**Smart Shopping Cart Robot**

**Higher National Diploma in Software Engineering**

**Final Report**

**2024.1F**

**Group 13**

Group Members

|  |  |
| --- | --- |
| Index Number: | Name: |
| KAHDSE241F-020 | W.S.M. Wedage |
| KAHDSE241F-021 | C.D. Senaratne |
| KAHDSE241F-019 | T.M.N.M. Tennakoon |
| KAHDSE241F-068 | K.M.K.K.S.S. Bandara |



School of Computing and Engineering

National Institute of Business Management

# Abstract

The Smart Shopping Cart Robot is designed to assist customers by following them and automatically scanning items using ultrasonic and infrared sensors, barcode scanning technology, and computer vision. The system incorporates an Arduino UNO microcontroller, ESP32 camera, and various sensors to allow the cart to follow the user and scan barcodes of items, displaying the total bill on an LCD screen. This project aims to enhance the shopping experience by minimizing manual effort and streamlining the checkout process.

# Acknowledgement

We would like to express our heartfelt gratitude to our supervisors for their unwavering guidance, encouragement, and support throughout the duration of this project. Their expertise and insightful feedback were instrumental in helping us navigate through the various challenges we encountered. They consistently provided us with constructive advice and fostered an environment that encouraged learning and growth. Their patience, dedication, and valuable insights contributed significantly to the success of this project. We are truly grateful for the time and effort they invested in guiding us towards the completion of this work.

# List of Tables

[Table 1: Gantt Chart 5](#_Toc177414899)

# List of Figures

[Figure 1: Prototype 1 6](#_Toc177808390)

[Figure 2: Prototype 2 7](#_Toc177808391)

[Figure 3: Prototype 2 7](#_Toc177808392)

[Figure 4: Prototype 3 8](#_Toc177808393)

[Figure 5: Prototype 3 8](#_Toc177808394)

[Figure 6: Prototype 4 9](#_Toc177808395)

[Figure 7: Prototype 4 9](#_Toc177808396)

Table of Contents

[Abstract i](#_Toc177415058)

[Acknowledgement i](#_Toc177415059)

[List of Tables ii](#_Toc177415060)

[List of Figures ii](#_Toc177415061)

[Chapter 1: Introduction 1](#_Toc177415062)

[Problem Statement 1](#_Toc177415063)

[Project Objective 1](#_Toc177415064)

[Project Scope 1](#_Toc177415065)

[Chapter 2: Literature Reviews 2](#_Toc177415066)

[Autonomous Shopping Carts 2](#_Toc177415067)

[Barcode Scanning Systems 2](#_Toc177415068)

[Chapter 3: System Design 3](#_Toc177415069)

[Hardware Components 3](#_Toc177415070)

[Software Components 3](#_Toc177415071)

[Customer Following Mechanism 3](#_Toc177415072)

[Barcode Scanning and Billing System 4](#_Toc177415073)

[Chapter 4: Methodology 5](#_Toc177415074)

[Implementation Plan 5](#_Toc177415075)

[Prototype Design 6](#_Toc177415076)

[Prototype 1 6](#_Toc177415077)

[Prototype 2 7](#_Toc177415078)

[Prototype 3 8](#_Toc177415079)

[Prototype 4 9](#_Toc177415080)

[Chapter 5: Discussion 10](#_Toc177415081)

[Advantages of the System 10](#_Toc177415082)

[Challengers Faced 10](#_Toc177415083)

[Chapter 6: Future Implementations 12](#_Toc177415084)

[Potential Improvements 12](#_Toc177415085)

[Scalability and Commercial Use 12](#_Toc177415086)

[Chapter 7: Conclusion 13](#_Toc177415087)

[Reference 14](#_Toc177415088)

[Appendices 15](#_Toc177415089)

# Chapter 1: Introduction

## Problem Statement

Shopping can be time-consuming, and traditional shopping carts require manual effort. Customers often must push heavy carts and manually scan items at checkout. This process can be tedious and inefficient.

## Project Objective

The objective of this project is to develop a smart shopping cart that follows the customer and scans items automatically. This will reduce the manual effort involved in shopping and create a more efficient, user-friendly experience.

## Project Scope

The project focuses on integrating sensors, microcontrollers, and barcode scanning technology to create a cart capable of autonomous movement and item scanning. This prototype will serve as a basis for further development in retail automation.

# Chapter 2: Literature Reviews

## Autonomous Shopping Carts

Autonomous shopping carts are designed to follow customers without manual control, offering a more convenient shopping experience. These carts rely on sensors like ultrasonic and infrared to track the customer's movement and maintain a consistent distance. By reducing the need for pushing or pulling a cart, they aim to enhance comfort and efficiency in shopping environments.

## Barcode Scanning Systems

Barcode scanning has advanced with the use of modern technologies like computer vision. Tools such as OpenCV and Pyzbar library make it possible to scan barcodes quickly and accurately, even from camera feeds. These systems allow for automatic item recognition and billing, simplifying the checkout process and reducing the need for manual input.

# Chapter 3: System Design

## Hardware Components

* Arduino uno R3
* L293D Motor Driver module
* Infrared (IR) Sensor Module
* hc-sr04 ultrasonic distance sensor
* 3.7V 5000mA 18650 Li-ion Rechargeable Battery
* Esp32 Devkit v1
* ESP32 OV2640 Camera Bluetooth Wi-Fi Board
* I2C Serial Interface Adapter Module
* DM0001 - 1602 16x2 Blue Backlight LCD Display
* 4WD Car Chassis Kit
* Gear Motor DC 3-6v

## Software Components

* **Python with OpenCV and Pyzbar Libraries**: Used for barcode recognition and billing.
* **Arduino IDE**: Used to write all the programs needed for Arduino uno, esp32 and esp32 cam module

## Customer Following Mechanism

The cart uses ultrasonic sensors (HC-SR04) and infrared sensors to detect and follow the customer by adjusting the motor speeds accordingly. If the customer turns or changes direction, the cart adjusts its path using sensor data.

## Barcode Scanning and Billing System

The barcode scanning and billing system is a critical part of the smart shopping cart's functionality. It uses an ESP32 camera module to capture real-time images of product barcodes. The camera is integrated with the OpenCV library, which processes the image, while the Pyzbar library decodes the barcode information. Once the barcode is identified, the system cross-references it with an internal inventory stored in the Python code.

Each barcode corresponds to a specific product, which is listed in the inventory along with its price. When a barcode is scanned, the product's information is retrieved, and the price is added to the total bill. The system keeps track of all scanned items and displays the updated total on the 16x2 LCD screen connected to the ESP32 board.

In cases where an item is scanned more than once, the system recognizes the duplicate and avoids adding it to the bill again. If an unrecognized barcode is scanned, the system will notify the user that the item is not found in the inventory. Communication between the Python-based barcode scanner and the ESP32 is facilitated through a Wi-Fi connection, allowing the cart to display real-time billing information to the customer.

This setup eliminates the need for manual checkout, enabling an efficient, automated billing process directly from the cart itself.

# Chapter 4: Methodology

## Implementation Plan

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Time  Task | June | | | | July | | | | August | | | | September | | | |
| 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 |
| Planning |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Analysis |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Designing |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Testing |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Implementation |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Table : Gantt Chart

The project was executed in four stages, with each prototype progressively adding more features to achieve the final design.

**Prototype 1** focused on developing the base of the system. A 4WD car chassis was used, with motors connected to a motor driver and controlled by an Arduino Uno. The system was powered by batteries, establishing a basic platform for further modifications.

**Prototype 2** introduced sensors to the system. Two IR sensors and an ultrasonic sensor were added to implement a human-following functionality. A program was developed to enable the prototype to follow a user based on these sensors, improving the system's automation capabilities.

**Prototype 3** expanded the system's capabilities by incorporating an ESP32-CAM module. This module allowed the system to scan barcodes, and the scanned data was processed using Python code to calculate the total bill. This feature marked the transition from basic automation to a more complex, functional system.

**Prototype 4** built upon the previous prototype by using an ESP32 module to retrieve the calculated bill from the Python script via local Wi-Fi. Additionally, an LCD display was added to show the total bill directly on the cart. This made the system more user-friendly, providing real-time feedback to the user about their purchases and enhancing the overall shopping experience.

Each stage of the implementation added critical functionality to the system, leading to the final, fully integrated prototype.

## Prototype Design

### Prototype 1

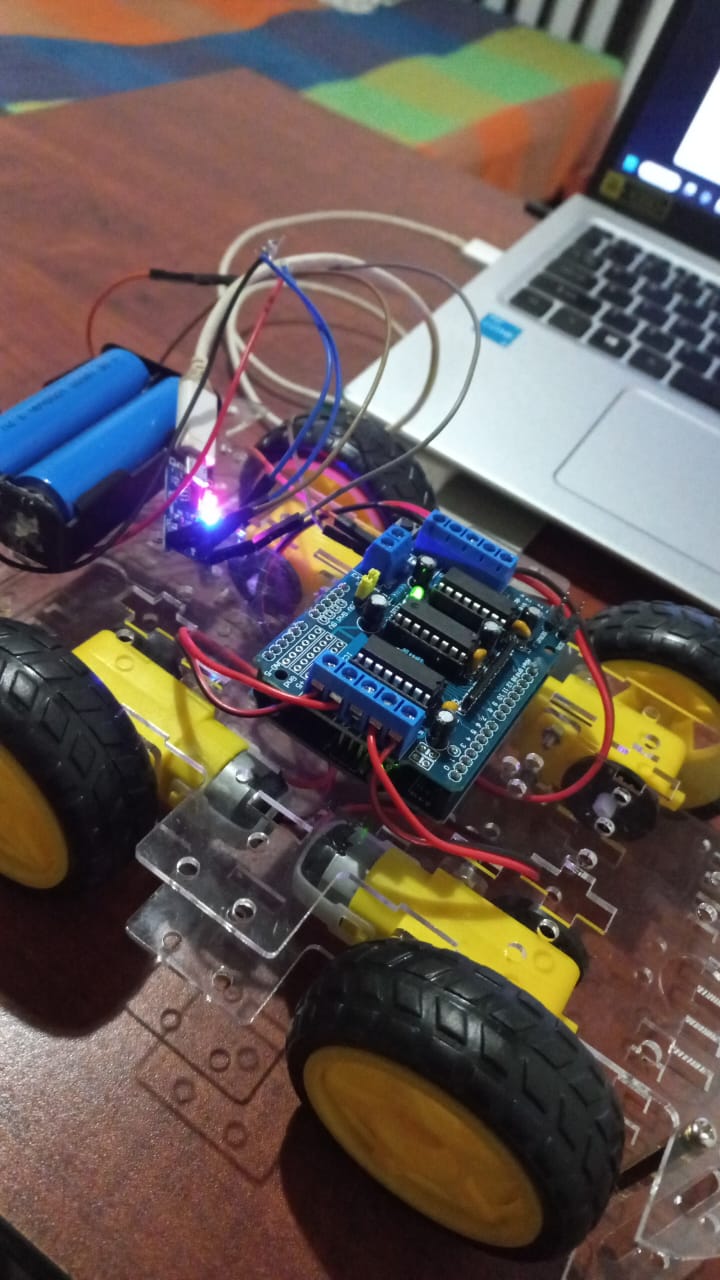
Prototype 1 consists of a basic 4-wheel drive (4WD) car chassis with four motors connected to a motor driver and controlled by an Arduino Uno. The system is powered by a set of batteries, which provide the necessary energy for the motors to drive the wheels. This setup forms the foundation of the prototype, allowing for movement and testing of basic control functions, with the Arduino Uno handling motor operations and overall system control.

Figure 1: Prototype 1

### A robot with a computer Description automatically generated with medium confidencePrototype 2

Figure 3: Prototype 2

Figure 2: Prototype 2

Prototype 2 enhances the initial design by integrating two infrared (IR) sensors and an ultrasonic sensor into the system. These additions enable the prototype to detect and respond to environmental inputs, specifically for human following capabilities. The ultrasonic sensor measures distance and detects obstacles, while the IR sensors help in navigating by detecting surface variations. The entire system is programmed with a human-following algorithm, which allows the prototype to track and follow a person autonomously, making it more interactive and functional in real-world scenarios.

### Prototype 3

Figure 5: Prototype 3

Figure 4: Prototype 3

Prototype 3 builds upon previous iterations by incorporating an ESP32-CAM module for barcode scanning functionality. This allows the robot to identify products by scanning their barcodes. The scanned data is then processed using Python code, which calculates the total bill for the items. This addition transforms the prototype into a more practical solution for tasks such as automated billing in retail environments, combining vision-based sensing with real-time data processing to streamline the shopping experience.

### Prototype 4

Figure 7: Prototype 4



Figure 6: Prototype 4

Prototype 4 enhances the functionality of the system by integrating an ESP32 module to retrieve the final bill calculated by the Python script over local Wi-Fi. An LCD display is also added to the cart, allowing the bill amount to be shown in real-time as the customer adds items. This upgrade improves the user experience, making it easier to track purchases and view the total cost directly on the cart, offering a more seamless and interactive shopping process.

# Chapter 5: Discussion

## Advantages of the System

The smart shopping cart offers several key advantages that improve the shopping experience:

* **Hands-Free Operation**: By following the customer automatically, the cart removes the need for manual pushing, allowing customers to shop more comfortably without worrying about moving a heavy cart.
* **Automated Billing**: The integrated barcode scanning system enables real-time item scanning and billing. As items are placed in the cart, they are automatically scanned and added to the bill, which eliminates the need to wait in long checkout lines.
* **Error Reduction**: The system avoids human error in price calculation by automatically fetching product information from the inventory. Duplicates are recognized, preventing double billing, and only valid items are added to the bill.
* **Improved Efficiency**: The cart streamlines the entire shopping process by combining tracking, scanning, and billing into a single system. Customers can complete their shopping faster, as both the cart movement and payment processes are automated.
* **User-Friendly**: The system is simple and intuitive for customers, with clear real-time updates displayed on the LCD screen, ensuring transparency in billing and minimizing confusion.

These benefits make the system more convenient and efficient for both customers and store owners.

## Challengers Faced

During the development of the project, several challenges were encountered that required problem-solving and adjustments.

One of the first issues faced was power management. Initially, 1000mAh batteries were used to power the DC motors, but they quickly proved insufficient, leading to frequent power drops. To solve this, we upgraded to 5000mAh batteries, which provided the necessary power to run the motors consistently.

Another challenge arose when attempting to upload code to the ESP32-CAM module. Using the standard ESP32 board setup caused multiple issues during code uploading. To overcome this, we switched to an FTID module, which provided a more reliable way to upload code. Additionally, connecting to Wi-Fi was problematic at first, especially when managing the local IP address. This required troubleshooting to ensure stable local network communication.

When implementing the barcode scanning functionality, scanning barcodes became an unexpected challenge. The low resolution of the ESP32-CAM caused difficulties in scanning some printed barcodes, particularly those with finer details or lower print quality.

Finally, integrating the Python code for calculating the bill had its own hurdles. We had to convert the scanned barcode data into a readable string format to calculate the total bill. Once calculated, the result was then sent back to the ESP32 over local Wi-Fi, which required further debugging to ensure seamless data transfer.

Each of these challenges required adaptations but overcoming them led to a more robust and functional final system.

# Chapter 6: Future Implementations

## Potential Improvements

While the smart shopping cart prototype is functional, there are several areas for improvement that could enhance its performance and user experience:

* **Advanced Tracking System**: The current system uses ultrasonic and infrared sensors for customer tracking, which can sometimes be imprecise. Implementing a more advanced tracking system, such as using cameras with machine learning or AI for object detection, would improve accuracy and allow the cart to better navigate crowded environments.
* **Voice Control Integration**: Adding voice recognition technology would allow users to control the cart with simple voice commands, making it more user-friendly, especially for those with mobility challenges. Commands like "stop" or "follow" could be used to manage the cart’s movement.
* **Larger Battery Capacity**: The cart is currently powered by a 3.7V Li-ion battery, which may not be sufficient for prolonged use. Increasing the battery capacity would allow the cart to operate for extended periods without needing frequent recharging.
* **Mobile Payment Integration**: In addition to displaying the bill on the LCD, integrating mobile payment options such as NFC (Near Field Communication) would allow customers to pay directly through their smartphones, creating a fully automated shopping experience.
* **Improved Obstacle Avoidance**: Enhancing the cart’s obstacle detection capabilities by incorporating additional sensors or cameras would allow it to navigate around objects more smoothly, ensuring safer movement in complex store layouts.

By addressing these potential improvements, the smart shopping cart could become more reliable, efficient, and user-friendly, making it a more attractive option for commercial deployment.

## Scalability and Commercial Use

The smart shopping cart system can be scaled for larger retail environments by managing multiple carts through a centralized server, allowing for smooth operation even in busy stores. It can also integrate with store inventory systems for real-time stock updates and use cloud services for remote monitoring and management.

With improvements in battery life and sensor technology, the system can be deployed in larger stores or malls. Its modular design allows for easy upgrades and customization, making it adaptable to the needs of different retailers.

# Chapter 7: Conclusion

The smart shopping cart project demonstrates the potential of automation in enhancing the retail shopping experience. By integrating sensors, barcode scanning, and autonomous movement, the cart reduces the physical effort for customers and streamlines the checkout process. The system effectively combines hardware and software components to create a functional prototype that can track customers, scan products, and calculate bills in real time.

Although the prototype has shown promising results, there are opportunities for further improvements, such as enhancing tracking accuracy, increasing battery life, and integrating mobile payment solutions. With continued development, this technology could significantly impact the retail industry, offering a more efficient, convenient, and user-friendly shopping experience for both customers and store operators.

# Reference

Santos, S. and Santos, S. (2024) *I2C LCD with ESP32 on Arduino IDE - ESP8266 compatible | Random Nerd Tutorials*. <https://randomnerdtutorials.com/esp32-esp8266-i2c-lcd-arduino-ide/>.

Admin (2022). *QR Code Scanner with ESP32 CAM Module & OpenCV*. [online] How To Electronics. Available at: <https://how2electronics.com/qr-code-scanner-with-esp32-cam-module-opencv/>.

www.youtube.com. (n.d.). *How To Make Arduino Human Following Robot - YouTube*. [online] Available at: <https://www.youtube.com/watch?v=yAV5aZ0unag>.

circuitdigest.com. (n.d.). *Obstacle Avoiding Car Using Arduino*. [online] Available at: <https://circuitdigest.com/microcontroller-projects/arduino-obstacle-avoiding-car>.

OpenCV (2019). *OpenCV library*. [online] Opencv.org. Available at: <https://opencv.org/>.

Hudson, L. (n.d.). *pyzbar: Read one-dimensional barcodes and QR codes from Python 2 and 3.* [online] PyPI. Available at: <https://pypi.org/project/pyzbar/>.

Anon, (2019). *I2C LCD with ESP32 on Arduino IDE - ESP8266 compatible | Random Nerd Tutorials*. [online] Available at: <https://randomnerdtutorials.com/esp32-esp8266-i2c-lcd-arduino-ide/>.

Tec-it.com. (2019). *Free Online Barcode Generator: Create Barcodes for Free!* [online] Available at: <https://barcode.tec-it.com/en>.

Code, Q. (2022). *QR Code Reader/Scanner with ESP32 Camera Module & OpenCV*. [online] YouTube. Available at: [https://youtu.be/dxyivPrSFG8?si=NLUx\_K67v4NLzjvV](https://youtu.be/dxyivPrSFG8?si=NLUx_K67v4NLzjvV%20) [Accessed 16 Sep. 2024].

wokwi.com. (n.d.). *ESP32 on Wokwi - Online ESP32, STM32, Arduino Simulator*. [online] Available at: <https://wokwi.com/esp32>.

# Appendices

* Arduino code used for human following

#include<NewPing.h>

#include<AFMotor.h>

#define RIGHT A2

#define LEFT A3

#define TRIGGER\_PIN A1

#define ECHO\_PIN A0

#define MAX\_DISTANCE 500

unsigned int distance = 0;

unsigned int Right\_Value = 0;

unsigned int Left\_Value = 0;

NewPing sonar(TRIGGER\_PIN, ECHO\_PIN, MAX\_DISTANCE);

AF\_DCMotor Motor1(1,MOTOR12\_1KHZ);

AF\_DCMotor Motor2(2,MOTOR12\_1KHZ);

AF\_DCMotor Motor3(3,MOTOR34\_1KHZ);

AF\_DCMotor Motor4(4,MOTOR34\_1KHZ);

void setup() {

  Serial.begin(9600);

  pinMode(RIGHT, INPUT);

  pinMode(LEFT, INPUT);

}

void loop() {

  delay(50);

  distance = sonar.ping\_cm();

  Serial.print("distance");

  Serial.println(distance);

  Right\_Value = digitalRead(RIGHT);

  Left\_Value = digitalRead(LEFT);

  Serial.print("RIGHT");

  Serial.println(Right\_Value);

  Serial.print("LEFT");

  Serial.println(Left\_Value);

  if((distance > 10) && (distance < 200)){

  //Move Forward:

  Motor1.setSpeed(130);

  Motor1.run(FORWARD);

  Motor2.setSpeed(130);

  Motor2.run(FORWARD);

  Motor3.setSpeed(130);

  Motor3.run(FORWARD);

  Motor4.setSpeed(130);

  Motor4.run(FORWARD);

  }else if((Right\_Value==0) && (Left\_Value==1)) {

    //Turn Left

    Motor1.setSpeed(150);

    Motor1.run(FORWARD);

    Motor2.setSpeed(150);

    Motor2.run(FORWARD);

    Motor3.setSpeed(150);

    Motor3.run(BACKWARD);

    Motor4.setSpeed(150);

    Motor4.run(BACKWARD);

    delay(150);

  }else if((Right\_Value==1)&&(Left\_Value==0)) {

    //Turn Right

    Motor1.setSpeed(150);

    Motor1.run(BACKWARD);

    Motor2.setSpeed(150);

    Motor2.run(BACKWARD);

    Motor3.setSpeed(150);

    Motor3.run(FORWARD);

    Motor4.setSpeed(150);

    Motor4.run(FORWARD);

    delay(150);

  }else if(distance > 200) {

    //Stop

    Motor1.setSpeed(0);

    Motor1.run(RELEASE);

    Motor2.setSpeed(0);

    Motor2.run(RELEASE);

    Motor3.setSpeed(0);

    Motor3.run(RELEASE);

    Motor4.setSpeed(0);

    Motor4.run(RELEASE);

  }else if(distance < 10) {

    //Stop

    Motor1.setSpeed(0);

    Motor1.run(RELEASE);

    Motor2.setSpeed(0);

    Motor2.run(RELEASE);

    Motor3.setSpeed(0);

    Motor3.run(RELEASE);

    Motor4.setSpeed(0);

    Motor4.run(RELEASE);

  }

}

* Esp32-cam module code

#include <WebServer.h>

#include <WiFi.h>

#include <esp32cam.h>

const char\* WIFI\_SSID = "wifi SSID";

const char\* WIFI\_PASS = "wifi PWD";

WebServer server(80);

static auto loRes = esp32cam::Resolution::find(320, 240);

static auto midRes = esp32cam::Resolution::find(350, 530);

static auto hiRes = esp32cam::Resolution::find(800, 600);

void serveJpg()

{

  auto frame = esp32cam::capture();

  if (frame == nullptr) {

    Serial.println("CAPTURE FAIL");

    server.send(503, "", "");

    return;

  }

  Serial.printf("CAPTURE OK %dx%d %db\n", frame->getWidth(), frame->getHeight(),

                static\_cast<int>(frame->size()));

  server.setContentLength(frame->size());

  server.send(200, "image/jpeg");

  WiFiClient client = server.client();

  frame->writeTo(client);

}

void handleJpgLo()

{

  if (!esp32cam::Camera.changeResolution(loRes)) {

    Serial.println("SET-LO-RES FAIL");

  }

  serveJpg();

}

void handleJpgHi()

{

  if (!esp32cam::Camera.changeResolution(hiRes)) {

    Serial.println("SET-HI-RES FAIL");

  }

  serveJpg();

}

void handleJpgMid()

{

  if (!esp32cam::Camera.changeResolution(midRes)) {

    Serial.println("SET-MID-RES FAIL");

  }

  serveJpg();

}

void  setup(){

  Serial.begin(115200);

  Serial.println();

  {

    using namespace esp32cam;

    Config cfg;

    cfg.setPins(pins::AiThinker);

    cfg.setResolution(hiRes);

    cfg.setBufferCount(2);

    cfg.setJpeg(80);

    bool ok = Camera.begin(cfg);

    Serial.println(ok ? "CAMERA OK" : "CAMERA FAIL");

  }

  WiFi.persistent(false);

  WiFi.mode(WIFI\_STA);

  WiFi.begin(WIFI\_SSID, WIFI\_PASS);

  while (WiFi.status() != WL\_CONNECTED) {

    delay(500);

  }

  Serial.print("http://");

  Serial.println(WiFi.localIP());

  Serial.println("  /cam-lo.jpg");

  Serial.println("  /cam-hi.jpg");

  Serial.println("  /cam-mid.jpg");

  server.on("/cam-lo.jpg", handleJpgLo);

  server.on("/cam-hi.jpg", handleJpgHi);

  server.on("/cam-mid.jpg", handleJpgMid);

  server.begin();

}

void loop()

{

  server.handleClient();

}

* Esp32 module code

#include <WiFi.h>

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

LiquidCrystal\_I2C lcd(0x27, 16, 2);

// Replace with your network credentials

const char\* ssid     = "Chamila's Galaxy A32";

const char\* password = "chamila321";

// Server port

WiFiServer server(80);

void scrollText(int row, String message, int delayTime, int lcdColumns) {

  for (int i=0; i < lcdColumns; i++) {

    message = " " + message;

  }

  message = message + " ";

  for (int pos = 0; pos < message.length(); pos++) {

    lcd.setCursor(0, row);

    lcd.print(message.substring(pos, pos + lcdColumns));

    delay(delayTime);

  }

}

void setup() {

  Serial.begin(115200);

  WiFi.setHostname("esp32\_board");

  // Connect to Wi-Fi network

  Serial.print("Connecting to ");

  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL\_CONNECTED) {

    delay(1000);

    Serial.println("Connecting...");

  }

  Serial.println("Connected to Wi-Fi");

  // Start the server

  server.begin();

  Serial.println("Server started");

  Serial.println(WiFi.localIP());

  //LCD

  lcd.init();

  lcd.backlight();

  lcd.clear();

  lcd.setCursor(4, 0);

  lcd.print("WELCOME!");

}

void loop() {

  // Listen for incoming clients

  WiFiClient client = server.available();

  if (client) {

    Serial.println("New client connected");

    // Wait until the client sends some data

    while (client.connected() && !client.available()) {

      delay(1);

    }

    // Read and print the message

    String message = client.readStringUntil('\n');

    Serial.println("Message received: " + message);

    //lcd.setCursor(0, 0);

    //lcd.print(message);

    //delay(5000);

    if (message.indexOf("Total") != -1){

      lcd.setCursor(0, 1);

      lcd.print(message);

      delay(3000);

    }

    else{

      scrollText(1, message, 500, 16);

    }

    // Send a response to the client

    client.println("Message received");

    // Close the connection

    client.stop();

    Serial.println("Client disconnected");

  }

}

* Python code

import cv2

import numpy as np

import pyzbar.pyzbar as pyzbar

import urllib.request

import socket

font = cv2.FONT\_HERSHEY\_PLAIN

url = 'http://192.168.50.219/'  #ip address of esp32 cam

cv2.namedWindow("live transmission", cv2.WINDOW\_AUTOSIZE)

ESP32\_IP = '192.168.50.221' #ip address of esp32 board

PORT = 80

# send values to esp32

def send\_message(message):

    try:

        # Create a socket object

        client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

        # Connect to ESP32

        client\_socket.connect((ESP32\_IP, PORT))

        print("Connected to ESP32")

        # Send message to ESP32

        client\_socket.sendall((message + '\n').encode('utf-8'))

        print(f"Sent: {message}")

        # Receive acknowledgment from ESP32

        response = client\_socket.recv(1024).decode('utf-8')

        print(f"ESP32 Response: {response}")

    except Exception as e:

        print(f"Error: {e}")

    finally:

        # Close the socket

        client\_socket.close()

prev = ""

pres = ""

item\_value=""

inventory = [

                ["ABC", 200],

                ["ACB", 300],

                ["BAC", 450],

                ["BCA", 350],

                ["CAB", 550],

                ["CBA", 600]

            ]

items = [

            "ABC00123", "ABC00132", "ABC00213", "ABC00231", "ABC00312", "ABC00321",

            "ACB00123", "ACB00132", "ACB00213", "ACB00231", "ACB00312", "ACB00321",

            "BAC00123", "BAC00132", "BAC00213", "BAC00231", "BAC00312", "BAC00321",

            "BCA00123", "BCA00132", "BCA00213", "BCA00231", "BCA00312", "BCA00321",

            "CAB00123", "CAB00132", "CAB00213", "CAB00231", "CAB00312", "CAB00321",

            "CBA00123", "CBA00132", "CBA00213", "CBA00231", "CBA00312", "CBA00321"

        ]

bill = []

tot = 0

st\_tot=""

while True:

    try:

        # Try to retrieve the image from the URL

        img\_resp = urllib.request.urlopen(url + 'cam-hi.jpg', timeout=20)

        imgnp = np.array(bytearray(img\_resp.read()), dtype=np.uint8)

        frame = cv2.imdecode(imgnp, -1)

        # If frame is empty or None, skip processing

        if frame is None:

            print("Failed to retrieve the image. Skipping frame.")

            continue

        decodedObjects = pyzbar.decode(frame)

        for obj in decodedObjects:

            pres = obj.data

            item\_value = obj.data.decode("utf-8")

            if prev == pres:

                pass

            else:

                prev = pres

                #Bill calculation

                print(item\_value)

                item = item\_value[:3]

                itemId = item\_value[5:]

                if item\_value in items:

                    if item\_value not in bill:

                        bill.append(item\_value)

                        for i in range(len(inventory)):

                            if inventory[i][0] == item:

                                add = int(inventory[i][1])

                                tot += add

                                print(f"Total value: {tot}")

                                st\_tot=str(tot)

                                send\_message(st\_tot)

                                break

                    else:

                        print("Item already added to bill")

                        send\_message("Item already added to bill")

                else:

                    print("Item not found")

                    send\_message("Item not found")

            cv2.putText(frame, str(obj.data), (50, 50), font, 2,

                        (255, 0, 0), 3)

        cv2.imshow("live transmission", frame)

    except urllib.error.URLError as e:

        print(f"Failed to connect to the server: {e.reason}")

        # Optionally break the loop or retry after some time

        break

    except Exception as e:

        print(f"An unexpected error occurred: {e}")

        break

    key = cv2.waitKey(1)

    if key == 27:

        break

cv2.destroyAllWindows()