Helwan University

Faculty of Engineering

Computer Engineering Department

Frontline Service Robot

Prepared By:

Sief Eldien Eslam Mohamed Abdellatif

Sherif Ashraf Farouk Younis

Fady Tawadrous Ayoub Tawadrous

Under Supervision of:

Dr. Rania Darwish

Assistant Professor at Faculty of Engineering

Helwan University

July 2021

Acknowledgement

We would like to thank Dr. Rania Darwish for her guidance and support. Also, thanks to everyone who contributed to supporting and spreading open-source projects and free software like the Arduino platform and the Free Software Foundation that helps us and many other people to achieve their goals.

Abstract

Service robots assist humans by performing a job that is distant, dangerous, or repetitive, reducing labor cost and providing quick and accurate services. Service robots has many possible applications in the current time and keeps advancing to take more tasks.

In our project we are concerned about Frontline Service Robots which are autonomous systems that interact, communicate, and deliver service to an organization's customer, specifically indoor service robots. We are also exploring the challenges facing indoor robots and different indoor localization and navigation techniques and applying navigation technique using near field communication for navigation and mapping to build an indoor service robot that can be used in restaurants, hospitals, factories and more.

Keywords

- Service Robots
- Frontline Service Robots
- Near Field Communication
- NFC
- Radio-Frequincy Identification
- RFID
- Indoor Posioning System
- Robot Navigation
- Indoor Navigation

Table of Contents

Table of figures

Table of tables

List of Acronyms and Abbreviations

FSR	Frontline Service Robot
NFC	Near Field Communication
RFID	Radio-Frequency Identification
IDE	Integerated Development Kit
API	Application Programming Interface
ML	Machine Learning
AI	Artificial Intelligence
IPS	Indoor Posioning System
LCD	Liquid Crystal Display
GPS	Global Posioning System
Wi-Fi	Wireless Fidelity
QR	Quick Response
IR	Infrared
P2P	Peer-to-Peer
I2C	Inter-Integerated-Circuit

Table 0-1: List of Acronyms and Abbreviations

Chapter 1 : Introduction

1.1. Problem Statement

We try to reduce the human factor in many services to achieve better quality services and lower the labor cost.

Frontline workers are employees who must physically show up to their jobs and interact with the customer like waiters in restaurants, or nurses delivering food and medicine to patients.

By automating these services using indoor service robots we will achieve better quality services, lower cost, healthier environment, and overall better customer satisfaction.

1.2. Problem Importance

Frontline occupations usually are repetitive, dull, labor intensive, and face a variety of health risks to both workers and customers, also these jobs are very important because better services lead to better customer experiences and improve customer loyalty.

The current pandemic led to the shutdown of many businesses to limit virus infection and spread, by replacing human workers with service robots in different environments like restaurants we drastically lowering the risk of virus infection by limiting human interaction.

Service robots in hospitals that can carry the service of delivering the food and medicine to patients can help the hospital staff in times of congestion, and lower the human interaction resulting in healthier environment.

1.3. Current Solutions and Previous work

- SERVEXA: The Serving Robot (2020)

  Reference: https://www.ijert.org/research/servexa-the-serving-robot-IJERTCONV8IS14015.pdf

  By Shruti B P, Anuson JoseHarshitha, Sagar Shah, Shree Lakshmi

  Faculty of Information Science and Engineering, Sri Krishna Institute of Technology, Bangalore, India.

Figure 1-1: ServeXa: The Serving Robot

  A restaurant service robot made to deliver food to customers by tracking lines on the floor using Infrared sensor to navigate the restaurant and interacting with the workers and customer by attached keypad and LCD screen.

- Design of Restaurant Service Robot for Contact less and Hygienic Eating Experience (August 2020)

  Reference: https://www.irjet.net/archives/V7/i8/IRJET-V7I8496.pdf

  By Prejitha.CT, Vikram Raj.N, Harshavardhan Vibhandik, Gayatri Wadyalkar, Pushpak Tiple, Khushi Kapoor, Sai Prajna Manduri, Ankush Oswal, Manasi Sanjivan Kulkarni

  School of Electrical Engineering, MITAOE, Alandi, Pune, India

  Dept. of Computer Science, SIES Graduate School of Technology, Mumbai, India

Dept. of mechanical engineering, SRKR Engineering College

Dept. of Computer Science, Modern Education Society college of Engineering, Pune

9Student, Dept. of Computer Science Shri Vile Parle Kelvani Mandal Institute of Technology, Dhule, India

Figure 1-2: Restaurant Service Robot 2

A line following robot designed to take orders from customers by working with GUI on customer smartphones and using attached keypad to select the order and deliver the food to customers while avoiding obstacles utilizing ultrasonic sensors.

- Servi Robot by Softbank

Figure 1-3: Servi by Softbank

Servi, the new service robot made by the Japanese company Softbank, designed to deliver the food to customers.

It will use 3D cameras and LIDAR technology to navigate around tables and customers.

When the robot launches, companies can lease the robot for $950 per month over a three-year period.

1.4. Objectives

- Design and build a service robot that can be implemented in different environment like restaurants and hospitals to deliver food and medicine.
- The robot must be simple to interact with by both customers and workers.
- The robot must be inexpensive and easy to maintain.
- The implementation of the robot must not require a drastic change in environment must be able to adapt and avoid obstacles.

1.5. Motivation

Our motivation is to help as many people as possible their everyday life by automating repetitive and dull tasks.

We also aim to help restaurants, hospitals, and other businesses to deliver higher quality services with better and more reliable work environment.

And least we aim to limit human interaction to lower the risk of spread of viruses and to produce a healthier environment overall.

1.6. Project Idea

Frontline Service Robot (FSR) is an indoor service robot designed to deliver things (food, medicine) from starting point to a destination while avoiding obstacles.

By using RFID tags as landmarks and compass sensor the robot can identify its location and orientation and navigate through the environment to reach the target location.

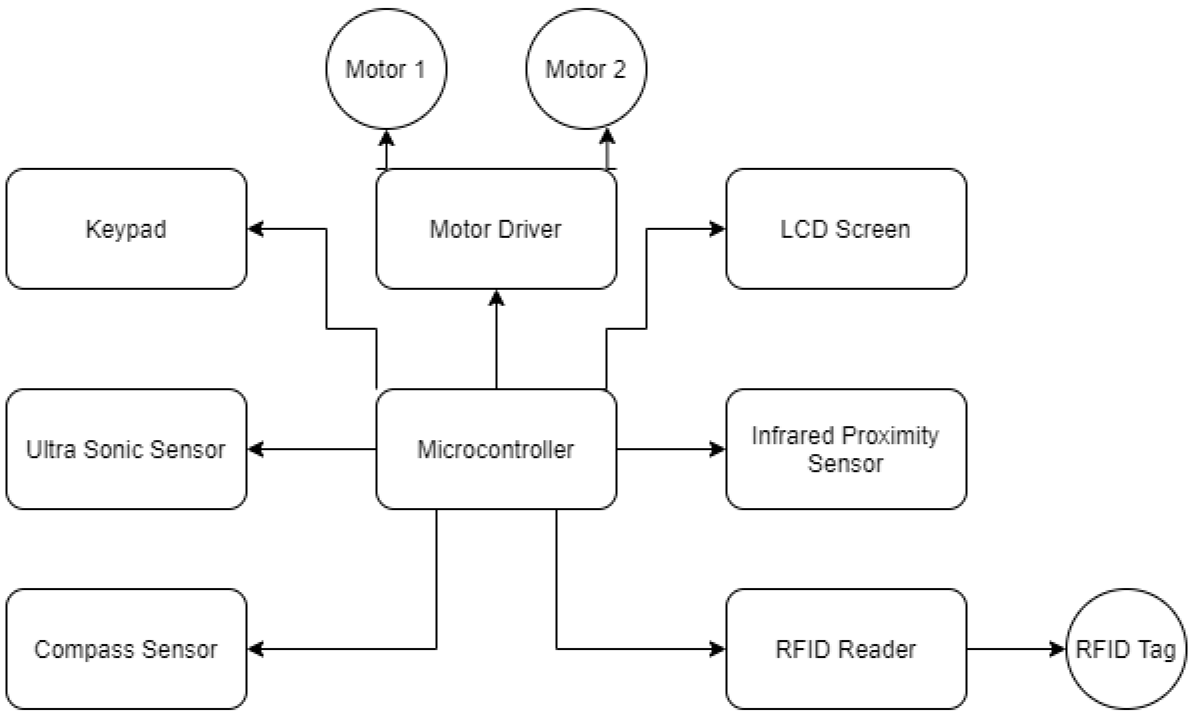


Figure 1-4: Project Diagram

1.7. Project Applications

- In restaurant: to deliver food to customers.
- In hospitals: to deliver food and medicine to patients.
- In factories and warehouses: to move parts and goods from certain location to a desired target location repeatedly.

1.8. Document Overview

- Chapter 1: Introduction and overview.
- Chapter 2: Explanation of necessary background knowledge.
- Chapter 3: Market analysis and future expectation.
- Chapter 4: Design specification and features.
- Chapter 5: Presenting the hardware components and software components used in this project, and the mechanical design.
- Chapter 6: Implementation of the hardware and software components and project result
- Chapter 7: Summary, conclusion, and exploring future work.

Chapter 2 : Necessary Background Knowledge

2.1. Service Robots Definition

The International Organization for Standardization defines a “service robot” as a robot “that performs useful tasks for humans or equipment excluding industrial automation applications. According to ISO 8373 robots require “a degree of autonomy”, which is the ability to perform intended tasks based on current state and sensing, without human intervention. For service robots this ranges from partial autonomy - including human robot interaction - to full autonomy without active human robot intervention. Service robots have many forms and structures as well as many application areas. The possible applications of robots to assist in human chores is widespread. At present there are a few main categories that is these robots fall into:

- Industrial Service Robots: Can be used to carry out simple tasks, such as examining welding, as well as more complex and dangerous tasks, such as aiding in the dismantling of nuclear power stations.

Figure 2-1: Industrial Service Robot

- Frontline Service Robots: Autonomous and adaptable robots that interact, communicate, and deliver services to an organization’s customers.

Figure 2-2: Frontline Service Robot

- Domestic Service Robots: Perform tasks that humans regularly perform in non-industrial environments, like people's homes such as cleaning floors, mowing the lawn and pool maintenance.

Figure 2-3: Domestic Robot

- Scientific Service Robots: Perform many functions such as repetitive tasks performed in research. These range from the multiple repetitive tasks made by gene samplers and sequencers to systems which can almost replace the scientist in designing and running experiments, analyzing data, and even forming hypotheses. Also, they can perform tasks which humans would find difficult or impossible, from the deep sea to outer space.

Figure 2-4: Scientific Robot

2.2. Indoor Localization and Navigation

Indoor localization and navigation problems are the biggest challenges facing service robots that are designed to be used indoors. The robot needs to always know its location and direction and provide the required service while navigating the environment smoothly without colliding with obstacles.

An indoor positioning system (IPS) is a network of devices used to locate people or objects inside closed areas where GPS and other satellite technologies lack precision or fail entirely.

Many techniques and devices are used to provide indoor positioning ranging using many different technologies such as Wi-Fi and Bluetooth beacons placed throughout a defined space. Lights, radio waves, magnetic fields, and acoustic signals are all used in IPS networks.

Robot localization implies the robot’s ability to recognize its own position and orientation. Path planning is an extension of localization, in that it requires the determination of the robot's current position and a position of a goal location, both within the same frame of reference or coordinates.

Robot navigation means the robot's ability to determine its own position in its frame of reference and then to plan a path towards some goal location. In order to navigate in its environment, the robot requires representation of this environment, basically a map of the environment and the ability to interpret that representation. So, navigation can be defined as the combination of the three fundamental competences: Self-localization, Path planning, and Map building and map interpretation.

There are many available navigation techniques that differ in complexity, cost, techniques for static routes, techniques for dynamic routes, required sensors, and required technologies. Available navigation techniques:

- By Vision: Vision-based navigation or optical navigation uses computer vision algorithms and optical sensors, including cameras and laser-based range finder to extract the visual features required to the localization and navigation in the surrounding environment.

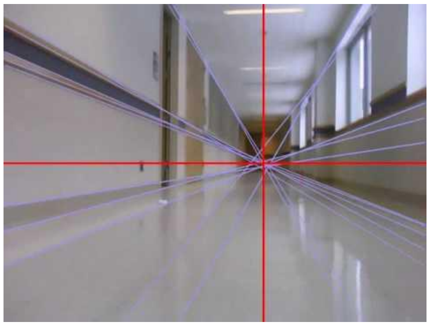


Figure 2-5: Navigation by vision

- By RFID: By using RFID tags as landmarks on the floor to represent a point in the environment’s map the robot can navigate to its destination by moving from a tag to another while comparing its current position with the required destination.

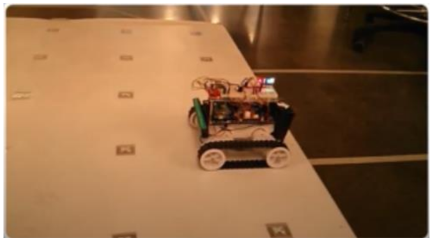


Figure 2-6: Navigation by RFID

- By QR Codes: Similar to RFID technique we use QR code on the floor as landmarks to guide the robot around its environment.

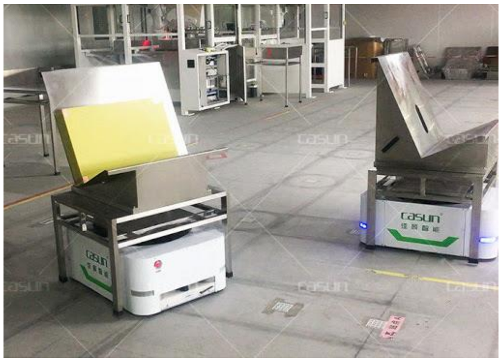


Figure 2-7: Navigation by QR codes

- By Line tracking System: By using line following sensor using infrared and marking every intersection between two lines as a point in the environment map the robot can navigate following the lines to the goal destination.

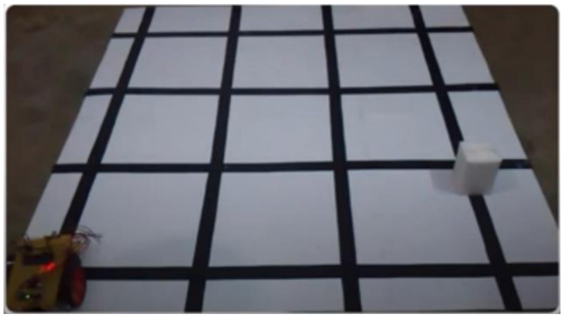


Figure 2-8: Navigation by line tracking

2.3. RFID and NFC

2.3.1. Radio-frequency identification

Radio-frequency Identification or (RFID) is a one-way communication method that uses electromagnetic fields to identify and track RFID tags.

An RFID system consists of a tiny radio transponder, a radio receiver and transmitter. When triggered by an electromagnetic interrogation pulse from a nearby RFID reader device, the tag transmits digital data, usually an identifying inventory number, back to the reader.

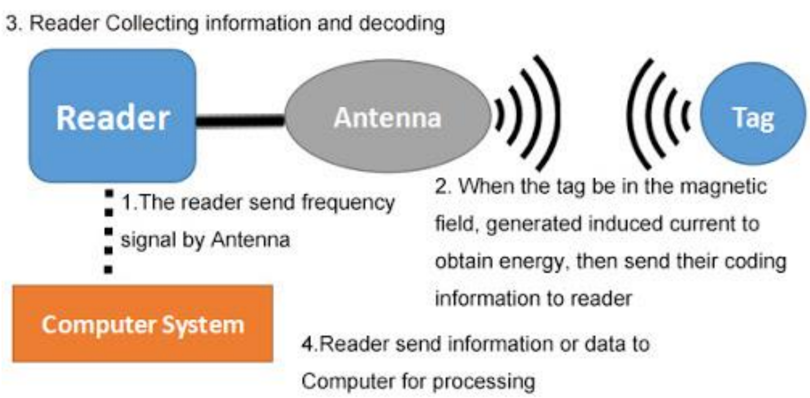


Figure 2-9: RFID System

RFID systems can be classified by the type of tag and reader into:

- Passive Reader Active Tag (PRAT): system has a passive reader which only receives radio signals from active tags. The range of PRAT systems is (0-600 m).

- Active Reader Passive Tag (ARPT): system has an active reader, which transmits interrogator signals and receives replies from passive tags.
- Active Reader Active Tag (ARAT): system uses active tags activated with an interrogator signal from the active reader.

RFID tags: are made of three pieces: a microchip to store and process information and modulates and demodulates radio frequency (RF) signals, an antenna for receiving and transmitting the signal and a substrate.

The tag information is stored in a non-volatile memory. The RFID tag includes either fixed or programmable logic for processing the transmission and sensor data.

RFID tags also comes in two different forms of transponders:

- Passive tags: Tags that need to be powered by the energy from the RFID reader's interrogating radio waves. With no battery inside the tag itself passive tags can last very long time, and have very small size, allowing them to be less noticeable.
- Active tags: Tags that are powered by a battery and thus can be read at greater distance from the RFID reader. They are unique because unlike passive RFID tags, they are continually transmitting a signal out. That means as the active RFID tag approaches a scanner, the scanner recognizes the signal being transmitted. But with battery inside the tag, an active tag can be bigger in size and more expensive than a passive tag.

The scanner can notice the RFID tags, but they have no additional uses on their own. RFID can utilize different frequency levels that dictates the RFID's distance:

- Low frequency (LF) is the shortest.
- High frequency (HF) is a medium distance.
- Ultra-high frequency (UHF) can go up to ten feet.

Examples of RFID use cases:

- Finding lost goods

- Inventory management
- Personal racking
- Access management
- Animal tracking
- Billing processes

2.3.2 Near Field Communication

Near-Field Communication or (NFC) is a two-way communication method using a set of communication protocols for peer to peer (p2p) communication between two electronic devices over a very short distance, around (4 cm) or less.

NFC can be described as a special version of RFID technology using special communication protocols and only works over very short distance and on specific frequency (13.56 MHz).

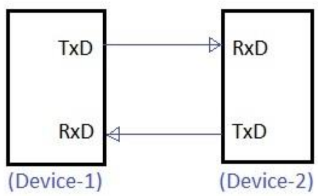
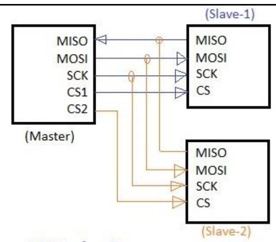
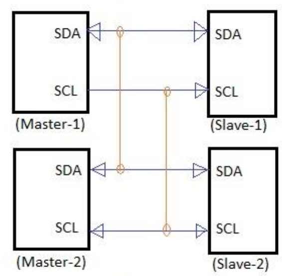
NFC devices can work in three modes:

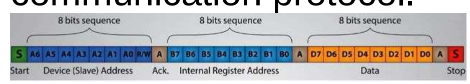
- NFC peer-to-peer: Enables two NFC-enabled devices to communicate with each other to exchange information.
- NFC reader/writer: Enables NFC-enabled devices to read information stored on NFC tags.
- NFC card emulation: Enables NFC-enabled devices such as smartphones to act like smart cards.

Example of NFC use cases:

- Contactless payments
- Access control
- Automation
- Landmark identification

2.4. Communication Protocols (UART – SPI – I2C)

Features	**UART**	**SPI**	**I2C**
Full Form	Universal Asynchronous Receiver/Transmitter	Serial Peripheral Interface	Inter-integrated Circuit
Interface Diagram	 UART Interface Diagram	 SPI Interface Diagram	 I2C Interface Diagram
Pin Designatious	TxD: Transmit Data RxD: Receive Data	SCLK: Serial Clock MOSI: Master Output, Slave Input MISO: Master Input, Slave Output SS: Slave Select	SDA: Serial Data SCL: Serial Clock
Data Rate	As this is asynchronous communication, data rate between two devices wanting to communicate should be set to equal value. Maximum data rate supported is about 230 Kbps to 460kbps.	Maximum data rate limit is not specified in SPI interface. Usually supports about 10 Mbps to 20 Mbps	I2C supports 100 kbps, 400 kbps, 3.4 Mbps. Some variants also support 10 Kbps and 1 Mbps.
Distance	About 50 feet	Highest	Higher
Type of communication	Asynchronous	synchronous	synchronous

Number of Masters	None	One	One or more
Clock	No Common Clock signal is used. Both the devices will use their independent clocks.	There is one common serial clock signal between master and slave devices.	There is common clock signal between multiple masters and multiple slaves.
Hardware Complexity	lesser	less	more
Protocol	For 8 bits of data one start bit and one stop bit is used.	Each company or manufacturers have got their own specific protocols to communicate with peripherals. Hence one needs to read datasheet to know read/write protocol for SPI communication to be established. For example, we would like SPI communication between microcontroller and EPROM. Here one need to go through read/write operational diagram in the EPROM data sheet.	It uses start and stop bits. It uses ACK bit for each 8 bits of data which indicates whether data has been received or not. Figure depicts the data communication protocol. 

Software Addressing	As this is one to one connection between two devices, addressing is not needed.	Slave select lines are used to address any particular slave connected with the master. There will be 'n' slave select lines on master device for 'n' slaves.	There will be multiple slaves and multiple masters and all masters can communicate with all the slaves. Up to 27 slave devices can be connected/addressed in the I2C interface circuit.
Advantages	• It is simple communication and most popular which is available due to UART support in almost all the devices with 9 pin connector. It is also referred as RS232 interface.	•It is simple protocol and hence so not require processing overheads. •Supports full duplex communication. •Due to separate use of CS lines, same kind of multiple chips can be used in the circuit design. •SPI uses push-pull and hence higher data rates, and longer ranges are possible. •SPI uses less power compared to I2C	•Due to open collector design, limited slew rates can be achieved. •More than one masters can be used in the electronic circuit design. •Needs fewer i.e., only 2 wires for communication. •I2C addressing is simple which does not require any CS lines used in SPI and it is easy to add extra devices on the bus. •It uses open collector bus concept. Hence there is bus voltage flexibility on the interface bus. •Uses flow control.
Disadvantages	• They are suitable for communication between only two devices.	• As number of slave increases, number of CS lines increases,	•Increases complexity of the circuit when number of slaves and masters increases. •I2C interface is half duplex.

	• It supports fixed data rate agreed upon between devices initially before communication otherwise data will be garbled.	this results in hardware complexity as number of pins required will increase. • To add a device in SPI requires one to add extra CS line and changes in software for particular device addressing is concerned. •Master and slave relationship cannot be changed as usually done in I2C interface. •No flow control available in SPI.	•Requires software stack to control the protocol and hence it needs some processing overheads on microcontroller/microprocessor.

Table 2-1: Communication Protocols

Chapter 3 : Market Analysis and Predictions

3.1. Statistics

The International Federation of Robotics (IFR) Statistical Department carries out annual statistical survey on service robotics sales. The data is evaluated and published in the World Robotics Service Robotics report.

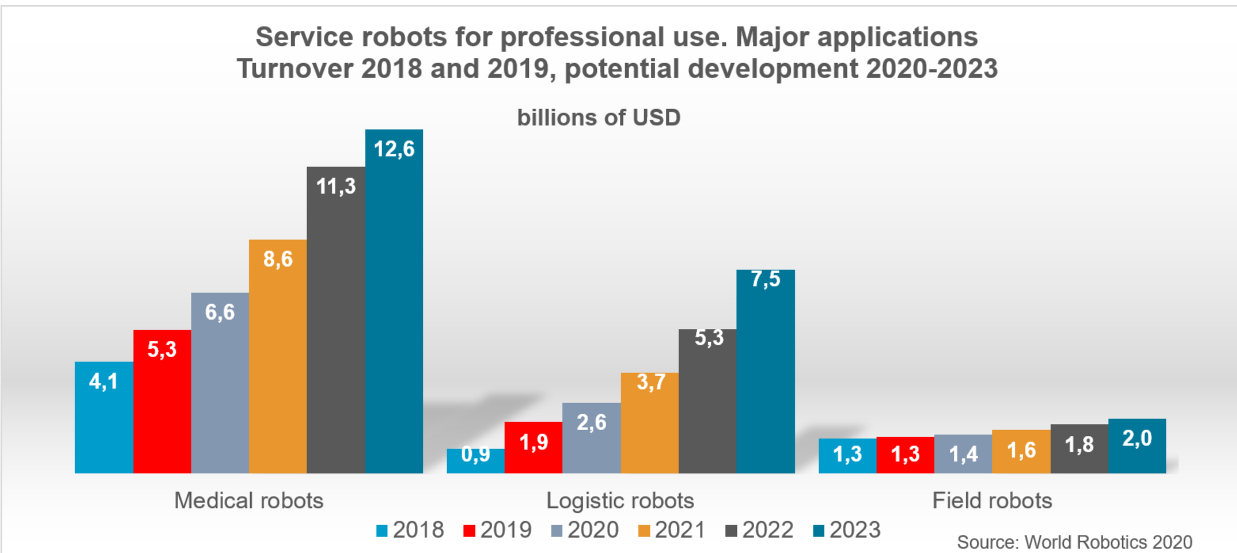


Figure 3-1: Service robots' potential development

The rise of demand of service robot increases every year and becoming one of the biggest industries in the world.

Global service robotics market size is expected to garner $34.7 billion by 2022.

The service robotics market is led by America, with an estimated sales value of 5.6 billion U.S. dollars in 2018. This value is forecast to increase to some 12 billion U.S. dollars in 2022 just in the USA.

According to MarketsandMarkets, the service robotics market is projected to grow from USD 37.0 billion in 2020 to USD 102.5 billion by 2025; it is expected to grow at a Compound Annual Growth Rate (CAGR) of 22.6%

from 2020 to 2025. The increase in adoption of service robots due to the decrease in labor costs and the more reliable services, there has been a sharp increase in the funding for research on robots by both the government as well as key market players.

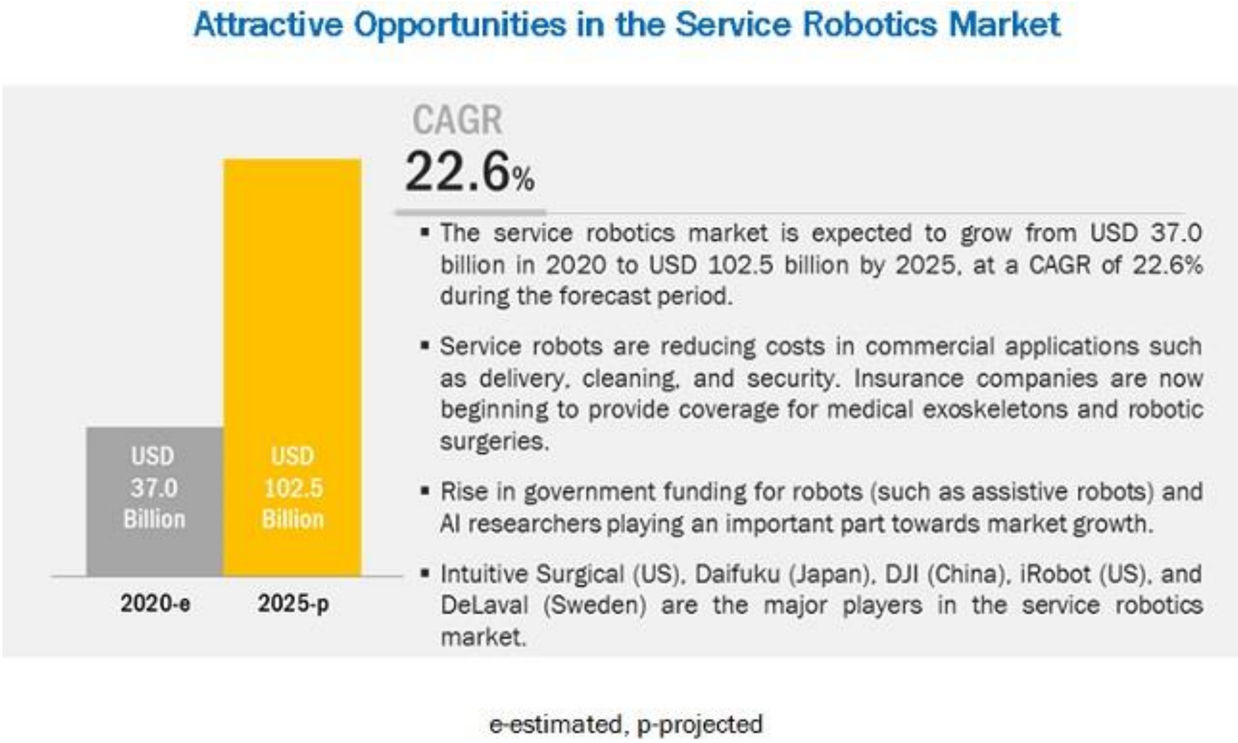


Figure 3-2: Service robots CAGR

Market for service robots estimated to grow at Compound Annual Growth Rate (CAGR) of 22.6%, with personal and domestic robots projected to grow at higher CAGR compared to professional robots.

Hardware components sales will increase rapidly, due to the innovative designs required on the hardware side. Sensors and control units on both domestic and commercial type service robots remain expensive. Batteries often contribute to a major portion to the cost and. Although hardware component will have the larger share of the market, it is expected to decrease over time due to economies of scale.

North America is expected to account for largest share of service robotics market throughout forecast period.

3.2. Key Market Players

- SoftBank Robotics Group (Japan): A holding company of the SoftBank Group that oversees the robotics business in the SoftBank Group, established in 2012.
- iRobot (US): A global consumer robot company specialized in making robot vacuums like the Roomba.
- Intuitive Surgical (US): An American corporation that develops, manufactures, and markets robotic products designed to improve clinical outcomes of patients through minimally invasive surgery.
- DJI (China): A Chinese technology company that manufactures commercial unmanned aerial vehicles (drones) for aerial photography and videography.
- Boston Dynamics (US): An American company specializes in manufacturing mobile robots such as Spot, Handle, Pick, and Atlas (Humanoid), which are generally used in warehouses, The also design humanoid robots and autonomous cleaning solutions. The company got acquired by SoftBank in 2017.
- Other major companies like DeLaval (Sweden), Daifuku (Japan), CYBERDYNE (Japan), and a few emerging companies worldwide.

3.3. Customers Perspective on Service Robots

During the service encounter, customers often place a premium on pleasant relations with service employees, and so providing emotional and social value. However, in the next few years, it is estimated that 85 percent of all customer interactions will take place without a human agent.

According to the technology acceptance model (TAM), a customer's intention to use a new technology depends on the cognitive evaluation of its perceived usefulness and ease of use. However, the service must not just deliver the core, but frequently also social-emotional and relational elements of the service.

It is assumed that service robots will perform well on the functional dimensions and, therefore, will not hinder adoption. In fact, this is an important difference to customers' technology acceptance in a service context, that is, the adoption of Self-Serving Technology (SSTs). Specifically, SSTs face frequently a long adoption period by customers who often fear they do not know how to operate an SST, may get stuck and cannot complete a transaction. It can be assumed that the adoption of service robots will be faster and smoother than for most SSTs.

Dimension	SST	Service Robots
Service scripts and roles	Customers have to learn the service script and role, and follow it carefully	Can guide the customer through the service process, with flexible interactions
Customer error tolerance	Generally, do not function well when customers make errors	Will be more error tolerant
Service recovery	The service process tends to break down when there is a failure, so the recovery is unlikely within the technology	Can recover from customer error, and offer alternative solutions

Table 3-1: SST and Service Robots Comparison

Service robots can guide customers through the process. Even customer errors can be corrected by the robot, making robot-delivered service much more robust than existing SSTs. Customers will be able to interact with the robot much like with a service employee. That is, usefulness and ease of use seem to be a given in most cases but would be a barrier if not provided at a level required by customers.

Chapter 4 : Design Specifications

4.1. Requirement Analysis

4.1.1. Functional Requirements

Functional Requirement	Description
Derliver Food/Medicine	Order the robot to go to a specific destiniation to deliver and item
Dismiss the robot	Through IR sensor the customer can dimiss the robot and the robot will return to its origin location
Avoid obstacles	Through an ultrasonic sensor the robot must navigate its environment without colliding with any obstacle

Table 4-1: Functional Requirements

4.1.2. Nonfunctional Requirements

Nonfunctional Requirements	Description
Availability	The robot should be available during working hours.
Security	No one can access the robot from outside the restaurant. Only the manager of the restaurant can update the system
Safety	The robot must avoid colliding with people or objects while moving around

Table 4-2: Nonfunctional Requirements

4.1.3. Functional Requirements Specification

- Stakeholders

  Engineer: Design and build the robot.

  Worker: Order the robot to go to specific location.

Customer/Patient: Receive food/medicine and dismiss the robot.

Hospital/Restaurant Manager

- Users:

  Worker

  Customer

- Use Cases:

  Deliver item: A worker can order the robot to go to specific location to deliver food/medicine.

  Dismiss the robot: A customer can dismiss the robot; the robot will leave and return to its main location.

4.2. Project Diagrams

4.2.1 Block Diagram

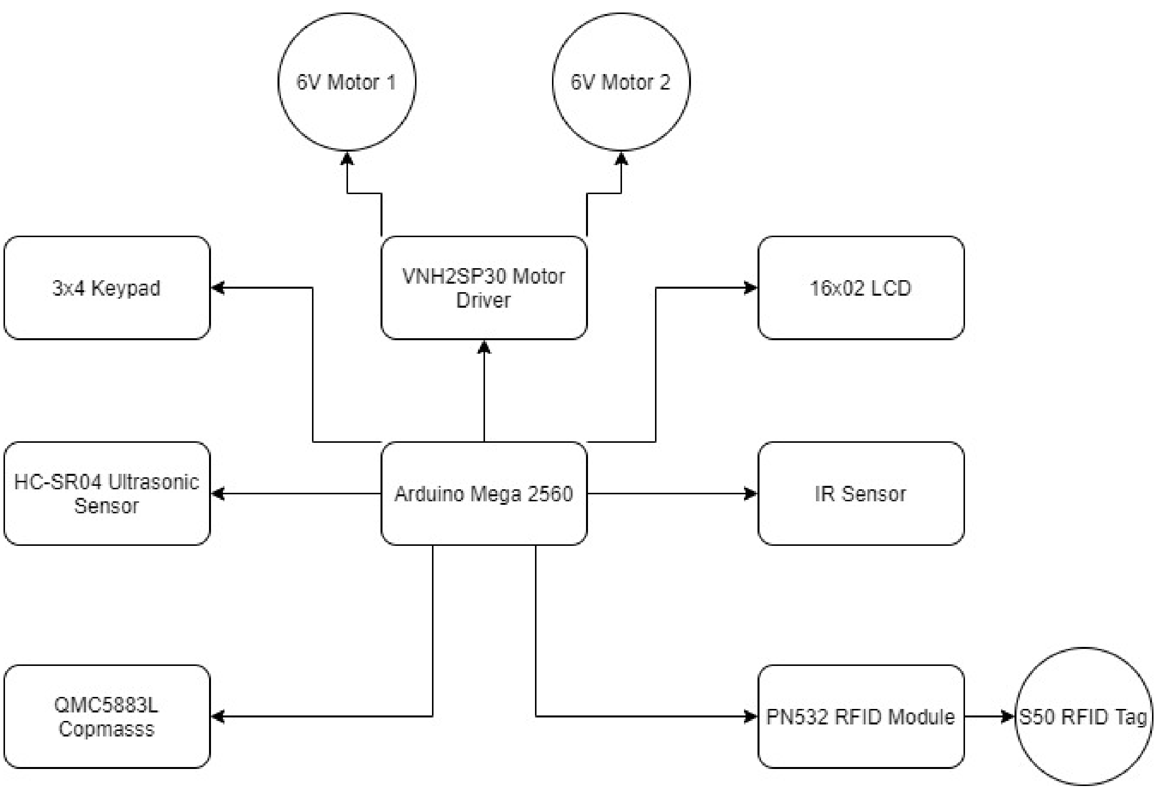


Figure 4-1: Block Diagram

4.2.2. Flowchart

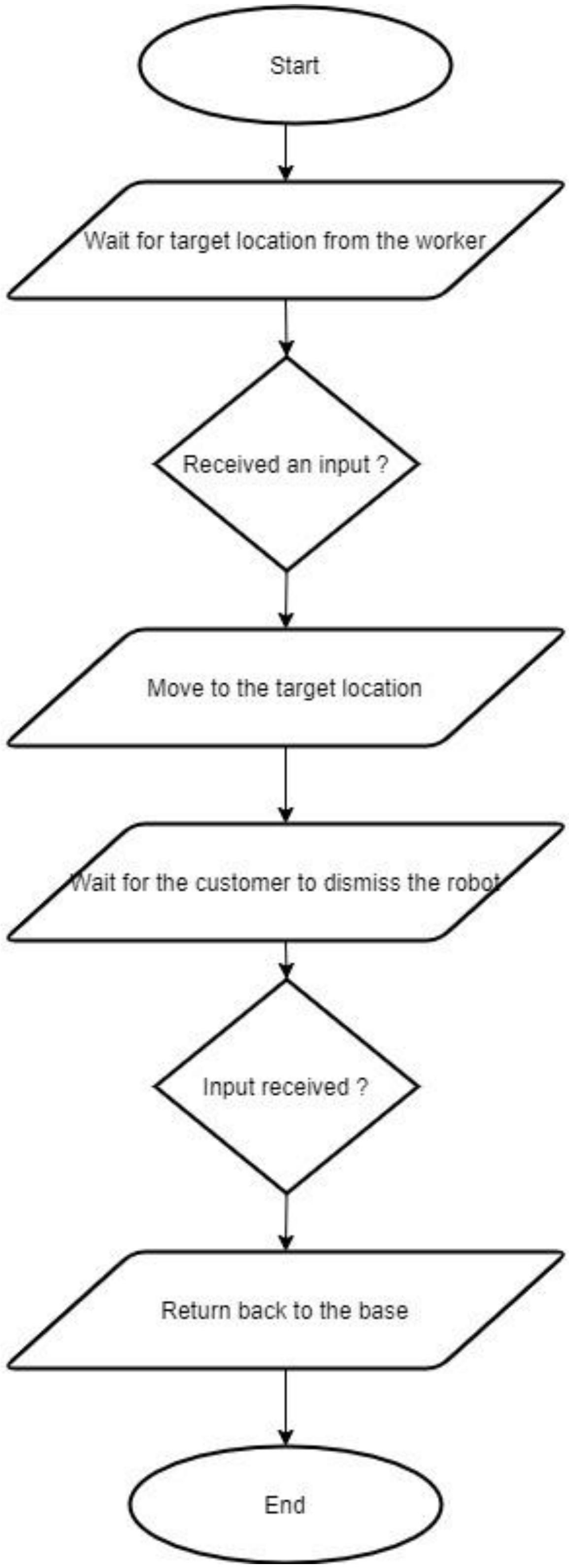


Figure 4-2: Flowchart

4.2.3. Use Case Diagram

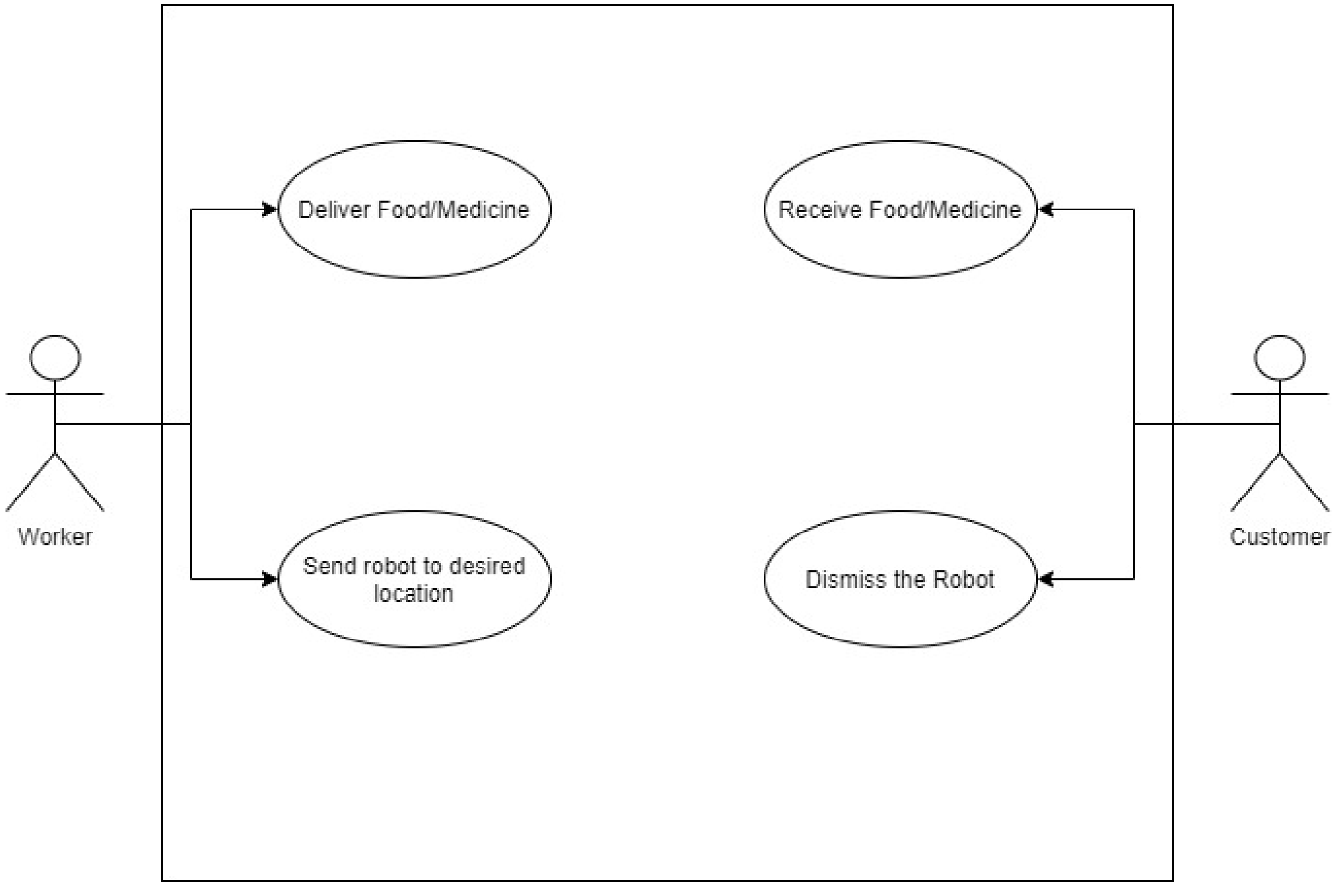


Figure 4-3: Use case Diagram

4.2.4. Sequence Diagram

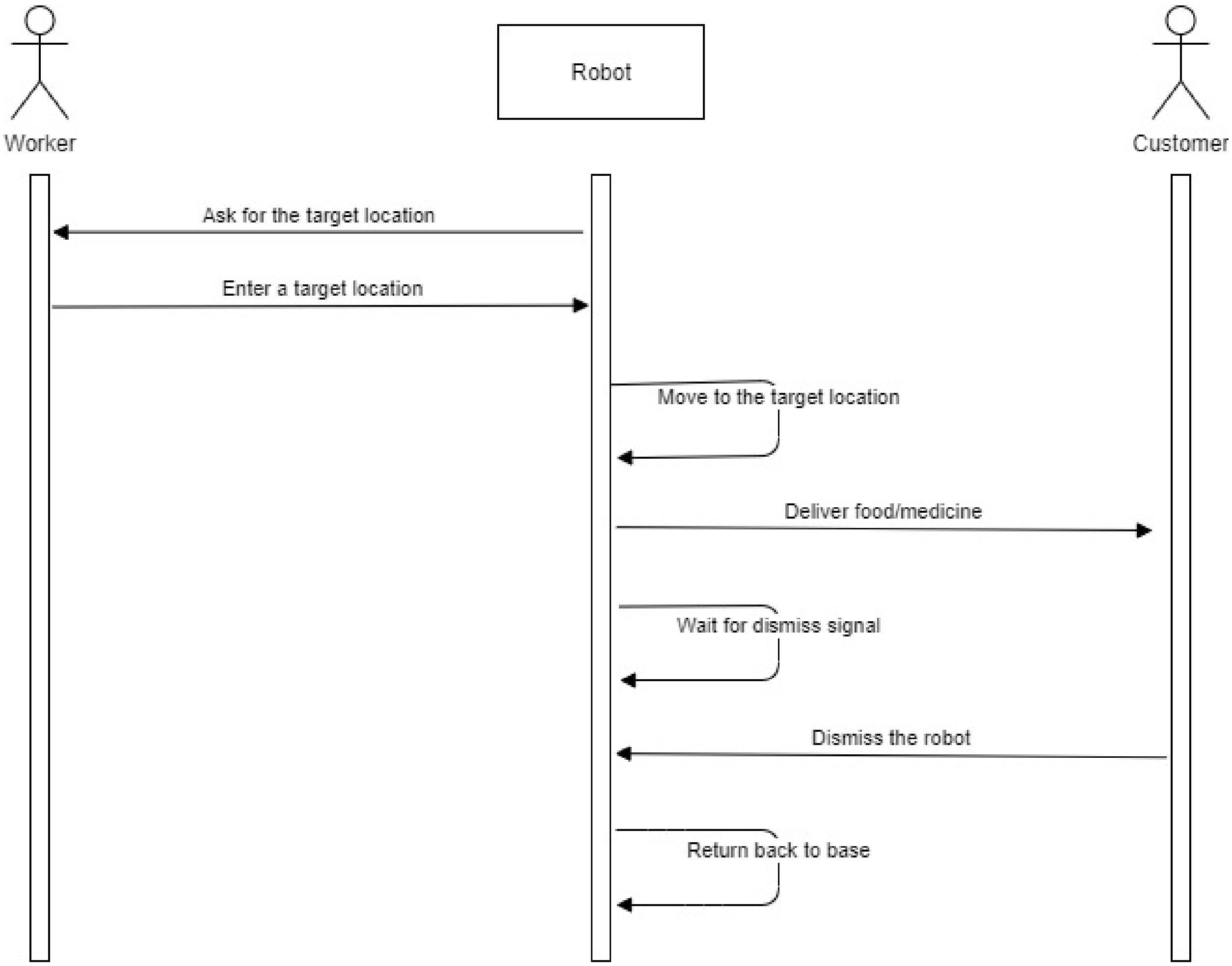


Figure 4-4: System Sequence Diagram

Chapter 5 : Mechanical Design

5.1. Mechanical Design

It is a hexagonal shape structure to provide good distribution for loads and ease to manufacturing.

All parts are cut by laser cutting machine for wood plate with (4.3mm & 3mm) thickness.

It consists of two main assemblies one is base assembly and other is body assembly.

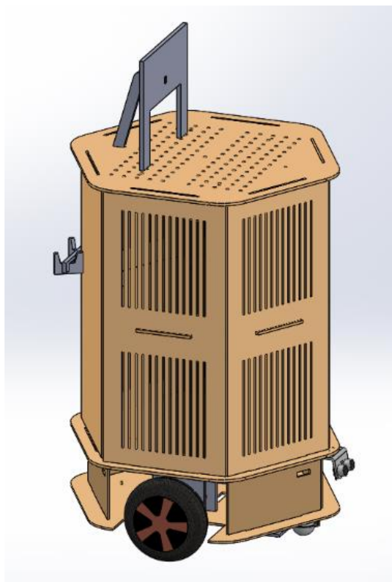


Figure 5-1: Robot Mechanical Design

1. Base assembly

It consists of two horizontal plates held together with four vertical plates and four screws acts as support

Base contains:

- Motors
- Wheels and coaster wheels
- Ultrasonic holder

- RFID module holder

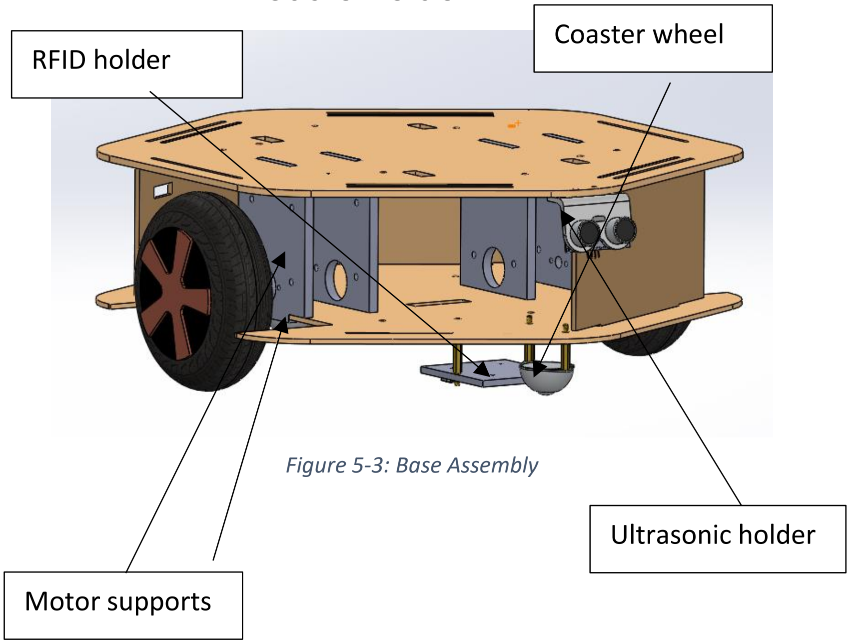


Figure 5-3: Base Assembly

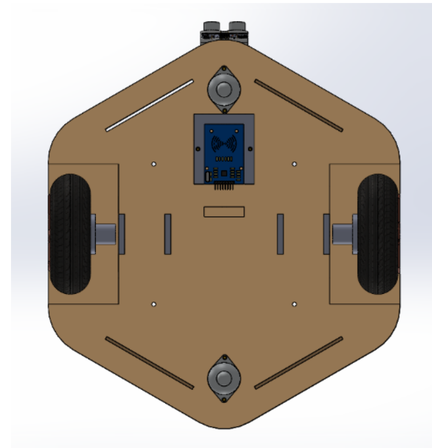


Figure 5-2: Robot Base

2. Body assembly

It consists of three horizontal plates held together by five vertical plates and four long screws acts as a support.

The sixth vertical plate is the door to provide ease of access for the rest of components and electronics inside the structure.

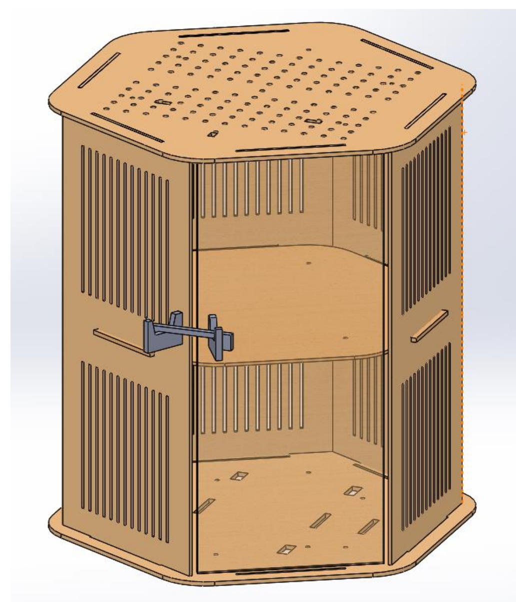


Figure 5-4: Robot Body

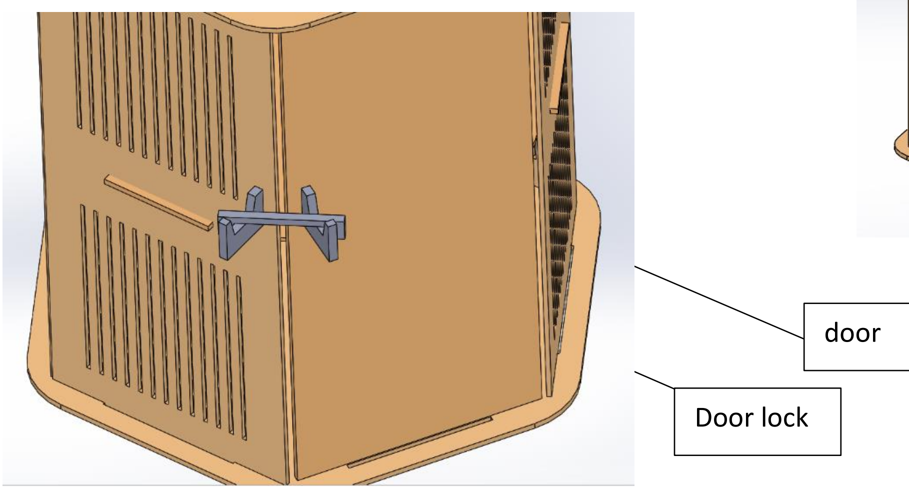


Figure 5-5: Body Assembly

3. Lcd and keypad holder

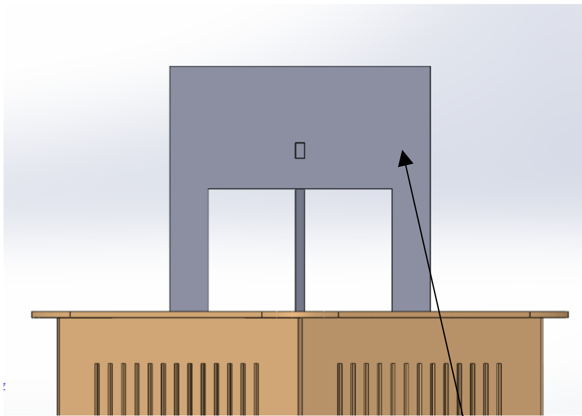


Figure 5-7: Keypad Holder (Front)

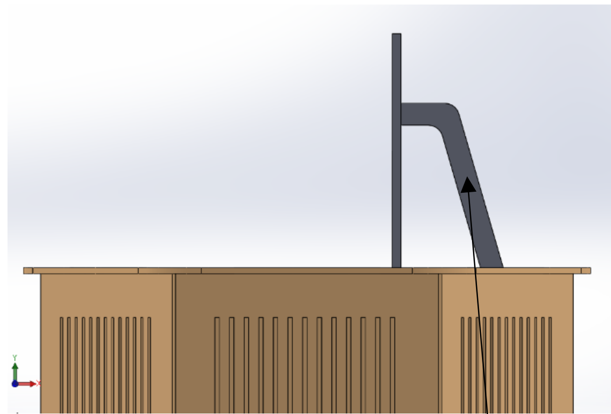


Figure 5-6: Keypad Holder (Side)

- Robot Dimensions:
  - 65 cm height
  - 35 cm width
- Mechanical Components:
  - Wooden plates
  - 2 DC motors
  - 2 Wheels (12 cm diameter)
  - 2 Caster wheels (40 mm)
  - Bolts, nuts, and screws
- Machines needed:
  - Laser cutter
  - Drill

5.2. Hardware Components

5.2.1. Arduino Mega 2560

Figure 5-8: Arduino Mega 2560

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started.

Specifications:

- Microcontroller: ATmega2560
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V
- Input Voltage (limits): 6-20V
- Digital I/O Pins: 54 (of which 15 provide PWM output)
- Analog Input Pins: 16
- DC Current per I/O Pin: 40 mA
- DC Current for 3.3V Pin:50 mA
- Flash Memory: 256 KB of which 8 KB used by bootloader
- SRAM: 8 KB
- EEPROM: 4 KB
- Clock Speed: 16 MHz

Pinout:

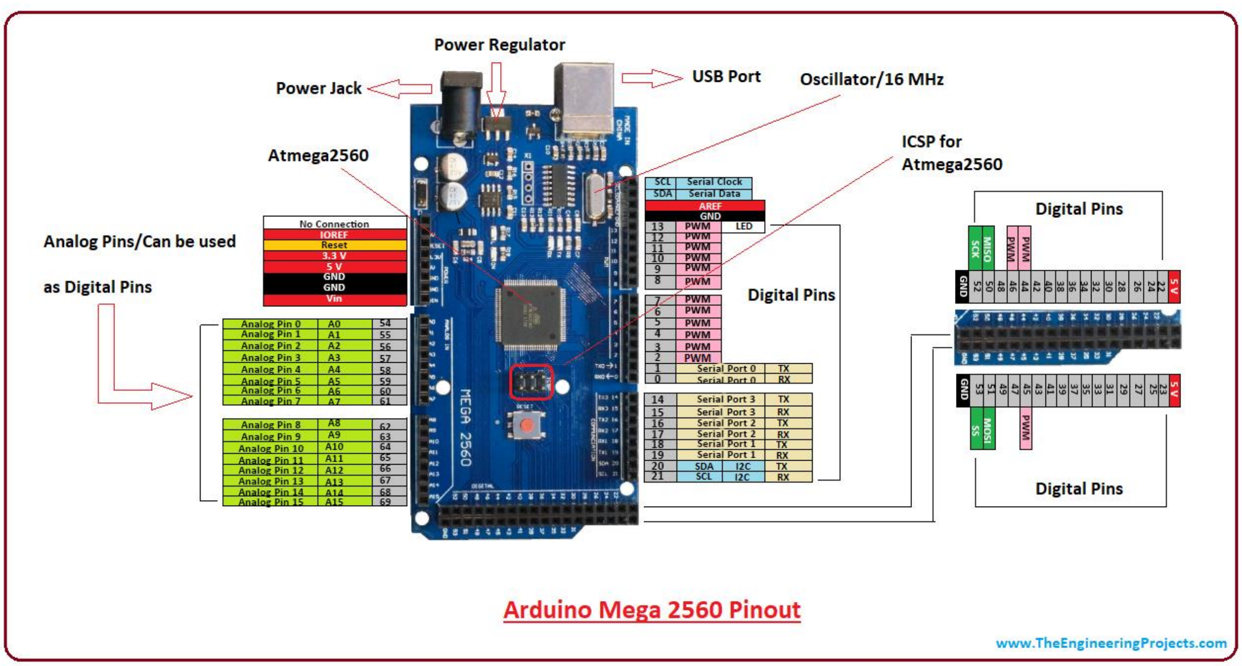


Figure 5-9: Arduino Mega 2560 Pinout

5.2.2. Motor Shield VNH2SP30

Figure 5-10: Motor Driver VNH2SP30

VNH2SP30 is a full bridge motor driver used for a wide range of automotive applications. The device includes a dual monolithic high side driver and two low side switches.

The VIN and motor out are pitched for 5mm screw terminals, making it easy to connect larger gauge wires. INA and INB control the direction of each motor, and the PWM pins turns the motors on or off. For the VNH2SP30, the current sense (CS) pins will output approximately 0.13 volts per amp of output current.

Specifications:

- Voltage Range: 5.5V - 16V
- Maximum Current rating: 30A
- Practical Continuous Current: 14 A
- Current sense output proportional to motor current
- MOSFET on-resistance: 19 mΩ (per leg)

- Maximum PWM frequency: 20 kHz
- Thermal Shutdown
- Undervoltage and Overvoltage shutdown

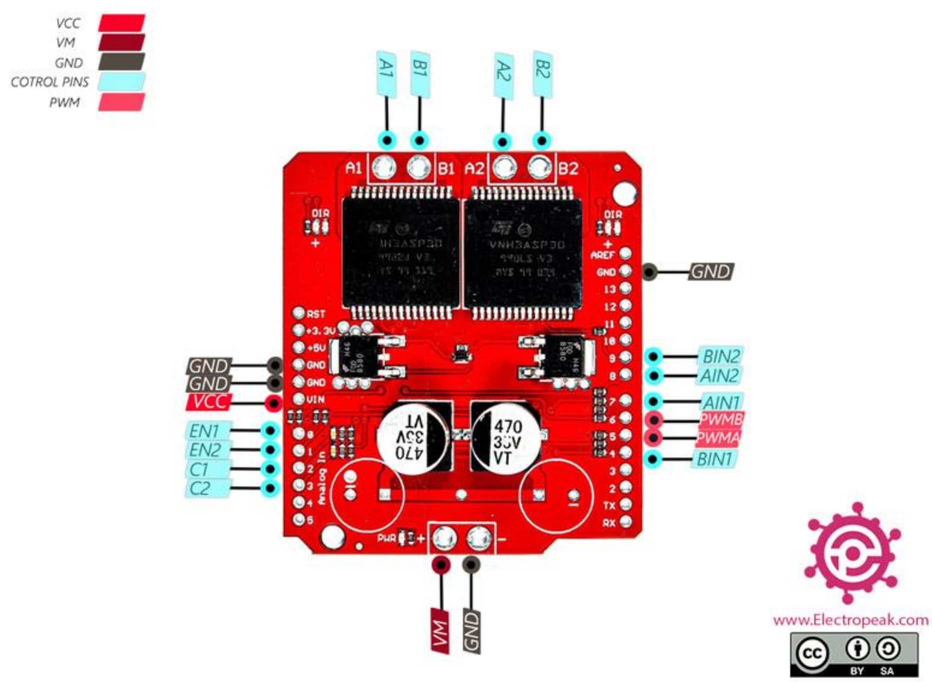


Figure 5-11: Motor Driver VNH2SP30 Pinout

Pinout:

- **VM**: Module voltage
- **VCC**: Module power supply
- **GND**: Ground
- **A1**: Positive end for motor A
- **A2**: Negative end for motor A
- **B1**: Positive end for motor B
- **B2**: Negative end for motor B
- **PWMA:** Speed control signal for motor A
- **PWMB:** Speed control signal for motor B
- **AIN1:** Control signal for motor A

- **AIN2:** Control signal for motor A
- **BIN1:** Control signal for motor B
- **BIN2:** Control signal for motor B
- **EN1:** Activation signal for motor A
- **EN2:** Activation signal for motor B
- **C1:** current measurement signal for motor A
- **C2:** current measurement signal for motor B

5.2.3. Ultrasonic Sensor HC-SR04

Ultrasonic Sensor HC-SR04 is a sensor that can measure distance. It emits an ultrasound at 40 000 Hz (40kHz) which travels through the air and if there is an object or obstacle on its path It will bounce back to the module. Considering the travel time and the speed of the sound you can calculate the distance.

Pinout:

- **Pin 1:** VCC (5V)
- **Pin 2:** Trigger
- **Pin 3:** Echo
- **Pin 4:** Ground

The supply voltage of VCC is +5V and you can attach TRIG and ECHO pin to any Digital I/O in your Arduino Board.

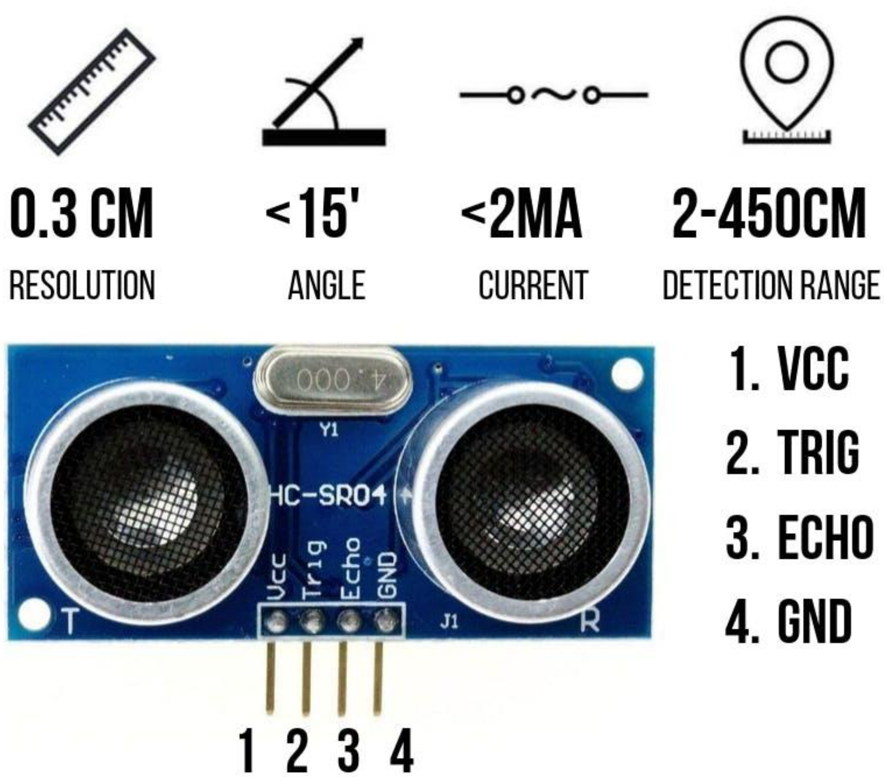


Figure 5-12: Ultrasonic Sensor HC-SR04

To generate an ultrasound wave, we will set the Trigger Pin on a High State for 10 µs. That will get an 8-cycle sonic burst which will travel by the speed of sound, and it will be received in the Echo Pin. The Echo Pin will output the time of the sound wave traveled.so we can get the distance from the speed of sound and the time of the travelled sound.

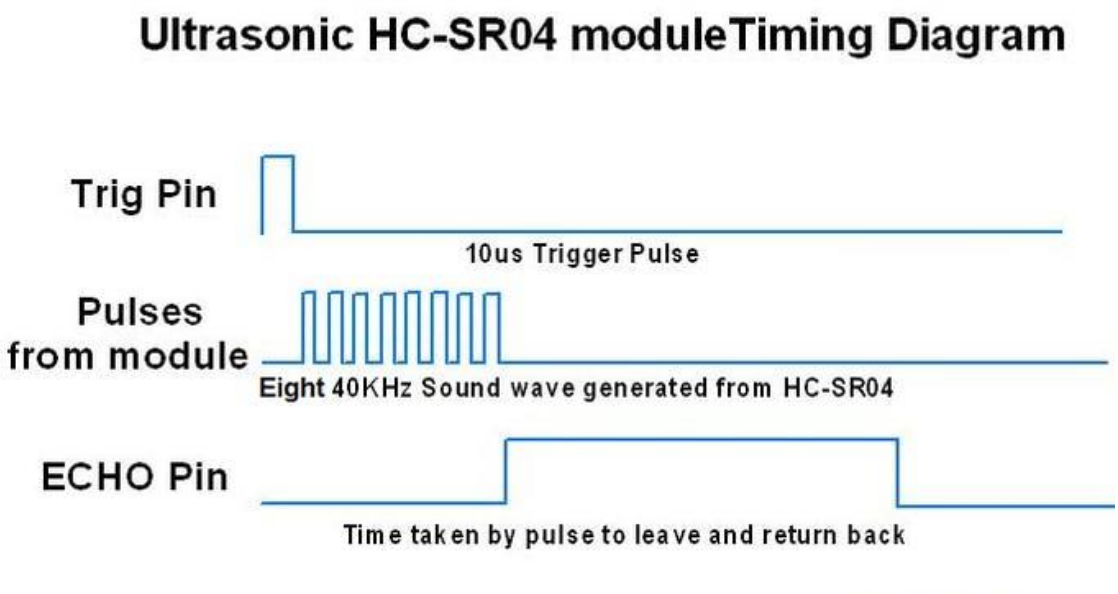


Figure 5-13: Ultrasonic HC-SR04 Timing Diagram

If we have an object 10 cm away from the ultrasonic sensor as shown in the next figure we have the speed of sound which is 0.034 cm/ µs, and the sound will take about 588 µs to travel from echo to the trigger ,now we have speed and time so we can get the distance which is equal to 0.034 * 588 = 20 cm but that distance is the distance of all the travelled from echo to object then back to trigger so that we will divide the distance by 2 to get the distance from echo to object which will be 10 cm.

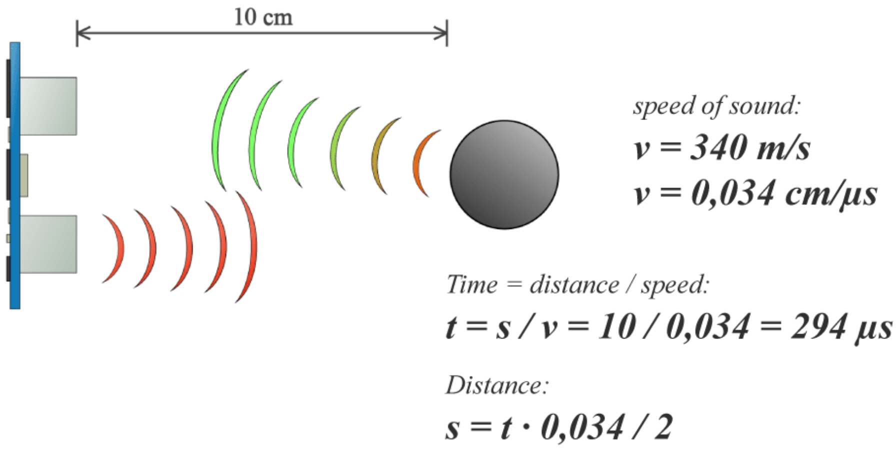


Figure 5-14: Distance Measurements Using HC-SR04

Specifications:

- Input voltage: 5V
- Current draw: 20mA
- Distance range: 2cm to 400cm
- Angle: 15°
- Frequency of ultrasonic waves: 40KHz
- Working temperature: -15°C to 70°C

5.2.4. I2C LCD Screen 16x02

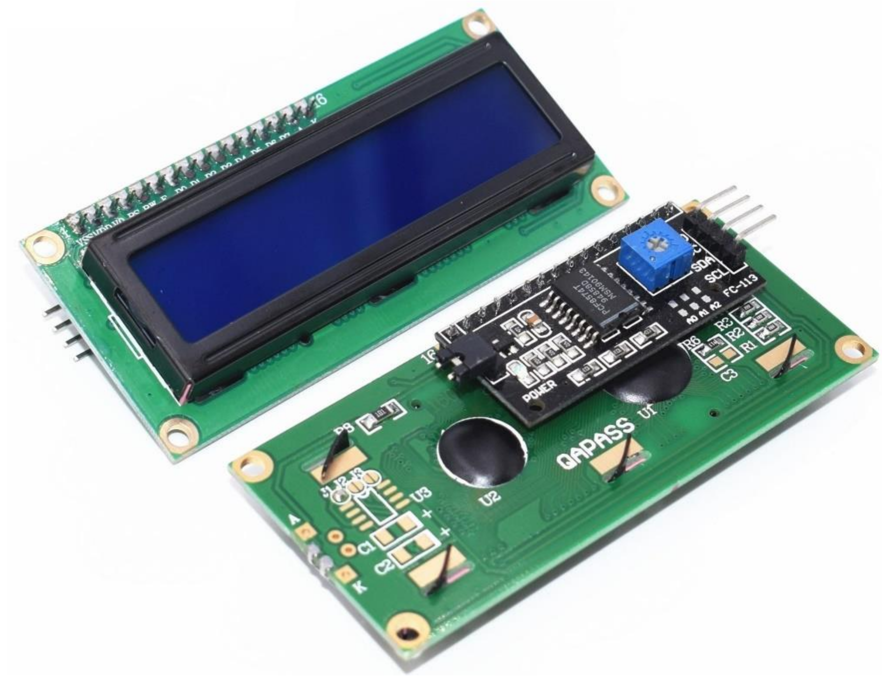


Figure 5-15: LCD Screen 16x02

We will use character display and its size is 16*2. LCD consist of (LCD panel – LCD controller – CGROM – DDRAM- CGRAM).

LCD panel is the place which you can see the character through it, that panel depends on Liquid Crystal which can be changed through the effect of signal which can be controlled by light.

LCD controller: control which place on panel need to be turned on and which one need to be turned off.

DDRAM: it has all the addresses of each place on the panel.

CGROM: it has all the addresses of each character.

Pinout:

- **GND:** Ground
- **VCC:** Input voltage
- **SDA:** Serial lock line
- **SCL:** Serial data line

Specifications:

- LCD operate with voltage between (4.7V – 5.3V)
- Has 2 rows and 16 columns with total of 32 characters
- Operating current is 1mA
- The LCD can write letters and numbers
- It has two modes (4-bit or 8-bit)

5.2.5. Keypad 3x4

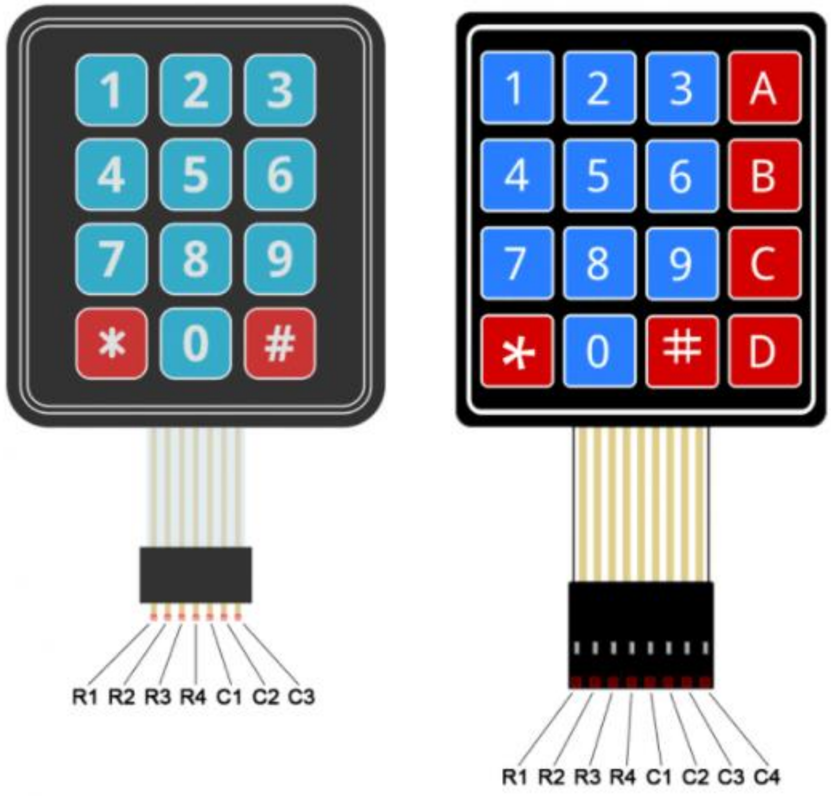


Figure 5-16: Keypad

The keypad is a matrix of buttons, and those buttons are arranged in rows and columns. A 3X4 keypad has 3 columns and 4 rows. A 4X4 keypad has 4 columns and 4 rows.

1. If there are no buttons pressed, all the column pins are HIGH, and all the row pins are LOW:

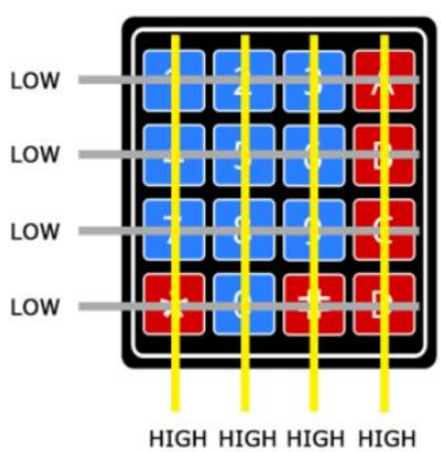


Figure 5-17: Keypad Operation 1

2. If a button is pressed, the column pin will change to LOW:

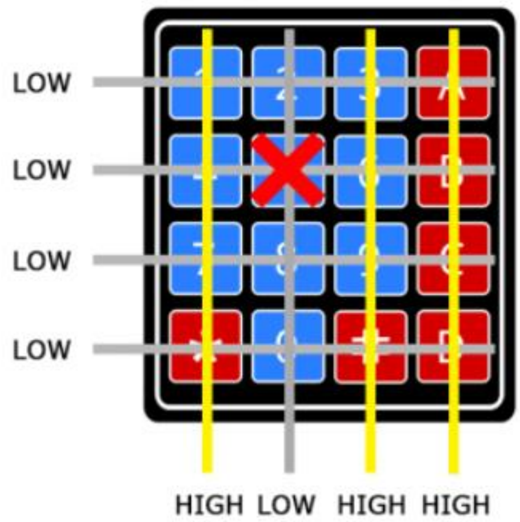


Figure 5-18: Keypad Operation 2

3. Now we know which column the button is in. but we do not know which row the button is in, so we need to find the row of the button. We will do this by changing each one of the row pins from LOW to HIGH, and at the same time reading all the column pins to detect which column pin returns to HIGH:

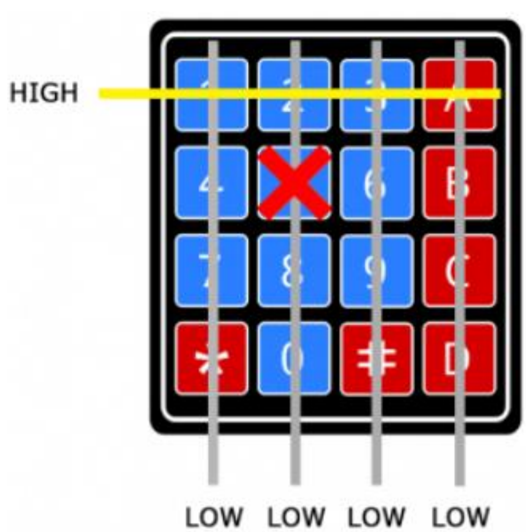


Figure 5-19: Keypad Operation 3

4. When the column pin goes HIGH again, so now we found the row pin that is connected to the button:

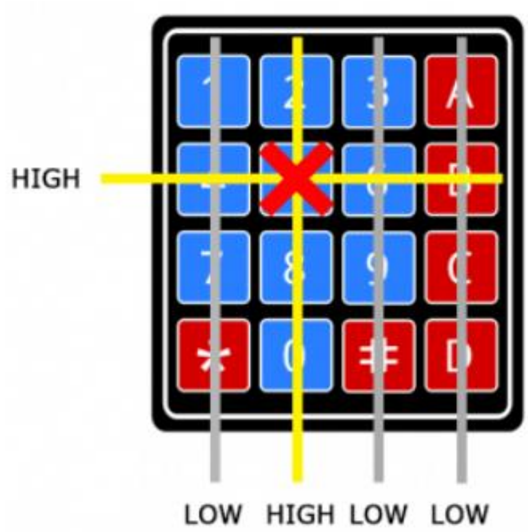


Figure 5-20: Keypad Operation 4

From the diagram above, you can see that the number of the row is 2 and the column is 2 which mean that the number 5 button was pressed.

Specifications:

- Size: 76 x 54 mm
- Cable length: 85 mm

5.2.6. 3-Axis Digital Compass HMC5883L

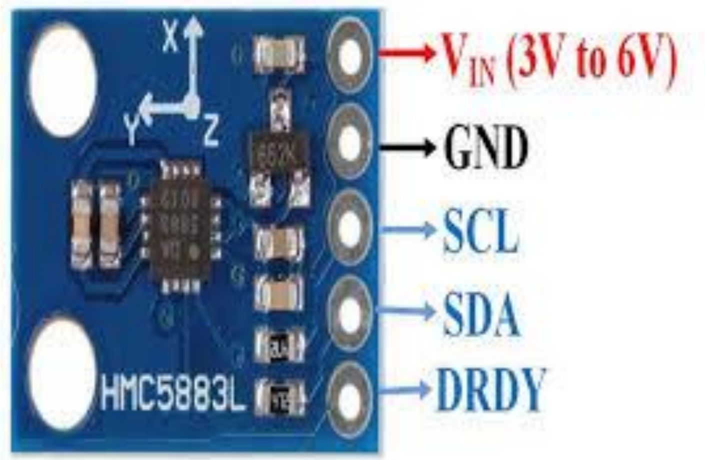


Figure 5-21: Digital Compass HMC5883L

HMC5883L is used as magnetometer module. That module depends on anisotropic magnetoresistance technology. It behaves as a digital compass which help to find the direction and measure the magnitude and direction of the magnetic field along X, Y and Z-axis. The module converts the magnetic field to voltages on the 3-axis pins.

Magnetoresistance technology is the property of material to change the value of its electrical resistance when an external magnetic field is applied to it, that effect was first discovered by William Thompson in 1857.

The robots need to know the direction, So, they need the compass module to know the direction. The compass module will sense the magnetic field and depending on that magnetic field the compass will now the direction.

SCL and SDA are used to send data to the processor depending on I2C communication.

Specifications:

- Input voltage: 3V to 5V
- Supports I2C communication protocol
- Data rate: 160Hz
- Measuring range: ± 1.3-8 Gauss
- Dimensions: 14.8mm x 13.5mm x 3.5mm

5.2.7. Infrared Proximity Sensor

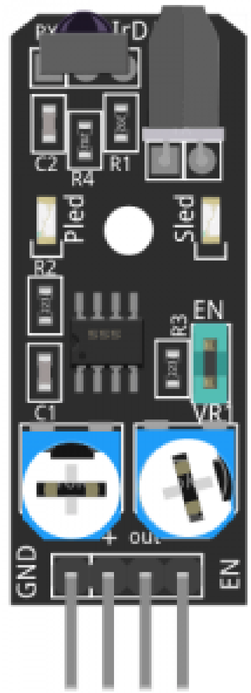


Figure 5-22: IR Proximity Sensor

An infrared (IR) sensor is a device that detects infrared radiation in its surrounding environment. If an object comes close to the IR sensor, the infrared light from the LED reflects to the receiver of IR the same idea as ultrasonic sensor.

The module has an infrared LED as an emitter, and an infrared receiver, the emitter sends infrared lights and if that light hit an object, it will be reflected to the receiver.

Pinout:

- **GND:** Ground
- **+:** Input voltage (VCC)
- **Out:** Detection pin
- **EN:** Enable

Specification:

- Input voltage: 3.3V to 5V
- Operating current: 20mA
- Working temperature: -10°C to 50°C
- Detection distance: 2cm to 40cm
- Effective angle: 35°

5.2.8. RFID Module PN532

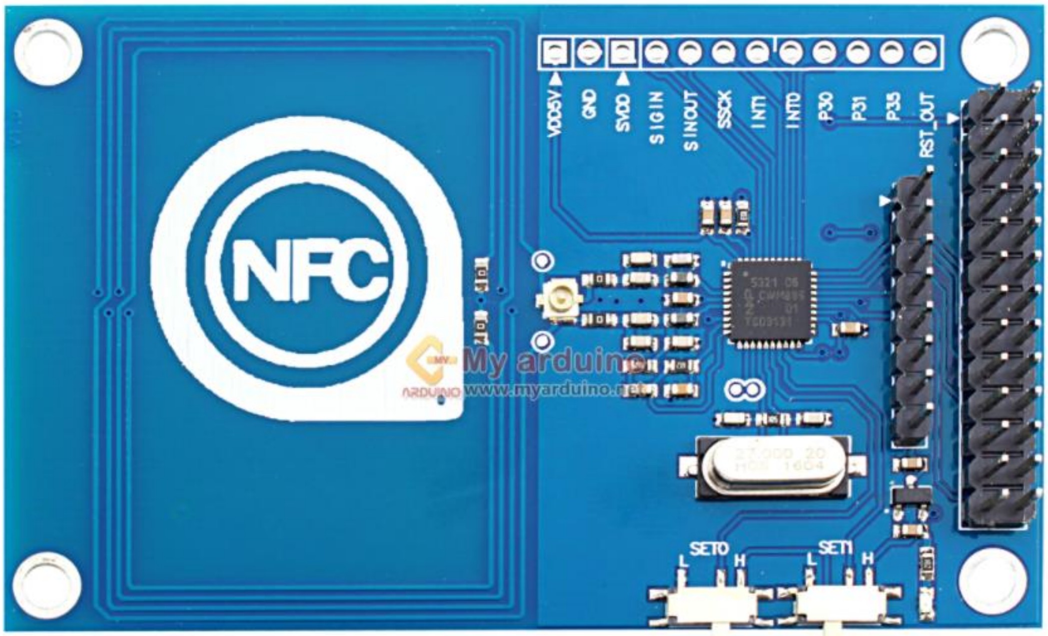


Figure 5-23: RFID Module PN532

PN532 NFC RFID Module is a highly integrated transmission module for Near Field Communication at 13.56MHz. With the mode switch on board, you can change easily between I2C, SPI, and UART modes. In addition, it supports RFID reading and writing, and NFC function.

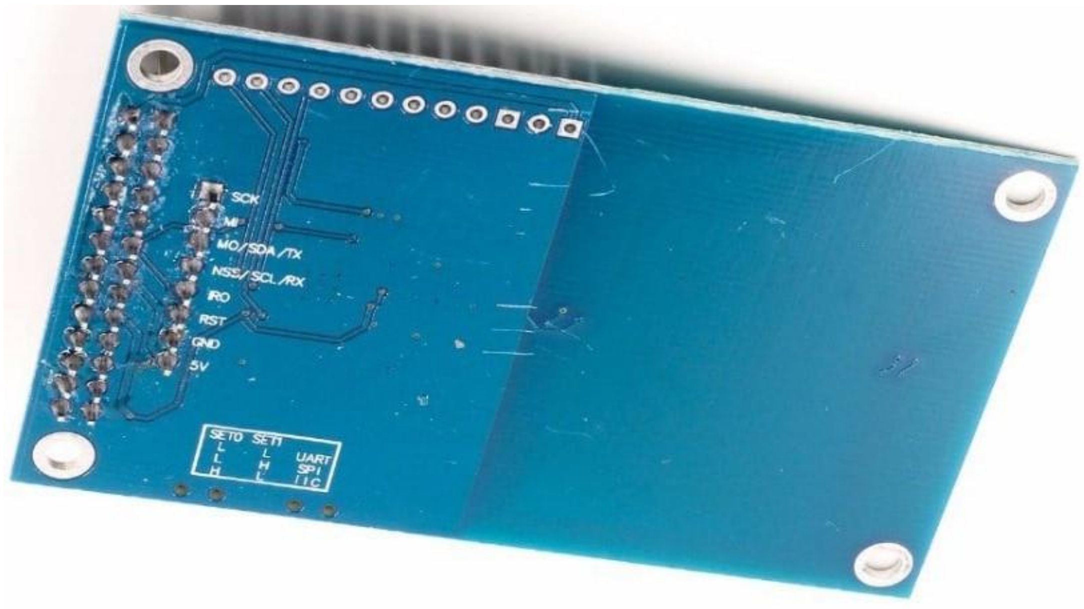


Figure 5-24: RFID Module PN532 Pinout

Pinout:

- **VCC:** Input voltage (3.3V to 5.5V)
- **GND:** Ground
- **RST:** Reset and power-down. When this pin goes low, hard power-down is enabled. This turns off all internal current sinks including the oscillator and the input pins are disconnected from the outside world. On the rising edge, the module is reset
- **IRQ:** Interrupt pin that can alert the microcontroller when RFID tag comes into its vicinity
- **NSS/SCL/RX:** Pin acts as slave select when SPI interface is enabled, acts as serial clock when I2C interface is enabled and acts as serial data input when UART interface is enabled
- **MO/SDA/TX:** Pin acts as Master-Out-Slave-In when SPI interface is enabled, acts as serial data when I2C interface is enabled and acts as serial data output when UART interface is enabled
- **MI:** Master-In-Slave-Out
- **SCK:** Serial clock, accepts clock pulses provided by the SPI bus Master (i.e., Arduino)

Features:

- Effective communication distance of 0 to 1 cm (depend on TAG type)
- Supports switching of SPI, IIC and UART interface.
- Can be used for 13.56M non-contact communication
- Compatible with ISO14443 Type A and Type B standards

Specifications:

- IC NXP PN532 chip
- Operating Voltage: 3.3V
- Power Supply Voltage: 3.3 to 5.5V
- Max Supply Current: 150mA
- Working Current (Standby Mode): 100mA
- Working Current (Write Mode): 120mA
- Working Current (Read Mode): 120mA

5.3. Software Components

5.3.1. SolidWorks

Figure 5-25: SolidWorks

SolidWorks is a solid modeling computer-aided design (CAD) and computer-aided engineering (CAE) computer program.

We used it for the mechanical design of our robot.

5.3.2. Arduino IDE

Figure 5-26: Arduino IDE

The Arduino Integrated Development Environment is an open-source, cross-platform application that is written in functions from C and C++. It is used to write and upload programs to Arduino compatible boards.

We used it for writing and uploading sketches to the Arduino.

5.3.3. Keypad library

we use that library in our code. in that library we have many functions which help us to deal with keypad, but we will talk about some of them which are used in our project.

Function	parameters	return	usage
makeKeymap ()	keys*	void	We use that function to make a map of our keys
Keypad ()	makeKeymap row_pins col_pins rows columns	void	It is a constructor used to prepare the virtual keypad
waitForKey ()	void	char	This function waits until you press on a key then it will return it

Table 5-1: Keypad Driver

5.3.4. LiquidCrystal_I2C library

This library helps us to interface with the LCD screen in our project.

Function	parameters	return	usage
LCD ()	lcd_Addr lcd_cols lcd_rows	void	It is a constructor used to prepare the LCD by giving rows and columns numbers and the LCD I2C address
Clear ()	void	void	Used to clear the display
SetCursor ()	row, column	void	We use that function to choose where to write on the LCD
Print ()	string	void	We use that function to print on the LCD
Backlight ()	void	void	We use that function to the backlight of the screen
init ()	void	void	Used to initialize the LCD screen object

Table 5-2: LCD Screen Driver

5.3.5. QMC5883LCompass library

This library helps us to interface with the compass module.

Function	parameters	return	usage
Read ()	void	void	Start reading data through I2C
getAzimuth ()	void	int	We use that function to get the rotation angel of the robot
init ()	void	void	Used as initialization and prepare I2C communication
setCalibration ()	X-min X-max Y-min Y-max Z-min Z-max	void	We use that function for calibration by setting values for X, Y, and z

Table 5-3: Compass Driver

5.3.6. Adafruit_PN532 library

This library helps us to interface with the RFID module.

Function	parameters	return	usage
readPassiveTargetID ()	cardbaudrate uid& uidLength&	bool	Used to read the ID of a RFID tag
nfc ()	SCK MISO MOSI SS	void	It is a constructor use to set pins of RFID module
begin ()	void	void	It is used to initialize SPI communication
SAMConfig ()	void	bool	Configure SAM module to read RFID tags

Table 5-4: RFID Module Driver

5.3.7. SPI library

This library helps us to use SPI communication.

Function	parameters	return	usage
begin ()	void	void	Used to initialize the SPI communication

Table 5-5: SPI library

5.3.8. Wire library

This library helps us to use SPI communication.

Function	parameters	return	usage
begin ()	void	void	Used to initialize the I2C communication

Table 5-6: Wire library

Chapter 6 : Project Implementation

In this chapter we will discuss the interface of each part with the Arduino, connection, and the output.

6.1. RFID Module PN532

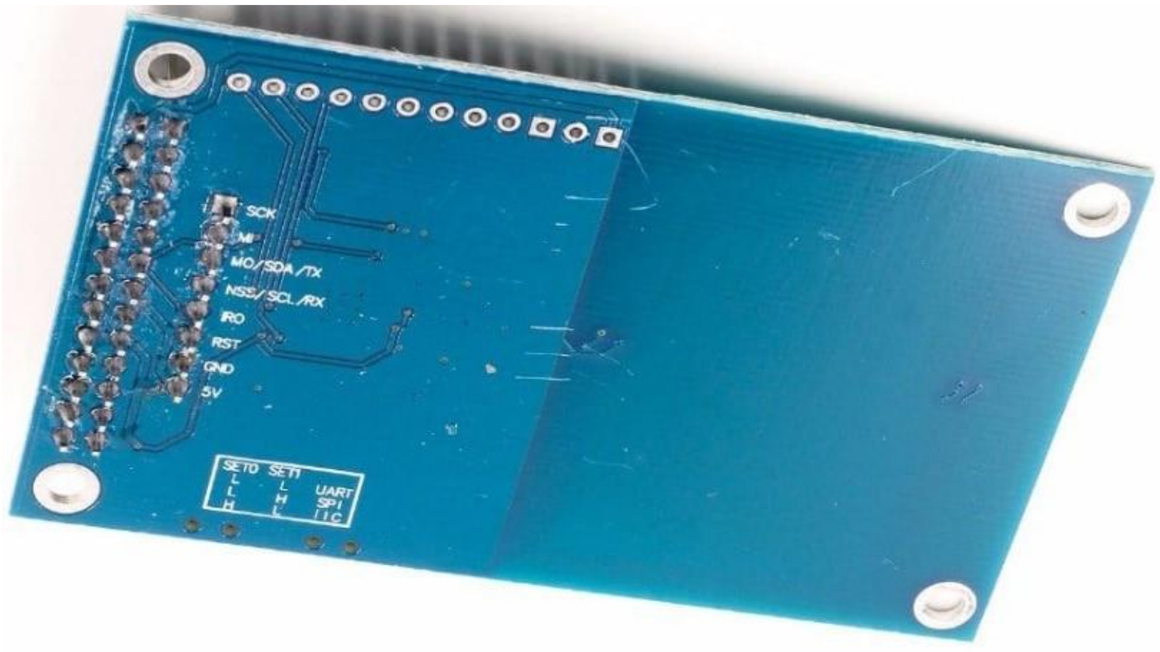


Figure 6-1: PN532 Pinout

Connection to Arduino:

Module	Arduino
SCK	52
MI	50
MO	51
NSS	53
IRQ	2
RST	3
GND	Ground
5V	5v

Table 6-1: RFID Module Pn532 Connection

Also, we need to select SPI as the interface, so on SEL1 place the jumper in the ON position. for SEL0 place the jumper in the OFF position.

Code:

readrfid

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_PN532.h>

// Define the pins for SPI communication.
#define PN532_SCK  (52)
#define PN532_MOSI (51)
#define PN532_SS   (53)
#define PN532_MISO (50)

// Define just the pins connected to the IRQ and reset lines.
#define PN532_IRQ   (2)
#define PN532_RESET (3)  // Not connected by default on the NFC Shield


// Use this line for a breakout with a software SPI connection (recommended):
Adafruit_PN532 nfc(PN532_SCK, PN532_MISO, PN532_MOSI, PN532_SS);

// Use this line for a breakout with a hardware SPI connection.  Note that
// the PN532 SCK, MOSI, and MISO pins need to be connected to the Arduino's
// hardware SPI SCK, MOSI, and MISO pins.
//Adafruit_PN532 nfc(PN532_SS);


void setup(void) {
  Serial.begin(115200);
  while (!Serial) delay(10); // for Leonardo/Micro/Zero
  Serial.println("Hello!");

  nfc.begin();

  uint32_t versiondata = nfc.getFirmwareVersion();
  if (! versiondata) {
    Serial.print("Didn't find PN53x board");
    while (1); // halt
```

Figure 6-2: RFID Module Code 1

readrfid

```
  uint32_t versiondata = nfc.getFirmwareVersion();
  if (! versiondata) {
    Serial.print("Didn't find PN53x board");
    while (1); // halt
  }

  // Got ok data, print it out!
  Serial.print("Found chip PN5"); Serial.println((versiondata>>24) & 0xFF, HEX);
  Serial.print("Firmware ver. "); Serial.print((versiondata>>16) & 0xFF, DEC);
  Serial.print('.'); Serial.println((versiondata>>8) & 0xFF, DEC);

  // Set the max number of retry attempts to read from a card
  // This prevents us from waiting forever for a card, which is
  // the default behaviour of the PN532.
  //nfc.setPassiveActivationRetries(0xFF);

  // configure board to read RFID tags
  nfc.SAMConfig();

  Serial.println("Waiting for an ISO14443A card");
}

void loop(void) {
  uint8_t checksum1;
  boolean success;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the returned UID
  uint8_t uidLength;        // Length of the UID (4 or 7 bytes depending on ISO14443A card type)

  // Wait for an ISO14443A type cards (Mifare, etc.).  When one is found
  // 'uid' will be populated with the UID, and uidLength will indicate
  // if the uid is 4 bytes (Mifare Classic) or 7 bytes (Mifare Ultralight)
  success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid[0], &uidLength);

```

Figure 6-3: RFID Module Code 2

```
void loop(void) {
  uint8_t checksum1;
  boolean success;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the returned UID
  uint8_t uidLength;        // Length of the UID (4 or 7 bytes depending on ISO14443A card type)

  // Wait for an ISO14443A type cards (Mifare, etc.).  When one is found
  // 'uid' will be populated with the UID, and uidLength will indicate
  // if the uid is 4 bytes (Mifare Classic) or 7 bytes (Mifare Ultralight)
  success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid[0], &uidLength);

   if (success) {
    // Display some basic information about the card
    Serial.println("Found an ISO14443A card");
    Serial.print("  UID Length: ");Serial.print(uidLength, DEC);Serial.println(" bytes");
    Serial.print("  UID Value: ");
    nfc.PrintHex(uid, uidLength);
    Serial.println("");
    checksum1 = uid[0] ^ uid[1] ^ uid[2] ^ uid[3];
    Serial.println(checksum1);
  }
}
```

Figure 6-4: RFID Module Code 3

- At first, we include the needed libraries. Adafruit_PN532 is library can read MiFare cards, including ID numbers that are hard-coded, as well as authenticate and read/write EEPROM portions. It can communicate through SPI or I2c. We use SPI in this situation.
- At setup() function we begin serial communication and print [hello] then check the firmware version.
- At loop() function we define array to store the returned id, then check if it succeeded to read RFID card it will print the length and the value of the Id.
- We need to give every RFID card a unique integer id so we can do checksum

Output:

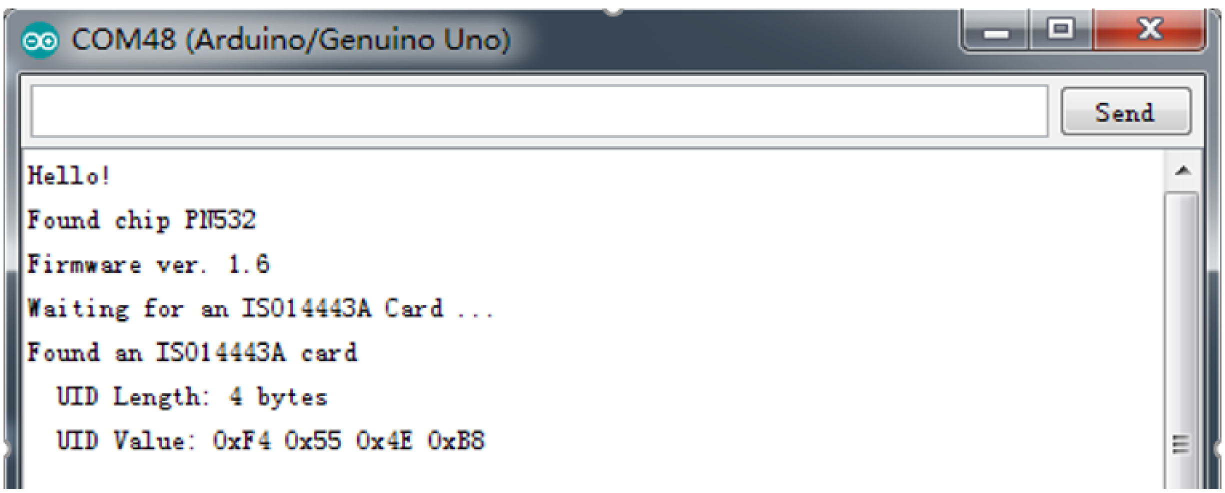


Figure 6-5: RFID Module Output

6.2. Ultrasonic Sensor HC-SR04

Figure 6-6: HC-SR04 Pinout

Connection to Arduino:

Module	Arduino
VCC	5V
Trigger	10
Echo	11
GND	Ground

Table 6-2: Ultrasonic Module Connection

Code:

ultasonice_code

```
const int trigPin = 10;
const int echoPin = 11;
// defines variables
long duration;
int distance;
void setup() {
pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
pinMode(echoPin, INPUT); // Sets the echoPin as an Input
Serial.begin(9600); // Starts the serial communication
}
void loop() {
// Clears the trigPin
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
// Reads the echoPin, returns the sound wave travel time in microseconds
duration = pulseIn(echoPin, HIGH);
// Calculating the distance
distance= duration*0.034/2;
// Prints the distance on the Serial Monitor
Serial.print("Distance: ");
Serial.println(distance);
}
```

Figure 6-7: Ultrasonic Module Code

- At first, we define the trigger pin and the echo pin to pins 10 and 11 in Arduino, and define two variables (duration) of type long and the second is (distance) of type integer.
- At setup() function we make the trig pin output and the echo pin input and start the serial communication.
- At loop() function we should make sure that the trig pin is clear, so we make its state low for 2 microseconds, then we set the state of trig pin to high to generate the ultrasound wave.
- We use the pulsein() function to return the time and put it on the duration variable and this function should take 2 parameters the first is name of the echo pin and the second should be low or high, in this example we use the second variable high this is meaning that the function should wait until the pin become high to start timing then wait until the pin become low and then end timing and calculate the total time, and then calculate the distance.

Output:

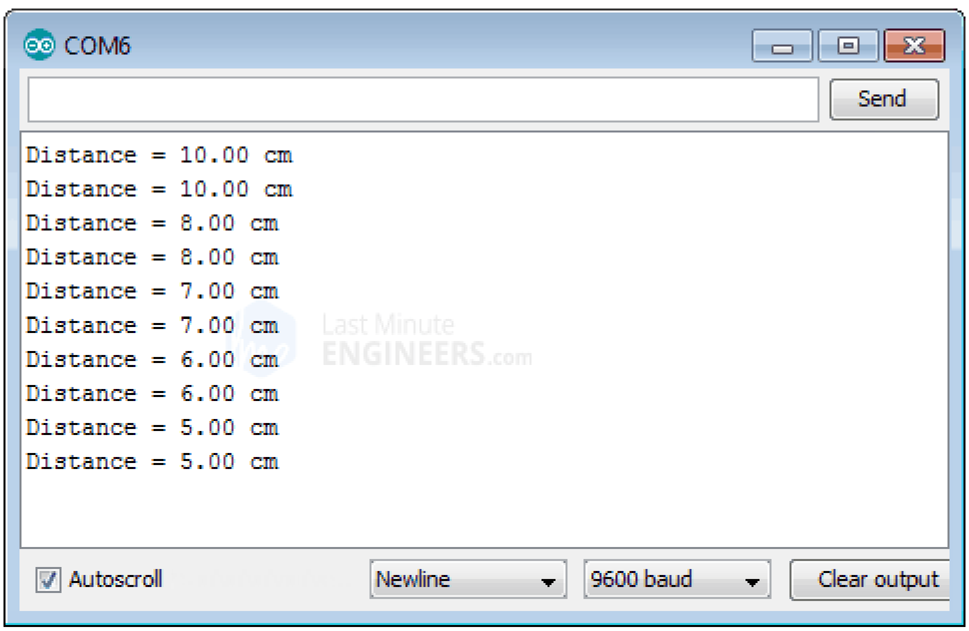


Figure 6-8: Ultrasonic Module Output

6.3. Infrared Proximity Sensor

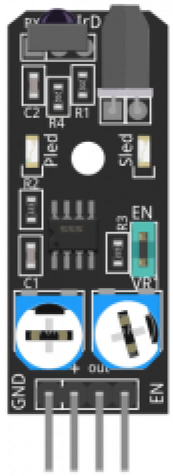


Figure 6-9: IR Module Pinout

Connection to Arduino:

Module	Arduino
GND	ground
+	5V
out	12
EN	None

Table 6-3: IR Module Connection

Code:

Proximity

```
int detect_pin = 12;
int val;
void setup() {
  // put your setup code here, to run once:
  pinMode(detect_pin, INPUT);
  Serial.begin(9600);

}

void loop() {
  // put your main code here, to run repeatedly:
  val = digitalRead(detect_pin);
  if (val == LOW)
  Serial.println("object detected");
  else
  Serial.println("NO object");
}
```

Figure 6-10: IR Module Code

- At first, we define variable of type integer to store the value of the detection pin number, and another variable val.
- We do not need the enable pin as the sensor is always on.
- At setup() function we make the detect_pin input and start the serial communication.
- At loop() function we read the value of the detect_ pin and store the value in val, if the value of val is low this is meaning that the sensor detect object and if it is high this is meaning there is no object.

Output:

Figure 6-11: IR Module Output

6.4. I2C LCD Screen

Figure 6-12: LCD Screen Pinout

Connection to Arduino:

Module	Arduino
GND	Ground
VCC	5V
SDA	20
SCL	21

Table 6-4: LCD Screen Connection

Code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup(){
  lcd.backlight();
  lcd.init();
  lcd.clear();
  lcd.setCursor(0, 0);
}

void loop(){
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Hello World");
  }
}
```

Figure 6-13: LCD Screen Code

- At first, we include the needed libraries, then initialize lcd object by providing the I2C address and the size of the screen.
- At setup() function we initiate the lcd, turn on the backlight, clear the screen, then set the cursor to the first place.
- At loop() function we can print messages on the screen

Output:

Figure 6-14: LCD Screen Output

6.5. Keypad 3x4

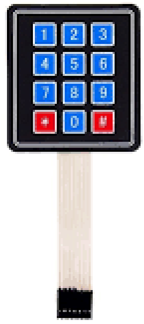


Figure 6-15: Keypad Pinout

Connection to Arduino:

Module	Arduino
R1	9
R2	8
R3	7
R4	6
C1	5
C2	4
C3	3

Table 6-5: Keypad Connection

Code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Keypad.h>

const byte rows = 4;
const byte cols = 3;

char keys[rows][cols] = {
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};

byte row_pins[rows] = {9, 8, 7, 6};
byte col_pins[cols] = {5, 4, 3};

Keypad my_keypad = Keypad(makeKeymap(keys), row_pins, col_pins, rows, cols);

LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup(){
  lcd.backlight();
  lcd.init();
  lcd.clear();
  lcd.setCursor(0, 0);
}

void loop(){
  char input_key = my_keypad.getKey();
  if (input_key){
    //lcd.clear();
    //lcd.setCursor(0, 0);
    lcd.print(input_key);
  }
}
```

Figure 6-16: Keypad Code

- In this code we use lcd I2C to print the number we enter.
- At first, we include the needed libraries [wire - LiquidCrystal_I2C - Keypad], keypad library is an Arduino library for utilizing matrix style keypads.
- We define 2 variables rows and columns to store the number of rows and columns, then we define two arrays row_pin and col_pin to store the pins numbers which connected to the Arduino.
- At setup() function we initiate the lcd, turn on the backlight, clear the screen, then set the cursor to the first place.
- At loop() function we receive the entered key and print it on the LCD screen.

Output:

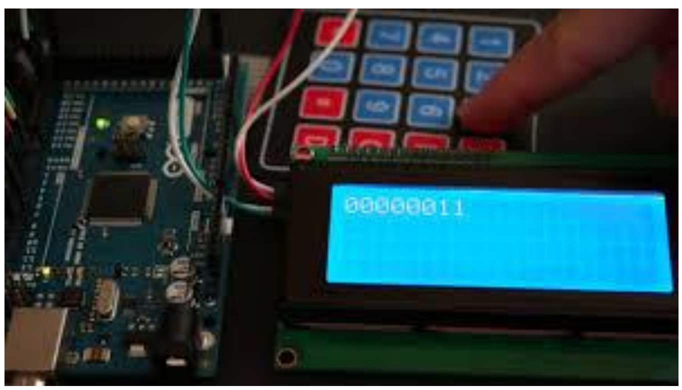


Figure 6-17: LCD Screen Output

6.6. Motor Driver VNH2SP30

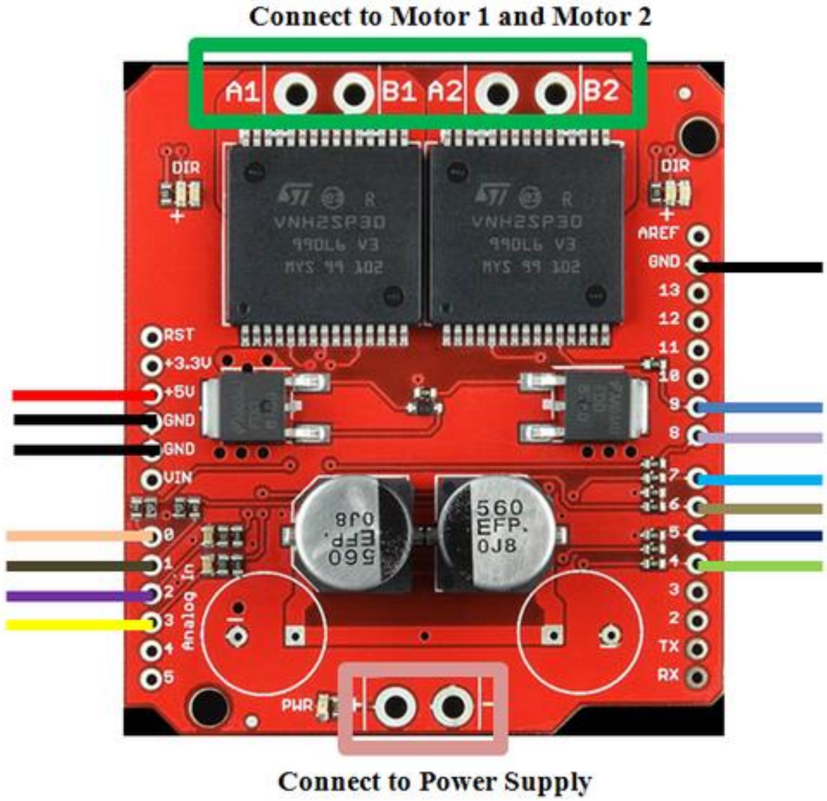


Figure 6-18: Motor Driver Pinout

Connection to Arduino:

Module	Arduino
Ground	Ground
VCC	VCC
0 (enable1)	A0
1 (enable2)	A1
2 (current sensor1)	A2
3 (current sensor 2)	A3
4 (clockwise 2)	D4
5 (PWM 1)	D5
6 (PWM 2)	D6
7 (clockwise 1)	D7
8 (counterclockwise 1)	D8
9 (counterclockwise 2)	D9

Table 6-6: Motor Driver Connection

Code:

```
#define BRAKE 0
#define CW    1
#define CCW   2
#define CS_THRESHOLD 15   // Definition of safety current (Check: "1.3 Monster Shield Example").

//MOTOR 1
#define MOTOR_A1_PIN 7
#define MOTOR_B1_PIN 8

//MOTOR 2
#define MOTOR_A2_PIN 4
#define MOTOR_B2_PIN 9

#define PWM_MOTOR_1 5
#define PWM_MOTOR_2 6

#define CURRENT_SEN_1 A2
#define CURRENT_SEN_2 A3

#define EN_PIN_1 A0
#define EN_PIN_2 A1

#define MOTOR_1 0
#define MOTOR_2 1

short usSpeed = 150;  //default motor speed
unsigned short usMotor_Status = BRAKE;
```

Figure 6-19: Motor Driver Code 1

```
void setup()
{
  pinMode(MOTOR_A1_PIN, OUTPUT);
  pinMode(MOTOR_B1_PIN, OUTPUT);

  pinMode(MOTOR_A2_PIN, OUTPUT);
  pinMode(MOTOR_B2_PIN, OUTPUT);

  pinMode(PWM_MOTOR_1, OUTPUT);
  pinMode(PWM_MOTOR_2, OUTPUT);

  pinMode(CURRENT_SEN_1, OUTPUT);
  pinMode(CURRENT_SEN_2, OUTPUT);

  pinMode(EN_PIN_1, OUTPUT);
  pinMode(EN_PIN_2, OUTPUT);

  Serial.begin(9600);              // Initiates the serial to do the monitoring
  Serial.println("Begin motor control");
  Serial.println(); //Print function list for user selection
  Serial.println("Enter number for control option:");
  Serial.println("1. STOP");
  Serial.println("2. FORWARD");
  Serial.println("3. REVERSE");
  Serial.println("4. READ CURRENT");
  Serial.println("+. INCREASE SPEED");
  Serial.println("-. DECREASE SPEED");
  Serial.println();

}
```

Figure 6-20: Motor Driver Code 2

```
void loop()
{
  char user_input;



  while(Serial.available())
  {
    user_input = Serial.read(); //Read user input and trigger appropriate function
    digitalWrite(EN_PIN_1, HIGH);
    digitalWrite(EN_PIN_2, HIGH);

    if (user_input =='1')
    {
       Stop();
    }
    else if(user_input =='2')
    {
      Forward();
    }
    else if(user_input =='3')
    {
      Reverse();
    }
    else if(user_input =='+')
    {
      IncreaseSpeed();
    }
    else if(user_input =='-')
    {
      DecreaseSpeed();
    }
    else
    {
      Serial.println("Invalid option entered.");
    }

  }
```

Figure 6-21: Motor Driver Code 3

```
void Stop()
{
  Serial.println("Stop");
  usMotor_Status = BRAKE;
  motorGo(MOTOR_1, usMotor_Status, 0);
  motorGo(MOTOR_2, usMotor_Status, 0);
}

void Forward()
{
  Serial.println("Forward");
  usMotor_Status = CW;
  motorGo(MOTOR_1, usMotor_Status, usSpeed);
  motorGo(MOTOR_2, usMotor_Status, usSpeed);
}

void Reverse()
{
  Serial.println("Reverse");
  usMotor_Status = CCW;
  motorGo(MOTOR_1, usMotor_Status, usSpeed);
  motorGo(MOTOR_2, usMotor_Status, usSpeed);
}

void IncreaseSpeed()
{
  usSpeed = usSpeed + 10;
  if(usSpeed > 255)
  {
    usSpeed = 255;
  }

  Serial.print("Speed +: ");
  Serial.println(usSpeed);

  motorGo(MOTOR_1, usMotor_Status, usSpeed);
  motorGo(MOTOR_2, usMotor_Status, usSpeed);
}
```

Figure 6-22: Motor Driver Code 4

```
void DecreaseSpeed()
{
  usSpeed = usSpeed - 10;
  if(usSpeed < 0)
  {
    usSpeed = 0;
  }

  Serial.print("Speed -: ");
  Serial.println(usSpeed);

  motorGo(MOTOR_1, usMotor_Status, usSpeed);
  motorGo(MOTOR_2, usMotor_Status, usSpeed);
}

void motorGo(uint8_t motor, uint8_t direct, uint8_t pwm)         //Function that controls the variables: motor(0 ou 1), direction (cw ou ccw) e pwm (entra 0 e 25
{
  if(motor == MOTOR_1)
  {
    if(direct == CW)
    {
      digitalWrite(MOTOR_A1_PIN, LOW);
      digitalWrite(MOTOR_B1_PIN, HIGH);
    }
    else if(direct == CCW)
    {
      digitalWrite(MOTOR_A1_PIN, HIGH);
      digitalWrite(MOTOR_B1_PIN, LOW);
    }
    else
    {
      digitalWrite(MOTOR_A1_PIN, LOW);
      digitalWrite(MOTOR_B1_PIN, LOW);
    }

    analogWrite(PWM_MOTOR_1, pwm);
  }
```

Figure 6-23: Motor Driver Code 5

```
  else if(motor == MOTOR_2)
  {
    if(direct == CW)
    {
      digitalWrite(MOTOR_A2_PIN, LOW);
      digitalWrite(MOTOR_B2_PIN, HIGH);
    }
    else if(direct == CCW)
    {
      digitalWrite(MOTOR_A2_PIN, HIGH);
      digitalWrite(MOTOR_B2_PIN, LOW);
    }
    else
    {
      digitalWrite(MOTOR_A2_PIN, LOW);
      digitalWrite(MOTOR_B2_PIN, LOW);
    }

    analogWrite(PWM_MOTOR_2, pwm);
  }
}
```

xt file   length : 3,965   lines : 199   Ln : 1   Col : 1   Sel : 0 | 0   Unix (LF)   UTF-8   INS

Figure 6-24: Motor Driver Code 6

- At first, we define the pins connected to the Arduino.
- At setup() function we configure the state of the pins, and print some messages to the user to tell him the character that the user should enter to control the motors.
- At loop() function we check the user input and then call function which corresponds to the user input.
- Stop(): Function to stop the motors
- Forward(): Function to start the rotation of the motors clockwise
- Reverse(): Function to start the rotation of the motors counterclockwise
- Increasespeed(): Function to increase the speed of the motors
- Decreasespeed(): Function to increase the speed of the motors
- MotorGO(): Takes motor number and motor status and speed as input parameters to enable the specified motor

Output:

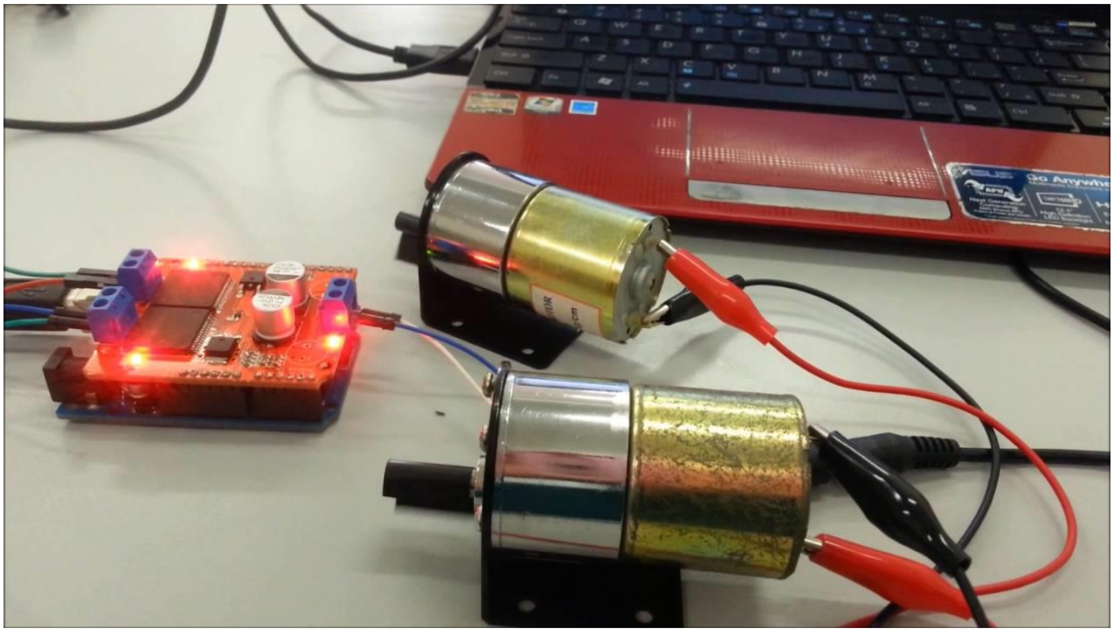


Figure 6-25: Motor Driver Output

6.7. Compass Module HMC5883L

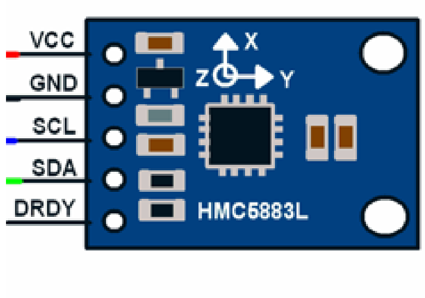


Figure 6-26: Compass Module Pinout

Connection to Arduino:

Module	Arduino
VCC	5V
GND	Ground
SCL	21
SDA	20
DRDY	None

Table 6-7: Compass Module Connection

Code:

Compass §

```
#include <QMC5883LCompass.h>

QMC5883LCompass compass;

void setup() {
  Serial.begin(9600);
  compass.init();
  //compass.setCalibration(-1175, 1537, -707, 2022, -1138, 808);

}

void loop() {
  int a;

  // Read compass values
  compass.read();

  // Return Azimuth reading
  a = compass.getAzimuth();

  Serial.print("A: ");
  Serial.print(a);
  Serial.println();

  delay(250);
}
```

Figure 6-27: Compass Module Code

- At first, we include the needed libraries, and create a compass object
- At setup() function we initialize the compass module, initiate I2C communication, and start the serial communication.
- At loop() function we read the output of the compass (X, Y, Z), calculate the azimuth angle, and print it.

Output:

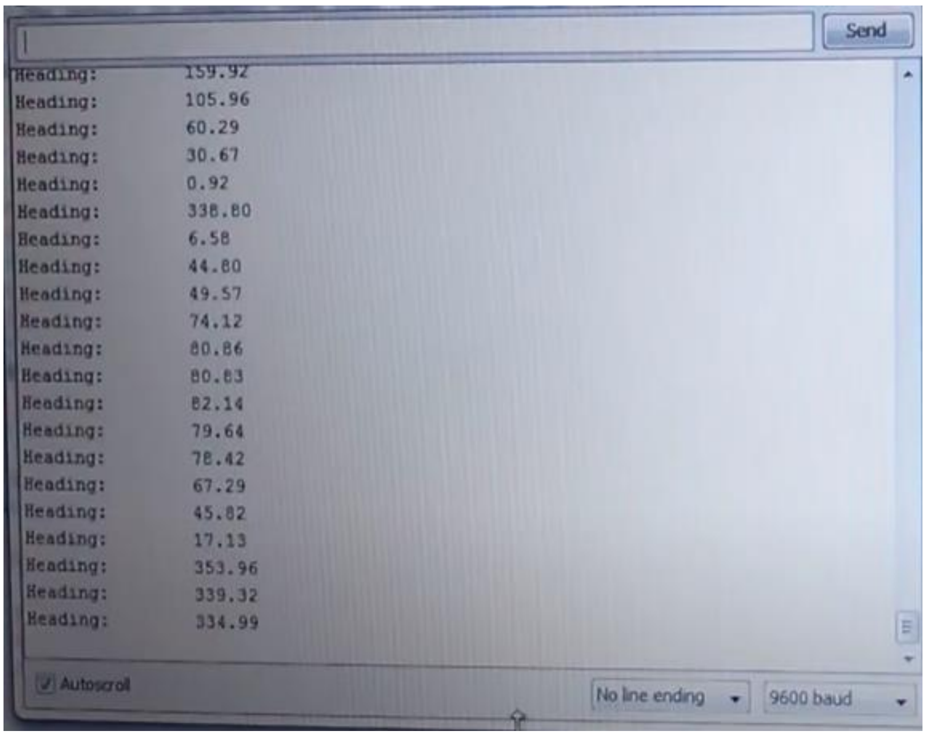


Figure 6-28: Compass Module Output

6.8. Project Assembly

Figure 6-29: The Robot

Code:

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_PN532.h>
#include <liquidCrystal_I2C.h>
#include <keypad.h>
#include <QMC5883LCompass.h>


// RFID
// If using the breakout with SPI, define the pins for SPI communication.
#define PN532_SCK  (52)
#define PN532_MOSI (51)
#define PN532_SS   (53)
#define PN532_MISO (50)

// If using the breakout or shield with I2C, define just the pins connected
// to the IRQ and reset lines.  Use the values below (2, 3) for the shield!
#define PN532_IRQ   (2)
#define PN532_RESET (3)  // Not connected by default on the NFC Shield

// Uncomment just _one_ line below depending on how your breakout or shield
// is connected to the Arduino:

// Use this line for a breakout with a software SPI connection (recommended):
Adafruit_PN532 nfc(PN532_SCK, PN532_MISO, PN532_MOSI, PN532_SS);


// UltraSonic
const int trigPin = 10;
const int echoPin = 11;
// defines variables
long duration;
int distance;


// Motors
#define BRAKE 0
#define CW    1
```

Figure 6-30: Final Project Code 1

```
#define CCW   2
#define CS_THRESHOLD 15   // Definition of safety current (Check: "1.3 Monster Shield Example").

//MOTOR 1
#define MOTOR_A1_PIN 7
#define MOTOR_B1_PIN 8

//MOTOR 2
#define MOTOR_A2_PIN 4
#define MOTOR_B2_PIN 9

#define PWM_MOTOR_1 5
#define PWM_MOTOR_2 6

#define CURRENT_SEN_1 A2
#define CURRENT_SEN_2 A3

#define EN_PIN_1 A0
#define EN_PIN_2 A1

#define MOTOR_1 0
#define MOTOR_2 1

short usSpeed = 150;  //default motor speed
unsigned short usMotor_Status = BRAKE;


// Proximity
int detect_pin = 12 ;
int val;


// Keypad
const byte keypad_rows = 4;
const byte keypad_cols = 3;

char keys[keypad_rows][keypad_cols] = {
  {'1', '2', '3'},
```

Figure 6-31: Final Project Code 2

```
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};

byte row_pins[keypad_rows] = {24, 26, 28, 30};
byte col_pins[keypad_cols] = {32, 34, 36};

Keypad my_keypad = Keypad(makeKeymap(keys), row_pins, col_pins, keypad_rows, keypad_cols);
char key;


// LCD
LiquidCrystal_I2C lcd(0x27, 16, 2);


// Compass
QMC5883LCompass compass;


// Algorithm
int row, col;
#define rows 3
#define cols 4


int origin_loc = 156;
int origin_row = 2;
int origin_col = 0;

int current_loc = origin_loc;
int current_row = origin_row;
int current_col = origin_row;

int target_loc = 0;
int target_row = 99;
int target_col = 99;

```

Figure 6-32: Final Project Code 3

```
int heading = 0; // The heading of the robot in degrees




// Grid 3x4 (0,0 to 2,3)
int grid[rows][cols] = {

  {134,151,63,5},

  {237,77,214,225},

  {156,131,31,248}};


void find_loc(int loc){
  int a,b;
  for (a=0; a<3; a++){
    for (b=0; b<4; b++){
      if (loc == grid[a][b]){
        current_row = a;
        current_col = b;}
    }}
}


// Move Rows
void move_rows() {
  digitalWrite(EN_PIN_1, HIGH);
  digitalWrite(EN_PIN_2, HIGH);

  if (target_row > current_row) {
    // look down using compass
    while (60>heading || heading>80){
        TurnRight();
        compass.read();
        heading = compass.getAzimuth();
        }
```

Figure 6-33: Final Project Code 4

```
    // move forward
    Forward();
    // stop when sensing rfid tag
    uint8_t checksum1;
    boolean success;
    uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the returned UID
    uint8_t uidLength;        // Length of the UID (4 or 7 bytes depending on ISO14443A card type)

    // Wait for an ISO14443A type cards (Mifare, etc.).  When one is found
    // 'uid' will be populated with the UID, and uidLength will indicate
    //success = nfc.startPassiveTargetIDDetection(PN532_MIFARE_ISO14443A); //106000
    success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid[0], &uidLength);

   if (success) {
    Stop();
    checksum1 = uid[0] ^ uid[1] ^ uid[2] ^ uid[3];
   }

   current_loc = checksum1;
   find_loc(current_loc);
      }

  if (target_row < current_row) {

    // look up
    while (60>heading || heading>80){
        TurnRight();
        compass.read();
        heading = compass.getAzimuth();
        }

    // move forward
    Forward();

    // stop when sensing rfid tag
    uint8_t checksum1;
    boolean success;
    uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the returned UID
```

Figure 6-34:: Final Project Code 5

```
    uint8_t uidLength;        // Length of the UID (4 or 7 bytes depending on ISO14443A card type)

    // Wait for an ISO14443A type cards (Mifare, etc.).  When one is found
    // 'uid' will be populated with the UID, and uidLength will indicate
    //success = nfc.startPassiveTargetIDDetection(PN532_MIFARE_ISO14443A); //106000
    success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid[0], &uidLength);

   if (success) {
    Stop();
    checksum1 = uid[0] ^ uid[1] ^ uid[2] ^ uid[3];
   }

   current_loc = checksum1;
   find_loc(current_loc);

  }

}


// Move Columns
void move_cols() {
  digitalWrite(EN_PIN_1, HIGH);
  digitalWrite(EN_PIN_2, HIGH);

  if (target_col > current_col) {
    // look right using compass
    while (60>heading || heading>80){
        TurnRight();
        compass.read();
        heading = compass.getAzimuth();
        }
    // move forward
    Forward();
    // stop when sensing rfid tag
    uint8_t checksum1;
    boolean success;
    uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the returned UID
```

Figure 6-35:: Final Project Code 6

```
    uint8_t uidLength;        // Length of the UID (4 or 7 bytes depending on ISO14443A card type)

    // Wait for an ISO14443A type cards (Mifare, etc.).  When one is found
    // 'uid' will be populated with the UID, and uidLength will indicate
    //success = nfc.startPassiveTargetIDDetection(PN532_MIFARE_ISO14443A); //106000
    success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid[0], &uidLength);

   if (success) {
    Stop();
    checksum1 = uid[0] ^ uid[1] ^ uid[2] ^ uid[3];
   }

   current_loc = checksum1;
   find_loc(current_loc);
     }

  if (target_col < current_col) {

    // look left
    while (60>heading || heading>80){
        TurnRight();
        compass.read();
        heading = compass.getAzimuth();
        }

    // move forward
    Forward();

    // stop when sensing rfid tag
    uint8_t checksum1;
    boolean success;
    uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the returned UID
    uint8_t uidLength;        // Length of the UID (4 or 7 bytes depending on ISO14443A card type)

    // Wait for an ISO14443A type cards (Mifare, etc.).  When one is found
    // 'uid' will be populated with the UID, and uidLength will indicate
    //success = nfc.startPassiveTargetIDDetection(PN532_MIFARE_ISO14443A); //106000
    success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid[0], &uidLength);
```

Figure 6-36: Final Project Code 7

```

   if (success) {
    Stop();
    checksum1 = uid[0] ^ uid[1] ^ uid[2] ^ uid[3];
   }

   current_loc = checksum1;
   find_loc(current_loc);

  }

}





void setup(void) {

  SPI.begin();
  Wire.begin();


  // Compass
  compass.init();
  //compass.setMode(0x00, 0x0C, 0x10, 0x00);
  compass.setCalibration(-1175, 1537, -707, 2022, -1138, 808);


  // Ultrasonic
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input


  // Rfid
  nfc.begin();
  // Set the max number of retry attempts to read from a card
  // This prevents us from waiting forever for a card, which is
```

Figure 6-37: Final Project Code 8

```
  // the default behaviour of the PN532.
  //nfc.setPassiveActivationRetries(0xFF);
  // configure board to read RFID tags
  nfc.SAMConfig();


  // Proximity
  pinMode(detect_pin, INPUT);


  // Motors
  pinMode(MOTOR_A1_PIN, OUTPUT);
  pinMode(MOTOR_B1_PIN, OUTPUT);

  pinMode(MOTOR_A2_PIN, OUTPUT);
  pinMode(MOTOR_B2_PIN, OUTPUT);

  pinMode(PWM_MOTOR_1, OUTPUT);
  pinMode(PWM_MOTOR_2, OUTPUT);

  pinMode(CURRENT_SEN_1, OUTPUT);
  pinMode(CURRENT_SEN_2, OUTPUT);

  pinMode(EN_PIN_1, OUTPUT);
  pinMode(EN_PIN_2, OUTPUT);


  // keypad and Screen
  lcd.backlight();
  lcd.init();
  lcd.clear();
  lcd.setCursor(0, 0);


}

void loop(void) {

```

Figure 6-38: Final Project Code 9

```
  // Current direction
  compass.read();
  heading = compass.getAzimuth();


  // Current location
  uint8_t checksum1;
  boolean success;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the returned UID
  uint8_t uidLength;        // Length of the UID (4 or 7 bytes depending on ISO14443A card type)

  // Wait for an ISO14443A type cards (Mifare, etc.).  When one is found
  // 'uid' will be populated with the UID, and uidLength will indicate
  //success = nfc.startPassiveTargetIDDetection(PN532_MIFARE_ISO14443A); //106000
  success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid[0], &uidLength);

   if (success) {
    checksum1 = uid[0] ^ uid[1] ^ uid[2] ^ uid[3];
   }

   current_loc = checksum1;
   find_loc(current_loc);


  // Getting user inputs
  while(target_row<0 || target_row>(rows-1)){
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Enter Row");
    target_row = my_keypad.waitForKey();
    }
    while(target_col<0 || target_col>(cols-1)){
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Enter Column");
    target_col = my_keypad.waitForKey();
    }

```

Figure 6-39: Final Project Code 10

```
  target_loc = grid[target_row][target_col];

  move_rows();
  move_cols();

  val = digitalRead(detect_pin);
  while(val == HIGH){
    continue;
    }

   target_loc = origin_loc;
   target_row = origin_row;
   target_col = origin_col;

   move_rows();
   move_cols();

}



void Stop()
{
  usMotor_Status = BRAKE;
  motorGo(MOTOR_1, usMotor_Status, 0);
  motorGo(MOTOR_2, usMotor_Status, 0);
}

void Forward()
{
  usMotor_Status = CW;
  motorGo(MOTOR_1, usMotor_Status, usSpeed);
  motorGo(MOTOR_2, usMotor_Status, usSpeed);
}

void Reverse()
{
  usMotor_Status = CCW;
```

Figure 6-40: Final Project Code 11

```
  motorGo(MOTOR_1, usMotor_Status, usSpeed);
  motorGo(MOTOR_2, usMotor_Status, usSpeed);
}

void IncreaseSpeed()
{
  usSpeed = usSpeed + 10;
  if(usSpeed > 255)
  {
    usSpeed = 255;
  }


  motorGo(MOTOR_1, usMotor_Status, usSpeed);
  motorGo(MOTOR_2, usMotor_Status, usSpeed);
}

void DecreaseSpeed()
{
  usSpeed = usSpeed - 10;
  if(usSpeed < 0)
  {
    usSpeed = 0;
  }


  motorGo(MOTOR_1, usMotor_Status, usSpeed);
  motorGo(MOTOR_2, usMotor_Status, usSpeed);
}

void TurnRight()
{
  usMotor_Status = CW;
  motorGo(MOTOR_1, usMotor_Status, 100);
  motorGo(MOTOR_2, usMotor_Status, 0);
}

void TurnLeftt()
```

Figure 6-41: Final Project Code 12

```
{
  usMotor_Status = CW;
  motorGo(MOTOR_1, usMotor_Status, 0);
  motorGo(MOTOR_2, usMotor_Status, 100);
}

void motorGo(uint8_t motor, uint8_t direct, uint8_t pwm)         //Function that controls the variables: motor(0 ou 1), direction (cw ou ccw) e pwm (entra 0 e 25
{
  if(motor == MOTOR_1)
  {
    if(direct == CW)
    {
      digitalWrite(MOTOR_A1_PIN, LOW);
      digitalWrite(MOTOR_B1_PIN, HIGH);
    }
    else if(direct == CCW)
    {
      digitalWrite(MOTOR_A1_PIN, HIGH);
      digitalWrite(MOTOR_B1_PIN, LOW);
    }
    else
    {
      digitalWrite(MOTOR_A1_PIN, LOW);
      digitalWrite(MOTOR_B1_PIN, LOW);
    }

    analogWrite(PWM_MOTOR_1, pwm);
  }
  else if(motor == MOTOR_2)
  {
    if(direct == CW)
    {
      digitalWrite(MOTOR_A2_PIN, LOW);
      digitalWrite(MOTOR_B2_PIN, HIGH);
    }
    else if(direct == CCW)
    {
```

Figure 6-42: Final Project Code 13

```
    {
      digitalWrite(MOTOR_A2_PIN, HIGH);
      digitalWrite(MOTOR_B2_PIN, LOW);
    }
    else
    {
      digitalWrite(MOTOR_A2_PIN, LOW);
      digitalWrite(MOTOR_B2_PIN, LOW);
    }

    analogWrite(PWM_MOTOR_2, pwm);
  }
}
```

Figure 6-43: Final Project Code 14

The Algorithm:

- At first, we include the libraries which we need, and we discussed them in the previous section
- Then we define pins for each component
- We assume that the origin location is (2,0) which has id = 165
- We define an array grid that has the checksums of the ids of RFID tags
- To be able to reach the target location we need to know the target location id, and always monitor the orientation and location of the robot for navigation
- find_loc() function: takes the id of RFID card and find its row and its column in the grid
- move_row() function: at first, we enable the two motors ,then check if the target row is greater than the current row then the robot should look down, or if the target row is smaller than the current row then the robot should look up. Then the robot keeps rotating until it faces the right orientation using the compass then it goes forward. It should stop when it reads an RFID tag, if it reads the tag, it will use function find_loc() to know and update its current row and column
- move_col() function: at first, we enable the two motors, then check if the target column is greater than the current column then the robot should look right, or if the target column is smaller than the current column then the robot should look left. Then the robot keeps rotating until it faces the right orientation using the compass then it goes forward. It should stop when it reads an RFID tag, if it reads the tag, it will use function find_loc() to know and update its current row and column
- The robot will use move_row(), and move_col() function to reach its target location
- At setup() function:
    1. Begin SPI, and I2C communication

2. Initialize the compass module
3. Configure the pins of the ultrasonic module
4. Configure the RFID module
5. Configure the pins of IR proximity sensor
6. Configure the pins of the motor driver
7. Initialize the keypad and the LCD screen

- At loop() function:
    1. The robot deduces the current location of the robot by reading the current id of the RFID tag using the RFID module and calculating the current row and column, and read the output of the compass module to know the current orientation
    2. The robot receives the target row and column from the user using the keypad
    3. First, the robot moves to the target row then to the target column to reach the target location
    4. After reaching the target location the robot will stop and wait for the customer to dismiss it using the proximity sensor
    5. When the robot is dismissed, it returns to its original location (its base)
- we use timer and interrupt, timer will wait for a fixed period and at the end of each period interrupt the code and goes to ISR, and the ultrasonic sensor start working, if it detects any obstacle the robot stops till the object is removed

Chapter 7 : Conclusion and future work

7.1. Conclusion

Our project is designed to provide services in restaurants and hospitals.

By automating certain tasks like delivering food to customers or delivering food and medicine to patients we can achieve more reliable services at lower cost. We can also achieve healthier environment by limiting the human interaction thus reducing the spread of viruses and diseases.

We explored different indoor navigation techniques, and we utilized RFID navigation to accomplish:

- High quality service
- High accuracy navigation
- Simple design
- Ease of use
- Low build and maintenance cost
- Ease of implementation in exiting environments

7.2. Future Work

Our goal is to automate repetitive and labor-intensive tasks like frontline jobs, with our biggest challenge to face is indoor navigation.

We suggest in future work:

- Improve the responsiveness of the robot
- Improve the mechanical design to build bigger, stronger, and faster robot for more challenging environments

- Add wireless communication to interact with the robot both by the keypad and wirelessly
- Provide a way (hardware, APIs, …) to facilitate the integration of the robot with existing systems
- Add bigger battery to achieve longer operation time

References

- Arduino documentation

  https://store.arduino.cc/usa/mega-2560-r3

- RFID PN532 datasheet

  https://www.nxp.com/docs/en/nxp/data-sheets/PN532_C1.pdf

- Compass Module datasheet

  https://img.filipeflop.com/files/download/Datasheet-QMC5883L-1.0%20.pdf

- LCD Screen datasheet

  https://circuitdigest.com/article/16x2-lcd-display-module-pinout-datasheet

- Motor Driver datasheet

  https://www.pololu.com/file/0J52/vnh2sp30.pdf

- Brave new world: service robots in the frontline by Jochen Wirtz , Paul G. Patterson , Werner H. Kunz , Thorsten Gruber , Vinh Nhat Lu , Stefanie Paluch , Antje Martins

  https://www.emerald.com/insight/content/doi/10.1108/JOSM-04-2018-0119/full/html#sec002

- Design and implementation of a service robot for a restaurant

  https://www.researchgate.net/publication/245419381_Design_and_implementation_of_a_service_robot_for_a_restaurant

- https://github.com/mprograms/QMC5883LCompass
- https://github.com/adafruit/Adafruit-PN532
- https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library
- https://www.instructables.com/Monster-Motor-Shield-VNH2SP30