | 39 |
| Figure 5.7: GUI interfaced of robotic arm control | 39 |
| Figure 5.8: Open video streaming server for the esp-32 camera. | 40 |
| Figure 5.9: Robotic gripper in an open state | 41 |
| Figure 5.10: Robotic gripper in close state | 41 |

LIST OF ABBREVIATIONS

| | |
|-------|-----------------------------------|
| APMBC | Anti Personal Mine Ban Convention |
| CCM | Convention on Cluster Munitions |
| GUI | Graphical User Interface |
| RMR | Remote Maintenance Robot |
| DOF | Degree of Freedom |
| GIPO | General- Purpose Input/Output |
| CAD | Computer- Aided Design |

ABSTRACT

Design and Fabrication of a Low-Cost Bomb Disposal Robot

Since their inception in the 1970s, remotely controlled bomb disposal robots have saved many lives, moving from their civil defense roots to serve the military in asymmetric combat. The explosive ordnance disposal (EOD) robot's job description is relatively straightforward: disable or safely explode a charge while the operator stays at a safe distance. The latest bomb disposal robot programs demonstrate the exceptional level of versatility, dexterity, and diversity offered by new platforms to support the EOD team of defense services in achieving their goals more safely and efficiently than ever. Bombs are often regarded as one of humanity's most hazardous inventions. Combining chemical components with technology results in the construction of gadgets capable of wreaking havoc on a global scale. This devastation does not stop at property damage; it also includes loss of life. To fight this, technology once again rises against technology. What is designed to destroy may also be designed to defuse. This project paper examines the possibility of developing equipment capable of disassembling bombs from a great distance, protecting valuable blood, and averting death. We intended to design and build a 4-wheel mobile robot capable of independent drive on each wheel. We also equipped our prototype with a robotic arm with six degrees of freedom movement. The robotic arm is also designed to assist the rover for multiple purposes such as surveillance, sampling, etc. Our robotic arm utilizes forward kinematics to achieve various tasks. Our prototype can be controlled by a GUI interface from a remote location. We tried to accomplish some bomb disposal tasks with our prototype built from available components in Bangladesh within our limitations.

সারসংক্ষেপ

Design and Fabrication of a Low-Cost Bomb Disposal Robot

1970-এর দশকে তাদের সূচনা হওয়ার পর থেকে, দূরবর্তীভাবে নিয়ন্ত্রিত বোমা নিষ্ক্রিয়করণ রোবটগুলি (bomb disposal robots) অনেকের জীবন বাঁচিয়েছে, অসমমিতিক যুদ্ধে সামরিক বাহিনীকে সেবা দেওয়ার জন্য তাদের নাগরিক প্রতিরক্ষা শিকড় থেকে সরে এসেছে। বিস্ফোরক অস্ত্র নিষ্পত্তি (EOD) রোবটের কাজের বিবরণ তুলনামূলকভাবে সহজ: অপারেটর নিরাপদ দূরত্বে থাকার সময় একটি চার্জ নিষ্ক্রিয় বা নিরাপদে বিস্ফোরিত করুন। সাম্প্রতিক বোমা নিষ্পত্তি রোবট (bomb disposal robots) প্রোগ্রামগুলি আগের চেয়ে আরও নিরাপদে এবং দক্ষতার সাথে তাদের লক্ষ্য অর্জনে প্রতিরক্ষা পরিষেবাগুলির EOD টিমকে সমর্থন করার জন্য নতুন প্ল্যাটফর্মগুলি দ্বারা প্রস্তাবিত বহুমুখিতা, দক্ষতা (dexterity) এবং বৈচিত্র্যের ব্যতিক্রমী স্তর প্রদর্শন করে। বোমাগুলিকে প্রায়শই মানবতার অন্যতম বিপজ্জনক আবিষ্কার হিসাবে বিবেচনা করা হয়। প্রযুক্তির সাথে রাসায়নিক উপাদানগুলিকে একত্রিত করার ফলে গ্যাজেটগুলি তৈরি হয় যা বিশ্বব্যাপী বিপর্যয় ঘটাতে সক্ষম। সম্পত্তির ক্ষতিতেই এই ধ্বংসাত্মক থামে না; এতে প্রাণহানিও অন্তর্ভুক্ত। এর বিরুদ্ধে লড়াই করতে আবারও প্রযুক্তির বিরুদ্ধে মাথাচাড়া দেয় প্রযুক্তি। যা ধ্বংস করার জন্য ডিজাইন করা হয়েছে তা নিষ্ক্রিয় (defuse) করার জন্যও ডিজাইন করা হতে পারে। এই প্রজেক্ট পেপারটি অনেক দূর থেকে বোমা বিচ্ছিন্ন (disassembling) করতে, মূল্যবান রক্ত রক্ষা করতে এবং মৃত্যু এড়াতে সক্ষম সরঞ্জাম তৈরির সম্ভাবনা পরীক্ষা করে। আমরা প্রতিটি চাকায় স্বাধীন ড্রাইভ (independent drive) করতে সক্ষম একটি 4-হুইল মোবাইল রোবট ডিজাইন এবং তৈরি করতে চেয়েছিলাম। আমরা আমাদের প্রোটোটাইপকে একটি রোবোটিক আর্ম দিয়ে সজ্জিত করেছি যার 6 ডিগ্রি অফ ফ্রিডম (degrees of freedom)। রোবোটিক আর্মটি একাধিক উদ্দেশ্যে যেমন নজরদারি, স্যাম্পলিং (sampling) ইত্যাদির জন্য রোভারকে সহায়তা করার জন্য ডিজাইন করা হয়েছে। আমাদের রোবটিক আর্ম বিভিন্ন কাজ অর্জনের জন্য ফরোয়ার্ড গতিবিদ্যা (forward kinematics) ব্যবহার করে। আমাদের প্রোটোটাইপ দূরবর্তী অবস্থান থেকে একটি GUI ইন্টারফেস (interface) দ্বারা নিয়ন্ত্রণ করা যেতে পারে। আমরা আমাদের সীমাবদ্ধতার মধ্যে বাংলাদেশে উপলব্ধ উপাদান থেকে তৈরি আমাদের প্রোটোটাইপ দিয়ে বোমা নিষ্ক্রিয় করার কিছু কাজ সম্পন্ন করার চেষ্টা করেছি।

CHAPTER ONE

INTRODUCTION

1.1 Introduction

The term "robot" was borrowed from the Czech word "robota," which means "forced laborer". Later on, a renowned Russian science fiction writer named Isaac Asimov coined the term "robotics". As the industry moves from its current stage of automation to robotization, many advances in the field of robotics have been successfully created in the form of humanoids, micro-robots, manipulators, and so on. As a result, robot technology is rapidly evolving. For the past few years, robotics research has been the talk of the town. Human beings are, as you may have guessed, the driving force behind this.

The difficulties that humans have faced over time prompted them to seek a simpler option that would save time, be more efficient, and be safer to employ. Many efforts have been made to produce robots that can be utilized to replace humans in tasks that are dangerous to humans, such as bomb disposal and laser cutting. Surprisingly, robots have become a significant aspect of human existence, and we can see robotics taking over in a variety of domains.

The primary idea behind this project is to create a robot that can be used to dispose of bombs and other explosives, ensuring that no human lives are endangered. The bomb disposal squads will be given a line of defense against the potentially lethal threats they encounter when dealing with explosives.

1.2 What is Robot

A robot is a programmable mechanical device equipped to complete an intricate arrangement of tasks automatically and interact with its environment. It contains sensors, power supplies, manipulators, and software to perform the designated tasks.

Technology is a multi-dimensional field that is experiencing advancements with rapid speed. Robotics is one of the sub-divisions of technology. This ambit of technology deals with Robots. Robots are machines that can be programmed to perform specific tasks based

on the given instructions. This series of actions is either performed autonomously or semi-autonomously by the Robots.

As per the available knowledge of Robotics, there are the following three factors in its formation:

- (a) Robots interact with the physical world with the help of actuators and sensors.
- (b) Robots can be programmed.
- (c) Robots are often autonomous or semi-autonomous.

The essence of the technology of robotics is to secure the life of human beings while carrying out specific actions where life is threatened by any natural calamity or any other dangerous resort. Robots like Mars Rover, Sojourner, and Exploration Rover and underwater robots collect information or assess the condition without involving direct human physical contact.

1.3 Types of Robot

Robots can be classified into following categories considering their style of movement.

1.3.1 Legged Robot

A legged robot is a robot that moves about using mechanical limbs (legs). Compared to other robots in the field, this robot has a lot more maneuverability.

1.3.2 Wheeled Robot

Wheeled robots are defined as robots with wheels attached to their construction. The installation of the wheels is responsible for the machine's motion.

1.3.3 Stationary or Fixed Robots

These gadgets are made up of connected pieces or segments that allow them to travel with a certain degree of freedom in an area that is inaccessible to humans. They alter the articles or pieces using a fixed base. In such areas, the devices grip or move certain items. An

example of a manipulator is Canada's arm. Remote or programmed control mechanisms can be used to operate these robots.

1.4 Bomb Disposal Robot

Robots go where humans fear not to. There are many different applications of robotic systems. One of the most hazardous applications of robotic systems is bomb disposal, where many humans' lives could be at risk. With the advancement in robotic technology, it is now possible to perform dangerous activities such as bomb disposal without risking human lives. (Richa Parmar, Mar -2017)

A network of ropes controlled the early bomb disposal robots. However, orders were transmitted to the robots through a communications connection when technology advanced. However, the robots were restricted to a specific operational radius due to the cable, and there was also the risk of the cable getting tangled. (Sagar Randive, Mar 2012)



Figure 1.1: Early bomb disposal robot controlled via cumbersome control cables

Nowadays, Bomb disposal robots are controlled through wireless communication. This increases their operational range allowing the operator to control them from a very safe distance.

As the main idea remains the same, the design of bomb disposal robots has changed just a little since they were first designed.

1.5 Project Objectives

We create a prototype for our suggested project to complete the primary mission. Our primary mission is as follows:

- (a) To Provide visual identification for analysis of a suspicious packet (or bomb).
- (b) To design and implement the gripper with a robotic arm.
- (c) Provide mobile control from a distance.

1.6 Organization of the Project

The organization of the project is as follows:

Chapter 2 reviews the basic theories and robot fundamentals related to the project. literature review and motivation for this project are mainly discussed here.

Chapter 3 discusses different components used in the project. A theoretical study on design and construction for the project has been presented.

Chapter 4 highlights the steps of the simulation for the system. A gradual workflow from the collection of radar raw data up to transmission has been presented. Also, the robotic system architecture has been discussed.

Chapter 5 mainly focuses on results and final output from this project.

Chapter 6 Summarizes the limitations and the problems we have faced.

Chapter 7 concludes with recommendations for further improvement. gives conclusion and recommendation on this project and provides directions on possibilities for future work.

CHAPTER TWO

LITERATURE REVIEW

2.1 Review Summary

While significant progress has been made since the adoptions of APMBC and Convention on Cluster Munitions (CCM) in curbing the use of these indiscriminate weapons, ensuring the destruction of existing stockpiles, and clearing contaminated lands, challenges persist. Large areas of the world remain contaminated by anti-personnel mines and explosive remnants, posing a daily threat to civilians, hampering agriculture, trade, and development, and hindering humanitarian operations. In the paper, (Homsup, 22 April 2008) the author has developed a Remote Maintenance Robot (RMR) System for use in the SPR III facility. The RMR should reduce occupational radiation exposure by a factor of four and decrease reactor downtime due to maintenance. In authors (Reddy, 02, February-2015) report that the TIRAMISU project in the European Union proposes to use multisensory data fusion techniques for combining information from a metal detector and a chemical sensor to improve the location and detection accuracy of bombs. In the proposed project, we plan to design a low cost remotely controlled vehicle using Arduino Mega to merge data from the controller & the gripper. In, (S.Keerthana, August 2019) it is stated that COMRADES Robots is having two robots called the Explorer and Corobot, manufactured by Coroware Inc. Both robots are four-wheeled and use a skid-steer mechanism for maneuvering. In author (Oke Alice O., March 2014) stated that the structure of the bomb-disposal robot consists of a remotely operated vehicle and a manipulator fixed on it with five DOF. This kind of structure has been adopted in several commercial bomb-disposal robots and is considered an effectual design. But to attain high accuracy of target positioning of the manipulator, automatic operation or intelligent techniques are necessary for such a robot. Three-dimensional coordinates of the perilous target are obtained with high accuracy in the robot-based coordinate system through the binocular stereo vision technique. While after acquiring the coordinates, the manipulator will reach for the target position automatically, instead of the manual controlling of the joints by the operator. But due to the cost of the project, only one camera will be used in our project and the idea of a manipulator with six degrees of freedom will be taken as a basic design.

Keeping all of the above incidents in mind, planning this project includes the necessity for a single elaborated system that takes care of all the problems mentioned above. Thus, our

project is to design & fabricate a robotic arm on a moving body that will allow us to control the movement & attain our goal of identifying & disposing of explosives.

2.2 Motivation

Handling explosive materials such as IEDs often becomes life-threatening for humans. Land mines and hidden IEDs are rarely seen by naked eyes, which becomes a significant threat to everyday civilian lives. Throughout the development in different fields of robotics, life-threatening tasks like disposing of bombs have been considered in the research field. Two main aspects of development such robots like-disarmament of a bomb with minor direct interaction of human to the bomb and saving crucial Evidence of Bomb material To trace technology, identifying the developer. Although technology and research have come a long way to develop crucial robots to pursue the critical task of bomb disposal, we intend to take one step further and build a cheaper and available solution concerning Bangladesh. For now, our prototype can do surveillance, basic object grasping, and remote navigation by human intervention, but with further funding and research, we hope to build more crucial and accomplished solutions that can someday dispose of bombs autonomously. Pursuing the research locally will save us time and money and build up trained professionals in this industry.

2.3 Existing Bomb Disposal Robot and Their Limitations

Some existing bomb disposal robots around the world are discussed below:

2.3.1 Talon

Talon is a tracked, autonomous, lightweight military robot developed and manufactured by Foster-Miller, a subsidiary of QinetiQ North America. The robot is designed to safeguard soldiers and first responders against hazardous threats. The robot has a high payload-to-weight ratio and is capable of being deployed in various types of environments and terrain. It provides infantry units with increased protection. The robot has a maximum maneuverability speed of 8.37 kilometers per hour. It can handle up to 43° stairwells, 45° side slopes, 38cm snow or demolition rubble, and through rock piles. At barely 30 pounds, the grip pressure is light. (Anon, 2000)



Figure 2.1: Talon bomb disposal robot

2.3.2 MK3 – Caliber

The MK3 - CALIBER EOD is an excellent choice for teams requiring robotic capabilities for EOD or SWAT missions. The MK3 - CALIBER is a medium-sized robot with the pulling and dragging skills of larger platforms with smaller, lighter platforms' speed and stair climbing capabilities. The MK3 - CALIBER Advanced, the third of three design variants, is a two-armed, multi-purpose robot equipped with a heavy-duty robotic claw and a Twin-Disrupter / Weapons Turret. Users can use two robotic arms to lift more objects with the claw without affecting the disrupter's targeting capabilities. Simple surveillance chores may be accomplished with any of the robot's six-color cameras. However, the weight is excessive (35 kg).



Figure 2.2: MK3 – CALIBER

2.3.3 Knight

KNIGHT was developed for surveillance, reconnaissance, hazardous item handling, suspected package manipulation, and the neutralization and management of improvised explosive devices, poisonous chemicals, and radioactive objects. KNIGHT is a multi-terrain, all-weather mobile robot that has been designed for usage both inside and outdoors (buildings, public institutions, airports, houses, etc.). It features cutting-edge robotic and computer technology and is designed for use in hazardous environments. It consists of a sturdy platform, a dexterous robot arm, an operator control station, communication links, and remotely operated operational equipment and accessories. It cannot be used in rugged terrain.



Figure 2.3: KNIGHT

2.3.4 LT2-F Bulldog

Bulldogs are capable of climbing stairs and other obstacles found in the home. The LT2-F is equipped with rear flipper arms that allow the robot to be raised and manoeuvred over more challenging terrain and obstacles. Additionally, they act as stabilisers, keeping the car upright on steep inclines. Due to the Bulldog's changeable multi-axis arm, it is a very adaptable instrument. It's ideal for emptying rooms, removing suspicious packages, and watching from a distance, among other things! The arm may be detached for a variety of

missions and clearance requirements. Without the arm, the Bulldog can fit under automobiles, beds, and into tiny spaces. The robot can now open doors, move and inspect objects, and inspect the furniture with the arm connected. Perhaps it lacks a zoom camera.



Figure 2.4: LT2-F Bulldog

2.3.5 HD2-S Mastiff

Due to the HD2's tracked construction, it can climb over the majority of obstacles, including stairs. A zoom camera with a 20x magnification and a 360-degree pan is incorporated. Additionally, it features infrared capabilities, enabling you to see in low-light conditions. The inclusion of a multi-axis arm enhances the flexibility of the HD2-S Doberman. The operator may utilize the multi-axis arm to open doors, assist in scaling obstacles, move garbage, inspect suspicious objects, and conduct EOD from a safe distance. When fully extended, the strong arm can lift to 20 pounds. The Mastiff may be equipped with a fifth axis that allows for 400-degree rotation and a sixth axis that allows for 360-degree wrist rotation. (Zeng Jian-Jun, June 1, 2007)



Figure 2.5: HD2-S Mastiff

CHAPTER THREE

DESIGN AND CONSTRUCTION

3.1 Background

Human life is becoming more dependent on the autonomy of robotics, and the reason behind this is that robots can often perform such daily activities with tremendous speed, accuracy and aptitude despite the level of difficulties. Mechanical autonomy innovation has already developed so much that the behavioural design of human beings can be studied by them and act according to fulfil that errand while the individual isn't displaying there. Instructions are conveyed on the robots by state-of-the-art programming language, and many advanced electronic gadgets are working along to increase efficiency while pursuing various activities.

Disposing of explosive ordinances such as IEDs, bombs, or other explosive material is one of the most important fields where robots have significantly impacted. In this chapter, we will conclude the design and fabrication of our prototype of a bomb disposal robot.

3.2 Design

Designing any robotic system consists of three major parts such as mechanical, electrical, and software components. All of these mentioned components are as important as any other ones. In the following sections, we will depict the design procedures of mechanical, electrical, and software components. Designing any robotic system consists of three major parts such as mechanical, electrical, and software components. All of these mentioned components are as important as any other ones. The following sections will depict the design procedures of mechanical, electrical, and software components.

3.3 Mechanical Body

We designed our robot using SOLIDWORKS software that is well known in industrial CAD design applications. SOLIDWORKS is used to design and create mechatronic systems from start to finish. The program is mainly used for planning, visual ideation, modelling, prototyping, and project management in the early stages. After that, the program is used to design and construct mechanical, electrical, and software components. We developed a prototype of our robot using SOLIDWORKS. After an initial prototype was

ready, we moved forward to the fabrication part of our robot. We developed a CAD prototype of all the parts used in our robot and simulated the design by SOLIDWORKS. Doing so helped us visualize and get a better insight into our robot before moving forward with the fabrication process. Some CAD snips of our robot design are given below for reference.

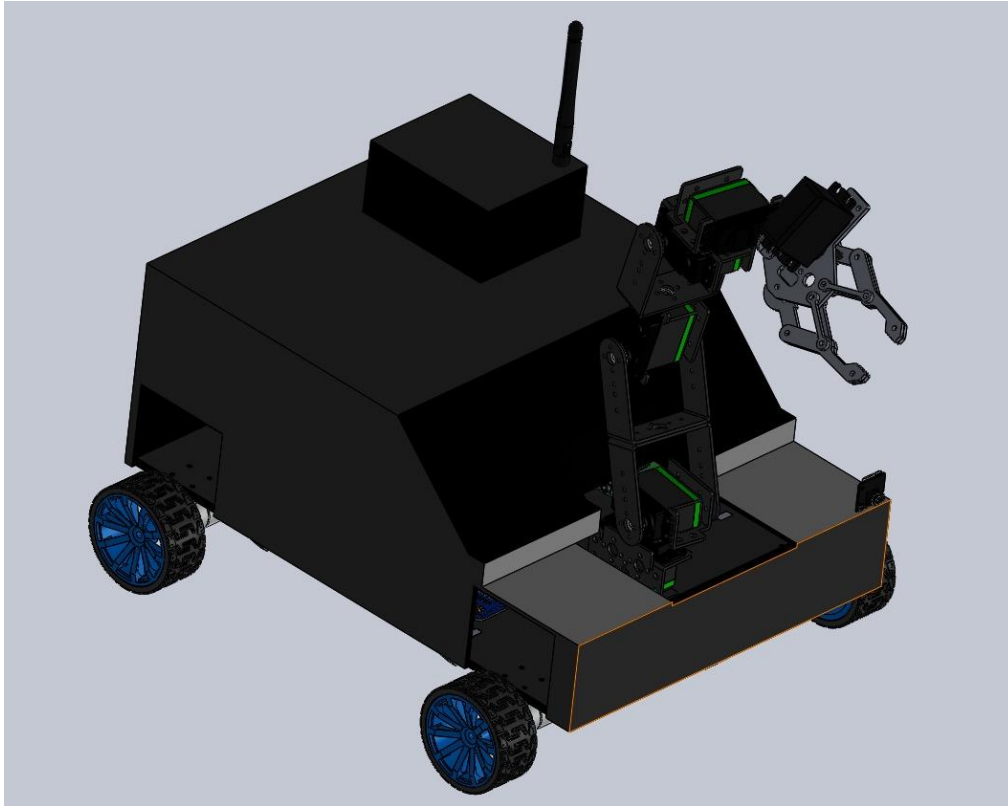


Figure 3.1: 3D design of our prototype

3.4 Mechanical Components

We used various mechanical components to build our prototype of a bomb disposal robot. A brief introduction of these components is given below

3.4.1 Acrylic Sheets

Acrylic, known as Plexiglass, is a flexible plastic material that serves various functions and offers a variety of benefits. It is available in various colours and opacities. Acrylic plastic is a transparent thermoplastic homopolymer that is sometimes referred to informally as Plexiglass. This plastic has unusual qualities that make it appropriate for various

applications, from everyday home goods to the world's fibre optic cables. These features have contributed to acrylic's enormous appeal in industrial and do-it-yourself projects:

- a) High impact resistance
- b) High optical clarity
- c) Innate weather ability and UV resistance
- d) Excellent dimensional stability
- e) Lightweight
- f) Excellent chemical resistance

We used acrylic sheets consisting of 6mm thickness coloured in pitch black. The skeletal body of our robot is made of the above-mentioned acrylic sheets. (Anon, 2019)



Figure 3.2: Acrylic sheets in pitch-black colour

3.4.2 Mounting Brackets

A bracket is an architectural element. It is a structural or decorative member. Mounting brackets provide a way to mount various electromechanical components such as servo motors or dc motors to our robot chassis. Mounting Brackets are used to secure things together, most often at an angle to one another. The bracket may be threaded or unthreaded, hinged, or constructed of aluminium, brass, nylon, polypropylene, or steel with a short L, tall L, straight block, or Z form. In our case, we used several mounting brackets made of steel shaped as long U, straight block, short L to mount motors in our prototype.



Figure 3.3: Multifunctional servo bracket



Figure 3.4: Long U bracket



Figure:3.5 Short L bracket

3.4.3 End Effector

An end effector is a peripheral device that attaches to a robot's wrist, allowing it to interact with its task. Most end effectors are mechanical or electromechanical and serve as grippers, process tools, or sensors. We used a Mechanical Gripper made of aluminium alloy. This has The Degree of Freedom (DOF): 2.



Figure 3.6: Mechanical gripper

3.4.4 Wheels

A wheel is a spherical frame or disk that revolves around an axis. Such as those seen on or in vehicles or machines. The creation of wheels cleared the stage for humanity's industrial revolution and subsequent transition to a mechanical civilization. Wheels are necessary components of every mechanical item for it to be mobile. Our prototype was propelled by four 85mm diameter wheels.



Figure 3.7: CAD design of 85mm anti-skid wheel

3.4.5 Nuts & Bolts

While creating any mechanical structures, nuts and bolts are always essential parts of it. The whole structure is held together by nuts and bolts. We used nuts and bolts of diameter 3mm and length 16mm to build our prototype.



Figure 3.8: M3 nuts and bolts

3.5 Electromechanical Components

Electromechanical components are components or devices that transform mechanical to electrical activity or electrical to mechanical action. In our project, we employed two distinct types of electromechanical components. Both motors turn electrical energy from a LiPo battery into mechanical energy that drives our robot.

3.5.1 High Torque DC Gear Motor

We used four 12V DC motors consisting of 500 RPM with high torque to drive our rover, enabling our rover to achieve a four-wheel drive that unlocks robust control and power in rough terrain. The high torque was enabled using a gear combination mechanism in our motor.

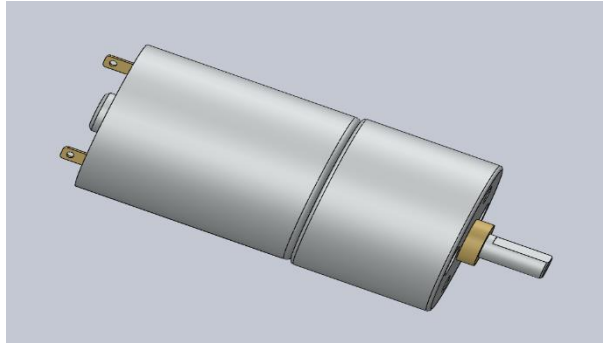


Figure 3.9: CAD design of the DC motor used in our prototype

3.5.2 Servo Motor

We used MG995 servo motors to generate rotating movements in our prototype's 6DOF arm. The MG995 servo motor is a rugged dual-bearing unit with all-metal gears. This servo has a maximum torque of 208 oz-in at 6V and a spinning speed of 0.13 seconds per 60°. Reduce the voltage to 4.8V, the torque remains 180oz-in, and the reaction time remains 0.17 seconds per 60°. MG995 servo motors are capable of 180° rotations and a lifting capacity of 10KG.



Figure 3.10: MG995 servo motor

PWM=Orange (\square)
 Vcc = Red (+)
 Ground=Brown (-)

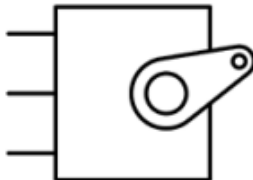
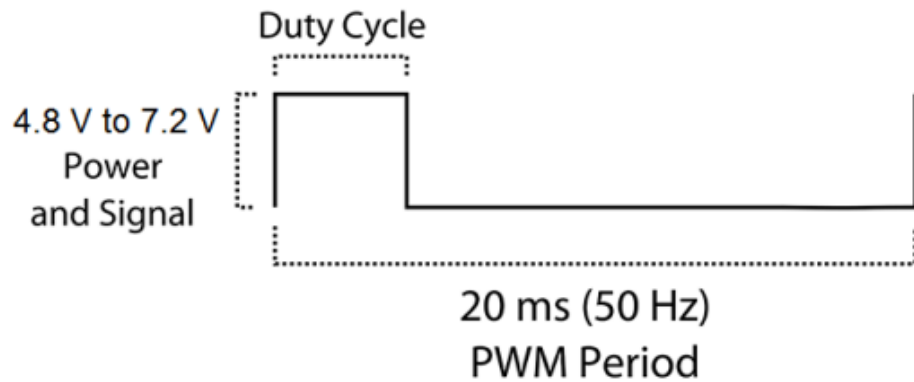



Figure 3.11: Wiring diagram of MG995

3.6 Electrical Components

We used several electrical components in building our prototype varying from simple wires to dc power sources to dc-dc converters.

3.6.1 Power Source

All of our robot's electrical, electromechanical, and embedded components are powered by a LiPo battery. A lithium polymer battery, or more precisely a lithium-ion polymer battery (abbreviated as LiPo, LIP, Li-poly, lithium-poly, and others), is a type of rechargeable lithium-ion battery that uses a polymer electrolyte rather than a liquid electrolyte. Our battery has a capacity of 4200 mA and can deliver up to 12V.



Figure 3.12: LiPo battery

3.6.2 DC-DC Buck Converter

A buck converter (sometimes known as a step-down converter) is a DC-to-DC power converter. Reduces the voltage from its input (supply) to its output (while drawing less average current) (load). The LM2596 DC-DC Buck Converter Step Down Module is used to power our embedded components by converting the battery's 12V to 5V.



Figure 3.13: LM2596 DC-DC Buck Converter

We also used the LM2596 DC-DC Step-Down Buck Converter with Digital Tube Display to supply power to our main computer board (Raspberry Pi). The main reason for using this is to monitor real-time voltage input into raspberry pi as stable voltage output within 5V is crucial to protect the board.

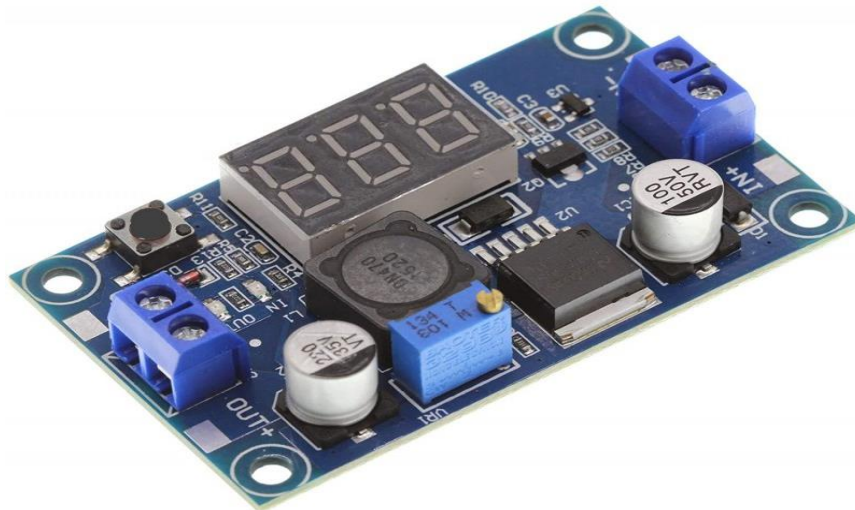


Figure 3.14: LM2596 DC-DC Buck Converter with a digital display and Voltmeter

3.6.3 Wires

All the wires used in this project are compatible with providing a stable 3A current supply within 5V to 12V range. We have maintained colour code in wiring any components such as black representing neutral/ground whereas red representing positive.

3.7 Embedded Components

Embedded components can be defined as the technology to embed electronics and computational components in a single printed circuit board. Embedded components have grown significantly in the defence, aerospace, telecommunications, medical, and consumer markets. We have used microcontrollers and a mini-computer in developing our prototype.

3.7.1 Raspberry Pi

The Raspberry Pi is an exceptionally low power computer that runs Linux. Moreover, it gives a set of GPIO pins, permitting to control of electronic elements for physical computing and investigating the Web of Things. The Raspberry Pi works within the open-source environment. The Raspberry Pi Foundation contributes to the Linux kernel and different other open-source ventures and discharges much of its program as open source. (William M. Davidson, 10 May 2017)

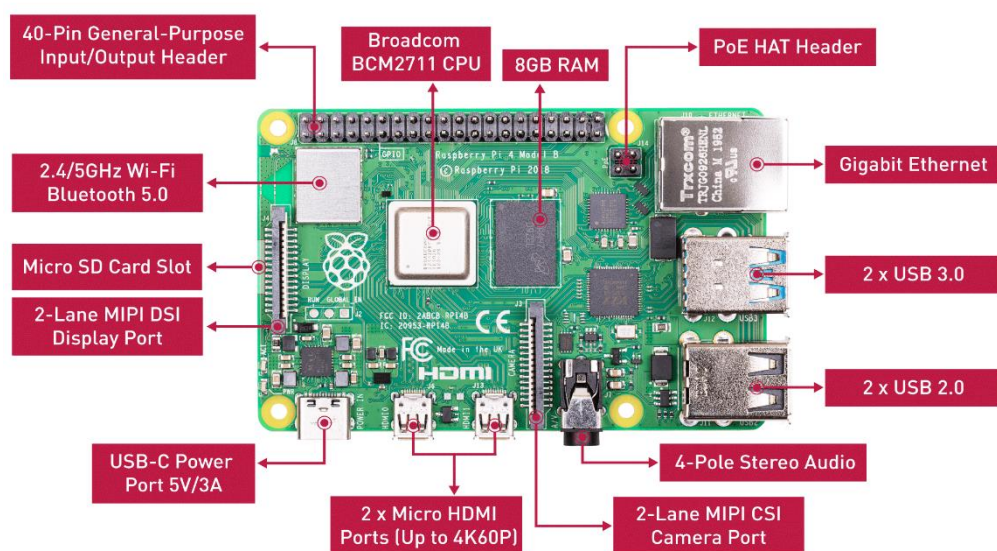



Figure 3.15: Raspberry Pi 4

Raspberry pi is a small yet powerful component that acts as a mini-computer that enables us to complete the complex task of operating our robot. Raspberry pi 4 offers 8GB of Random-access memory and Broadcom BCM2711 CPU based on arm processor that can run Debian based Linux operating system. The board also has 40 pinouts, of which 20 of them offer GPIO services that can directly interact with various sensors and electromechanical components. Raspberry pi also offers onboard WIFI and a Bluetooth chip, enabling the board to be wireless and robust in communication. While communicating with a microcontroller such as Arduino boards, no other board offers such versatile control and features within such low budget as raspberry pi does.



| Peripherals | GPIO | Particle | Pin # | Pin # | Particle | GPIO | Peripherals |
|-------------|--------|------------|-------|-------|----------|------|-------------|
| | 3.3V | | 1 | X | X | 2 | 5V |
| I2C | GPIO2 | SDA | 3 | X | X | 4 | 5V |
| | GPIO3 | SCL | 5 | X | X | 6 | GND |
| Digital I/O | GPIO4 | D0 | 7 | X | X | 8 | TX |
| | GND | | 9 | X | X | 10 | RX |
| Digital I/O | GPIO17 | D1 | 11 | X | X | 12 | D9/A0 |
| Digital I/O | GPIO27 | D2 | 13 | X | X | 14 | GND |
| Digital I/O | GPIO22 | D3 | 15 | X | X | 16 | D10/A1 |
| | 3.3V | | 17 | X | X | 18 | D11/A2 |
| | GPIO10 | MOSI | 19 | X | X | 20 | GND |
| SPI | GPIO9 | MISO | 21 | X | X | 22 | D12/A3 |
| | GPIO11 | SCK | 23 | X | X | 24 | CE0 |
| | GND | | 25 | X | X | 26 | CE1 |
| DO NOT USE | ID_SD | DO NOT USE | 27 | X | X | 28 | DO NOT USE |
| Digital I/O | GPIO5 | D4 | 29 | X | X | 30 | GND |
| Digital I/O | GPIO6 | D5 | 31 | X | X | 32 | D13/A4 |
| PWM 2 | GPIO13 | D6 | 33 | X | X | 34 | GND |
| PWM 2 | GPIO19 | D7 | 35 | X | X | 36 | D14/A5 |
| Digital I/O | GPIO26 | D8 | 37 | X | X | 38 | D15/A6 |
| | GND | | 39 | X | X | 40 | D16/A7 |

Figure 3.16: Raspberry pi pin diagram

3.7.2 Arduino UNO

Arduino Uno is a single-board computer with an ATmega328P microprocessor at its heart. It has 14 digital input/output pins (pulse width modulation outputs), six analogue inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power connector, an ICSP header, and a reset button. We utilized UNO to control the dc motors that power our robot's motion.



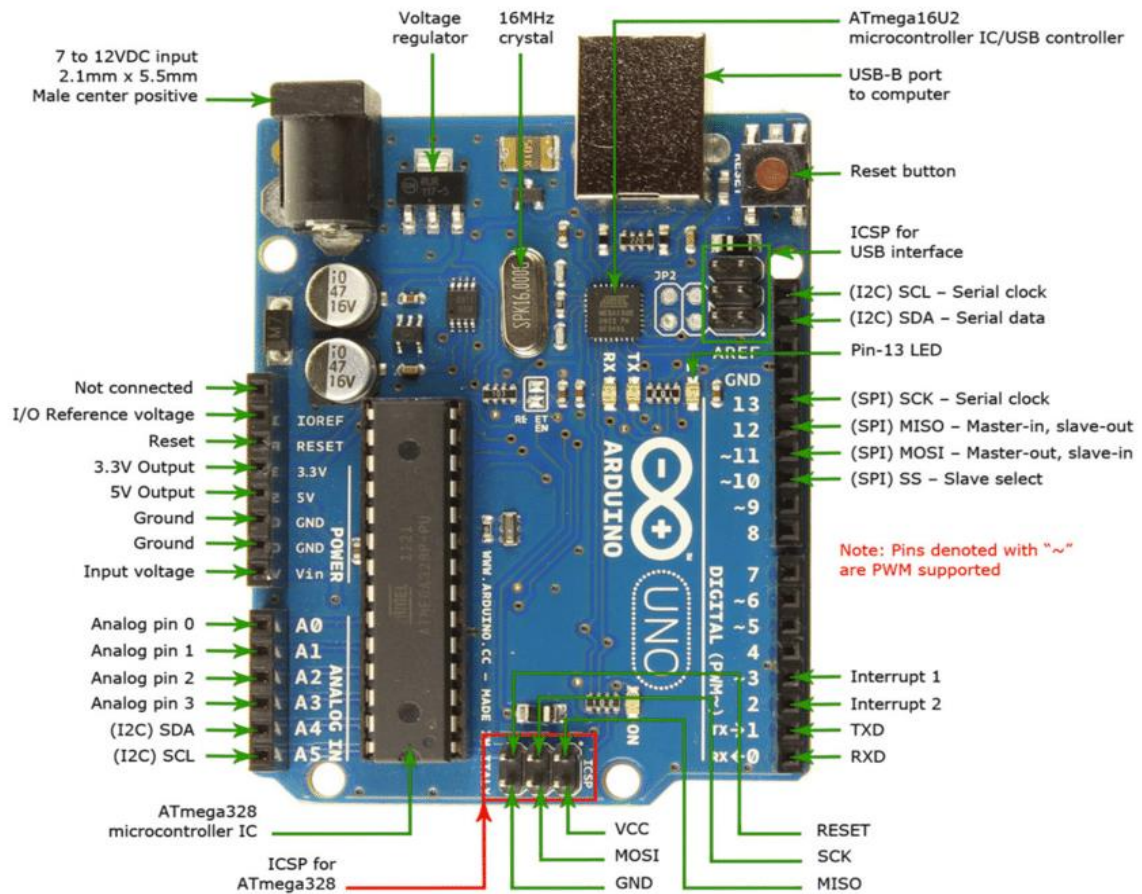
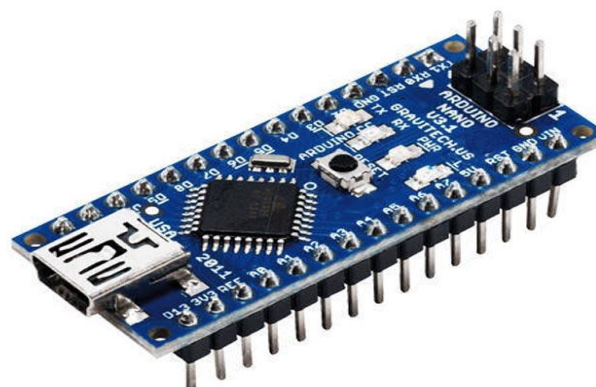


Figure 3.17: Pin diagram of UNO

3.7.3 Arduino Nano

The Arduino Nano is a small board designed to fit on a breadboard. It is based on the ATmega328 microcontroller (Arduino Nano 3. x). It is functionally similar to the Arduino Duemilanove but is packaged differently. It lacks a DC power connector and communicates through a Mini-B USB connection rather than a standard one. Arduino Nano was utilized to operate the servo motors in our robotic arm.



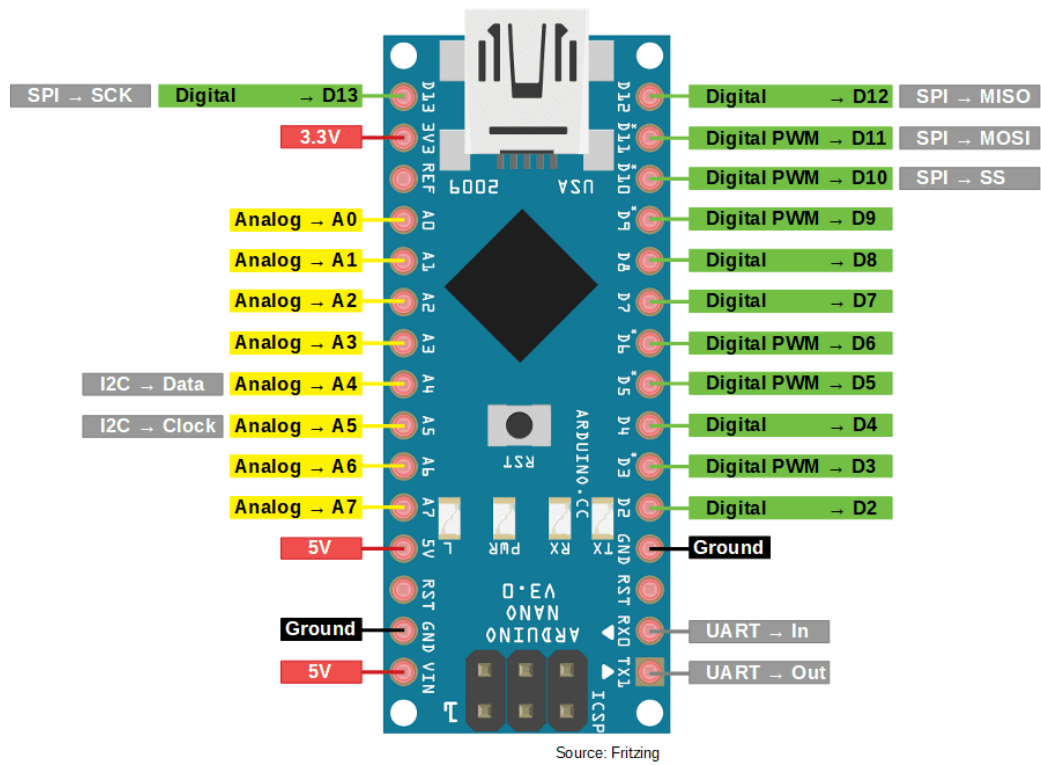


Figure 3.18: Arduino Nano Pin diagram

3.7.4 DC Motor Driver

Motor drivers serve as a link between the motors and the control circuits. In comparison, the motor consumes a significant amount of current, the controller circuit functions on low-current signals. Thus, the purpose of motor drivers is to convert a low-current control signal to a higher-current signal capable of driving a motor. An H bridge circuit is another type of motor driver circuit frequently employed. H bridge circuits are critical in robotic applications where the DC motor must operate in both directions. The term "H Bridge" refers to the circuit's diagrammatic form. The L293D is a dual H bridge motor driver integrated circuit. We utilized a dual L293D H bridge motor driver to control our four DC motors. (Anon, n.d.)

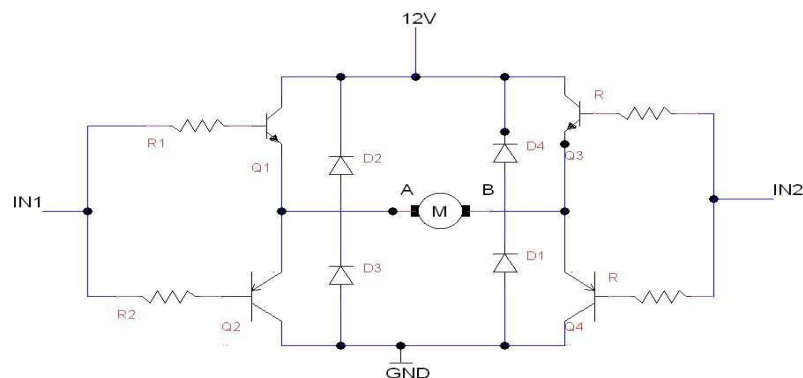


Figure 3.19: H-bridge circuit

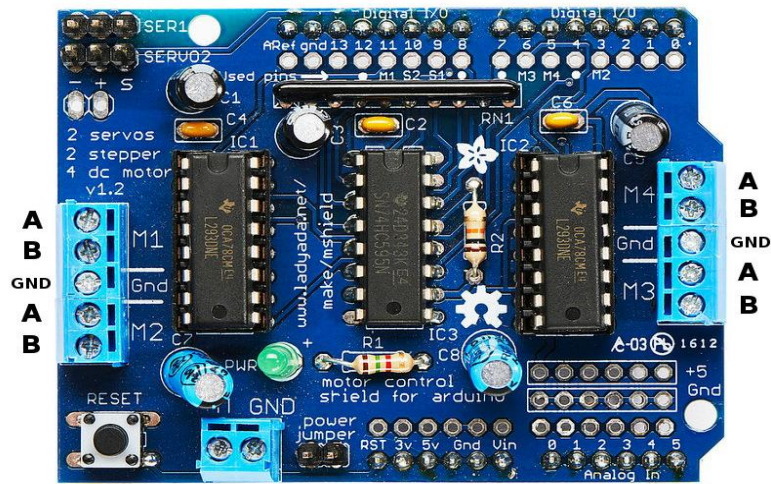


Figure 3.20: Dual L293D motor driver

3.7.5 Servo PWM Driver

Servos are operated by transmitting a variable width electrical pulse or pulse width modulation (PWM) signal through the control wire. There is a minimum and maximum pulse length and a repetition rate. Typically, a servo motor can only turn 90 degrees in either direction for 180° motions. The servo motor controller is a circuit that allows the position of a servo motor to be controlled. Additionally, it is referred to as a servo motor driver. A servo motor controller comprises three components: a controller, a servo motor, and a power supply unit. Servo motor drivers can operate a single servo or a group of servo motors.

To operate the servo motor on our 6DOF arm, we utilized the Adafruit PCA9685 16-Channel Servo Driver. The Adafruit 16-Channel 12-bit PWM/Servo Driver requires just two pins to drive up to 16 servos via I2C. Without extra Arduino processing overhead, the onboard PWM controller will drive all 16 channels concurrently.

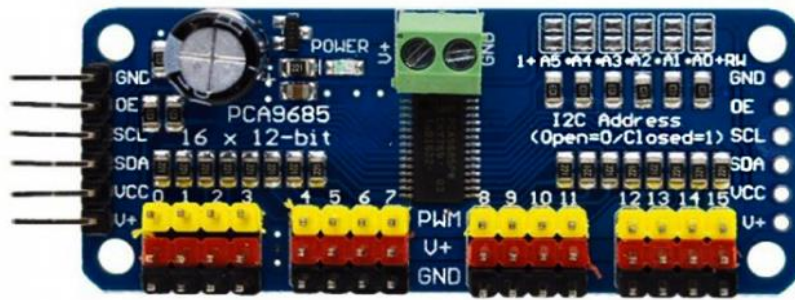


Figure 3.21: Servo PWM driver

3.7.6 Control Pins

3.7.6.1 SCL-I2C clock pin

Connect to the I2C clock line on our microcontroller. It supports logic operating at 3V or 5V and has not a strong pull-up to VCC.

3.7.6.2 SDA-I2C data pin

Connect to the I2C data line on our microcontroller. It supports logic operating at 3V or 5V and has a weak pull-up to VCC.

3.7.6.3 OE - output enable

OE allows for rapid disabling of all outputs. When this pin is low, all other pins become active. When this pin is high, it disables the outputs. By default, it is pulled low, making it an optional pin.

3.7.6 Output Ports

There are total of sixteen output ports. Each port is equipped with three pins: V+, GND, and a PWM output. Each PWM operates autonomously, but they must all operate at the same PWM frequency.

They're wired for servos, but they also work with LEDs! The maximum current allowed per pin is 25mA. All PWM pins are connected in series with 220-ohm resistors, and the output logic is identical to VCC, so bear this in mind when utilizing LEDs.

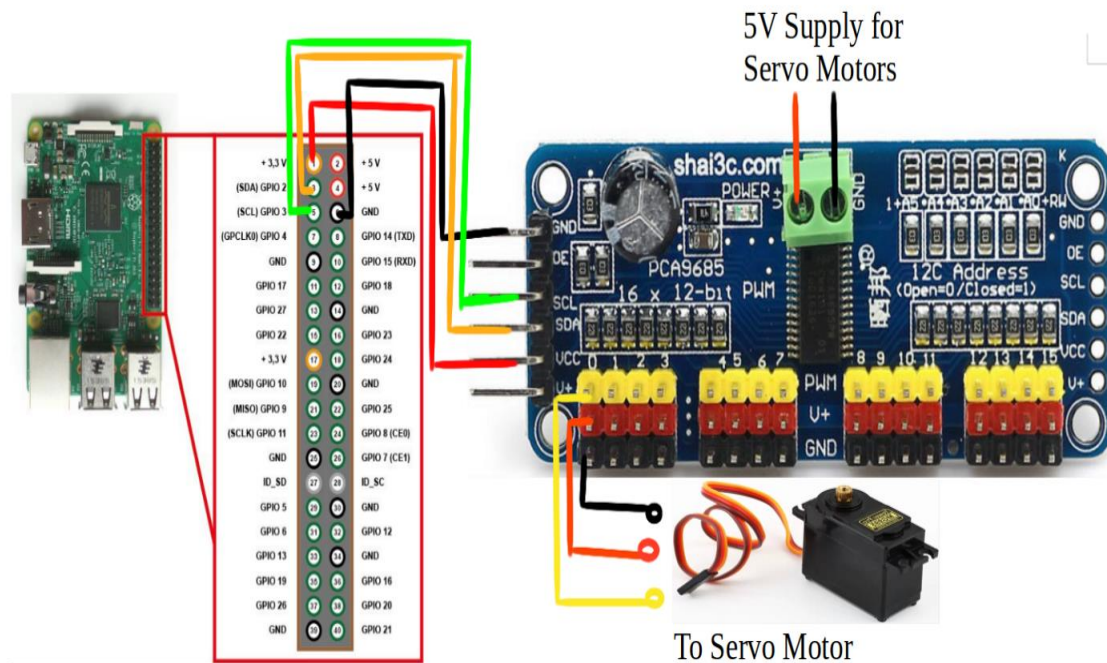


Figure 3.22: Servo driver with Raspberry pi

3.7.8 ESP-32 Camera

We are using an ESP-32 camera through a web server to send real-time visual input from our robot to our control interface. The ESP32-CAM development board is a low-cost board equipped with a WIFI camera. It allows the construction of IP camera projects with a range of video streaming resolutions. The ESP32-CAM incorporates a PCB antenna. The ESP32-CAM-UFL is identical to the ESP32-CAM-UFL, but with a U. Because the ESP32-CAM is based on the ESP32-S module, its specs are identical. (Argel A. Bandala, November 2012)

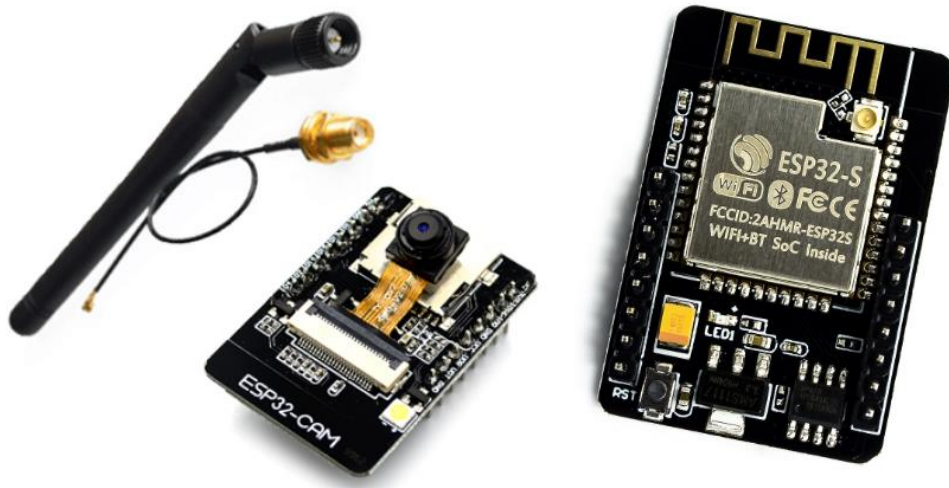


Figure 3.24: ESP-32 Camera

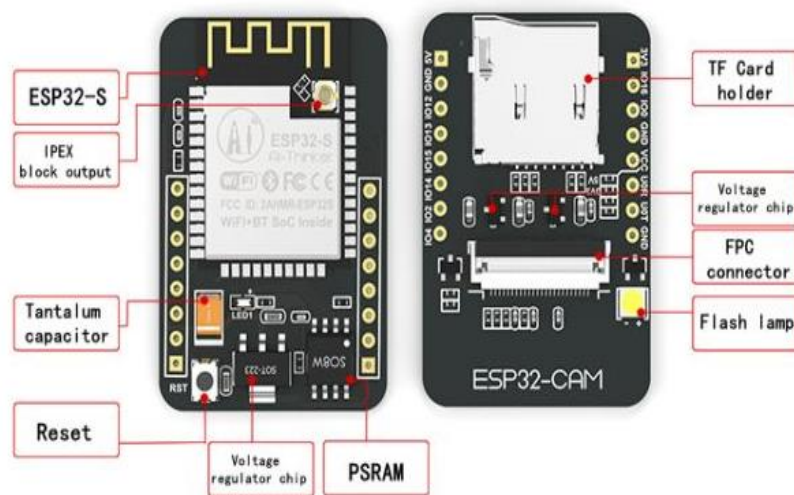


Figure 3.25: ESP-32 Camera (2)

3.8 Software Components

We utilized various open-sourced software to build our prototype. A brief introduction of them is given below.

3.8.1 Raspbian OS

Raspbian is a Debian-based OS made especially for the Raspberry Pi computer board, and it is the perfect general-purpose OS for projects that utilize raspberry pi. It's capable of running instances of python 3.7 and other higher versions. Raspbian is the Raspberry foundation's official supported OS and can accomplish a wide range of tasks on the go.

3.8.2 Python 3.9

Python 3 is the language's latest version, published in December 2008. This version was primarily released to address issues with Python 2. Python 3 was incompatible with Python 2 due to the nature of these modifications. It is irreconcilable with the past. Particular Python 3 features have been backported to Python 2. x versions to simplify the Python 3 conversion process. As a result, any business utilizing Python 2. x was required to make several adjustments to migrate to Python 3. x. These changes affect projects, apps, and all the Python ecosystem's libraries.

3.8.3 Arduino IDE

The free and open-source Arduino IDE makes it simple to write code and upload it to the board. This software runs on any Arduino board. This IDE was created and developed by the Arduino organization and may be used with any Arduino board currently available. GitHub hosts the Arduino software's active development.

3.8.4 Pyserial

We established a serial connection between Arduino boards and the raspberry pi using the pyserial package. This module restricts access to the serial port. It provides Python backends for Windows, Mac OS X, Linux, BSD, and perhaps any POSIX-compliant system and IronPython. The "serial" module automatically picks the correct backend.

CHAPTER FOUR

SIMULATION & EXPERIMENTAL SETUP

4.1 Introduction

We will discuss about the simulation of our software as well as the working procedure of the components and our prototype in here. We will also show circuit diagrams of the whole control system of our robot, workflow, and finally working interface of the prototype.

4.2 Hardware and Software Implementation

There was several software for the simulation of the project in accordance with the assembled hardware.

4.2.1 Electronics Simulation

We used PROTEUS, a Circuit simulation, to simulate all the electronics components before deploying them into our prototype. We tested for all possible voltage-current variation scenarios and selected the voltage-current input for those components based on the simulation result and the dataset provided for the components. Proteus is a software package for simulating, designing, and sketching electrical circuits. The Labcenter electronic invented it. (Hengnian Qi1, October 2007.)

Two primary electro-computational operations are going on in our prototype. One is to control the rover, and another one is to control the 6DOF arm of our prototype. Both operations are maintained centrally from the raspberry pi and two different microcontrollers for two separate operations such that failure in one operation doesn't affect the other.

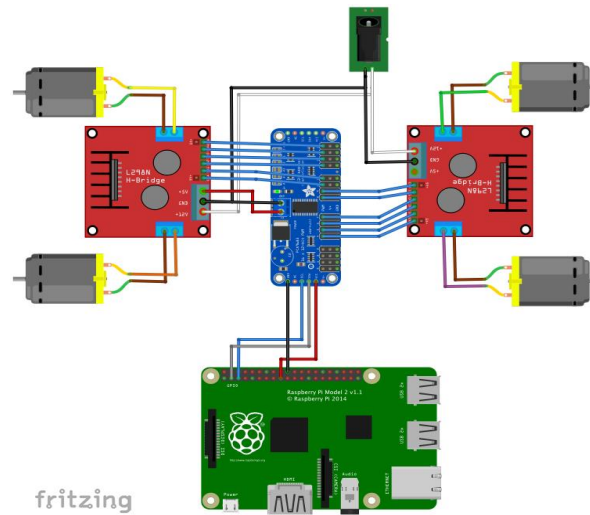


Figure 4.1: Circuit diagram for controlling the rover in four-wheel-drive mode

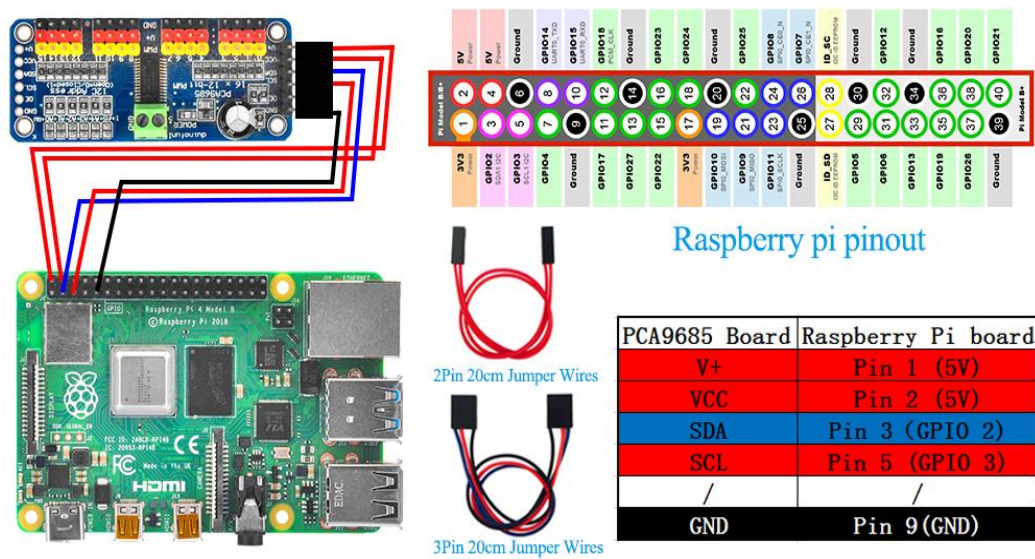


Figure 4.2: Connecting PCA9685 with raspberry pi

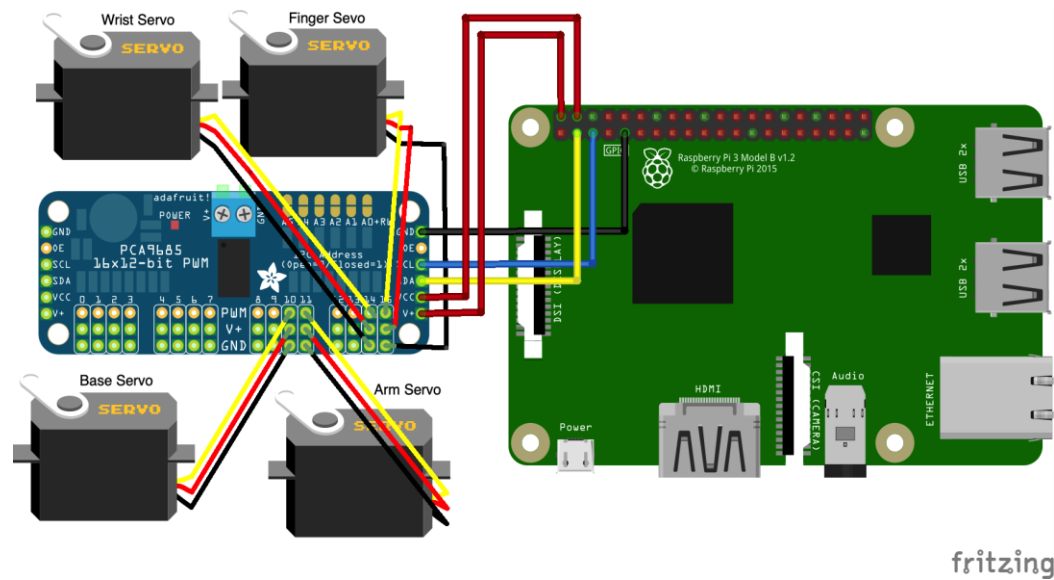


Figure 4.3: Connecting servos with PCA9685

4.2.2 Mechanical Model of the Arm

Here, we attempted to construct a mechanical arm with six degrees of freedom. DOF refers to the maximum number of conceptually independent values in a data sample or values that are not subject to change. Our example refers to the maximum number of rotations around the x, y, or z axes that our arm's joints may make. (Shinde, 2019) A robot's degrees of freedom are often stated in terms of the number of moving joints. A robot with three moveable joints will have three axes and three degrees of freedom; a robot with four moveable joints will have four axes and four degrees of freedom, and so on. A robot's degrees of freedom are often stated in terms of the number of movable joints. A robot with three movable joints will have three axes and three degrees of freedom; a robot with four moveable joints will have four axes and four degrees of freedom, and so on. (Anon, 2022). Detail diagram of our robotic arm is given below with 4 figures



Figure 4.4: Feature of 6 DOF arm

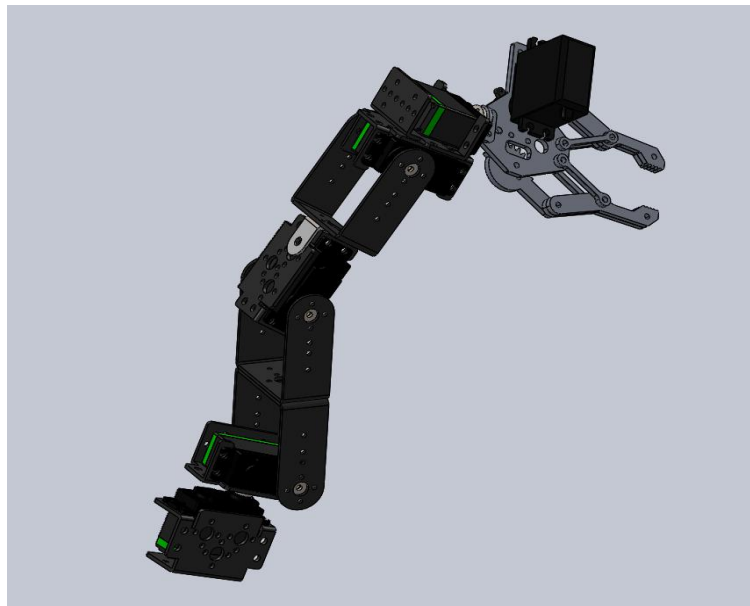


Figure 4.5: 3D model of our 6 DOF arm

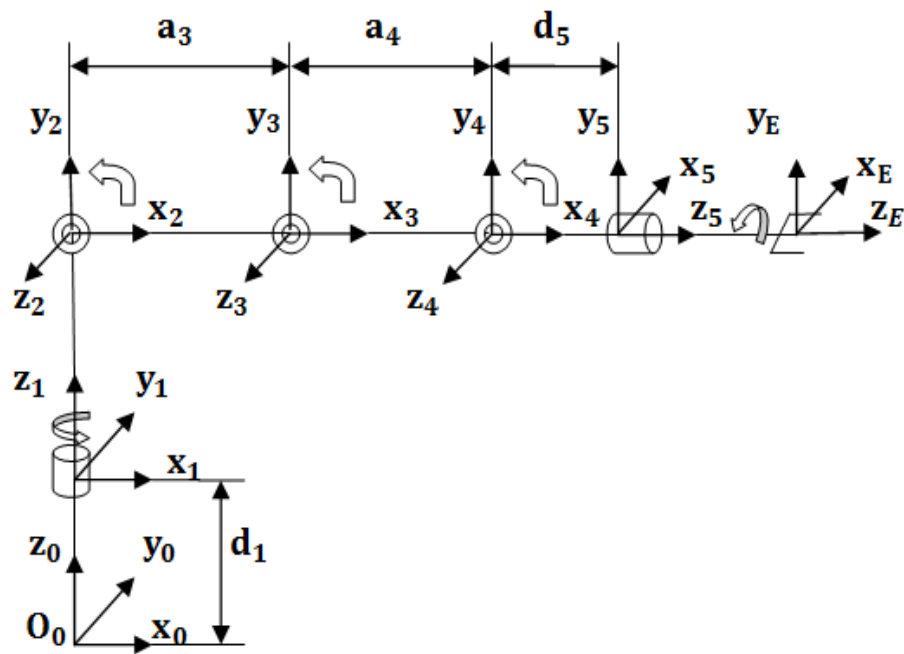


Figure 4.6: The robot coordinate frame

| Joint | Link | a_{i-1} mm | α_{i-1} degree | d_i mm | θ_i degree |
|-------|------|--------------|-----------------------|----------|-------------------|
| 0-1 | 1 | 0 | 0 | 45 | θ_1 |
| 1-2 | 2 | 0 | 90 | 0 | θ_2 |
| 2-3 | 3 | 90 | 0 | 0 | θ_3 |
| 3-4 | 4 | 90 | 0-90 | 0 | θ_4 |
| 4-5 | 5 | 0 | -90 | 30 | θ_5 |
| 5-6 | 6 | 0 | 0 | 0 | gripper |

Figure 4.7: DH parameters

4.2.3 Control Interface

We designed and coded our whole control interface from scratch using a high-level programming language called python. Python is a general-purpose language. It can be used to create various programs and isn't specialized for any specific problems. We used PyQt5 to design the front end of our control interface. The microcontroller is run by embedded C in the backend, and python communicates with the embedded system using serial communication.

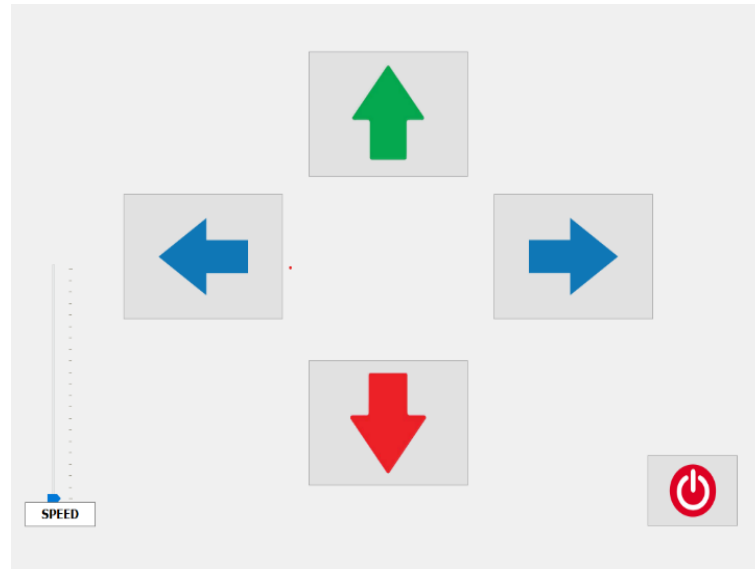


Figure 4.8: Rover control interface

Here, we can control our rover to move along any direction at our desired speed.

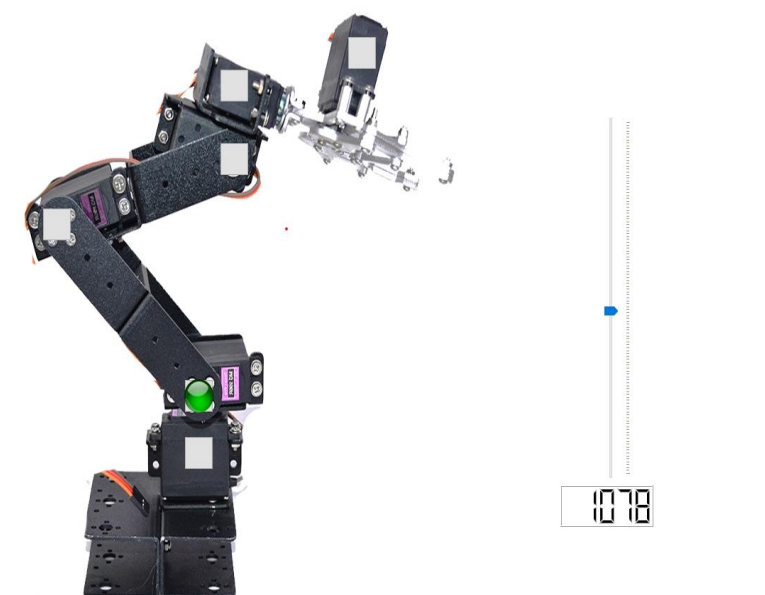


Figure 4.9: ARM control interface

In the above figure, the green dot represents which joints are we trying to rotate to a certain degree along a certain axis. And on the right side, the LCD screen shows the rotational degree.

4.3 Working Flow

The working flow of communication and power system and the flow diagram is shown below in figure 4.10 and 4.11

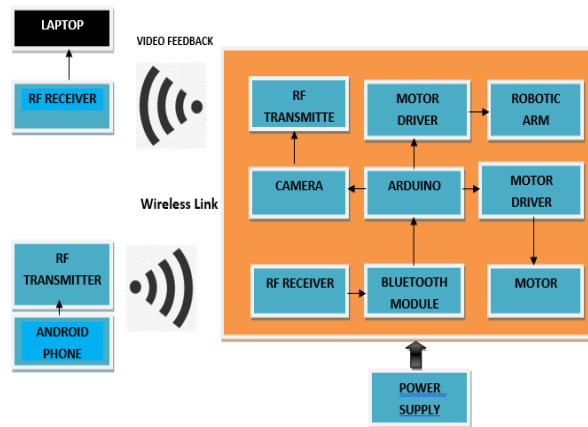


Figure 4.10: The working flow of communication and power system

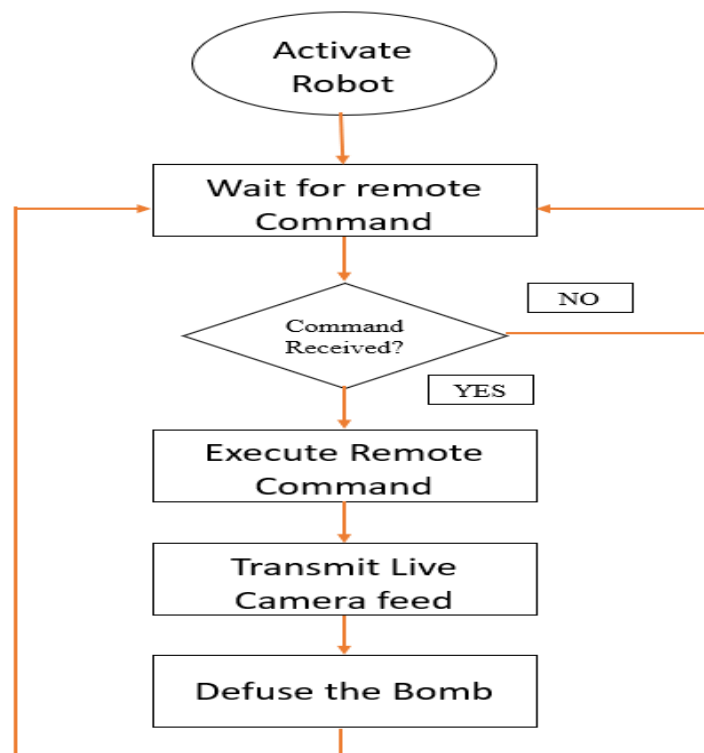


Figure 4.11: Experimental Flow Chart

4.4 Robotic System Architecture

The block diagram of system architecture is shown below in figure 4.12

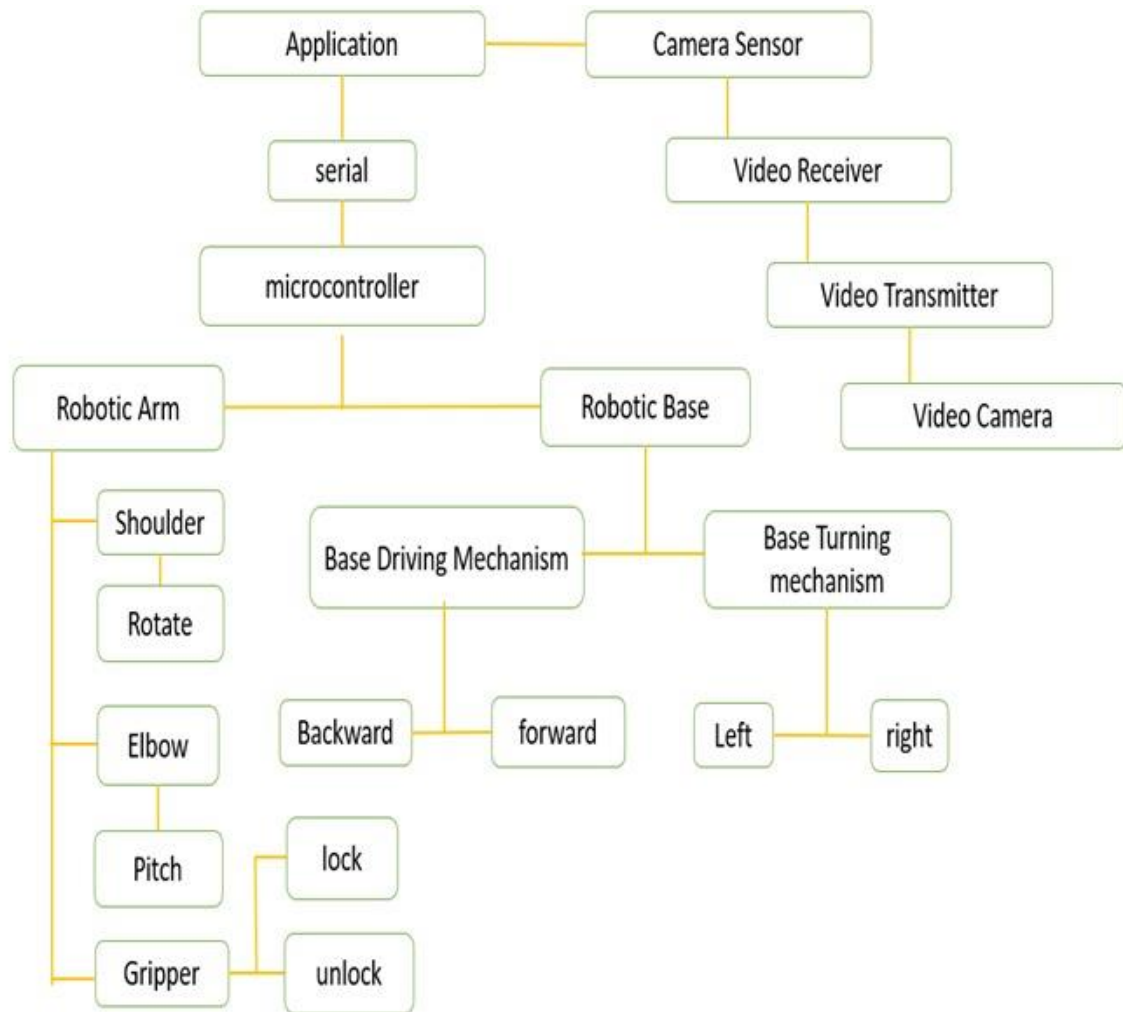


Figure 4.12: Block diagram of Robotics System architecture

CHAPTER FIVE

RESULT & DISCUSSION

5.1 Introduction

In this chapter, we will demonstrate the final outlook of our robot as well as the various configurations of our prototype.

5.2 Final Project Outlook

After assembling all the parts, plastic boards were used to cover the internal machinery to give it a better look. The camera with the antenna was kept outside to give visual feedback. The final outlook is shown in fig 5.1. The diagram of solidworks project is shown in 5.2



Figure 5.1: Final Project Outlook

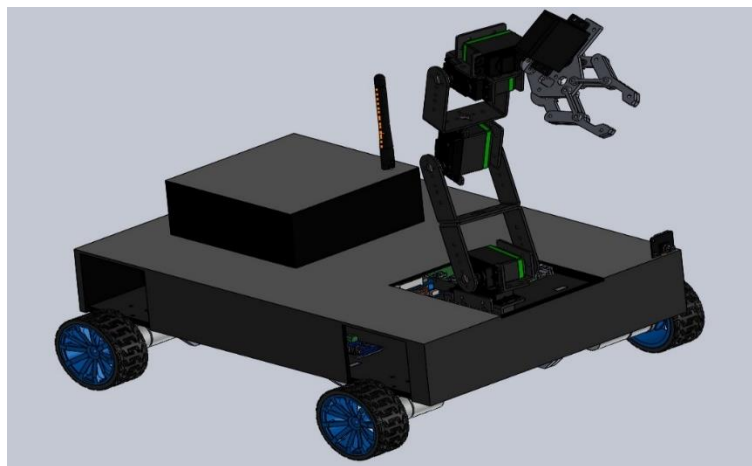


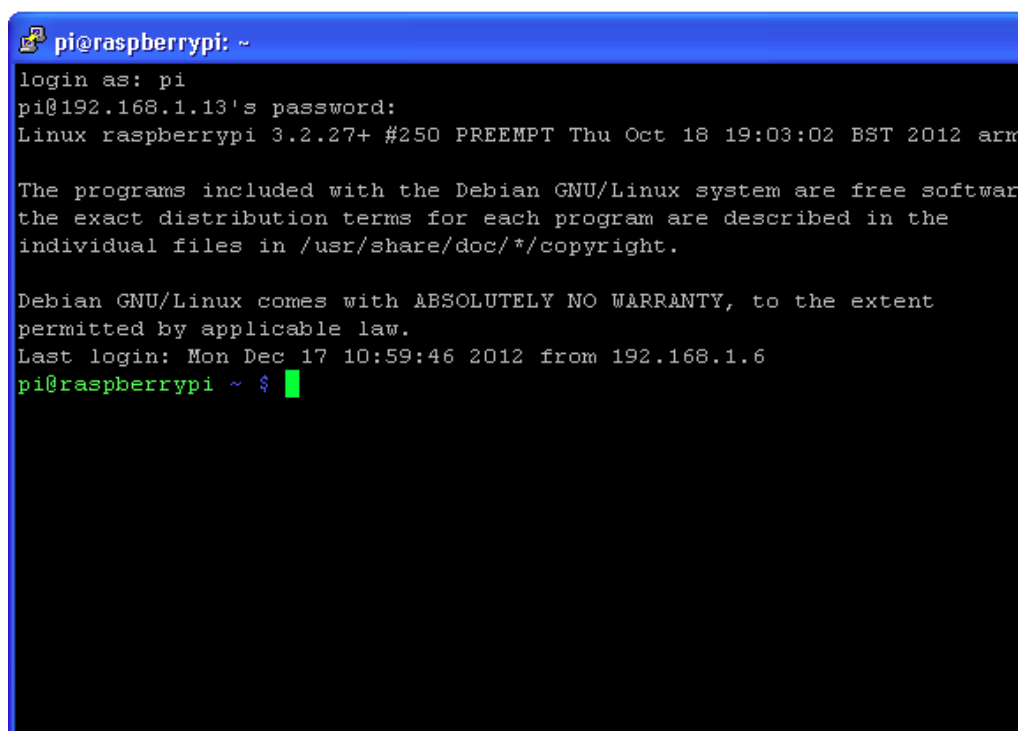
Figure 5.2: Final Project Outlook in Solid Work Design

5.3 Initialization & Testing

After assembling and initial setup, we moved to the testing part. The raspberry pi was first connected using the WIFI protocol. Then using VNC the codes for rover and arm was run for testing purpose.

5.3.1 Connect to Raspberry Pi by SSH Protocol

Raspberry Pi was connected to specific IP address using SSH protocol. The screenshot of the process is shown in figure 5.3

A screenshot of a terminal window showing an SSH login process. The window title is 'pi@raspberrypi: ~'. The text in the terminal is as follows:

```
login as: pi
pi@192.168.1.13's password:
Linux raspberrypi 3.2.27+ #250 PREEMPT Thu Oct 18 19:03:02 BST 2012 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Dec 17 10:59:46 2012 from 192.168.1.6
pi@raspberrypi ~ $
```

Figure 5.3: Connect to raspberry pi by SSH protocol

5.3.2 Setting Up VNC Server in Raspberry Pi

To set up the VNC server in the Raspberry Pi, we used the following coding. Screenshot of coding has been shown in figure 5.4

```

pi@raspberrypi: ~
File Edit Tabs Help
-rw----- 1 pi pi 427 Dec 26 18:17 .Xauthority
-rw----- 1 pi pi 706 Dec 26 18:17 .xsession-errors
-rw----- 1 pi pi 706 Dec 26 15:51 .xsession-errors.old
pi@raspberrypi:~ $ vncserver
VNC(R) Server 5.3.2 (RasPi) (r22164) ARMv6 (Sep 21 2016 14:09:31)
Copyright (C) 2002-2016 RealVNC Ltd.
RealVNC and VNC are trademarks of RealVNC Ltd and are protected by trademark
registrations and/or pending trademark applications in the European Union,
United States of America and other jurisdictions.
Protected by UK patent 2481870; US patent 8760366.
See http://www.realvnc.com for information on VNC.
For third party acknowledgements see:
http://www.realvnc.com/products/vnc/documentation/5.3/acknowledgements.txt

If a desktop environment fails to load for this virtual desktop, please see:
http://www.realvnc.com/doclink/kb-345?version=5.3.2.22164
Running applications in /etc/vnc/xstartup

VNC Server catchphrase: "Pinball rival oasis. Option forum yoyo."
signature: 92-48-b2-4f-12-b7-fe-7c

Log file is /home/pi/.vnc/raspberrypi:2.log
New desktop is raspberrypi:2 (192.168.0.109:2)
pi@raspberrypi:~ $

```

Figure 5.4: Setting up VNC server in raspberry pi

5.3.3 Connecting Raspberry Pi with Our Laptop Through VNC Server

Using the WIFI chip in Raspberry Pi, the controller was connected with the laptop. Same networks had to be matched. Screenshot of the process is shown in figure 5.5

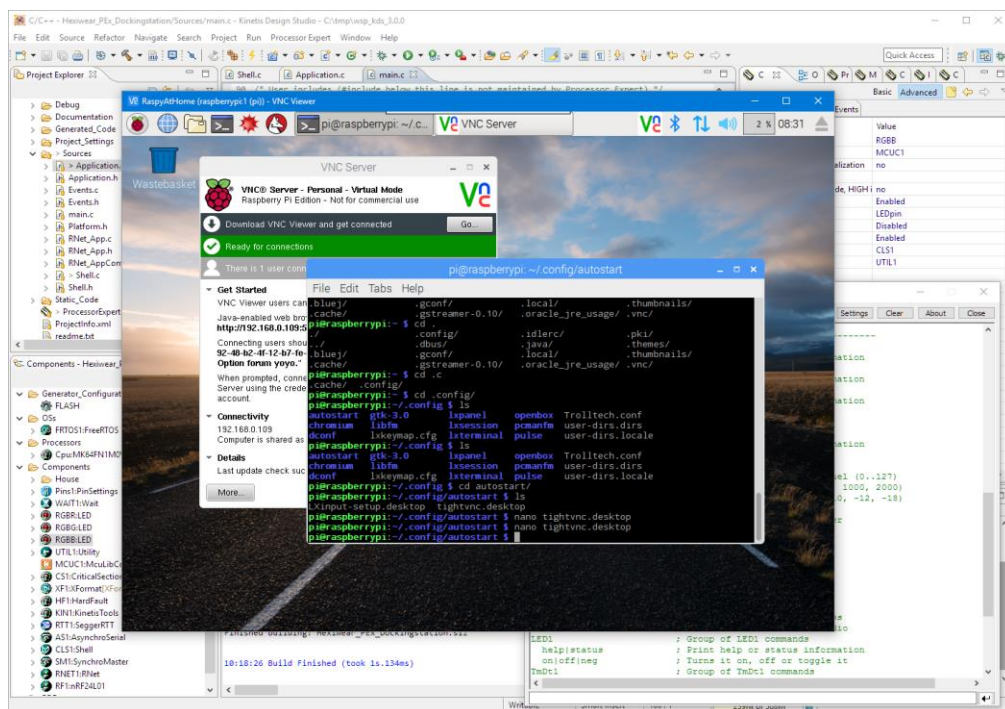


Figure 5.5: Connecting Raspberry pi with our laptop through VNC server

5.3.4 Open GUI interfaced from Raspberry pi

For controlling the rover and the robotic arm, the code written in python language had to be run using VNC server. The interface of the controller is shown in figure 5.6 and 5.7

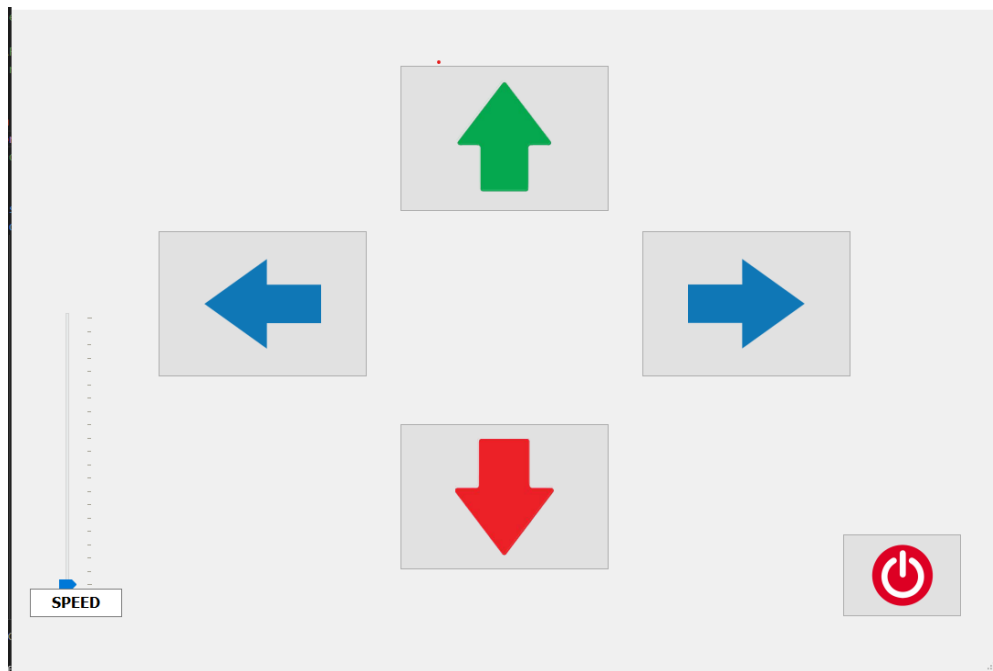


Figure 5.6: Open GUI interfaced from Raspberry pi

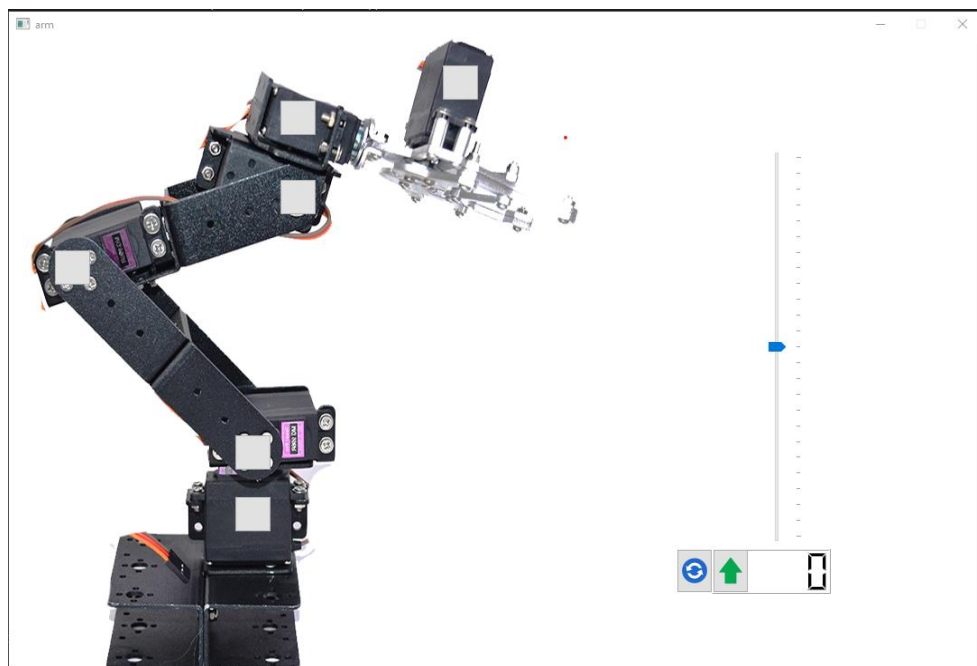


Figure 5.7: GUI interfaced of robotic arm control

5.3.5 Open Video Streaming Server for the ESP-32 Camera

The ESP-32 camera was used by connecting to the specific IP address allotted to the camera. Then streaming was started and the specific quality of video output was selected. The output of the camera is shown in figure 5.8

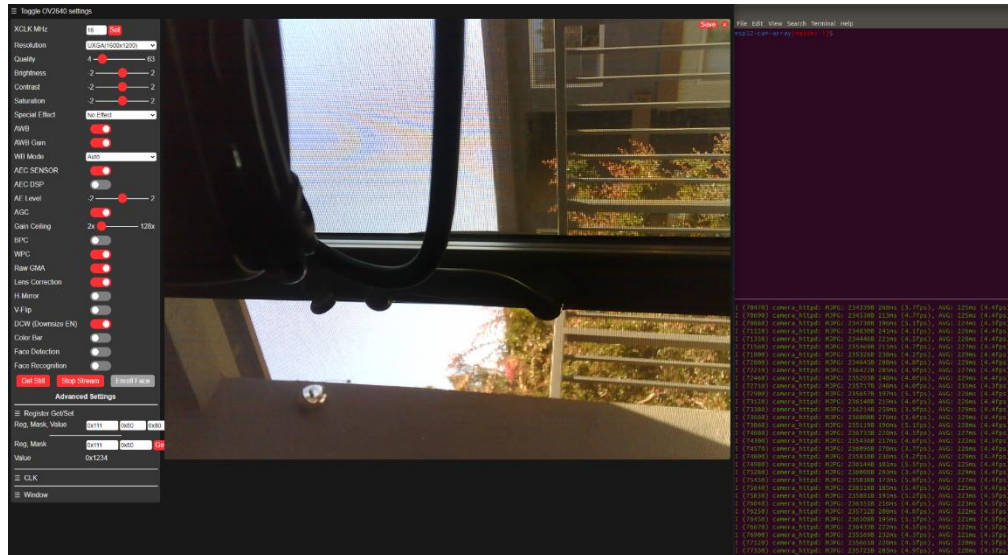


Figure 5.8: Open video streaming server for the esp-32 camera.

5.4 Result & its Analysis

After taking all the initial preparation, the robot was tested to fulfil its objectives. We could achieve all our objectives including wireless remote control, providing visual identification and implementing the gripper with the designed robotic arm. The final outcome is shown in figure 5.9 and 5.10

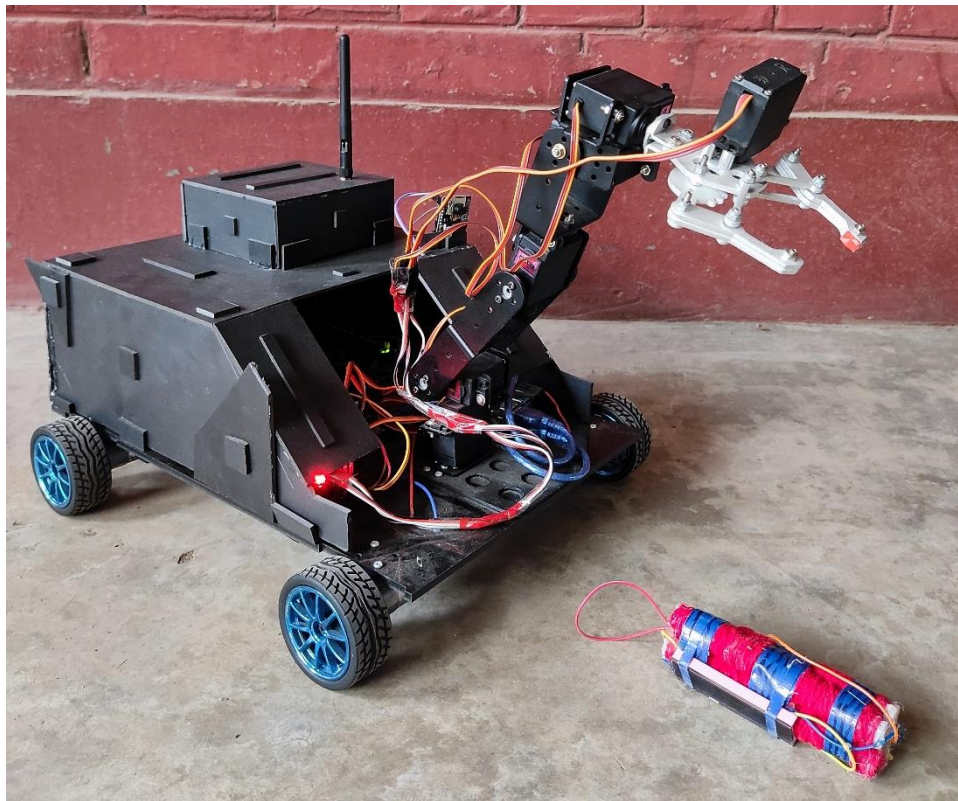


Figure 5.9: Robotic gripper in an open state



Figure 5.10: Robotic gripper in close state

CHAPTER SIX

CONCLUSION AND RECOMMENDATION

6.1 Conclusion

We intended to build a bomb disposal robot prototype utilizing the components that can be sourced from Bangladesh. Because of the pandemic and global chip shortage, sourcing electronics products wasn't easy. We saw a price hike of our intended components too. We had to plan our prototype within the limited amount of budget that was available to us.

6.2 Major Contribution

The major contributions of our project are discussed below:

- (a) The suggested system of the bomb-disposing robot will be very beneficial in the fields of security and spying on adversaries and regions inaccessible to humans; the robot will do the bomb-disposing operation.
- (b) Additionally, this robot is controlled remotely through the internet, ensuring that no human lives are endangered. A simple mechanical robot, IoT technology, and the interfacing of raspberry pi and Arduino modules and connections may combine to create the greatest bomb-disposing gadget, which would be very beneficial in saving human life over the internet.

6.3 Future Work

- (a) The rover can be made with a rocker-boogie mechanism, and suspensions can be added to withstand rough terrains.
- (b) A more robust processing unit such as the Jetson board by NVIDIA can be used in the future, which would enable our robot for enhanced computational power.
- (c) The mechanical gripper can be made with the help of 3D printing to be more precise to pursue delicate tasks.
- (d) Mine detection features can be introduced in our robot.

REFERENCES

- Anon, 2000. *TALON Tracked Military Robot*. [Online]
Available at: <https://www.army-technology.com/projects/talon-tracked-military-robot/>
- Anon, 2019. *What is Acrylic / Plexiglass?*. [Online]
Available at: <https://www.acmeplastics.com/what-is-acrylic-plexiglass>
- Anon, 2022. *What Is Degrees Of Freedom In Robotics?*. [Online]
Available at: <https://www.programatium.com/what-is-degrees-of-freedom-in-robotics/>
- Anon, n.d. *All About DC Motor Controllers - What They Are and How They Work*. [Online]
Available at: <https://www.thomasnet.com/articles/instruments-controls/dc-motor-controllers/>
- Anon, n.d. *RF Front End*. [Online]
Available at: <https://www.arrow.com/en/categories/rf-and-microwave/rf-ics/rf-front-end>
- Argel A. Bandala, E. P. D., November 2012. *Development and Design of Mobile Robot with IP-Based Vision System*. s.l., TENCON 2012 - 2012 IEEE Region 10 Conference.
- Hengnian Qi1, W. W. L.-z. J. L.-q. F., October 2007.. *Automatic Target Positioning of Bomb-disposal Robot Based on Binocular Stereo Vision*. s.l., Proceedings of the 2007 International Conference on Intelligent Systems and Knowledge Engineering (ISKE 2007).
- Homsup, N. H. T. J. W., 22 April 2008. *A Control of a Bomb Disposal Robot Using a Stereoscopic Vision (Non-Reviewed)*. s.l., IEEE.
- Oke Alice O., A. A., March 2014. *DEVELOPMENT OF A ROBOTIC ARM FOR DANGEROUS OBJECT*. s.l., s.n.
- Reddy, S. N. C. R., 02, February-2015. *Bomb Defusing Robotic Arm using Gesture*. s.l., International Journal of Engineering Research & Technology (IJERT).
- Richa Parmar, A. M. A. S. , B. P., Mar -2017. *Military Spying & Bomb Diffusing Robot with Night vision*. s.l., International Research Journal of Engineering and Technology (IRJET), pp. 2395 -0056.
- S.Keerthana, A. M., August 2019. *Bomb detection and disposal robot: Aid for risky Military Fields*. s.l., International Journal of Engineering and Advanced Technology (IJEAT).
- Sagar Randive, N. L. A. K. S. C. V. P., Mar 2012. *Hand Gesture Recognition BombDiffusing Surveillance Robot*. s.l., International Journal of Engineering Research and Applications (IJERA) .
- Shinde, R. U. V. K. P. S. S., 2019. A SECURE BOMB DIFFUSAL SPYROBOT CONTROLLED USING ANDROID APP. *IJCSMC*, 8(6), pp. 1-6.
- William M. Davidson, A. K. M. M. M. J. L. S. S. T. G. P. S., 10 May 2017. A Remote Maintenance Robot System for a Pulsed Nuclear Reactor. *Nuclear Technology* , 79 (3), pp. 249-259.
- Zeng Jian-Jun, Y. R.-Q. Z. W.-J. W. X.-H. Q. J., june 1, 2007. Research on Semi-automatic Bomb Fetching for an EOD Robot. *International Journal of Advanced Robotic Systems (IJARS)*.

APPENDIX

APPENDIX A: DC MOTOR CODE

```
#include <AFMotor.h>
AF_DCMotor motor1(1);
AF_DCMotor motor2(2);
AF_DCMotor motor3(3);
AF_DCMotor motor4(4);
char x;
String y;
int spd;
String sspd;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println("Rover test");
  spd = 70;
  motor1.setSpeed(spd);
  motor2.setSpeed(spd);
  motor3.setSpeed(spd);
  motor4.setSpeed(spd);
  delay(10);
}
void loop() {
  while (!Serial.available());
  y = Serial.readString();
  Serial.println(y);
  sspd = y.substring(1,4);
  spd = sspd.toInt();
  x = y[0];

  if (x=='w'){
    //allstop();
    delay(500);
    forward();
  }
  else if (x=='s'){
    //allstop();
    delay(500);
    backward();
  }
  else if (x=='d'){
    //allstop();
    delay(500);
    rightward();
  }
  else if (x=='a'){
```

```

    //allstop();
    delay(500);
    leftward();
}
else if (x=='f'){
    allstop();
}
}

void backward(){
    motor1.setSpeed(spд);
    motor2.setSpeed(spд);
    motor3.setSpeed(spд);
    motor4.setSpeed(spд);
    motor1.run(BACKWARD);
    motor2.run(BACKWARD);
    motor3.run(BACKWARD);
    motor4.run(BACKWARD);
}

void forward(){
    motor1.setSpeed(spд);
    motor2.setSpeed(spд);
    motor3.setSpeed(spд);
    motor4.setSpeed(spд);
    motor1.run(FORWARD);
    motor2.run(FORWARD);
    motor3.run(FORWARD);
    motor4.run(FORWARD);
}

void leftward(){
    motor1.setSpeed(spд);
    motor4.setSpeed(spд);
    motor2.run(RELEASE);
    motor3.run(RELEASE);
    motor1.run(FORWARD);
    motor4.run(FORWARD);
}

void rightward(){
    motor2.setSpeed(spд);
    motor3.setSpeed(spд);
    motor1.run(RELEASE);
    motor4.run(RELEASE);
    motor2.run(FORWARD);
    motor3.run(FORWARD);
}

void allstop(){
    motor1.run(RELEASE);
    motor2.run(RELEASE);
    motor3.run(RELEASE);
    motor4.run(RELEASE);
}

```

APPENDIX B: ARM CODE

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
#define USMIN 500 // This is the rounded 'minimum' microsecond length based on the
minimum pulse of 150
#define USMAX 1800 // This is the rounded 'maximum' microsecond length based on
the maximum pulse of 600
#define SERVO_FREQ 50 // Analog servos run at ~50 Hz updates
int servonum;
String x;
String arm;
String spos;
int pos1;
int pos2;
void setup() {
  Serial.begin(9600);
  Serial.println("6 DOF arm test!");
  pwm.begin();
  pwm.setOscillatorFrequency(27000000);
  pwm.setPWMFreq(SERVO_FREQ);
  delay(10);
}
void loop() {
  while (!Serial.available());
  x = Serial.readString();
  Serial.println(x);
  arm = x[0];
  spos = x.substring(1,5);
  pos1 = spos.toInt();
  spos = x.substring(5);
  pos2 = spos.toInt();
  servonum = arm.toInt();
  Serial.println(pos1);
  Serial.println(pos2);
  if (servonum==1){
    servonum = 0;
    moveServo(servonum, pos1, pos2);
  }
  else if (servonum==2){
    servonum = 2;
    pos1 = map(pos1, 600, 1800, 600, 1250);
    pos2 = map(pos2, 600, 1800, 600, 1250);
    moveServo(servonum, pos1, pos2);
  }
  else if (servonum==3){
    servonum = 4;
```



```

    moveServo(servonum, pos1, pos2);
}
else if (servonum==4){
    servonum = 6;
    moveServo(servonum, pos1, pos2);
}
else if (servonum==5){
    servonum = 8;
    moveServo(servonum, pos1, pos2);
}
else if (servonum==6){
    servonum = 15;
    pos1 = map(pos1, 600, 1800, 750, 1400);
    pos2 = map(pos2, 600, 1800, 750, 1400);
    moveServo(servonum, pos1, pos2);
}
else if (x[0]=='r'){
    // go to rest position
    pwm.writeMicroseconds(0, 1200);
    pwm.writeMicroseconds(2, 1250);
    pwm.writeMicroseconds(4, 1350);
    pwm.writeMicroseconds(6, 1150);
    pwm.writeMicroseconds(8, 1150);
    pwm.writeMicroseconds(15, 1150);
}
}
void moveServo(int servo, int pos1, int pos2){

// Drive each servo one at a time using writeMicroseconds(), it's not precise due to
calculation rounding!
// The writeMicroseconds() function is used to mimic the Arduino Servo library
writeMicroseconds() behavior.
if (pos1<pos2){
    for (uint16_t microsec = pos1; microsec < pos2; microsec++) {
        pwm.writeMicroseconds(servo, microsec);
    }
    delay(500);
}
else if (pos1>pos2){
    for (uint16_t microsec = pos1; microsec > pos2; microsec--) {
        pwm.writeMicroseconds(servo, microsec);
    }
    delay(500);
}
}
void moveServo1(int servo){
    for (uint16_t microsec = USMAX; microsec > USMIN; microsec--) {
        pwm.writeMicroseconds(servo, microsec);
    }
    delay(500);
}

```

}

APPENDIX C: ROVER PY

```
# arduinoData = serial.Serial('/dev/ttyACMO', '9600')
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        self.speed = 70
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1080, 720)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
        QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeightForWidth())
        MainWindow.setSizePolicy(sizePolicy)
        MainWindow.setMinimumSize(QtCore.QSize(1080, 720))
        MainWindow.setMaximumSize(QtCore.QSize(1080, 720))
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.down = QtWidgets.QPushButton(self.centralwidget)
        self.down.setGeometry(QtCore.QRect(425, 450, 230, 160))
        self.down.setText("")
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap("down.png"), QtGui.QIcon.Normal,
        QtGui.QIcon.Off)
        self.down.setIcon(icon)
        self.down.setIconSize(QtCore.QSize(128, 128))
        self.down.setObjectName("down")
        self.up = QtWidgets.QPushButton(self.centralwidget)
        self.up.setGeometry(QtCore.QRect(425, 60, 230, 160))
        self.up.setText("")
        icon1 = QtGui.QIcon()
        icon1.addPixmap(QtGui.QPixmap("up.png"), QtGui.QIcon.Normal,
        QtGui.QIcon.Off)
        self.up.setIcon(icon1)
        self.up.setIconSize(QtCore.QSize(128, 128))
        self.up.setObjectName("up")
        self.left = QtWidgets.QPushButton(self.centralwidget)
        self.left.setGeometry(QtCore.QRect(160, 240, 230, 160))
        self.left.setAutoFillBackground(False)
        self.left.setText("")
        icon2 = QtGui.QIcon()
        icon2.addPixmap(QtGui.QPixmap("left.png"), QtGui.QIcon.Normal,
        QtGui.QIcon.Off)
        self.left.setIcon(icon2)
        self.left.setIconSize(QtCore.QSize(128, 128))
        self.left.setObjectName("left")
        self.right = QtWidgets.QPushButton(self.centralwidget)
        self.right.setGeometry(QtCore.QRect(690, 240, 230, 160))
        self.right.setText("")
```

```

        icon3 = QtGui.QIcon()
        icon3.addPixmap(QtGui.QPixmap("right.png"), QtGui.QIcon.Normal,
QtGui.QIcon.Off)
        self.right.setIcon(icon3)
        self.right.setIconSize(QtCore.QSize(128, 128))
        self.right.setObjectName("right")
            self.verticalSlider = QtWidgets.QSlider(self.centralwidget)
        self.verticalSlider.setGeometry(QtCore.QRect(40, 280, 61, 350))
        self.verticalSlider.setOrientation(QtCore.Qt.Vertical)
        self.verticalSlider.setObjectName("verticalSlider")
        self.verticalSlider.setMinimum(0)
        self.verticalSlider.setMaximum(140)
        self.verticalSlider.setTickPosition(QtWidgets.QSlider.TicksRight)
        self.verticalSlider.setTickInterval(5)
        self.verticalSlider.valueChanged.connect(self.valuechange)
            self.lcdNumber = QtWidgets.QLCDNumber(self.centralwidget)
        self.lcdNumber.setGeometry(QtCore.QRect(0, 630, 120, 51))
        self.lcdNumber.setObjectName("lcdNumber")
            self.stop = QtWidgets.QPushButton(self.centralwidget)
        self.stop.setGeometry(QtCore.QRect(910, 570, 131, 91))
        self.stop.setText("")
        icon4 = QtGui.QIcon()
        icon4.addPixmap(QtGui.QPixmap("stop.png"), QtGui.QIcon.Normal,
QtGui.QIcon.Off)
        self.stop.setIcon(icon4)
        self.stop.setIconSize(QtCore.QSize(128, 128))
        self.stop.setObjectName("pushButton")
            MainWindow.setCentralWidget(self.centralwidget)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)
        self.up.clicked.connect(self.forward)
        self.down.clicked.connect(self.backward)
        self.left.clicked.connect(self.leftward)
        self.right.clicked.connect(self.rightward)
        self.stop.clicked.connect(self.allstop)
            self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)
    def valuechange(self):
        self.size = self.verticalSlider.value()
        self.lcdNumber.display(self.size)
        # print(self.size)
        def forward(self):
            print("forward") # do something with the button clicked
            # arduinoData.write('x')
        def backward(self):
            print("backward") # do something with the button clicked
        def rightward(self):
            print("rightward") # do something with the button clicked
        def leftward(self):

```

```

        print("leftward") # do something with the button clicked
        def allstop(self):
            print("stop") # do something with the button clicked
            def retranslateUi(self, MainWindow):
                _translate = QtCore.QCoreApplication.translate
                MainWindow.setWindowTitle(_translate("MainWindow", "ROVER"))
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())

```

APPENDIX D: ARM PY

```
# arduinoData = serial.Serial('/dev/ttyUSB0', '9600')
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        self.arm = None
        self.pos1 = 1200
        self.pos2 = 1350
        self.pos3 = 1350
        self.pos4 = 1150
        self.pos5 = 1150
        self.pos6 = 1150
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1080, 720)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeightForWidth())
        MainWindow.setSizePolicy(sizePolicy)
        MainWindow.setMinimumSize(QtCore.QSize(1080, 720))
        MainWindow.setMaximumSize(QtCore.QSize(1080, 720))
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(0, 0, 1080, 720))
        self.label.setMinimumSize(QtCore.QSize(1080, 720))
        self.label.setMaximumSize(QtCore.QSize(1080, 720))
        self.label.setText("")
        self.label.setPixmap(QtGui.QPixmap("arm1.png"))
        self.label.setObjectName("label")
        self.lcdNumber = QtWidgets.QLCDNumber(self.centralwidget)
        self.lcdNumber.setGeometry(QtCore.QRect(780, 580, 131, 51))
        self.lcdNumber.setObjectName("lcdNumber")
        self.verticalSlider = QtWidgets.QSlider(self.centralwidget)
        self.verticalSlider.setGeometry(QtCore.QRect(830, 130, 61, 441))
        self.verticalSlider.setOrientation(QtCore.Qt.Vertical)
        self.verticalSlider.setObjectName("verticalSlider_2")
        self.verticalSlider.setTickPosition(QtWidgets.QSlider.TicksRight)
        self.verticalSlider.setTickInterval(1)
        self.verticalSlider.valueChanged.connect(self.valuechange)
        self.verticalSlider.setMinimum(-60)
        self.verticalSlider.setMaximum(60)
        self.label_1 = QtWidgets.QLabel(self.centralwidget)
        self.label_1.setGeometry(QtCore.QRect(250, 520, 41, 41))
        self.label_1.setText("")
        self.label_1.setPixmap(QtGui.QPixmap("green_dot.png"))
        self.label_1.setScaledContents(True)
```

```

self.label_1.setObjectName("label_1")
self.label_1.setHidden(True)
    self.label_2 = QtWidgets.QLabel(self.centralwidget)
self.label_2.setGeometry(QtCore.QRect(250, 450, 41, 41))
self.label_2.setText("")
self.label_2.setPixmap(QtGui.QPixmap("green_dot.png"))
self.label_2.setScaledContents(True)
self.label_2.setObjectName("label_2")
self.label_2.setHidden(True)
    self.label_3 = QtWidgets.QLabel(self.centralwidget)
self.label_3.setGeometry(QtCore.QRect(300, 160, 41, 41))
self.label_3.setText("")
self.label_3.setPixmap(QtGui.QPixmap("green_dot.png"))
self.label_3.setScaledContents(True)
self.label_3.setObjectName("label_3")
self.label_3.setHidden(True)
    self.label_4 = QtWidgets.QLabel(self.centralwidget)
self.label_4.setGeometry(QtCore.QRect(50, 240, 41, 41))
self.label_4.setText("")
self.label_4.setPixmap(QtGui.QPixmap("green_dot.png"))
self.label_4.setScaledContents(True)
self.label_4.setObjectName("label_4")
self.label_4.setHidden(True)
    self.label_5 = QtWidgets.QLabel(self.centralwidget)
self.label_5.setGeometry(QtCore.QRect(480, 30, 41, 41))
self.label_5.setText("")
self.label_5.setPixmap(QtGui.QPixmap("green_dot.png"))
self.label_5.setScaledContents(True)
self.label_5.setObjectName("label_5")
self.label_5.setHidden(True)
    self.label_6 = QtWidgets.QLabel(self.centralwidget)
self.label_6.setGeometry(QtCore.QRect(300, 70, 41, 41))
self.label_6.setText("")
self.label_6.setPixmap(QtGui.QPixmap("green_dot.png"))
self.label_6.setScaledContents(True)
self.label_6.setObjectName("label_6")
self.label_6.setHidden(True)

self.pushButton1 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton1.setGeometry(QtCore.QRect(250, 520, 41, 41))
self.pushButton1.setText("")
self.pushButton1.setObjectName("pushButton1")
    self.pushButton6 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton6.setGeometry(QtCore.QRect(480, 30, 41, 41))
self.pushButton6.setText("")
self.pushButton6.setObjectName("pushButton2")
    self.pushButton5 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton5.setGeometry(QtCore.QRect(300, 70, 41, 41))
self.pushButton5.setText("")
self.pushButton5.setObjectName("pushButton3")

```

```

        self.pushButton4 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton4.setGeometry(QtCore.QRect(300, 160, 41, 41))
        self.pushButton4.setText("")
        self.pushButton4.setObjectName("pushButton4")
        self.pushButton3 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton3.setGeometry(QtCore.QRect(50, 240, 41, 41))
        self.pushButton3.setText("")
        self.pushButton3.setObjectName("pushButton5")
        self.pushButton2 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton2.setGeometry(QtCore.QRect(250, 450, 41, 41))
        self.pushButton2.setText("")
        self.pushButton2.setObjectName("pushButton6")
        self.up = QtWidgets.QPushButton(self.centralwidget)
        self.up.setGeometry(QtCore.QRect(780, 580, 40, 50))
        self.up.setText("")
        icon1 = QtGui.QIcon()
        icon1.addPixmap(QtGui.QPixmap("up.png"), QtGui.QIcon.Normal,
QtGui.QIcon.Off)
        self.up.setIcon(icon1)
        self.up.setIconSize(QtCore.QSize(32,32))
        self.up.setObjectName("up")
        self.r = QtWidgets.QPushButton(self.centralwidget)
        self.r.setGeometry(QtCore.QRect(740, 580, 40, 50))
        self.r.setText("")
        icon2 = QtGui.QIcon()
        icon2.addPixmap(QtGui.QPixmap("Background.png"), QtGui.QIcon.Normal,
QtGui.QIcon.Off)
        self.r.setIcon(icon2)
        self.r.setIconSize(QtCore.QSize(32,32))
        self.r.setObjectName("reset")

self.label.raise_()
self.verticalSlider.raise_()
self.lcdNumber.raise_()
self.pushButton1.raise_()
self.label_1.raise_()
self.pushButton2.raise_()
self.pushButton3.raise_()
self.pushButton4.raise_()
self.pushButton5.raise_()
self.pushButton6.raise_()
self.up.raise_()
self.r.raise_()
self.label_2.raise_()
self.label_4.raise_()
self.label_3.raise_()
self.label_6.raise_()
self.label_5.raise_()
MainWindow.setCentralWidget(self.centralwidget)
self.retranslateUi(MainWindow)

```

```

QtCore.QMetaObject.connectSlotsByName(MainWindow)
self.pushButton1.clicked.connect(self.amr1)
self.pushButton2.clicked.connect(self.amr2)
self.pushButton3.clicked.connect(self.amr3)
self.pushButton4.clicked.connect(self.amr4)
self.pushButton5.clicked.connect(self.amr5)
self.pushButton6.clicked.connect(self.amr6)
self.up.clicked.connect(self.upload)
self.r.clicked.connect(self.reset)
def reset(self):
    # arduinoData.write(text.encode())
    print("reset")
    def upload(self):
        text = None
        if self.arm == 1:
            text= str(self.arm)+str(self.pos1)+str(self.size)
            self.pos1 = self.size
        elif self.arm == 2:
            text= str(self.arm)+str(self.pos2)+str(self.size)
            self.pos2 = self.size
        elif self.arm == 3:
            text= str(self.arm)+str(self.pos3)+str(self.size)
            self.pos3 = self.size
        elif self.arm == 4:
            text= str(self.arm)+str(self.pos4)+str(self.size)
            self.pos4 = self.size
        elif self.arm == 5:
            text= str(self.arm)+str(self.pos5)+str(self.size)
            self.pos5 = self.size
        elif self.arm == 6:
            text= str(self.arm)+str(self.pos6)+str(self.size)
            self.pos6 = self.size
        print("sending data ",text)
        # arduinoData.write(text.encode())
    def valuechange(self):
        self.size = self.verticalSlider.value()
        self.lcdNumber.display(self.size)
        # print(self.size)
        def mapped(x, in_min, in_max, out_min, out_max):
            return int((x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min)
        self.size = mapped(self.size, -60, 60, 600, 1800)
        if self.size < 1000:
            self.size = '0'+str(self.size)
    def amr1(self):
        print("arm1")
        self.arm = 1
        # arduinoData.write(str(self.arm).encode())
        self.label_1.setHidden(False)
        self.label_2.setHidden(True)
        self.label_3.setHidden(True)

```



```

        self.label_4.setHidden(True)
        self.label_5.setHidden(True)
        self.label_6.setHidden(True)
def amr2(self):
    print("arm2")
    self.arm = 2
    #arduinoData.write(str(self.arm).encode())
    self.label_1.setHidden(True)
    self.label_2.setHidden(False)
    self.label_3.setHidden(True)
    self.label_4.setHidden(True)
    self.label_5.setHidden(True)
    self.label_6.setHidden(True)
def amr3(self):
    print("arm3")
    self.arm = 3
    #arduinoData.write(str(self.arm).encode())
    self.label_1.setHidden(True)
    self.label_2.setHidden(True)
    self.label_4.setHidden(False)
    self.label_3.setHidden(True)
    self.label_5.setHidden(True)
    self.label_6.setHidden(True)
def amr4(self):
    print("arm4")
    self.arm = 4
    #arduinoData.write(str(self.arm).encode())
    self.label_1.setHidden(True)
    self.label_2.setHidden(True)
    self.label_4.setHidden(True)
    self.label_3.setHidden(False)
    self.label_5.setHidden(True)
    self.label_6.setHidden(True)
def amr5(self):
    print("arm5")
    self.arm = 5
    #arduinoData.write(str(self.arm).encode())
    self.label_1.setHidden(True)
    self.label_2.setHidden(True)
    self.label_3.setHidden(True)
    self.label_4.setHidden(True)
    self.label_6.setHidden(False)
    self.label_5.setHidden(True)
def amr6(self):
    print("arm6")
    self.arm = 6
    #arduinoData.write(str(self.arm).encode())
    self.label_1.setHidden(True)
    self.label_2.setHidden(True)
    self.label_3.setHidden(True)

```

```

        self.label_4.setHidden(True)
        self.label_6.setHidden(True)
        self.label_5.setHidden(False)
    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "arm"))
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())

```

APPENDIX E: ESP32 CAMERA

```
#include "esp_camera.h"
#include <WiFi.h>
#define CAMERA_MODEL_AI_THINKER // Has PSRAM
#include "camera_pins.h"
const char* ssid = "helal";
const char* password = "90909090";
void startCameraServer();
void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  // if PSRAM IC present, init with UXGA resolution and higher JPEG quality
  //           for larger pre-allocated frame buffer.
  if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }
  #if defined(CAMERA_MODEL_ESP_EYE)
    pinMode(13, INPUT_PULLUP);
    pinMode(14, INPUT_PULLUP);
  #endif
  // camera init
```

```

esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
}
sensor_t * s = esp_camera_sensor_get();
// initial sensors are flipped vertically and colors are a bit saturated
if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1); // flip it back
    s->set_brightness(s, 1); // up the brightness just a bit
    s->set_saturation(s, -2); // lower the saturation
}
// drop down frame size for higher initial frame rate
s->set_framesize(s, FRAMESIZE_QVGA);

#if defined(CAMERA_MODEL_M5STACK_WIDE) ||
defined(CAMERA_MODEL_M5STACK_ESP32CAM)
    s->set_vflip(s, 1);
    s->set_hmirror(s, 1);
#endif
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");
startCameraServer();
Serial.print("Camera Ready! Use 'http://");
Serial.print(WiFi.localIP());
Serial.println("' to connect");
}

void loop() {
    // put your main code here, to run repeatedly:
    delay(10000);
}

```