Autonomous Shopping Cart

Group Members

M. Hamza Ali	19I-0755
Basit Shabbir	19I-0766
Usama Ishfaq	19I-0902

Project Supervisor

Dr. Arshad Hassan

Department of Electrical Engineering

National University of Computer and Emerging Sciences, Islamabad

2023

Developer's Submission

"This report is being submitted to the Department of Electrical Engineering of the National University of Computer and Emerging Sciences in partial fulfillment of the requirements for the degree of BS in Electrical Engineering"

Developer's Declaration

"We take full responsibility of the project work conducted during the Final Year Project (FYP) titled **"Autonomous Shopping Cart"**. We solemnly declare that the project work presented in the FYP report is done solely by us with no significant help from any other person; however, small help wherever taken is duly acknowledged. We have also written the complete FYP report by ourselves. Moreover, we have not presented this FYP (or substantially similar project work) or any part of the thesis previously to any other degree awarding institution within Pakistan or abroad.

We understand that the management of Department of Electrical Engineering of National University of Computer and Emerging Sciences has a zero-tolerance policy towards plagiarism. Therefore, we as an author of the above-mentioned FYP report solemnly declare that no portion of our report has been plagiarized and any material used in the report from other sources is properly referenced. Moreover, the report does not contain any literal citing of more than 70 words (total) even by giving a reference unless we have obtained the written permission of the publisher to do so. Furthermore, the work presented in the report is our own work and we have positively cited the related work of the other projects by clearly differentiating our work from their relevant work.

We further understand that if we are found guilty of any form of plagiarism in our FYP report even after our graduation, the University reserves the right to withdraw our BS degree. Moreover, the University will also have the right to publish our names on its website that keeps a record of the students who committed plagiarism in their FYP reports."

_____	_____	_____
M. Hamza Ali	Basit Shabbir	Usama Ishfaq
BS(EE) 2019-0755	BS(EE) 2019-0766	BS(EE) 2019-0902

_____

Certified by Supervisor

_____

Verified by Plagiarism Cell Officer

Dated: _____

Abstract

Shopping is a routine activity which is part of our daily lives. Traditionally, the customer manually pushes the shopping cart. This makes shopping inefficient and, at times, a cumbersome process. The automation of shopping carts has gained significant attention in recent years due to the potential for enhancing the shopping experience. This project aims to design and develop an automated shopping cart system capable of following the user and autonomously navigating through obstacles. By implementing various electrical engineering techniques and technologies such as sensor integration and motion control algorithms, the proposed solution aims to provide a seamless and efficient shopping experience.

The project begins with an analysis of the requirements and specifications of an automated shopping cart system. The design phase involves the selection and integration of various components to track the target and perceive the environment to detect obstacles accurately. Additionally, motion control systems such as motor drivers are incorporated to enable the cart to follow the user smoothly. Algorithms are developed to determine behavior of the cart in a variety of situations.

The system is implemented and tested using a prototype shopping cart equipped with the designed automation components. Thorough testing is conducted to evaluate the effectiveness and reliability of the automation system, including crowded environments and dynamic obstacles. Performance metrics such as tracking accuracy, obstacle detection rate, and response time are assessed to quantify the system's performance.

The automation technology developed in this project has the potential for future integration into retail environments, ultimately revolutionizing the way customers interact with shopping carts.

Acknowledgements

We would like to express our deepest gratitude to all individuals who have contributed to the successful completion of this project.

First and foremost, we would like to thank our project supervisor Dr. Arshad Hassan for his invaluable guidance, support, and expertise throughout the entire project. His continuous encouragement and insightful feedback were instrumental in shaping the direction of our work.

We would also like to extend our appreciation to the faculty members of the Electrical Engineering Department for providing us with a conducive learning environment and equipping us with the necessary knowledge and skills to undertake this project.

We are also indebted to the technical staff of the university for their assistance in providing access to laboratory facilities and equipment, which were crucial for conducting experiments and testing our system.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

With the passage of time, we have seen a gradual increase in our day-to-day requirements. The practice of repeatedly travelling to the market for the purchase of regular items is no longer feasible, and it is preferred to buy a month's supply of goods in a single trip. To this end, large shopping centers have opened in every major city to facilitate the customer. However, handling such a large volume of items can be difficult for the elderly, the ailing and the disabled. This problem persists to this day, very much so in our society where very little facilitation is provided to those in need.

Furthermore, customers generally feel embarrassed asking for the price of an item, or for help locating an item. These apprehensions hinder the shopping process and result in a waste of time and opportunity for sales.

Chapter 1 During the COVID-19 pandemic lockdown, great emphasis was placed on reducing face to face interactions. Several major retailers tested the feasibility of automated shopping carts. In the forefront of this was Kroger, which is a major retail company operating in the United States. According to an article published in the Washington Times in 2021 [1], the investment in smart shopping carts was recovered within a year.

1.1 Motivation

A survey conducted at a local shopping center shows that 67% of users are in favor of having an automated system (Figure 1). To this end, we have chosen a project which seeks to solve three main concerns for the customer:

1. Transportation of heavy shopping cart
2. Checking of item prices / total spending
3. Checking availability of items / locating item

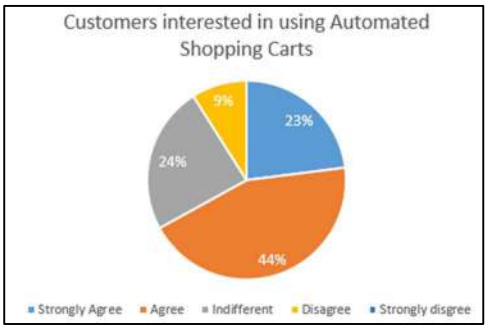

Figure 1.1 Survey results

1.2 Problem Statement

Customers are provided shopping carts when they enter large shopping malls to carry their items. Wholesale shopping is common practice, and shopping carts grow in size day by day. It is becoming increasingly difficult for large families, the disabled and the elderly to push carts during their shopping. Furthermore, wholesale shopping centers are giant structures where locating individual products is cumbersome if one is not familiar with the surroundings. Automation of shopping carts will overcome these issues, where the shopping cart will not only follow the

customer but also provide information about the location and price for any product desired. This will allow customers to shop more efficiently and with greater ease.

1.3 Literature Review

The biggest challenge of this project is to come up with an efficient tracking system which will not only be able to track the target but also avoid any obstacles which may appear along the path. Many methods of tracking and locating an object are available, the most efficient of which is with the use of GPS. However, GPS becomes redundant in an indoor environment. Several other methods are used for tracking which have relevant studies available.

In [2], image processing was used for tracking the target where a predefined custom tag was placed on the user. However, there are issues with image processing as continuously varying image background can make tracking difficult.

As discussed in [3] and [4], RFID tagging is also a common method of target identification and they have both used it in replacement of barcodes for counter-less billing. However, for the purpose of tracking a moving target they become less feasible as signal can be affected by metal and liquids in the surroundings. Reliability is questionable.

An implementation of a smart shopping cart was done in [5] where a robot (attached to a cart) used ultrasonic, Bluetooth and line sensors for the purpose of following a target. However, in this case the path used by the cart was fixed.

In [6], a tracking robot was designed using radio frequency for target localization and an ultrasonic system for distance measurement. Phase interference method was used for determining orientation of target.

Another execution of smart shopping cart was done by students of MBITS, India [7] where they utilized a configuration of IR sensors and a transmitter for the purpose of tracking of target. Ultrasonic sensors were used for distancing and obstacle detection. A similar implementation was done in [8].

Bluetooth Low Energy (BLE) beacons are extensively used for purpose of tracking targets. [9] and [10] have used it for indoor navigation and positioning using fingerprinting (mapping of data from several access points) and triangulation (distancing from 3 beacons to determine coordinates), while [11] has used BLE in combination with RF fingerprinting. Another study [12] has covered the use of BLE beacon utilizing RSSI and IMU readings to estimate position of the target.

After thoroughly reviewing the above literature, we have proposed the use of BLE for tracking of target while a LiDAR sensor will be used for obstacle detection and avoidance which is less prone to disturbance from external stimuli.

1.4 Existing products

The use of autonomous shopping carts is not a new concept. Many Western retailers have already incorporated such technologies into their services, such as Amazon and Sobeys.

The Amazon Dash Cart was initially introduced in July 2020 as a pilot program at an Amazon Fresh grocery store in Los Angeles. The smart carts leverage Amazon's "Just Walk Out" technology, which eliminates the need for cashiers, initially introduced in Amazon Go convenience stores. By combining computer vision and sensors, these carts can recognize items as shoppers place them inside designated bags. As customers add or remove items, the cart's display dynamically updates the total cost. When shoppers are ready to leave, they can conveniently exit through a designated lane, with Amazon automatically processing the payment using their credit card.

The Sobeys Smart Cart was designed to enhance the shopping experience by providing features such as automatic scanning of items, a built-in display for item information and promotions, and the ability to process payments directly on the cart. Sobeys Smart Carts use sensor fusion technology to identify items to help make checkouts seamless.

Both products mentioned above focus on counter-less billing by reducing the need to stand in long checkout queues. However, the products are quite expensive, relying on a large number of sensors and sophisticated processors. The average cost for a smart shopping cart manufactured in the United States is in the $ 5,000 to $ 10,000 range. These products also lack the ability to follow customers, and require manual maneuvering.

1.5 Methodology

The methodology adopted for the execution of the project involves two main processes, target tracking and obstacle avoidance.

For purpose of target tracking, an MPU sensor will be used. An MPU sensor provides 9 parameters based on its accelerometer, gyroscope and magnetometer observations. These nine values can be used to determine the pitch, roll and yaw of the sensor. These values vary whenever there is displacement of the sensor along a particular axis.

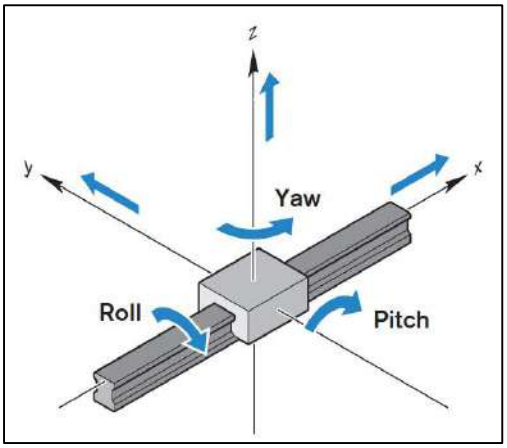

Figure 1.2 Pitch, roll and yaw for an object (positive x-axis is forward direction)

For our use, only yaw is relevant to determine left or right movement of the user. These values are computed via algorithms. These values will then be transmitted to the shopping cart via Bluetooth. Based on the strength of the signal received by the cart, distance will be estimated. With information about both distance and direction to the target, the shopping cart will follow the user.

For obstacle avoidance, a LiDAR sensor will be used. The LiDAR sensor determines distance to an object with a laser and measures time taken by the reflected light to reach the sensor. The LiDAR sensor will be mounted on a servomotor which will rotate the sensor 180°. This will allow the sensor to cover the front portion of the shopping cart and detect any obstacles in its path. With this information, the shopping cart will be able to make decisions to circumnavigate the obstacle.

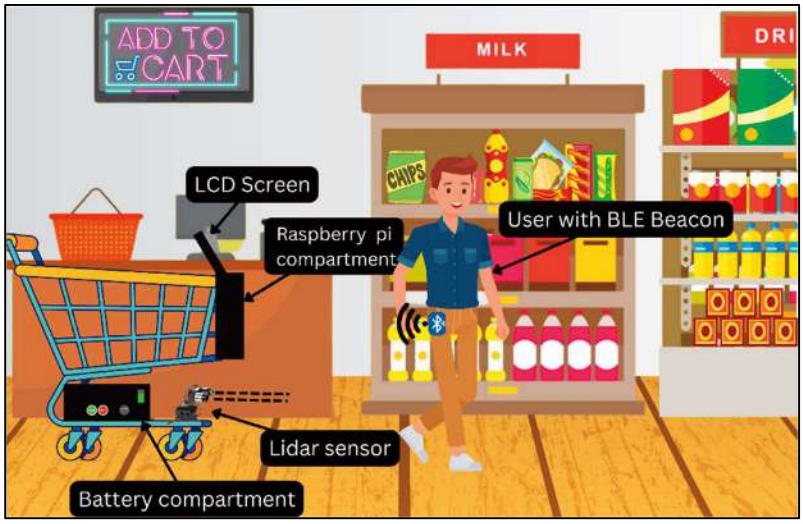

Figure 1.3 Project methodology visualized

1.6 Project Scope

1.6.1 Project Objective

To prepare a working prototype of an autonomous shopping cart by the end of semester Spring 2023 within the estimated project budget.

1.6.2 Project Deliverables

- Bluetooth Low Energy (BLE) beacon based on Bluetooth 4.0 for purpose of target tracking. Beacon will contain Bluetooth module, MPU sensor, Arduino mini, batteries, and compatible charger.
- Voltage converter unit to give 5V output to various control unit components from 24V input battery.

- Control unit integrated with shopping cart to control and drive the cart. The unit will contain Raspberry pi 4 model B, motor drivers, high torque motors, LiDAR sensor, touch LCD, and barcode scanner.
- Product location algorithm with inventory database on Raspberry pi.

1.6.3 Technical Requirements

- The prototype must be able to track the target with less than 5% margin for error.
- The prototype should detect and avoid obstacles with less than 1% margin for error.
- The control unit must be compatible with typical shopping cart chassis for installation.

1.6.4 Limits and Exclusions

- The prototype will be designed to operate for a maximum loading of 60 kgs.
- The prototype will be designed only for indoor environment.
- The inventory database will be limited to microprocessor internal memory and not be shifted to a cloud service.
- Work on project will be limited to university lab premises and timings.

1.7 Report Outline

The report is divided into multiple sections for ease of understanding.

Chapter 2 discusses the proposed solution in detail and includes relevant information such as the project block diagram, flow charts of various processes and a list of project deliverables. It also includes schematic diagrams and details of software and hardware implementations.

Chapter 3 discusses the results of the project and draws conclusions about the reliability and efficiency of the designed system. It also includes recommendations and opportunities for future enhancements.

Chapter 2 Design & Implementation

This chapter has detailed description of the proposed solution, the design parameters and the project implementation process. Section 2.1 details the scope of the project. Section 2.2 covers the block diagram and specifications of each module. Section 2.3 explains the flow charts for various algorithms. Section 2.4 has calculations for various specifications and ratings of components. Explanation of the software and hardware implementation of the project is covered in Sections 2.5 and 2.6 respectively.

2.1 Block Diagram

Block diagram for the project is shown in the figure below.

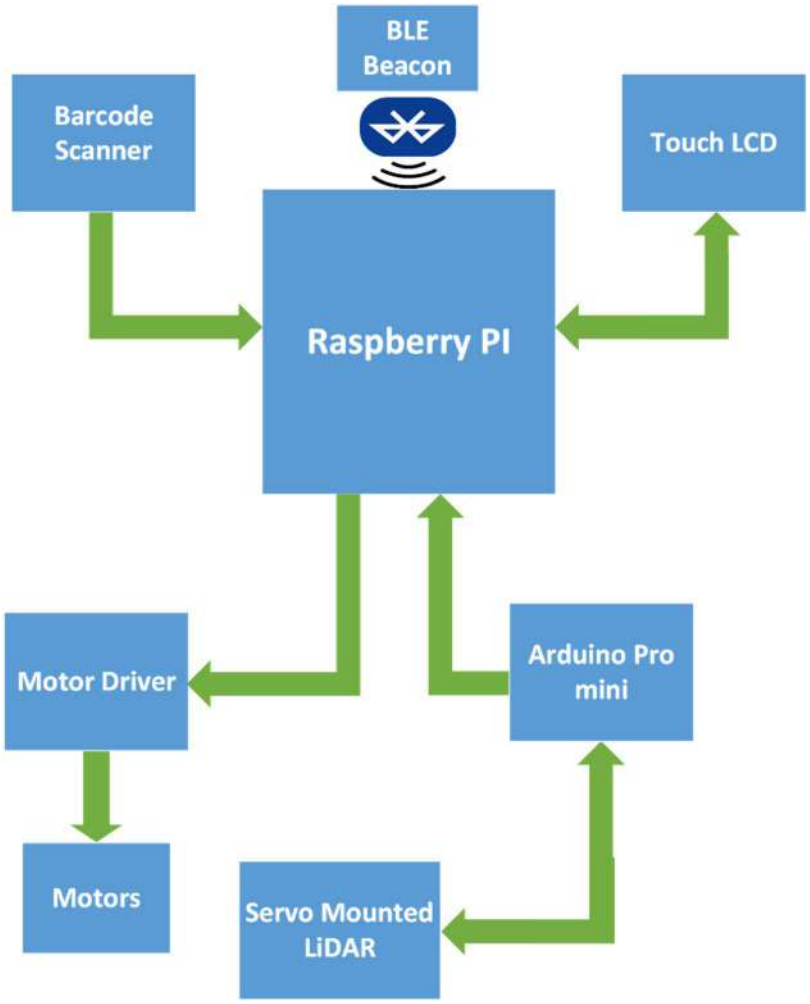

Figure 2.1 System block diagram

Explanation of each block is available on the next page.

- **BLE Beacon**
  The beacon consists of an Arduino microcontroller, an MPU sensor for providing inertial measurement unit (IMU) readings, and a Bluetooth module for transferring data. Based on the IMU readings, three parameters are computed (pitch, roll and yaw) which are used for estimating position and orientation of the BLE beacon by the control unit on the shopping cart. These parameters are communicated to the control unit via Bluetooth.
  Data input: IMU readings MPU sensor.
  Data output: pitch, roll, and yaw to Raspberry pi on the shopping cart.

- **Raspberry pi**
  The Raspberry pi will receive data from both the BLE beacon and the LiDAR sensor. It will control the motor drivers based on this data to follow the target as well as to avoid obstacles along the path. It will also be interfaced with a touch LCD to provide user access to the local database for information about products.
  Data input: pitch, roll, yaw from BLE beacon and obstacle distance from LiDAR sensor.
  Data output: control signals to motor drivers, data to LCD for display.

- **Touch LCD**
  LCD will provide user access to the local database with a graphical user interface (GUI). The user will be able to get information about products such as product price, location, availability etc.

- **Barcode Scanner**
  Barcode scanner will allow user to scan any item and instantly get information about the product. The user may also add item to cart to keep track of spending.

- **Servo mounted LiDAR**
  A LiDAR sensor is attached to a servo motor rotating along 180° to locate obstacles in the cart's path. This will allow the Raspberry pi to make decision to circumnavigate the obstacle.
  Data input: obstacle detection by sensor
  Data output: obstacle distance from the sensor

- **Motor Drivers**
  Motor drivers will receive control signals from the Raspberry pi to drive the motors attached to the shopping cart. This will allow the cart to follow its target.

- **Motors**
  The motors are attached to the rear wheels of the shopping cart. Based on the control signals from the motor drivers, the motors will drive the cart wheels to follow the target and to avoid obstacles.

2.2 Flow Charts

Various processes are taking place during the operation of the Autonomous Shopping Cart. The two main processes are target tracking and obstacle avoidance. The flowcharts with a brief summary for both these processes are given below.

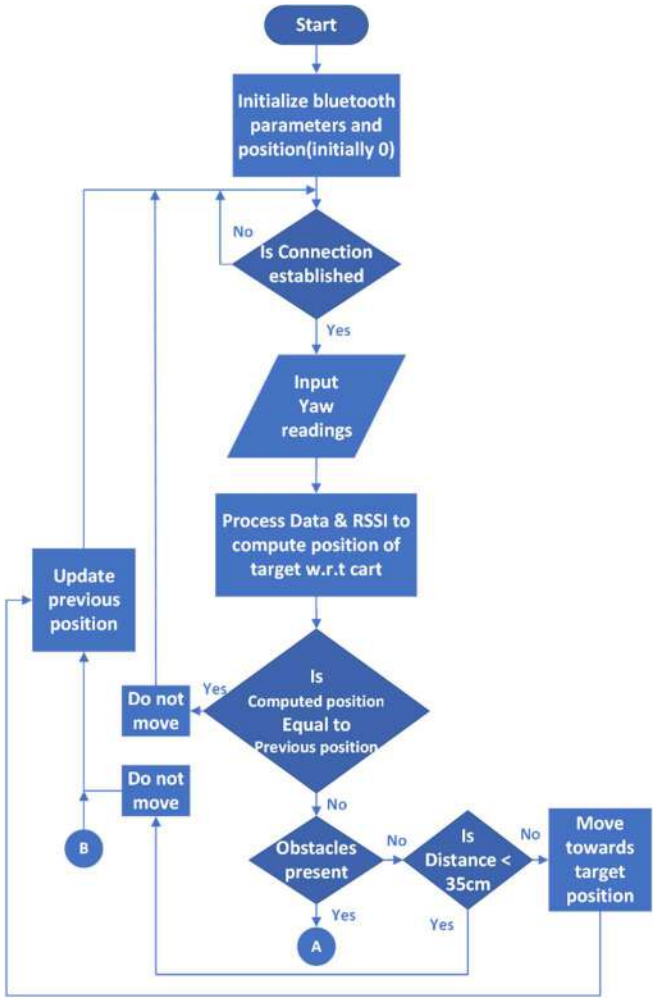

Figure 2.2 Target tracking algorithm

After initialization of parameters, connection is established between the BLE beacon and the shopping cart. The beacon transmits values of yaw to the Raspberry pi. With help of these values and the received signal strength (RSSI), the processor estimates the target location. If obstacle is not present, the motors are driven to follow the target up till the threshold value. If obstacle is detected, the obstacle avoidance algorithm comes into play.

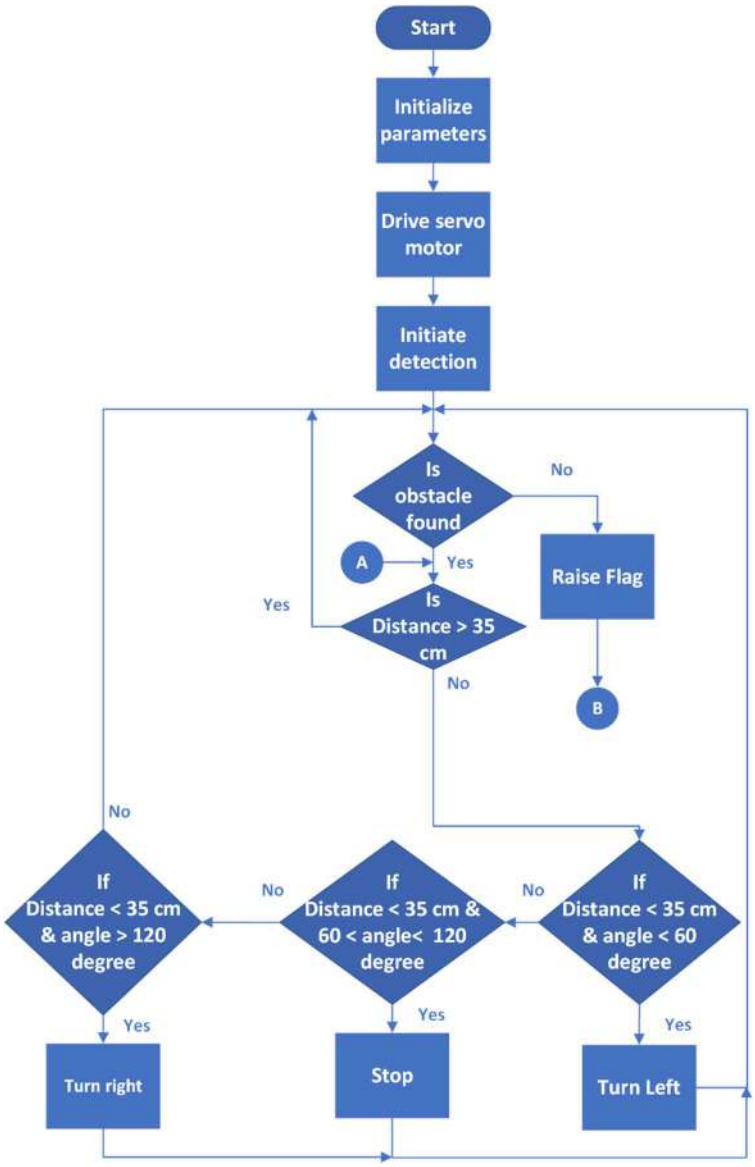

Figure 2.3 Obstacle avoidance algorithm

If obstacle is detected by the LiDAR sensor, its location with respect to the shopping cart will be determined (left, right or directly in front). Based on location of obstacle, the shopping cart will take remedial measures to change its course and circumnavigate it. Once obstacle is cleared, the tracking algorithm will continue.

2.3 Calculations

The project involved two particular calculations, the specifications for the motors and the power requirements of each component. The working for these calculations is discussed in this section.

2.3.1 Calculations for choice of motors

To determine what power source to use for the shopping cart, we need information about the motors which will be used to drive the cart. The project scope states that the minimum design load will be 60 kgs. Based on this requirement, the following calculations are made:

Total weight = load + cart chassis = 60 + 15 = 75 kgs

$F = mg$

$F = (75)(9.81)\ {}^{Kgm}/_{s^2} = 735.75\ {}^{Kgm}/_{s^2} = 735.75\ N$

Radius of coupled wheel : $r = 5\text{cm} = 0.05\text{m}$

estimated torque $= \tau_{est} = F * r$

$\tau_{est} = (735.75)(0.05)$

$\tau_{est} = 36.78\ \text{Nm}$

We know the relationship

$P_{out} = \tau * \omega$

$$\omega = \frac{(\text{speed in RPM})(2\pi)}{60}$$

Normally high torque motors operate in the range of 50-100 rpm. Taking speed as 60rpm:

$$\omega = \frac{(60)(2\pi)}{60} = 2\pi\ \text{rad/sec}$$

$P_{out} = \tau_{est} * \omega$

$P_{out} = (36.78)(2\pi) = 230.97\ \text{W}$

$P_{out} = 230.97\ \text{W}$ is the total power that will be needed as output to move 75kg

As two motors will be used so each motor can have half the power output. i.e.

$$P_{\text{per motor}} = \frac{P_{out}}{2} = \frac{230.97}{2} = 115.48\ \text{W}$$

$P_{\text{per motor}} = 115.48\ \text{W}$

We add safety factor of 10% so now

$P_{\text{per motor}}' = 127.028\ \text{W}$

Rounding off we get

$P_{\text{per motor}}' = 130\ \text{W}$

Specifications for a 24V DC motor:

Sr. No	Description	Value
1	Motor power rating	130 W
2	Motor current rating (for 24 V)	5.41 A
3	Motor speed	60 rpm

Table 2-1 Specifications for 24 V DC motor

2.3.2 Calculations for power requirements of shopping cart

The project consists of various components working at different voltage levels. The maximum voltage requirement is 24V, which will be provided to the motors. The remaining components all have a rating of 5V. For this reason, buck converters are needed. Based on current requirements, a design of two buck converters is proposed.

Calculations are shown as under:

Sr. No	Item Description	Quantity (Nos)	Nominal Voltage (V)	Nominal Current (A)	Power (W)
1	High torque motor	2	24	1.5	72.00
2	7-inch LCD display for raspberry pi	1	5	1.2	6.00
3	Raspberry pi 4b, 4gb ram	1	5	3	15.00
4	Lite-V3 LiDAR sensor	1	5	0.12	0.60
5	Servo motors	2	5	0.2	2.00
				Total	**95.60**

Table 2-2 Power requirement of various components of shopping cart

Battery voltage:	24 V
Cart Battery current @ 24 V:	3.98 A
Maximum current requirement @ 5 V:	4.52 A

Proposed design of 2 LM2596 buck converters to step battery voltage to 5 V.

A circuit diagram was made for the voltage converter with the help of Fritzing. A housing was also designed to enclose the converter circuit for safety, using AutoCAD. Both designs are shown in the figures below.

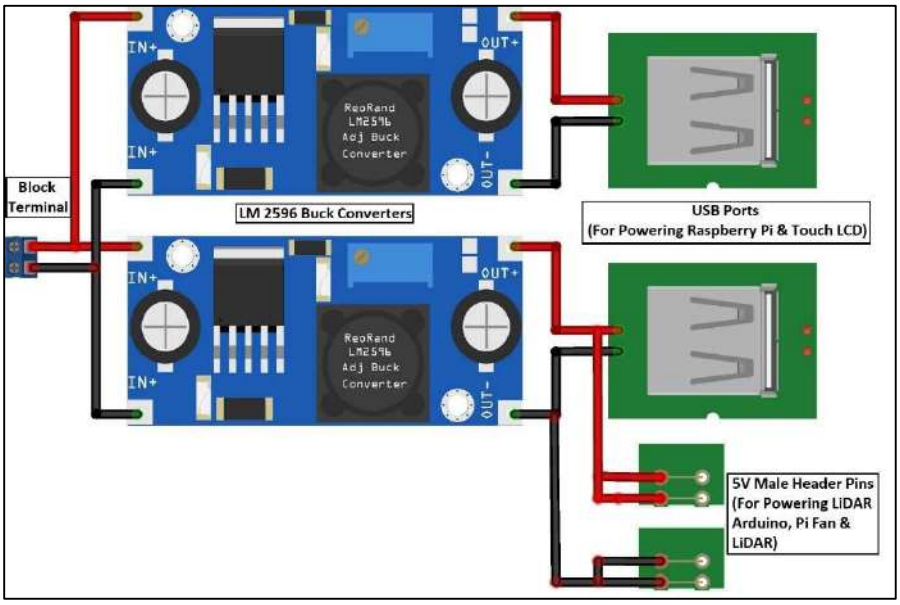

Figure 2.4 Schematic diagram for voltage converter circuit

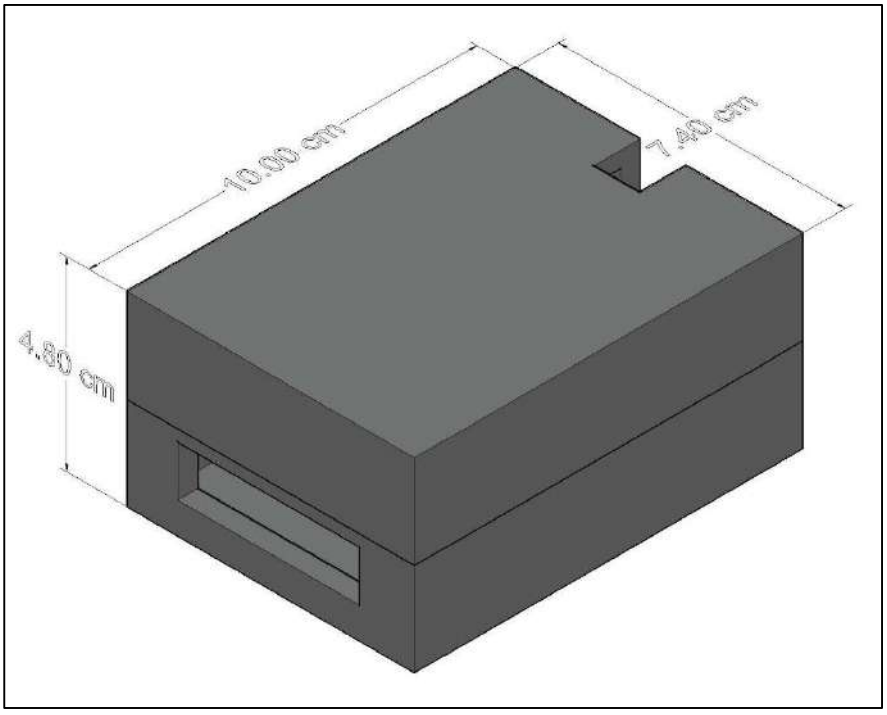

Figure 2.5 3D model of container for voltage converter

2.3.3 Calculations for power requirements of BLE beacon

The BLE beacon consists of various components, however each component has the same rated voltage. Hence, no voltage conversion is necessary. Calculations are shown below:

Sr. No	Item Description	Quantity (Nos)	Nominal Voltage (V)	Nominal Current (A)	Power (W)
1	Arduino pro mini	1	3.3	0.02	0.066
2	MPU 9250 sensor	1	3.3	0.005	0.016
3	Bluetooth module	1	3.3	0.01	0.033

Table 2-3 Power requirements for BLE beacon

Beacon Battery current @ 3.3 V: 0.035 A

2.4 Software Implementation

For the implementation of the project, algorithms needed to be implemented for target tracking, obstacle detection, obstacle avoidance and for the graphical user interface (GUI). A database also needed to be established for storing information on available items. The algorithms written for each of these tasks are discussed below.

2.4.1 Target Tracking

For target tracking, we needed two algorithms, one for the BLE beacon and one for Raspberry pi.

The algorithm for BLE beacon was written with Arduino IDE for Arduino pro mini (Appendix A section A-1). This algorithm received data from MPU sensor which was the values of accelerometer, gyroscope and magnetometer in x, y and z directions. With the help of these values, we determine the pitch, roll and yaw of the beacon. For purpose of tracking in 2D space, only the yaw parameter is of importance. This value was then to be transmitted via Bluetooth to the Raspberry pi.

The algorithm for Raspberry pi (Appendix A section A-2, A-3) had to establish connection with the beacon, receive the yaw values and also estimate distance till the user with help of the strength of the signal from the beacon. The pi then sent control signals to the motor drivers to follow the user. This was done in Python language.

Text files were used to store values for variables that were to be shared between the scripts for target tracking and obstacle avoidance.

2.4.2 Obstacle detection

For obstacle avoidance, an algorithm was needed for the LiDAR sensor to communicate with the Raspberry pi. This is available in Appendix A section A-4.

The LiDAR sensor communicated with the Raspberry pi via an Arduino pro mini. The algorithm written for the Arduino microcontroller received data from the LiDAR sensor, determined location of obstacle relevant to the shopping cart and sent this information to the Raspberry pi.

2.4.3 Obstacle avoidance

The Raspberry pi received information about obstacles from the Arduino attached to the LiDAR sensor, and sent control signals to the motors to circumnavigate the obstacle. This code is part of the main Raspberry pi algorithm (Appendix A section A-5).

2.4.4 Graphical user interface

An algorithm for the GUI needed to be written for the Raspberry pi, and is provided in Appendix A section A-7. For design of GUI, tkinter library of Python was used. The GUI was designed to provide the following functions:

1. Access to database by search bar, categories or barcode.
2. Estimation and tracking of expenditure by keeping record of items added to cart.
3. Access to map of mall/retail center.
4. Ability to edit items in the shopping cart.

Screenshots of various options available in the GUI are given below.

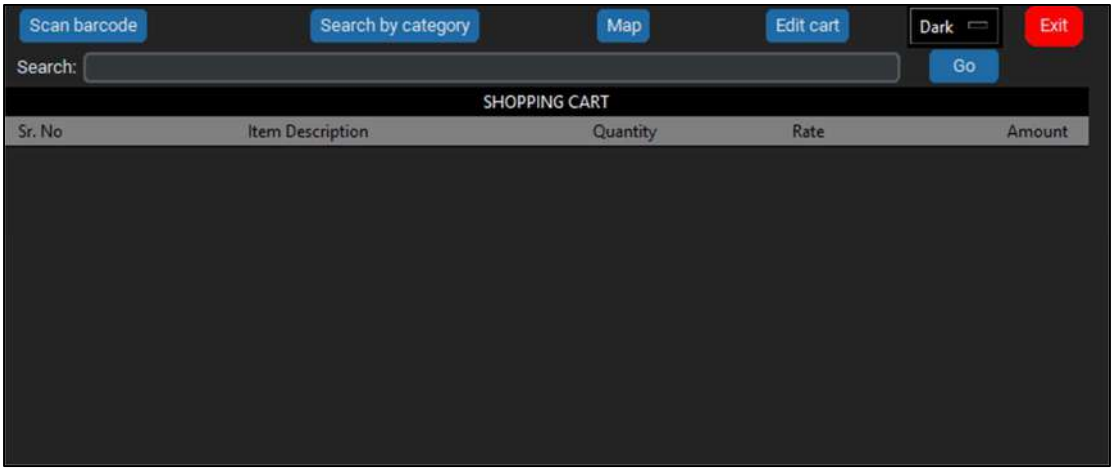

Figure 2.6 GUI main display

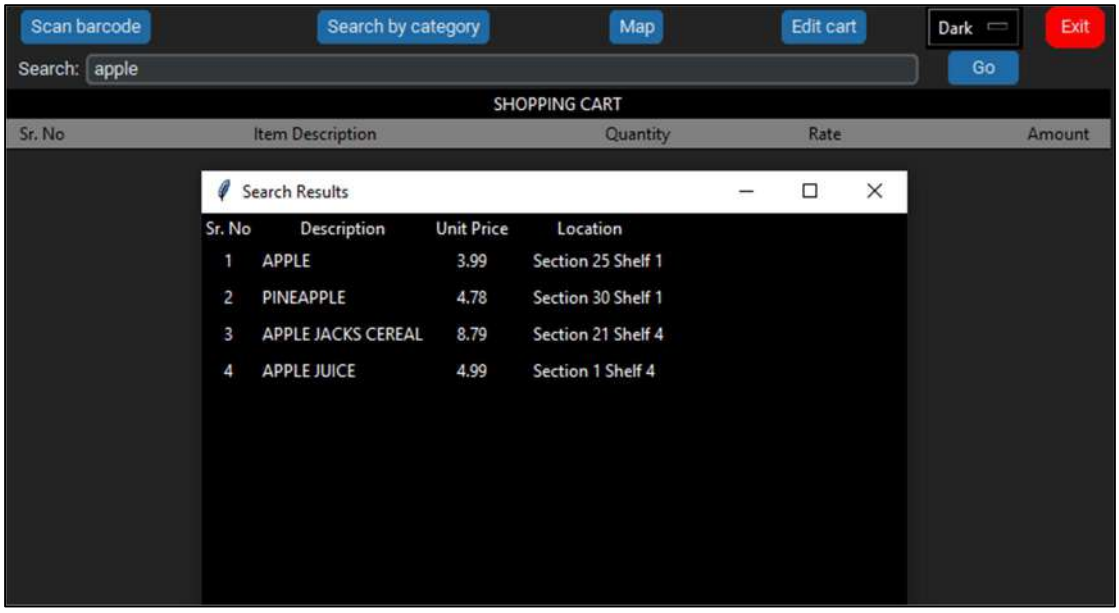

Figure 2.7 Results for item search with searchbar

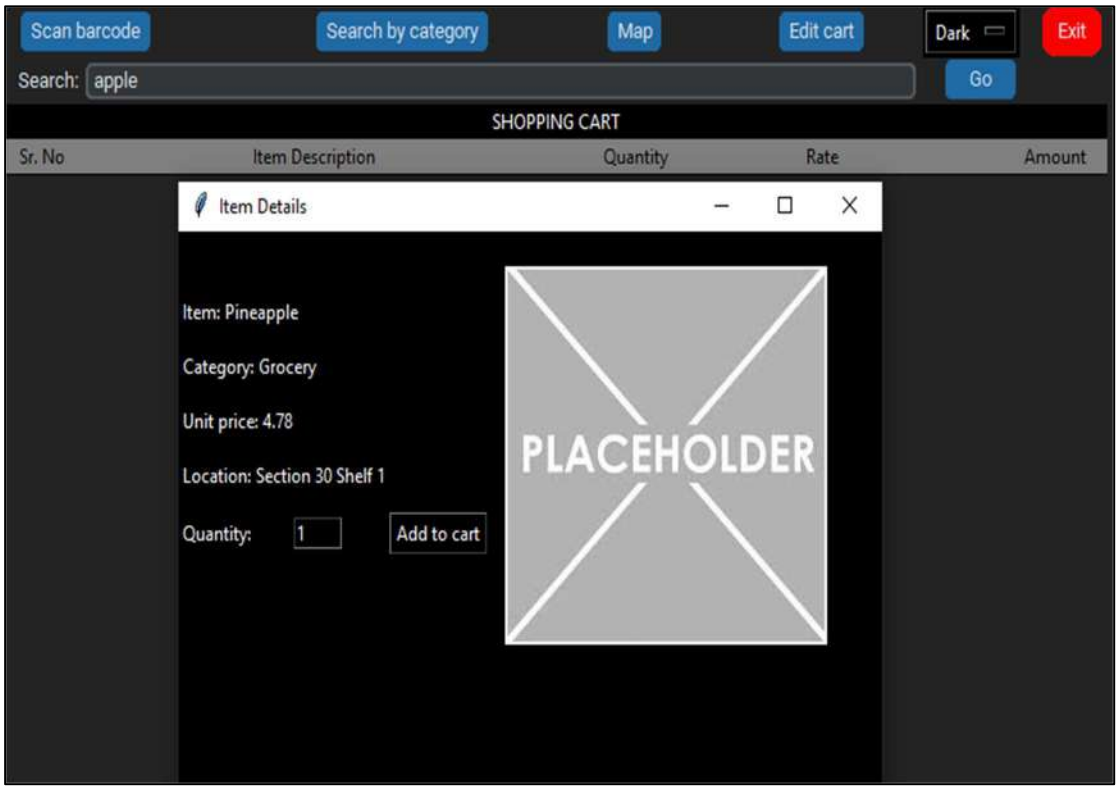

Figure 2.8 Item details and option to add to cart

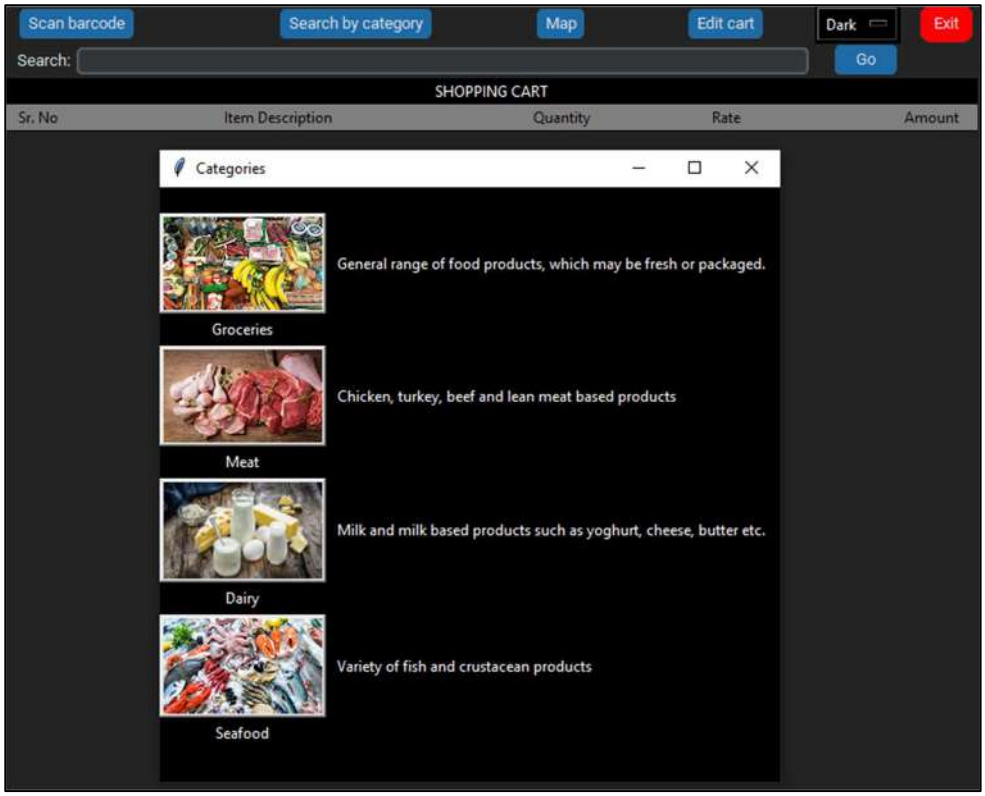

Figure 2.9 Various categories available in the database

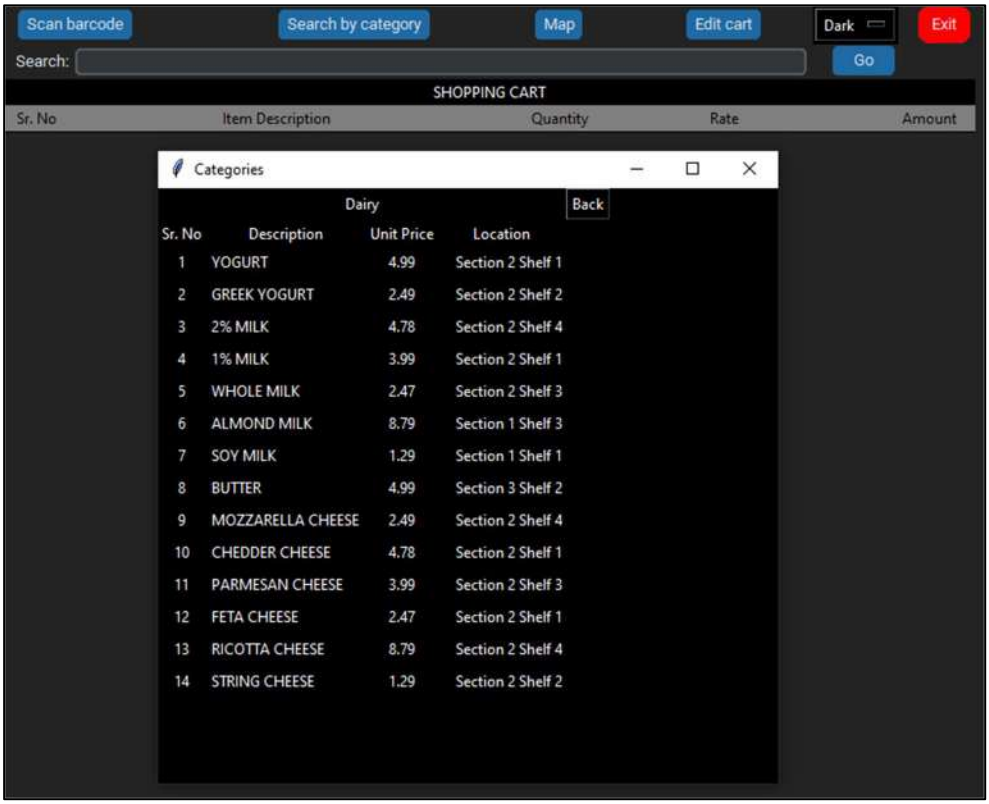

Figure 2.10 List of items for a category

2.4.5 Database

An algorithm was written to establish database in SQLite3 which is included in Python (Appendix A section A-6). For importing data to the database, an Excel spreadsheet with details of the items was used. Functions were written to add items to the database, as well as functions for accessing database in the GUI.

	A	B	C	D	E
1	Item name	Category	Unit price	Section	Shelf
2	APPLE	GROCERY	3.99	25	1
3	BANANA	GROCERY	2.47	26	1
4	BLACKBERRY	GROCERY	8.79	30	1
5	BLUEBERRY	GROCERY	1.29	12	1
6	COCUNUT	GROCERY	4.99	28	1
7	CRANBERRY	GROCERY	2.49	12	1
8	GRAPES	GROCERY	4.78	11	1
9	GRAPEFRUIT	GROCERY	3.99	29	1
10	KIWI	GROCERY	2.47	27	1
11	LEMON	GROCERY	8.79	25	1
12	LIME	GROCERY	1.29	30	1
13	MANGO	GROCERY	4.99	28	1
14	WATERMELON	GROCERY	2.49	27	1
15	CANTALOUPE	GROCERY	4.78	28	1
16	OLIVE	GROCERY	3.99	11	1
17	CLEMENTINE	GROCERY	2.47	28	1
18	MANDRINE	GROCERY	8.79	28	1
19	PEACH	GROCERY	1.29	26	1
20	PEAR	GROCERY	4.99	25	1
21	PLUM	GROCERY	2.49	25	1
22	PINEAPPLE	GROCERY	4.78	30	1
23	PUMPKIN	GROCERY	3.99	24	1

Figure 2.11 Excel spreadsheet for database

2.5 Hardware Implementation

Hardware of the project consists of two main parts:

1. BLE beacon
2. Autonomous shopping cart

The details of the components of each part are explained below:

2.5.1 BLE beacon

A schematic diagram for the BLE beacon is shown in Figure 2.17. It consists of a battery, an Arduino microcontroller, a Bluetooth module and an MPU sensor.

The MPU sensor detected displacements of the beacon along x, y and z-axis based on the movement of the user. The sensor was oriented in the beacon in such a way that the computed yaw values represented the left and right movement of the user. This information was transmitted via Bluetooth to the shopping cart control unit Raspberry pi.

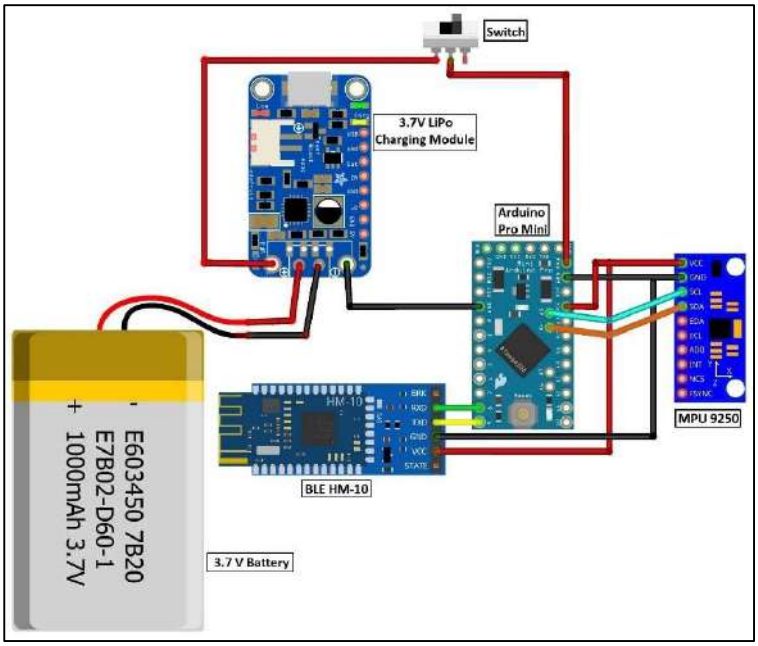

Figure 2.12 Schematic diagram for BLE beacon circuit

All components were assembled and tested to verify that correct values were being transmitted by the beacon. A container for the assembled components was also designed in AutoCAD to provide protection to the circuit.

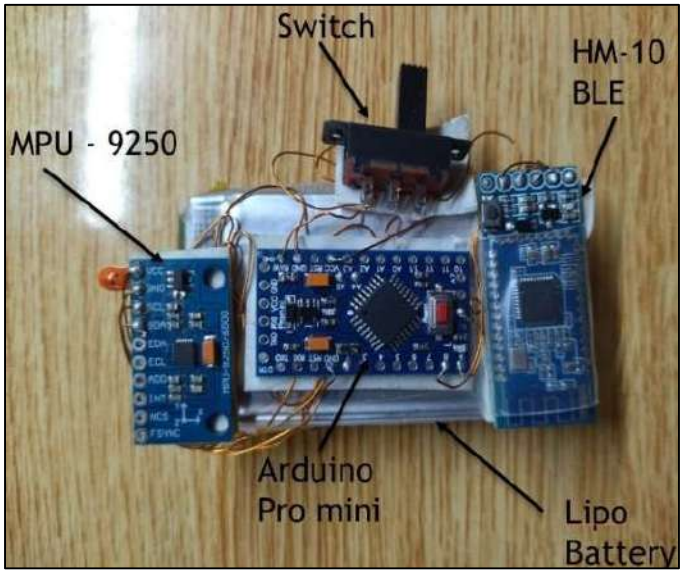

Figure 2.13 BLE beacon hardware implementation

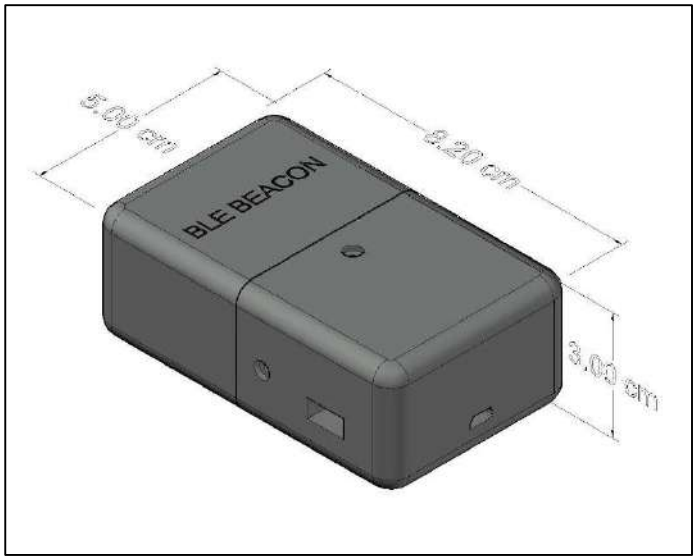

Figure 2.14 3D model of housing design for BLE beacon

2.5.2 Shopping cart

The shopping cart hardware working can be divided into 4 parts, namely

1. The installation of motors and wheels on the shopping cart.
2. The bottom compartment, consisting of the main battery and motor drivers.
3. The front compartment, consisting of the Raspberry pi and voltage converter.
4. The installation of LiDAR sensor and servomotor to the front of the cart.

Installation of motors and wheels

The existing wheels of the shopping cart needed to be removed before installation of motors and compatible wheels. Once removed, the motors were installed on the frame of the shopping cart, and coupled with the new wheels. Wiring connections between the motors and the motor drivers in the shopping compartment were done to finish this portion of the shopping cart.

Bottom compartment

The bottom compartment consisted of the main power supply i.e., the 24 V battery and the motor drivers. Other components were also present such as the charging module and a control switch. A schematic diagram for the compartment is shown.

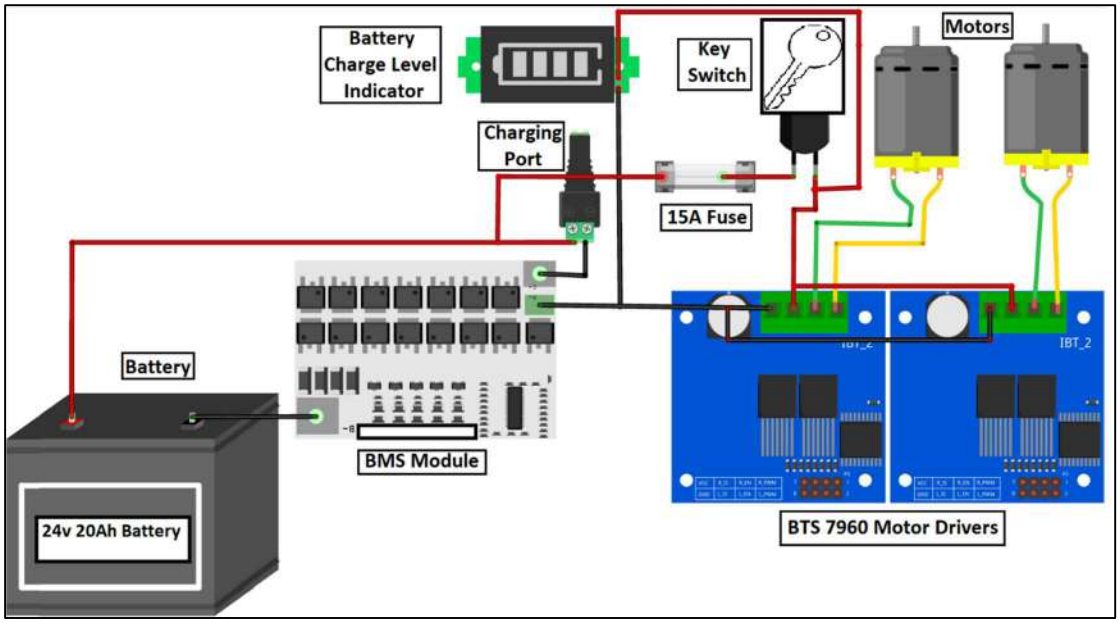

Figure 2.15 Schematic diagram for shopping cart bottom compartment

Front compartment

The front compartment consisted of the Raspberry pi microprocessor and the voltage converter (Figure 2.4). The touch LCD panel and the barcode scanner was also attached to the front compartment. The Raspberry pi was connected to the touch LCD, barcode scanner, motor drivers and the LiDAR sensor (via an Arduino microcontroller).

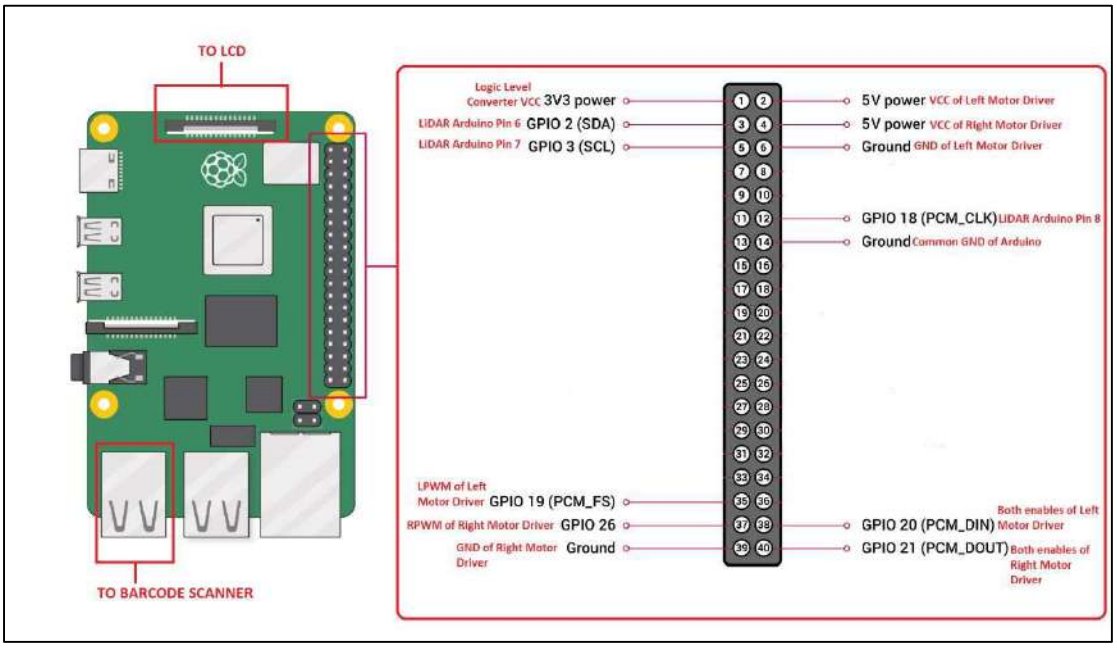

Figure 2.16 Raspberry pi pinout configuration

LiDAR sensor and servomotor

LiDAR sensor is to be used for obstacle detection, and hence is placed at the front of the cart. The LiDAR sensor is mounted on a servomotor which rotates it 180°, allowing detection of any obstacles which may appear in the path of the shopping cart.

As Arduino pro mini logic operates at 5 V and Raspberry pi logic level is 3 V, there was a need for logic level conversion to be implemented between the two. This was done with the help of a 5 to 3 V logic level converter circuit.

Schematic diagram of the circuit is shown below.

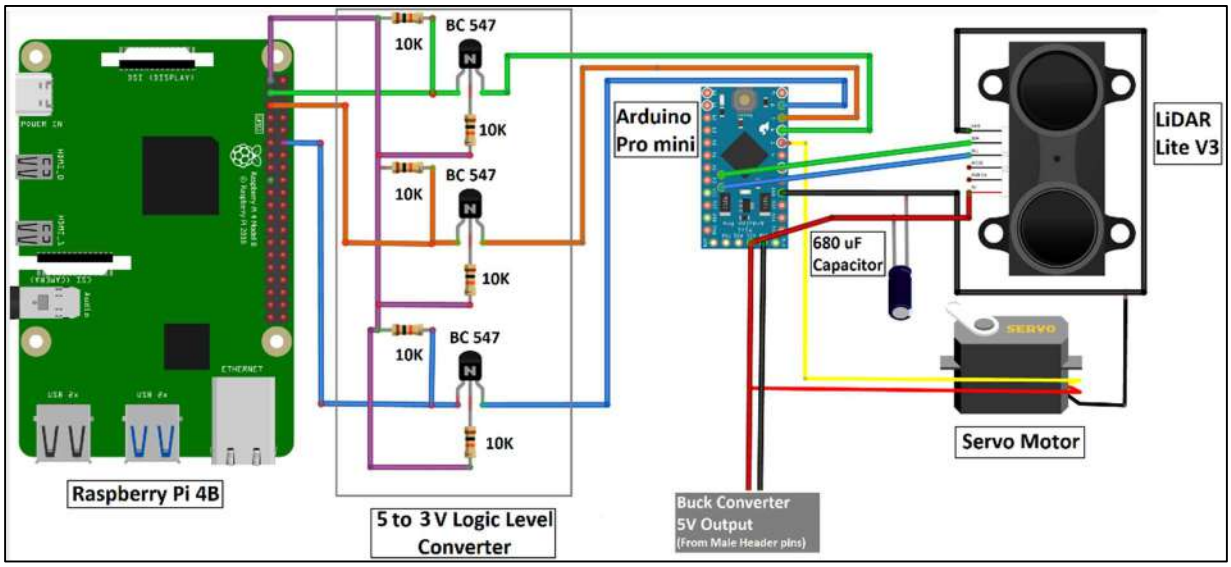

Figure 2.17 Schematic diagram for LiDAR sensor circuitry

The various stages of the shopping cart, as well as the final prototype after installation of all components are shown in the next few pages.

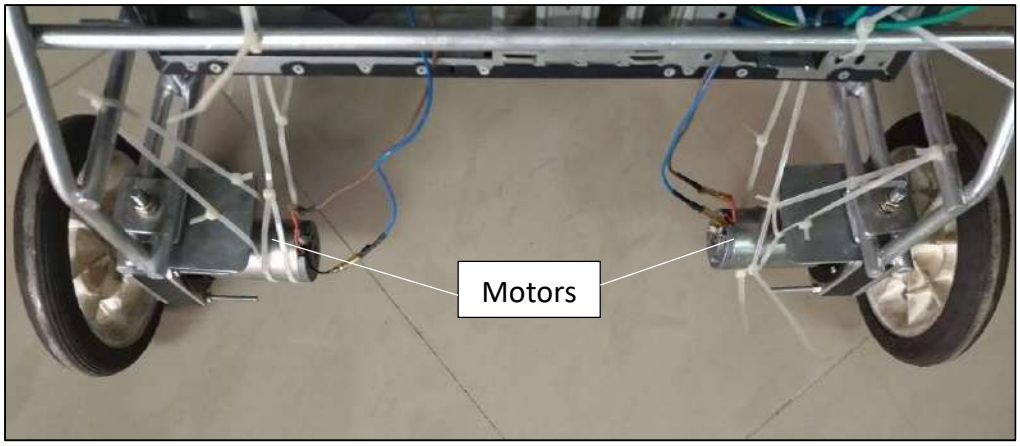

Figure 2.18 Motors and wheels on rear end of shopping cart

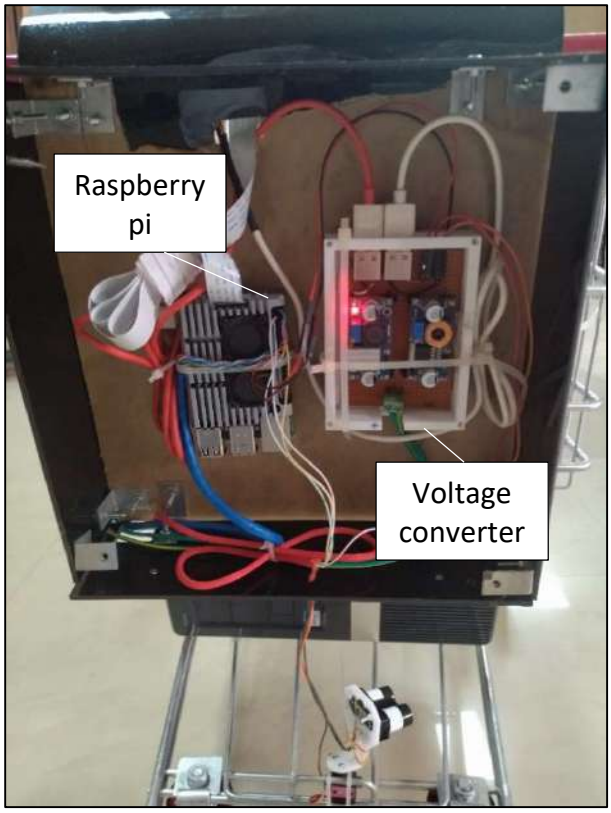

Figure 2.19 Shopping cart front compartment

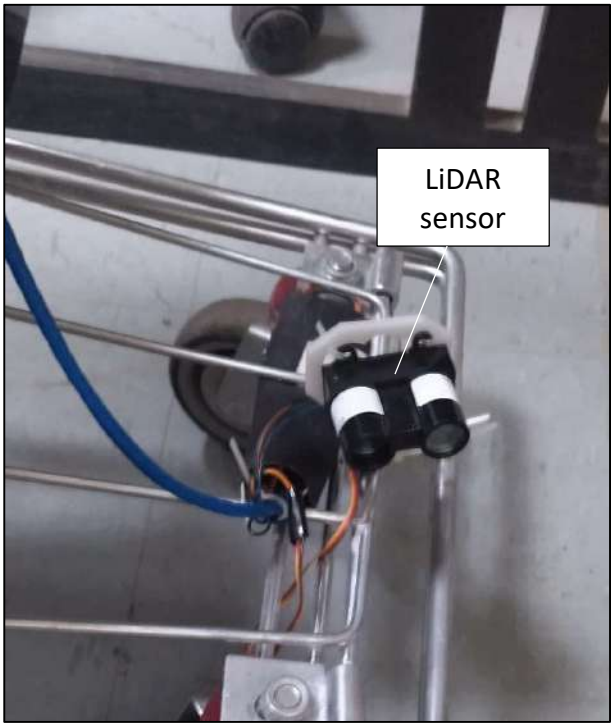

Figure 2.20 Servo mounted LiDAR sensor

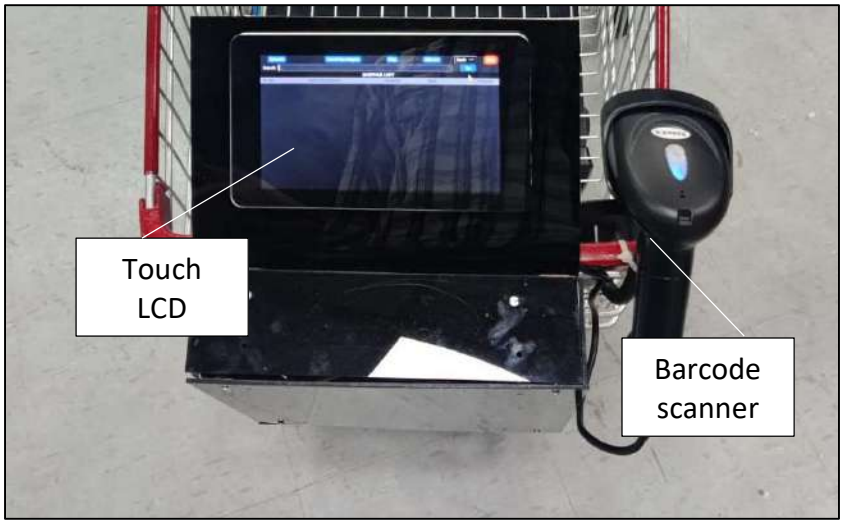

Figure 2.21 Barcode scanner and LCD on front compartment

Figure 2.22 Final BLE beacon

Figure 2.23 Final shopping cart prototype

Chapter 3 Results and Recommendations

3.1 Project Progress

Based on the project deliverables discussed in the project scope (section 2.1), the following key milestones were defined:

FYP-1

- Voltage converter
- Interfacing of motors, LCD, BLE sensors
- Assembly of BLE beacon
- Progress report

FYP-2

- Graphical User Interface (GUI)
- Tracking algorithm coding and hardware implementation
- LiDAR sensor interfacing
- Obstacle avoidance algorithm coding and interfacing
- Final project report

A project schedule was prepared with help of a planning and management software (Primavera P6). The above milestones were broken down into sub activities for easier monitoring and control. A screenshot of the schedule and the achieved progress by the end of final defense presentation of the project is shown below.

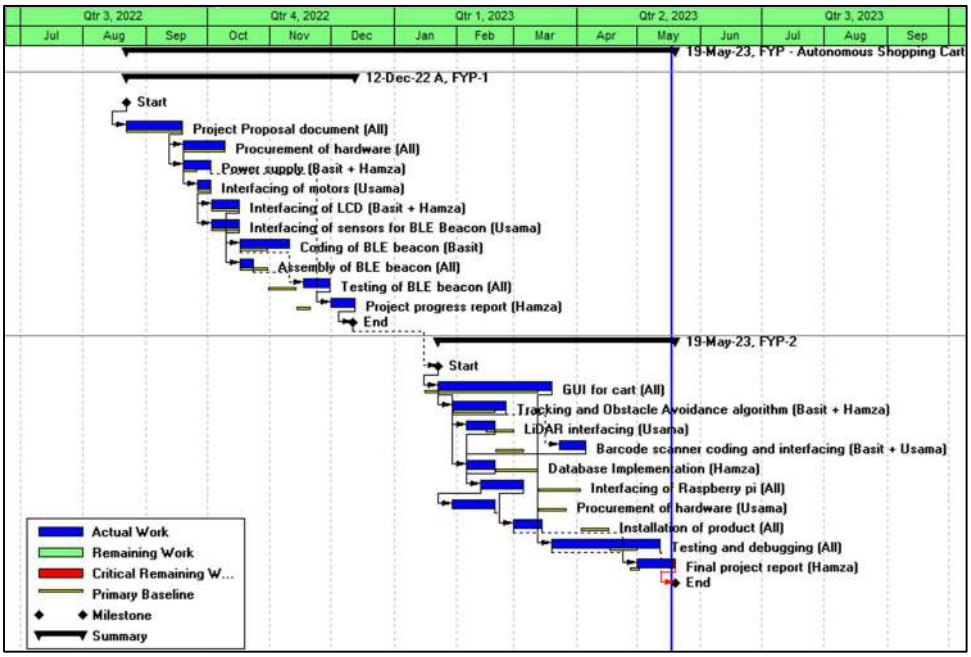

Figure 3.1 Project schedule and achieved progress (19-05-2023)

The work was distributed amongst the three group members for effective working on the project. The group worked on the project on a weekly basis, with dedicated targets for each week. Weekly meetings were also held with the project supervisor for discussion of any issues faced during execution and for guidance. A record of the meetings was maintained in the form of a minutes of meeting document.

For majority of the project, the execution of activities followed the project schedule. For FYP-1, the final testing and progress report were delayed as semester was extended. For FYP-2, some activities such as interfacing of pi, installation of product on shopping cart etc. were executed ahead of schedule by starting work on multiple activities simultaneously. This was necessary to be able to give demonstration of target tracking capabilities before the mid presentation, as requested by the presentation panel.

Overall, the project was completed within the designated period.

3.2 Project Results

Along with physical testing of the product, simulations were also run for the various processes involved in the project. For BLE beacon (target tracking) and LiDAR sensor (obstacle detection), the data from sensors was transmitted to Arduino IDE and verified via the Serial Monitor. For GUI, coding was done in Python using tkinter and simulations were run using Visual Studio Code.

The hardware portion of the project was executed in the FYP lab of EE block. After assembly of each deliverable, extensive testing was done both in lab and varying environments to incorporate multiple scenarios.

Results for the testing showed that the project was working accurately. Details of the testing conducted for various aspects of the project are discussed below.

- **Target Tracking**

Target tracking involves two aspects, distance from the target and target orientation.

Distance to the target was estimated with the help of the strength of the signal received from the BLE beacon by the Raspberry pi. For testing, the BLE beacon was placed at various distances from the shopping cart and these distances were physically measured. Comparisons were made with the estimated distance computed by the tracking algorithm, and errors were minimized by modifying various parameters in the algorithm.

The accuracy of distance estimation of the algorithm can be observed in the following graph.

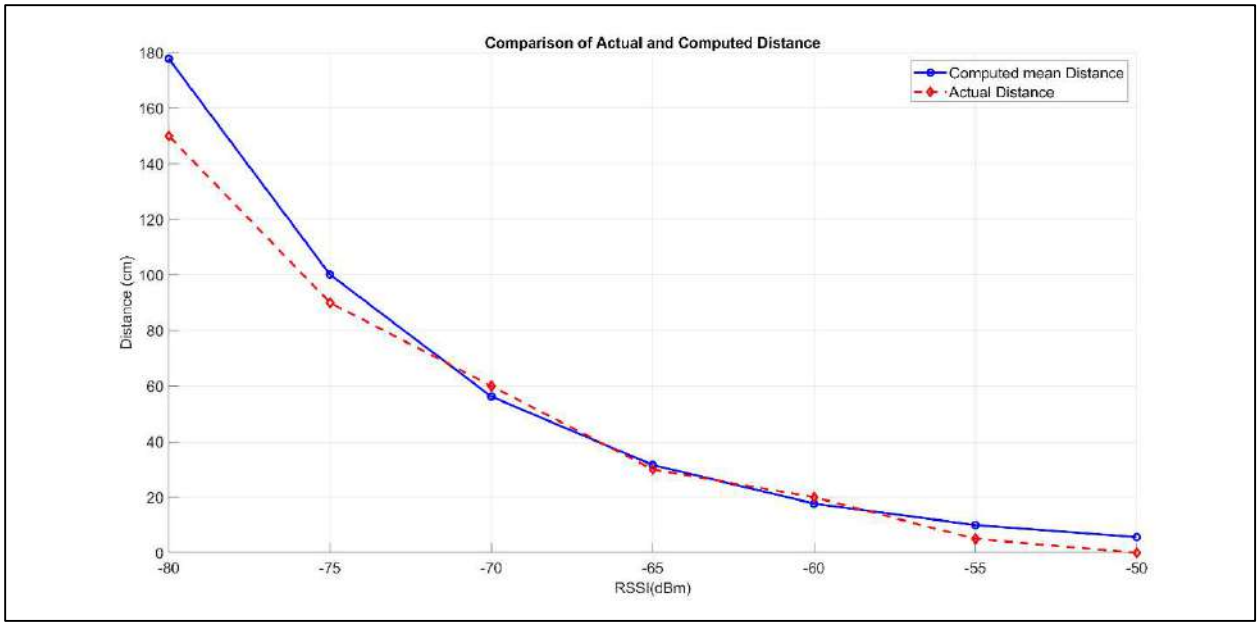

Figure 3.2 Comparison of actual and computed distances

It can be seen from the graph that the least variation is for a signal strength in the range of -70 dBm to -60 dBm, which translates to 20-60 cm from the target.

Next, the effectiveness of the shopping cart in following the target at various distances was tested. Target moved with the beacon to different distances and in various directions from the shopping cart, and it was observed how often the shopping cart followed the target accurately till the threshold value for each distance. Results of the testing are given below.

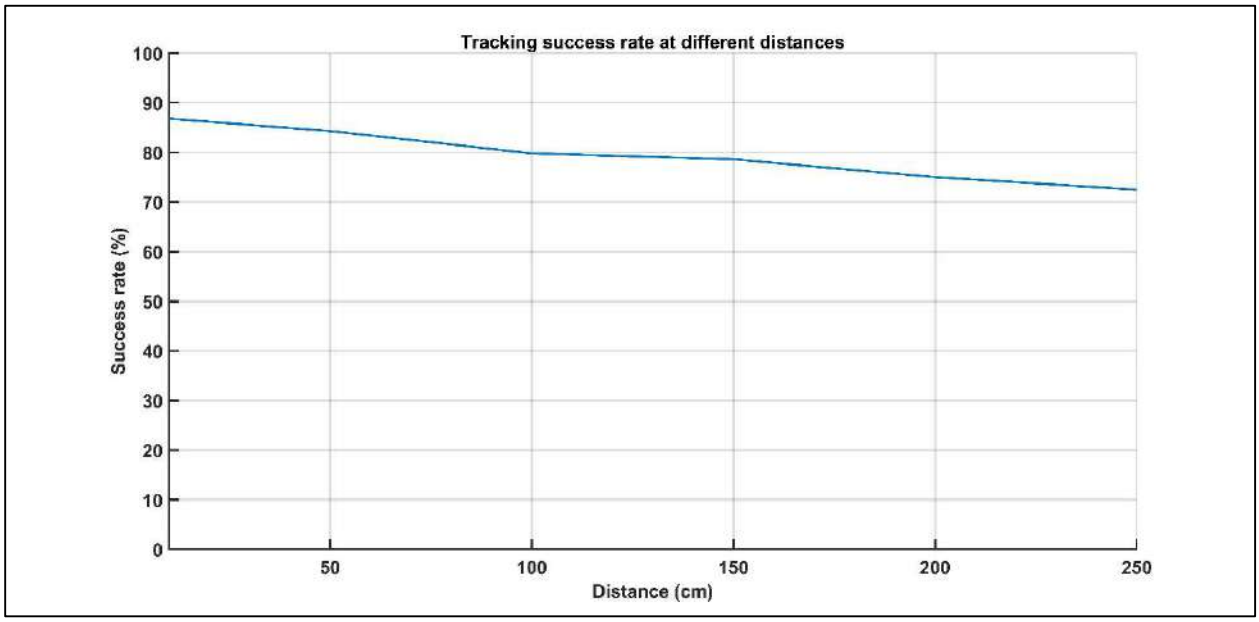

Figure 3.3 Tracking success rate for various distances to the target

It can be observed that tracking success rate falls drastically beyond the 150 cm range. This occurred due to the shopping cart not being able to effectively follow the direction in which the user turned as the data indicating the direction change occurred much sooner than when the cart reached the point of rotation. This can be improved and will be discussed in the following sections.

- **Obstacle avoidance**

For obstacle avoidance, LiDAR sensor provided obstacle distance and direction based on its orientation to the Raspberry pi. The pi then sent control signals to the motor drivers to circumnavigate the obstacle.

To test the accuracy of the LiDAR sensor in detecting obstacles and providing correct distances, extensive testing was done with various objects placed at multiple distances. The received data was compared with actual measurements to determine accuracy of the sensor. Results for the testing are given in the following graph.

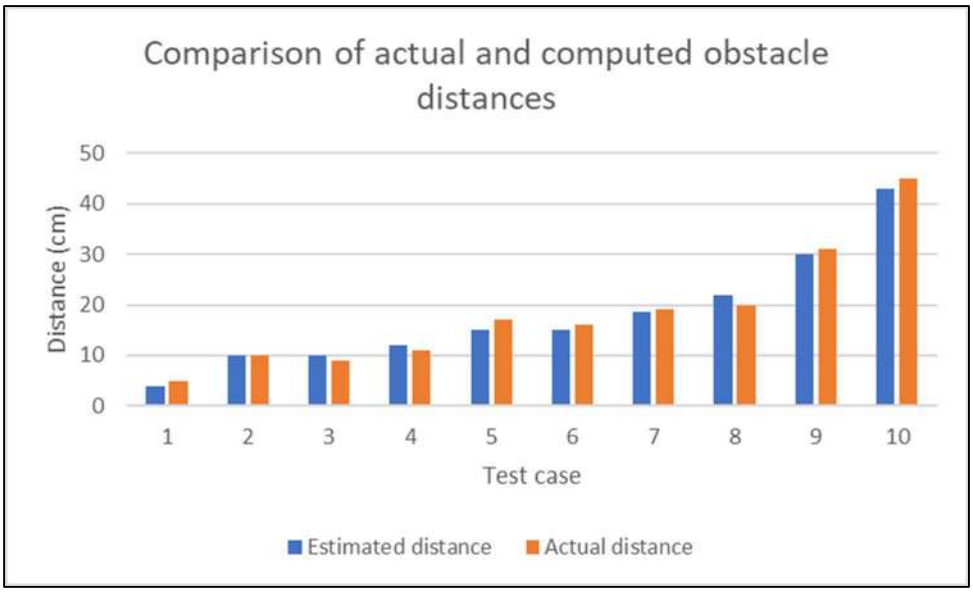

Figure 3.4 Comparison of actual and computed obstacle distances

Each case was tested for multiple obstacles at various orientations. At an average, the LiDAR sensor was able to detect obstacles and provide distances with 82% accuracy for all above test cases (distance ranging from 5 - 50 cm).

Next, the effectiveness of shopping cart in circumnavigating obstacles was to be determined. This was done by placing various obstacles along its path and testing if the cart was able to go around

the obstacle and continue following the target. The results of this testing are summarized by the following graphs.

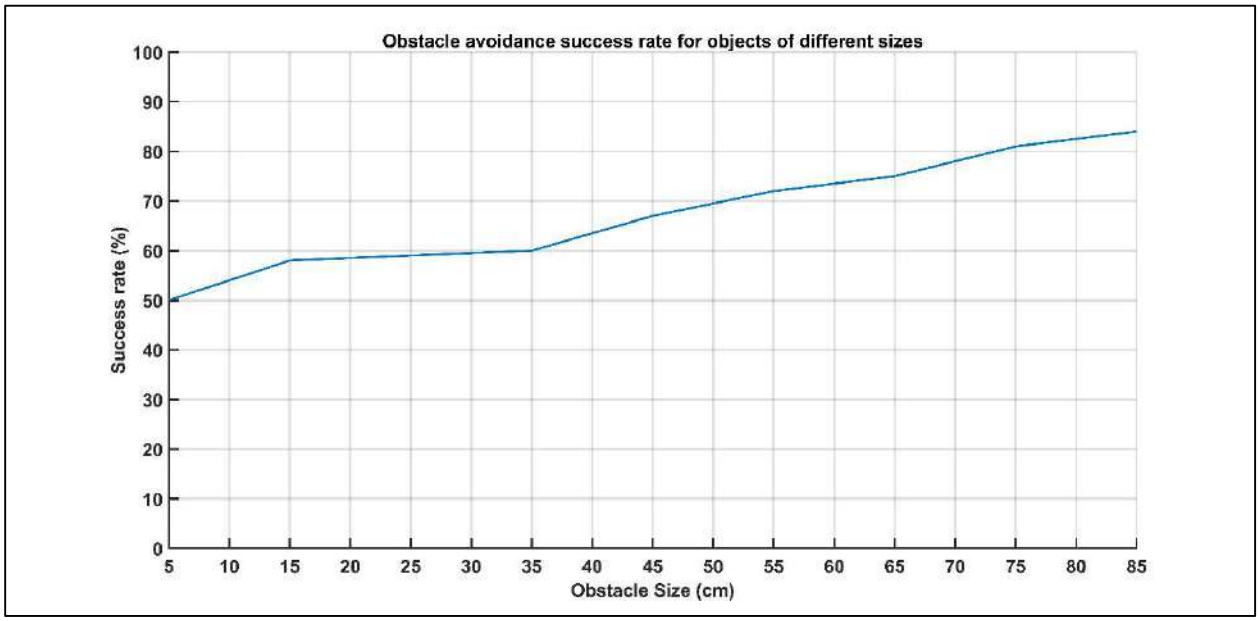

Figure 3.5 Success rate of obstacle avoidance based on size of object

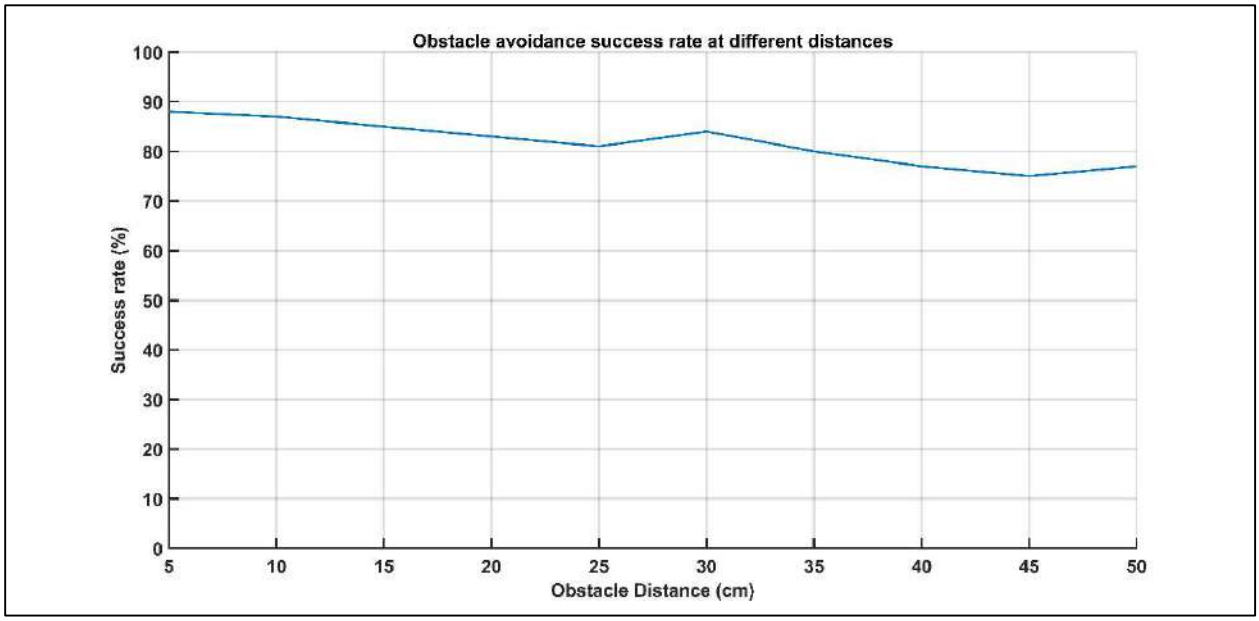

Figure 3.6 Success rate of obstacle avoidance based on distance of object

It can be seen from the above plots that the shopping cart is able to avoid obstacles of larger size more reliably and success rate is more frequent. Size refers to the width of the object. Obstacle avoidance algorithm is most effective for obstacles up to a distance of 30 cm from the shopping cart, beyond which success rate falls. For distances larger than 35 cm, it was observed that the

shopping cart struggled to clear the obstacles and was unable to continue following the target. This again can be improved and will be revisited in the recommendations section.

The testing shows that prototype is in working condition and can operate as per requirements for most conditions.

3.3 Recommendations / Future work

While the prototype is in working condition, there is room for improvement in both its operation as well as the methodology for its functioning. A few possibilities are listed below:

1. The inaccuracies in target tracking and obstacle avoidance can be further minimized by both algorithm tuning and hardware improvements. By incorporation of a feedback system for both processes, the errors in the output will drop. Accuracy can also be enhanced by improving the threshold values used for yaw variations and the delay functions for rotation of the shopping cart.
2. Distance estimation for target tracking is based on the strength of signal received from the beacon. The value of RSSI computed by the Raspberry pi is prone to variation based on environmental factors. The sampling of RSSI may be improved by including a larger number of values in the dataset, and with the use of filters to counteract distortions in the input values.
3. Another solution to the problems faced in target tracking can be the use of triangulation techniques. Triangulation is based on measuring distance of the transmitter from three stations which would allow computation of location based on the coordinates of the known stations. To implement such a process, additional Bluetooth beacons would be required to be placed at fixed locations and distances, such that the shopping cart is within range of at least three beacons at all times.
4. Simultaneous Localization and Mapping (SLAM) is a technique used in robotics to enable a device to build a map of its environment while also determining its own location within the map. It allows the device to localize itself in scenarios where GPS information is not available. Such a methodology may be adopted as an alternative to BLE based tracking.
5. With the current design, the shopping cart moves at a constant speed which is impractical. This can be revisited with use of motors with higher output rating, where speed of the shopping cart will be dependent on the speed of the moving target.
6. The project currently requires use of a pocket device i.e., the BLE beacon for target tracking. Most modern android smartphones come equipped with Bluetooth 4.0 and the required sensors to provide data for yaw computation. The same project can be implemented without a beacon by designing an application which will make use of the user's smartphone for the purpose of tracking.

7. Machine learning can also be used to enhance the project. The functioning of the autonomous shopping cart can be improved by incorporating object recognition, path planning, customer preference analysis, inventory management etc. to provide more optimized performance.

Overall, there are many possibilities for advancing this project which may be explored in the future.

3.4 Conclusions

A working prototype for an autonomous shopping cart was prepared. The final product was in accordance with the project scope, and catered to all design parameters such as load carrying capacity, required features etc. The project involved use of many techniques learnt throughout the engineering program, as well as the acquisition of new knowledge solely for the execution of the project such as GUI design, database implementation, use of Raspberry pi microprocessor etc. The project was completed successfully as per its specifications, within budget and on schedule.

A key aspect of any engineering project is its role and impact on society and the environment. The primary goal of the project was to offer equal opportunities to all and to provide ease in daily life activities. The project maps to three sustainable goals defined in the Sustainable Development Goals set by the United Nations General Assembly:

- SDG 08: Decent work and economic growth
- SDG 09: Industry, innovation and infrastructure
- SDG 11: Sustainable cities and communities

Statistical analysis shows improved sales numbers with use of automated shopping cart. The project aims to contribute to the economic growth of businesses, as well as creating jobs for skilled staff.

The retail industry is one of the oldest functioning sectors of the market. Our product provides an innovative feature for physical retailers that will result in growth of industry and have long lasting impact on infrastructure.

The project also emphasizes inclusiveness, which will lead to growth and development of society. It will provide safety and resilience to the marketplaces deploying this technology by reducing the chances of accident and injury.

The Autonomous Shopping Cart is equipped with several user-friendly features which make it an attractive product for both retailers and customers. The use of the product will not only facilitate the customer by providing ease of service but also benefit the retailer in terms of increased sales, reduction of staff for assistance, etc. The project also aims to achieve various sustainability goals which lead to improved industry and community environment. The successful completion of the project shows potential to develop it further for marketing.

Appendix-A Project Codes

- **A-1 Arduino code for BLE beacon**

```
#include "MPU9250.h"
#include <SoftwareSerial.h>
SoftwareSerial BLE(8,9);

double comp_pitch, comp_roll, comp_yaw;
String data="";
MPU9250 mpu;

void setup() {
  Serial.begin(9600);
  Wire.begin();
  BLE.begin(9600);

  if (!mpu.setup(0x68)) {
    while (1) {
      Serial.println("MPU connection failed. Please check your connection with `connection_check` example.");
      delay(5000);
    }
  }
}

void loop() {
  if (mpu.update()) {
    static uint32_t prev_ms = millis();
    if (millis() > prev_ms + 25) {
      comp_pitch = mpu.getPitch()*3.1415/180;
      comp_roll = mpu.getRoll()*3.1415/180;
      comp_yaw = mpu.getYaw()*3.1415/180;
      data += "|" + String(comp_roll,2)+ "|" +String(comp_pitch,2)+ "|" + String(comp_yaw,2) + "|";

      Serial.println(data);
      BLE.print(data);
      data="";
      delay(150);
      prev_ms = millis();
    }
  }
```

```
}

void print_roll_pitch_yaw() {
  Serial.print("Yaw, Pitch, Roll: ");
  Serial.print(mpu.getYaw(), 2);
  Serial.print(", ");
  Serial.print(mpu.getPitch(), 2);
  Serial.print(", ");
  Serial.println(mpu.getRoll(), 2);
}

void print_calibration() {
  Serial.println("< calibration parameters >");
  Serial.println("accel bias [g]: ");
  Serial.print(mpu.getAccBiasX() * 1000.f / (float)MPU9250::CALIB_ACCEL_SENSITIVITY);
  Serial.print(", ");
  Serial.print(mpu.getAccBiasY() * 1000.f / (float)MPU9250::CALIB_ACCEL_SENSITIVITY);
  Serial.print(", ");
  Serial.print(mpu.getAccBiasZ() * 1000.f / (float)MPU9250::CALIB_ACCEL_SENSITIVITY);
  Serial.println();
  Serial.println("gyro bias [deg/s]: ");
  Serial.print(mpu.getGyroBiasX() / (float)MPU9250::CALIB_GYRO_SENSITIVITY);
  Serial.print(", ");
  Serial.print(mpu.getGyroBiasY() / (float)MPU9250::CALIB_GYRO_SENSITIVITY);
  Serial.print(", ");
  Serial.print(mpu.getGyroBiasZ() / (float)MPU9250::CALIB_GYRO_SENSITIVITY);
  Serial.println();
  Serial.println("mag bias [mG]: ");
  Serial.print(mpu.getMagBiasX());
  Serial.print(", ");
  Serial.print(mpu.getMagBiasY());
  Serial.print(", ");
  Serial.print(mpu.getMagBiasZ());
  Serial.println();
  Serial.println("mag scale []: ");
  Serial.print(mpu.getMagScaleX());
  Serial.print(", ");
  Serial.print(mpu.getMagScaleY());
  Serial.print(", ");
  Serial.print(mpu.getMagScaleZ());
  Serial.println();
}
```

- **A-2 Python code for receiving data from BLE beacon**

```
import sys
import time
import asyncio
import platform
import re
from gpiozero import LED

from bleak import BleakClient
from bleak.backends.characteristic import BleakGATTCharacteristic

global yaw
global avg_distance

yaw = 0
Left = 0
Right = 0
Forward = 0
avg_distance = 2000

CHARACTERISTIC_UUID = "0000ffe1-0000-1000-8000-00805f9b34fb"
ADDRESS = (
   "d0:39:72:b4:5c:16"
   if platform.system() != "Darwin"
   else "00002a24-0000-1000-8000-00805f9b34fb"
)

def notification_handler(characteristic: BleakGATTCharacteristic, data:bytearray):

   global yaw
   global avg_distance

   avg_distance
   msg=data.decode("utf-8")
   x=re.findall("[+-]?\d+\.\d+",msg)
   tem = float(x[2]) * 57.3
   diff= tem - yaw
   print("\nYaw_Difference= " + str(diff))

   with open('shared_data.txt', 'r') as f:
      avg_distance = float(f.read())
```

```
    if avg_distance < 45:
        with open('shared_data1.txt', 'w') as f:
         f.write("Stop")
        print("\nStop")
    elif diff <= -18:
        with open('shared_data1.txt', 'w') as f:

         f.write("Left")

        print("\nLeft turn")

    elif diff >= 18:
        with open('shared_data1.txt', 'w') as f:
         f.write("Right")

        print("\nRight turn")

    else:
        with open('shared_data1.txt', 'w') as f:
         f.write("Forward")
        print("\nForward")

    yaw = tem

async def main(address, char_uuid):
    async with BleakClient(address) as client:
        print(f"Connected: {client.is_connected}")

        await client.start_notify(char_uuid, notification_handler)
        await asyncio.sleep(7200.0)
        await client.stop_notify(char_uuid)

if __name__ == "__main__":
    asyncio.run(
        main(
            sys.argv[1] if len(sys.argv) > 1 else ADDRESS,
            sys.argv[2] if len(sys.argv) > 2 else CHARACTERISTIC_UUID,
        )
    )
```

- **A-3 Python code for computing RSSI**

```
import time
import math
from gpiozero import LED
from time import sleep
from bluepy.btle import Scanner, DefaultDelegate

global avg_distance
class ScanDelegate(DefaultDelegate):
   def __init__(self):
      DefaultDelegate.__init__(self)

   def HandleDiscovery(self,dev,new_dev,new_dat):
      if new_dev:
         pass
      if new_dat:
         pass

scanner = Scanner().withDelegate(ScanDelegate())

while(1):
   sum = 0
   n = 0
   a = 6
   while a > 0:
      try:
         devices = scanner.scan(0.1)
         for ii in devices:
            if ii.addr == 'c4:be:84:22:99:22' :
               ratio = 10 ** ((-75 - ii.rssi) / (10 * 2))
               dis = float(math.sqrt(ratio)*100)
               print("Device %s, RSSI=%d dB, Distance=%f cm" % (ii.addr,ii.rssi,dis))
               sum = sum + dis
               n = n + 1
      except:
         continue

      a = a - 1

   if n == 0:
      print('No Device Connected')
      continue
   avg_distance = sum/n
```

```
    with open('shared_data.txt', 'w') as f:
        f.write(str(avg_distance))
    print('Average = %d cm' % avg_distance)
```

- **A-4 Arduino code for LiDAR sensor**

```
#include <Wire.h>
#include <LIDARLite.h>
#include <Servo.h>
Servo radarServo;
LIDARLite myLidarLite;

void setup()
{
  Serial.begin(115200);
  delay(20);
  radarServo.attach(5);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);

  myLidarLite.begin(0, true);
  myLidarLite.configure(0);
}

void loop()
{
  int distance = 0;
  Serial.println(myLidarLite.distance());

for(int i=0;i<=180;i=i+3)
  {
       digitalWrite(6, LOW);
       digitalWrite(7, LOW);
       digitalWrite(8, LOW);
    radarServo.write(i);
    delay(1);
    distance = myLidarLite.distance();
    if(distance)
    {
      Serial.print(distance);
      Serial.print("cm\t");
      if (distance <=75)
```

```
    {
     if (i < 75){
      Serial.print("Right Obstacle");
      digitalWrite(6, LOW);
      digitalWrite(7, HIGH);
      digitalWrite(8, LOW);
      }
     else if (i > 105){
      Serial.print("Left Obstacle");
      digitalWrite(6, HIGH);
      digitalWrite(7, LOW);
      digitalWrite(8, LOW);
      }
     else{
      digitalWrite(6, LOW);
      digitalWrite(7, LOW);
      digitalWrite(8, HIGH);
      Serial.print("Stop Front Obstacle");
      }
     while(distance <=75){
      distance = myLidarLite.distance();
      }
    }
}
}
for(int i=180;i>=0;i=i-3)
 {
    digitalWrite(6, LOW);
    digitalWrite(7, LOW);
    digitalWrite(8, LOW);
 radarServo.write(i);
 delay(1);
 distance = myLidarLite.distance();
 if(distance)
 {
  Serial.print(distance);
  Serial.print("cm\t");
  if (distance <=75)
  {
    if (i < 75){
      Serial.print("Right Obstacle");
      digitalWrite(6, LOW);
      digitalWrite(7, HIGH);
      digitalWrite(8, LOW);
```

```
        }
       else if (i > 105){
         Serial.print("Left Obstacle");
         digitalWrite(6, HIGH);
         digitalWrite(7, LOW);
         digitalWrite(8, LOW);
         }
       else{
         digitalWrite(6, LOW);
         digitalWrite(7, LOW);
         digitalWrite(8, HIGH);
         Serial.print("Stop Front Obstacle");
         }
       while(distance <=75){
         distance = myLidarLite.distance();
         }
     }
  }
  }
}
```

- **A-5 Python code for driving motors**

```
import RPi.GPIO as GPIO
import time

global Left
global Right
global Forward
global Ob

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)


L = 19;  # GPIO pin 19 to the RPWM on the BTS7960
R = 26;  # GPIO pin 26 to the RPWM on the BTS7960

# For enabling "Left" and "Right" movement
L_EN = 20;  # connect GPIO pin 20 to L_EN on the BTS7960
R_EN = 21;  # connect GPIO pin 21 to R_EN on the BTS7960

GPIO.setup(R, GPIO.OUT)
```

```
GPIO.setup(L, GPIO.OUT)
GPIO.setup(L_EN, GPIO.OUT)
GPIO.setup(R_EN, GPIO.OUT)
GPIO.setup(2, GPIO.IN)
GPIO.setup(3, GPIO.IN)
GPIO.setup(18, GPIO.IN)

# Enable "Left" and "Right" movement on the HBRidge
GPIO.output(R_EN, True)
GPIO.output(L_EN, True)
Ob=0

while(1):
   Lm = GPIO.input(2)
   Rm = GPIO.input(3)
   S = GPIO.input(18)
   if (Lm == 0 and Rm == 0 and S == 1):
      print("\nStop")
      GPIO.output(R, False)
      GPIO.output(L, False)
   elif (Lm == 1 and Rm == 0 and S == 0):
      print("\nRight turn")
      GPIO.output(R, False)
      GPIO.output(L, True)
   elif (Lm == 0 and Rm == 1 and S == 0):
      print("\nLeft turn")
      GPIO.output(R, True)
      GPIO.output(L, False)
   else:
      with open('shared_data1.txt', 'r') as f:
         Status = f.read()
      if (Status == "Stop" and Ob == 0):
         print("\nStop")
         GPIO.output(R, False)
         GPIO.output(L, False)
      elif (Status == "Right" and Ob == 0):
         print("\nRight turn")
         GPIO.output(R, False)
         GPIO.output(L, True)
         time.sleep(0.75)
      elif (Status == "Left" and Ob == 0):
         print("\nLeft turn")
         GPIO.output(R, True)
         GPIO.output(L, False)
```

```
        time.sleep(0.75)
    elif (Status == "Forward" and Ob == 0):
        print("\nForward")
        GPIO.output(R, True)
        GPIO.output(L, True)
    else:
        print("\nStop")
        GPIO.output(R, False)
        GPIO.output(L, False)
```

- **A-6 Python code to generate database**

```
from tkinter import *
from PIL import ImageTk, Image
import sqlite3
import csv

#Add Function
def submit_f():
    file = open('Store_Database.csv')
    csvreader = csv.reader(file)

    rows = []
    for row in csvreader:
        rows.append(row)

    conn = sqlite3.connect('data.db')
    c = conn.cursor()

    for x in rows:
        c.execute("INSERT INTO DATA VALUES (:item, :category, :rate, :section, :shelf, :bar)",
        {
            'item': x[0],
            'category': x[1],
            'rate': x[2],
            'section': x[3],
            'shelf':  x[4],
            'bar': x[5]
        })

    conn.commit()
    conn.close()
```

```
root = Tk()
root.title("Database")

Button(root,text="Read data from spreadsheet and generate database file", command= submit_f).grid(row=1, column= 1,pady=10)

#Create database
conn = sqlite3.connect('data.db')

c = conn.cursor()

c.execute("""CREATE TABLE DATA (
    item_name text,
    item_category text,
    item_rate integer,
    item_section integer,
    item_shelf integer,
    item_bar text
    )
    """)

root.mainloop()
```

- **A-7 Python code for Graphical User Interface**

```
from tkinter import *
from PIL import ImageTk, Image
from tkinter import messagebox
import sqlite3
import customtkinter

customtkinter.set_appearance_mode("light")
customtkinter.set_default_color_theme("blue")
root = customtkinter.CTk()
root.title("GUI")
root.attributes("-fullscreen",True)

## Variables
global row_ref, total, count, rate, quantity, amount, bg_color, fg_color, check, choice, bill
choice = StringVar()
check = 0
bill = []
quantity = 1
```

```
count = 1
row_ref = 4
total = 0
x = "White"
y = "Black"
bg_color = x
fg_color = y

## Functions
def add():
	global total
	total = 0

	for items in bill:
		total = total + items[3]

def map_btn():
	global x
	w3 = Toplevel()
	w3.title("Map")
	w3.attributes("-fullscreen",True)

	x = ImageTk.PhotoImage(Image.open("Mall.png"))
	xx = Label(w3, image=x)
	xx.grid(row=0)
	customtkinter.CTkButton(w3, text="Exit", command= w3.destroy,
width=50,height=30,corner_radius=10,fg_color="red").grid(row=1)

def cat_reset():
	global check
	check = 0
	cat_btn()

def cat_btn():
	global x1,x2,x3,x4, w1, check
	if (check == 0):
		w1 = Toplevel()
		w1.title("Categories")
		w1.attributes("-fullscreen",True)

	w1.config(bg=bg_color)
	for widgets in w1.winfo_children():
		widgets.destroy()
```

```
    Label(w1, text="", fg=fg_color, bg=bg_color).grid(row=0,column=4, sticky=W+E)

    x1 =ImageTk.PhotoImage(Image.open("G1.png"))
    Button(w1, image= x1, command= lambda:sub_cat_f("Grocery")).grid(row=1,column=0)
    Label(w1, text="", fg=fg_color, bg=bg_color).grid(row=1,column=1)
    Button(w1, text="Groceries", fg=fg_color, bg=bg_color, relief = FLAT, command= lambda:sub_cat_f("Grocery")).grid(row=2,column=0)
    Label(w1,text="General range of food products, which may be fresh or packaged.", fg=fg_color, bg=bg_color, anchor=W).grid(row=1,column=2, sticky=W+E)

    x2 =ImageTk.PhotoImage(Image.open("G2.png"))
    Button(w1, image=x2, command= lambda:sub_cat_f("Meat")).grid(row=4,column=0)
    Label(w1, text="", fg=fg_color, bg=bg_color).grid(row=4,column=1)
    Button(w1, text="Meat", fg=fg_color, bg=bg_color, relief = FLAT, command= lambda:sub_cat_f("Meat")).grid(row=5,column=0)
    Label(w1, text="Chicken, turkey, beef and lean meat based products", fg=fg_color, bg=bg_color, anchor=W).grid(row=4,column=2, sticky=W+E)

    x3 =ImageTk.PhotoImage(Image.open("G3.png"))
    Button(w1, image=x3, command= lambda:sub_cat_f("Dairy")).grid(row=6,column=0)
    Label(w1, text="", fg=fg_color, bg=bg_color).grid(row=6,column=1)
    Button(w1, text="Dairy", fg=fg_color, bg=bg_color, relief = FLAT, command= lambda:sub_cat_f("Dairy")).grid(row=7,column=0)
    Label(w1, text="Milk and milk based products such as yoghurt, cheese, butter etc.", fg=fg_color, bg=bg_color, anchor=W).grid(row=6,column=2, sticky=W+E)

    x4 =ImageTk.PhotoImage(Image.open("G4.png"))
    Button(w1, image=x4, command= lambda:sub_cat_f("Seafood")).grid(row=8,column=0)
    Label(w1, text="", fg=fg_color, bg=bg_color).grid(row=8,column=1)
    Button(w1, text="Seafood", fg=fg_color, bg=bg_color, relief = FLAT, command= lambda:sub_cat_f("Seafood")).grid(row=9,column=0)
    Label(w1, text="Variety of fish and crustacean products", fg=fg_color, bg=bg_color, anchor=W).grid(row=8,column=2, sticky=W+E)

    Button(w1, text = "Exit", command= w1.destroy, padx=5, pady=5 ).grid(row=0,column=2, sticky=E)

def sub_cat_f(name):
    global check
    check = 1
    w1.config(bg=bg_color)
    for widgets in w1.winfo_children():
```

```
        widgets.destroy()

    Label(w1,text=name, fg=fg_color, bg=bg_color).grid(row=0,column=0, columnspan=
5)
    Button(w1, text="Back", fg=fg_color, bg=bg_color, command=
cat_btn).grid(row=0,column=5)

    conn = sqlite3.connect('data.db')
    c = conn.cursor()
    c.execute("SELECT * from DATA WHERE item_category = ?", [str.upper(name)])

    records = c.fetchall()

    Label(w1,text = "Sr. No", bg=bg_color, fg=fg_color).grid(row=1, column=0)
    Label(w1,text = "Description", bg=bg_color, fg=fg_color).grid(row=1, column=1)
    Label(w1,text = "Unit Price", bg=bg_color, fg=fg_color).grid(row=1, column=2)
    Label(w1,text = "Location", bg=bg_color, fg=fg_color).grid(row=1, column=3)

    r = 2
    number = 1
    for record in records:
        Label(w1,text = str(number), bg=bg_color, fg=fg_color).grid(row=r, column=0,
sticky=W+E)
        Button(w1,text = str(record[0]), anchor=W, bg=bg_color, fg=fg_color,
relief=FLAT,command=lambda button_text =str(record[0]):
searchItem_f(button_text)).grid(row=r, column=1, sticky=W+E)
        Label(w1,text = str(record[2]), bg=bg_color, fg=fg_color).grid(row=r, column=2,
sticky=W+E)
        Label(w1,text = "   Section " + str(record[3]) + " Shelf " +str(record[4]), anchor=W,
bg=bg_color, fg=fg_color).grid(row=r, column=3, sticky=W+E)
        r = r+1
        number = number+1

def add_f(name, qty):
    global row_ref, total, count, rate, quantity, amount, bg_color, fg_color, bill
    quantity= qty
    amount = rate*int(qty)

    temp = [name, qty, rate, amount]
    bill.append(temp)
    add()
    print_f()

def searchItem_f(x):
```

```
    conn = sqlite3.connect('data.db')
    c = conn.cursor()

    c.execute("SELECT * from DATA WHERE item_name = ?", [str.upper(x)])
    data = c.fetchone()

    if data is None:
        response = messagebox.showerror("Error","No results found.")
    else:
        global row_ref, total, count, rate, quantity, amount, pic
        w2 = Toplevel()
        w2.title("Item Details")
        w2.geometry("500x400")
        w2.config(bg=bg_color)

        Label(w2,text="", bg=bg_color, fg=fg_color).grid(row=1,column=0)

        Label(w2,text= "Item: " + str.capitalize(data[0]), anchor=W, bg=bg_color,
fg=fg_color).grid(sticky=W+E,row=3,column=0,columnspan=3)
        Label(w2,text= "Category: " + str.capitalize(data[1]), anchor=W, bg=bg_color,
fg=fg_color).grid(sticky=W+E,row=4,column=0,columnspan=3)
        Label(w2,text= "Unit price: " + str(data[2]), anchor=W, bg=bg_color,
fg=fg_color).grid(sticky=W+E,row=5,column=0,columnspan=3)
        Label(w2,text= "Location: Section " + str(data[3]) + " Shelf " + str(data[4]),
anchor=W, bg=bg_color, fg=fg_color).grid(sticky=W+E,row=6,column=0,columnspan=3)

        rate = float(data[2])

        Label(w2,text="Quantity: ",anchor=W, bg=bg_color,
fg=fg_color).grid(sticky=W+E,row=7,column=0)
        qty = Entry(w2, width=5, bg=bg_color, fg=fg_color)
        qty.insert(0,"1")
        qty.grid(row=7,column=1)

        Label(w2,text=" ", bg=bg_color, fg=fg_color).grid(row=7,column=2)
        Button(w2,text="Add to cart", bg=bg_color, fg=fg_color, command=lambda:
[add_f(data[0], qty.get()), w2.destroy()]).grid(sticky=W+E,row=7,column=3)
        Label(w2,text="  ", bg=bg_color, fg=fg_color).grid(row=7,column=4)

        pic = ImageTk.PhotoImage(Image.open("Images\Placeholder.JPG"))
        Label(w2,image = pic).grid(row=2,column=5, rowspan=10)

def searchbar_f():
    s = str.upper(search_bar.get())
```

```
    conn = sqlite3.connect('data.db')
    c = conn.cursor()

    t = ('%'+s+'%',)
    c.execute('select * from DATA where item_name like ?', t)

    records = c.fetchall()

    if len(records) == 0:
        response = messagebox.showerror("Error","No results found.")
    else:
        w4 = Toplevel()
        w4.title("Search Results")
        w4.config(bg=bg_color)
        w4.geometry("500x400")

        Label(w4,text = "Sr. No", bg=bg_color, fg=fg_color).grid(row=1, column=0)
        Label(w4,text = "Description", bg=bg_color, fg=fg_color).grid(row=1, column=1)
        Label(w4,text = "Unit Price", bg=bg_color, fg=fg_color).grid(row=1, column=2)
        Label(w4,text = "Location", bg=bg_color, fg=fg_color).grid(row=1, column=3)

        r = 2
        number = 1
        for record in records:
            Label(w4,text = str(number), bg=bg_color, fg=fg_color).grid(row=r,
column=0, sticky=W+E)
            Button(w4,text = str(record[0]), anchor=W, bg=bg_color, fg=fg_color,
relief=FLAT,command=lambda button_text =str(record[0]):
searchItem_f(button_text)).grid(row=r, column=1, sticky=W+E)
            Label(w4,text = str(record[2]), bg=bg_color, fg=fg_color).grid(row=r,
column=2, sticky=W+E)
            Label(w4,text = "    Section " + str(record[3]) + " Shelf " +str(record[4]),
anchor=W, bg=bg_color, fg=fg_color).grid(row=r, column=3, sticky=W+E)
            r = r+1
            number = number+1

def check_length(event):
    global w2
    if len(search_bar.get()) > 8:
        searchbarcode_f()

def searchbarcode_f():
    global w2
    s = str(search_bar.get())
```

```
    print(s)
    conn = sqlite3.connect('data.db')
    c = conn.cursor()

    c.execute("SELECT * from DATA WHERE item_bar = ?", [s])
    data = c.fetchone()

    search_bar.delete(0, 'end')

    if data is None:
        response = messagebox.showerror("Error","No results found.")
    else:
        global row_ref, total, count, rate, quantity, amount, pic, check, w2

        w2 = Toplevel()
        w2.title("Item Details")
        w2.attributes("-fullscreen",True)

        w2.config(bg=bg_color)

        Label(w2,text="", bg=bg_color, fg=fg_color).grid(row=1,column=0)

        Label(w2,text= "Item: " + str.capitalize(data[0]), anchor=W, bg=bg_color,
fg=fg_color).grid(sticky=W+E,row=3,column=0,columnspan=3)
        Label(w2,text= "Category: " + str.capitalize(data[1]), anchor=W, bg=bg_color,
fg=fg_color).grid(sticky=W+E,row=4,column=0,columnspan=3)
        Label(w2,text= "Unit price: " + str(data[2]), anchor=W, bg=bg_color,
fg=fg_color).grid(sticky=W+E,row=5,column=0,columnspan=3)
        Label(w2,text= "Location: Section " + str(data[3]) + " Shelf " + str(data[4]),
anchor=W, bg=bg_color, fg=fg_color).grid(sticky=W+E,row=6,column=0,columnspan=3)

        rate = float(data[2])

        Label(w2,text="Quantity: ",anchor=W, bg=bg_color,
fg=fg_color).grid(sticky=W+E,row=7,column=0)
        qty = Entry(w2, width=5, bg=bg_color, fg=fg_color)
        qty.insert(0,"1")
        qty.grid(row=7,column=1)

        Label(w2,text=" ", bg=bg_color, fg=fg_color).grid(row=7,column=2)
        Button(w2,text="Add to cart", bg=bg_color, fg=fg_color, command=lambda:
[add_f(data[0], qty.get()), w2.destroy()]).grid(sticky=W+E,row=7,column=3)
        Label(w2,text="  ", bg=bg_color, fg=fg_color).grid(row=7,column=4)
```

```
        Button(w2,text="Exit", bg=bg_color, fg=fg_color, command=
w2.destroy).grid(sticky=E,row=0,column=3)

def print_f():
    global row_ref, total, count, rate, quantity, amount, bg_color, fg_color

    for temp in bill:
        Label(frame, text= str(count), bg= bg_color,
fg=fg_color,width=5).grid(sticky=W+E,row=row_ref,column=0, columnspan=2)
        Label(frame, text= "                    " + str.capitalize(temp[0]), bg= bg_color,
fg=fg_color, anchor=W, width=50).grid(sticky=W+E,row=row_ref,column=2,
columnspan=5)

        Label(frame, text=str(temp[1]), bg= bg_color, fg=fg_color, width=10,
anchor=W).grid(sticky=W+E,row=row_ref,column=7, columnspan=2)

        Label(frame, text=str(temp[2]), bg= bg_color, fg=fg_color,
width=12).grid(sticky=W+E,row=row_ref,column=9, columnspan=2)

        Label(frame, text=str(round(temp[3],3)), bg= bg_color, fg=fg_color, width= 16,
anchor = E).grid(sticky=W+E,row=row_ref,column=11)
        row_ref = row_ref + 1
        count = count + 1
        Label(frame, text="Total: ", bg= bg_color, fg=fg_color, anchor =
E).grid(row=row_ref,column=10, sticky=W+E)
        Label(frame, text= "$ " + str(round(total,3)), bg= bg_color,
fg=fg_color,relief=SOLID, anchor = E).grid(sticky=W+E,row=row_ref,column=11)
    row_ref= 4
    count = 1

def autofill(x):
    rec = Listbox(root,width=103)
    rec.grid(row=2,column=2, columnspan=5,rowspan=5)

    if len(search_bar.get()) == 0:
        for widgets in root.winfo_children():
            widgets.destroy()

def edit_btn():
    global bill, row_ref

    for items in bill:
        Button(frame, text="x", fg=fg_color, bg= bg_color, command=lambda
num=row_ref: delete_f(num)).grid(row=row_ref,column=20, pady=4,padx=2)
```

```
            row_ref = row_ref + 1
        row_ref = 4

def delete_f(x):
        global bill, choice

        bill.pop(x-4)
        add()

        for widgets in frame.winfo_children():
            widgets.destroy()

        print_f()

def theme(choice):
        global row_ref, total, count, rate, quantity, amount, bg_color, fg_color, search_bar

        if choice == "Light":
            root.configure(bg= x)
            frame.config(bg=x)
            bg_color = x
            customtkinter.set_appearance_mode("light")
            fg_color = "BLACK"
        if choice == "Dark":
            root.configure(bg=y)
            frame.config(bg=y)
            bg_color = y
            customtkinter.set_appearance_mode("dark")
            fg_color = "WHITE"

        customtkinter.CTkButton(root, text="Search by category",
command=cat_reset,height=25, width=25).grid(row=0, column=4, padx=12)
        customtkinter.CTkButton(root, text="Map", command=map_btn,height= 25,
width=25).grid(row=0, column=5, padx=10)
        customtkinter.CTkButton(root, text="Edit cart", command=edit_btn,height= 25,
width=25).grid(row=0, column=6, padx=10)
        GUI_theme.config(bg=bg_color, fg=fg_color, highlightbackground=bg_color)
        customtkinter.CTkButton(root, text="Exit", command= root.quit,
width=30,height=30,corner_radius=10,fg_color="red").grid(row=0,column=10)
        search_lbl = customtkinter.CTkLabel(root, text = "   Search:").grid(row=1,column=1)

        search_btn = customtkinter.CTkButton(root, text="Go", command=
searchbar_f,height= 25, width=50).grid(row=1, column=7, columnspan = 3)
```

```
    Label(root, text="SHOPPING CART", fg="WHITE", bg="BLACK").grid(row=2, column=1,
columnspan= 20, sticky=W+E)

    print_f()

#########################################################################
#####################################

## Shopping Cart UI
root.configure(bg= bg_color)

frame = LabelFrame(root)
frame.grid(row=row_ref,column=1, columnspan=11, sticky=W+E)
frame.config(bg=bg_color,fg=fg_color)

search_bar = customtkinter.CTkEntry(root,width = 590,height= 10)
search_bar.grid(row=1,column=2, columnspan=5, padx=5)
search_bar.bind('<KeyRelease>', check_length)
root.after(100, search_bar.focus)

choice.set("Light")
GUI_theme = OptionMenu(root, choice, "Light", "Dark", command=theme)
GUI_theme.config(bg=bg_color, fg=fg_color, highlightbackground=bg_color)
GUI_theme["menu"].config(bg=bg_color, fg=fg_color)
GUI_theme.grid(row=0,column= 7)
theme(choice.get())

# Row 3
Label(root, text="Sr. No", bg="GREY").grid(sticky=W+E,row=3,column=1)
Label(root,                text="Item                Description",
bg="GREY").grid(sticky=W+E,row=3,column=2,columnspan=3)
Label(root, text="Quantity", bg="GREY").grid(sticky=W+E,row=3,column=5)
Label(root, text="Rate", bg="GREY").grid(sticky=W+E,row=3,column=6)
Label(root, text="", bg="GREY").grid(sticky=W+E,row=3,column=7)
Label(root,    text="Amount          ",    bg="GREY").grid(sticky=W+E,row=3,column=8,
columnspan=3)

root.mainloop()
```

Appendix-B Hardware Components

- **B-1 Lists of Hardware components**

Sr. No.	Hardware description
1	MPU 9250 sensor
2	HM 10 Bluetooth module
3	ATmega328 pro mini
4	3.7 V LiPo 1000mAh battery
5	3.7 V LiPo battery charging module

Table B-1 Hardware components for BLE beacon

Sr. No.	Hardware description
1	Cart Chassis
2	LiDAR sensor (Lite V3)
3	MG90S servomotor
4	ATmega328 pro mini
5	LCD touchscreen
6	Barcode scanner
7	Raspberry Pi 4 Model B 8gb RAM
8	LM 2596 buck converters
9	24 V 20Ah $LiFePO_4$ Battery
10	BMS charging module
11	Battery level indicator
12	15 A fuse
13	BTS 7960 motor drivers
14	High torque DC motors
15	Wheels
16	Balance of system (wires, block terminals, USB ports, header pins etc.)

Table B-2 Hardware components for Autonomous Shopping Cart

- **B-2 Datasheets for various components**

Sr. No.	Attribute	Type / Value
1	Type of sensors	Accelerometer, gyroscope, magnetometer
2	Power consumption	16 mW
3	Number of channels	9
4	Sampling rate	50 Hz

Table B-3 Datasheet for MPU 9250 sensor

Sr. No.	Attribute	Type / Value
1	Voltage supply	3.6 – 6 V
2	RF power	23dbm, -6dbm, 0dbm, 6dbm
3	Default baud rate	9600
4	Maximum current	50 mA
5	Frequency	2.4 GHz

Table B-4 Datasheet for HM 10 Bluetooth module

Sr. No.	Attribute	Type / Value
1	Operating voltage	3.3 – 12 V
2	Operating current	200 mA
3	Processor chip	ATmega328 (8MHz)
4	SRAM	2 KB
5	Flash memory	32 KB
6	UART	1 port
7	Dimensions	33 mm x 18 mm

Table B-5 Datasheet for ATmega328

Sr. No.	Attribute	Type / Value
1	Operating voltage	4.8 – 6.0 V
2	Stall torque	2.2 kg/cm
3	Speed	0.1 s/60°
4	Operating angle	180°
5	Dimensions	22.9 mm x 12.2 mm x 28.5 mm
6	Gear type	Metal

Table B-6 Datasheet for MG90S servomotor

Sr. No.	Attribute	Type / Value
1	Operating voltage	4.5 – 6 V
2	Average power consumption	0.6 W
3	Operating range	40 m
4	Max operating range at 10% ref	5 m
5	Acceptable angle	2.3 °
6	Accuracy	1 % (less than 6 m) 2 % (6 m – 12 m)
7	Distance detection unit	Centimeters (cm)
8	Light sensitivity	70,000 Lux
9	Wavelength	850 nm
10	Communication interface	I2C / PWM
11	Serial port TTL voltage level	3.3 V
12	Electromagnetic compatibility	EN 55032 Class B

Table B-7 Datasheet for LiDAR sensor

Sr. No.	Attribute	Type / Value
1	Operating voltage	5 V
2	Operating current	150 mA
3	Touch type	Capacitive
4	Display resolution	800 x 480
5	Weight	120 g
6	Dimensions	194 mm x 110 mm x 20 mm

Table B-8 Datasheet for LCD touchscreen

Sr. No.	Attribute	Type / Value
1	Dimensions	67 mm x 97 mm x 165 mm
2	Weight	122.8 g
3	Ambient light	Fluorescent light 4000 lx max Direct sunlight 80,000 lx max White light 4000 lx max
4	Interface	USB HID KEYBOARD/USB VCP
5	Operating voltage	5 V_{dc}
6	Aim light source	Red bar LED
7	Identification 1D	UPC/EAN, Code 128, Code 93, Code 11, Matrix 2 of 5, Interleaved 2 of 5, Codabar, MSI

Table B-9 Datasheet for barcode scanner

Sr. No.	Attribute	Type / Value
1	Operating voltage	5 V
2	Operating current	3 A
3	Idle power consumption	2.7 W (540 mA)
4	Processor chip	Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
5	RAM	8 GB
6	Wireless criteria	802.11 b/g/n
7	Bluetooth	5.0 BLE
8	Dimensions	85.6 mm x 56.5 mm

Table B-10 Datasheet for Raspberry pi 4 Model B

Sr. No.	Attribute	Type / Value
1	Input voltage	4.5 – 40 V
2	Output voltage	1.25 – 37 V
3	Output current	Up to 3 A
4	Switching frequency	Up to 150 kHz
5	Ripple voltage	50 mV
6	Efficiency	Up to 92 %
7	Control method	PWM
8	Voltage regulation	± 2%
9	Features	Soft start, thermal shutdown
10	Control interface	External potentiometer

Table B-11 Datasheet for LM 2596 buck converter

Sr. No.	Attribute	Type / Value
1	Operating voltage	5 – 27 V
2	Peak current	86 A
3	Standby current	< 1mA
4	PWM frequency	Up to 25 kHz
5	Control signal voltage	3.3 – 5.0 V
6	Dimensions	53.8 mm x 50.5 mm x 24 mm
7	Weight	30 g
8	Motor type supported	DC brushed motors
9	Heat sink	Built-in

Table B-12 Datasheet for BTS 7960 motor drivers

Sr. No.	Attribute	Type / Value
1	Motor type	Brushed DC motor
2	Frame size	40 mm x 40 mm
3	Length	25 mm
4	Weight	500 g
5	Nominal voltage	24 V
6	Rated power output	144 W
7	Peak current	6 A
8	Speed range	3600 RPM
9	Gearbox	Yes (3600 to 24 RPM)
10	Mounting type	Flange or face

Table B-13 Datasheet for DC motors (Dunkermotoren GR40x25)

Sr. No.	Attribute	Type / Value
1	Battery type	LiPo
2	Nominal voltage	3.7 V
3	Nominal capacity	1000 mAh
4	Weight	20 g

Table B-14 Datasheet for BLE beacon battery

Sr. No.	Attribute	Type / Value
1	Battery type	$LiFePO_4$
2	Nominal voltage	24 V
3	Nominal capacity	20 Ah
4	Weight	5 kg
5	Cycle life	2000
6	Peak current	80 A

Table B-15 Datasheet for shopping cart battery

Bibliography

[1] The Washington Post. (2021, Jan. 29). Smart shopping carts on the rise as stores adapt to pandemic era [Online]. Available: https://www.washingtonpost.com/technology/2021/01/29/smart-shopping-carts-pandemic-innovations/

[2] R. J. C. Aquino, C. K. C. Beltran, J. W. A. Fajardo, J. L. A. Lopez, N. E. Sambajon and R. E. Tolentino, "Image Processing Based Human Following Cart Using 360° Camera," *2020 International Conference on Electronics and Sustainable Communication Systems* (ICESC), 2020, pp. 375-380, doi: 10.1109/ICESC48915.2020.9155956.

[3] Nemalidinne Sai Megana, "Design and Implementation of a Smart Shopping Cart by RFID Technology", *Asian Institute of Technology*, May 2018.

[4] H. -H. Chiang et al., "Development of smart shopping carts with customer-oriented service," *International Conference on System Science and Engineering (ICSSE)*, 2016, pp. 1-2, doi: 10.1109/ICSSE.2016.7551618.

[5] Leng Ng, Y., Siong Lim, C., A. Danapalasingam, K., Loong Peng Tan, M., & Wei Tan, C. (2015). *Automatic Human Guided Shopping Trolley with Smart Shopping System. Jurnal Teknologi*, 73(3). https://doi.org/10.11113/jt.v73.4246

[6] Zhou Fang, Lei Deng, Yongsheng Ou and Xinyu Wu, "A Tracking Robot Based on Wireless Beacon", *Conference: Intelligent Robotics and Applications - Third International Conference, ICIRA 2010, Shanghai, China*, November 2010.

[7] Leena Thomas, Renu Mary George, Amalasree Menon, Greeshma Rajan and Reshma Kurian, "Smart Trolley with Advanced Billing System", *International Research Journal of Engineering and Technology (IRJET)*, March 2017.

[8] Di Fan, Ying He, Xuyang Yao, "SMART SHOPPING CART", *ECE 445 Project Report, Illinois Institute of Technology*, April 2013.

[9] F. Campaña, A. Pinargote, F. Domínguez and E. Peláez, "Towards an indoor navigation system using Bluetooth Low Energy Beacons," 2017 *IEEE Second Ecuador Technical Chapters Meeting (ETCM)*, 2017, pp. 1-6, doi: 10.1109/ETCM.2017.8247464.

[10] Song Chai, Renbo An and Zhengzhong Du, "An Indoor Positioning Algorithm Using Bluetooth Low Energy RSSI", *International Conference on Advanced Materials Science and Environmental Engineering*, April 2016.

[11] A. A. Kherani, S. K. Bhogi and B. Shin, "Hybrid location tracking in BLE beacon systems with in-network coordination," *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2016, pp. 814-815, doi: 10.1109/CCNC.2016.7444890.

[12] B. V. Pradeep, E. S. Rahul and R. R. Bhavani, "Follow me robot using bluetooth-based position estimation," *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2017*, pp. 584-589, doi: 10.1109/ICACCI.2017.8125903.