Bangladesh University of Engineering and Technology
Department of Electrical and Electronic Engineering

EEE-404

Robotics and Automation Laboratory

Final Project Report

Section: G2 Group: 04

Smart Inventory management System

Course Instructors:

Dr. Celia Shahnaz, Professor, EEE
Md. Jawad Ul Islam, Lecturer, EEE
Aye Thein Maung, Adjunct Lecturer, EEE

Academic Honesty Statement:

IMPORTANT! Please carefully read and sign the Academic Honesty Statement, below. Type the student ID and name, and put your signature. *You will not receive credit for this project experiment unless this statement is signed in the presence of your lab instructor.*

"In signing this statement, We hereby certify that the work on this project is our own and that we have not copied the work of any other students (past or present), and cited all relevant sources while completing this project. We understand that if we fail to honor this agreement, We will each receive a score of ZERO for this project and be subject to failure of this course."

Table of Contents

1 Abstract

A Smart Inventory Management System based on Raspberry Pi, Arduino and a 6-DOF robotic hand for object detection, tracking and handling. The presented solution uses YOLOv5 for realtime tracking of an object by recognizing and locating different items in the specified workspace. A Pi sending the object's visual data to an Arduino microcontroller which in turn determine the upper middle point of the bounding box corresponding to that object. An Arduino board is used to activate switching of the 6 DOF robotic hand for picking, placing, and sorting of objects. All these automated functions decrease the amount of work required to manage the stock as well as enhance the accuracy of management thus making it usable in warehouses, retail, industrial, and other businesses.

2 Introduction

Efficient inventory management is essential for modern warehouses, retail stores, and supply chains where speed, accuracy, and automation are crucial. With this project we tried to implement a Smart Inventory Management System that integrates YOLOv5 for object detection, the Google Speech API for voice recognition, Raspberry Pi for processing, Arduino for control, and a 6-DOF robotic hand for automated object handling.

The system employs YOLOv5 to detect and locate objects within a defined stoppage, enabling real-time tracking of inventory items. Voice commands, processed via the Google Speech API, allow users to interact with the system hands-free, facilitating dynamic control of inventory operations. The Raspberry Pi coordinates the detection, recognition, and control processes, while the Arduino drives the movements of the 6-DOF robotic hand, enabling precise picking, sorting, and placement of items.

By combining computer vision, voice recognition, and robotics, this system offers a versatile and user-friendly approach to inventory management. It reduces human intervention, enhances operational efficiency, and provides an intuitive, voice-controlled interface for streamlined warehouse automation

3 Design

3.1 Problem Formulation (PO(b))

3.1.1 Formulation of Problem

The initial problems:

- **i) Real time Object detection is highly computationally heavy how to solve for this**
- **ii) How to accurately control the 6 DOF robotic arm**
- **iii) How to generate a map for the perfect stoppage points**
- **iv) How to enable Arduino with the voice commands**
- **v) How to get a real time view in a end computer device**

3.1.2 Analysis

1. Real-time Object Detection is Computationally Heavy

- **Cause**: YOLOv5, while efficient, requires significant computational resources, especially for real-time performance on edge devices like Raspberry Pi.
- **Constraints**: Limited processing power, memory, and energy efficiency of Raspberry Pi.

Possible Solutions:

- Use a **lightweight YOLOv5 model** (YOLOv5n or YOLOv5s) to reduce computation.

2. Accurate Control of the 6-DOF Robotic Arm

- **Cause**: Controlling a 6-DOF robotic arm involves complex kinematics, inverse kinematics, and precise coordination between multiple joints.
- **Constraints**: Limited control precision due to hardware limitations of servo motors and potential delays from Raspberry Pi/Arduino communication.

Possible Solutions:

- Calibrate each servo motor to minimize cumulative error

3. Generating a Map for Perfect Stoppage Points

- **Cause**: The robotic arm needs to know the exact locations where objects should be picked or placed.
- **Constraints**: Variations in object size, orientation, and the possibility of dynamic environments.

Possible Solutions:

- Implement **marker-based positioning** using fiducial markers (like ArUco) to identify and locate specific points

4. Enabling Arduino with Voice Commands

- **Cause**: Voice commands need to be recognized, interpreted, and converted into instructions for the Arduino.
- **Constraints**: Voice recognition accuracy can be affected by noise, accents, and limited computational capacity on Arduino.

Possible Solutions:

- Use **Google Speech API** on the Raspberry Pi for voice recognition.

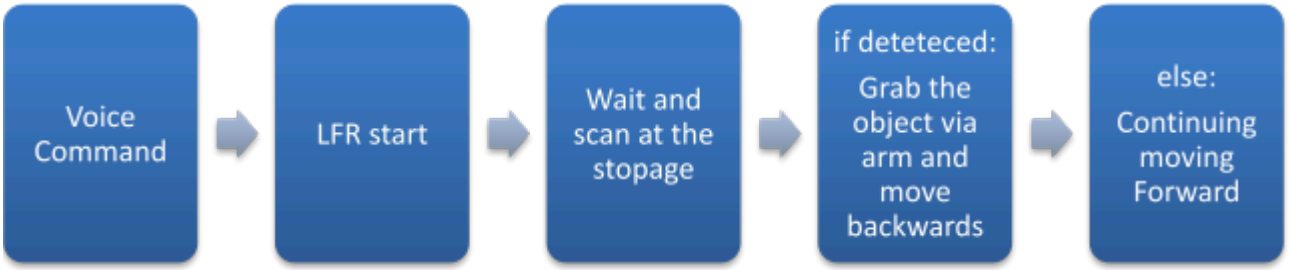
5. Real-time View on an End Computer Device

- **Cause**: The user requires live visual feedback for monitoring inventory operations.
- **Constraints**: Limited bandwidth, video compression, and processing power on Raspberry Pi.

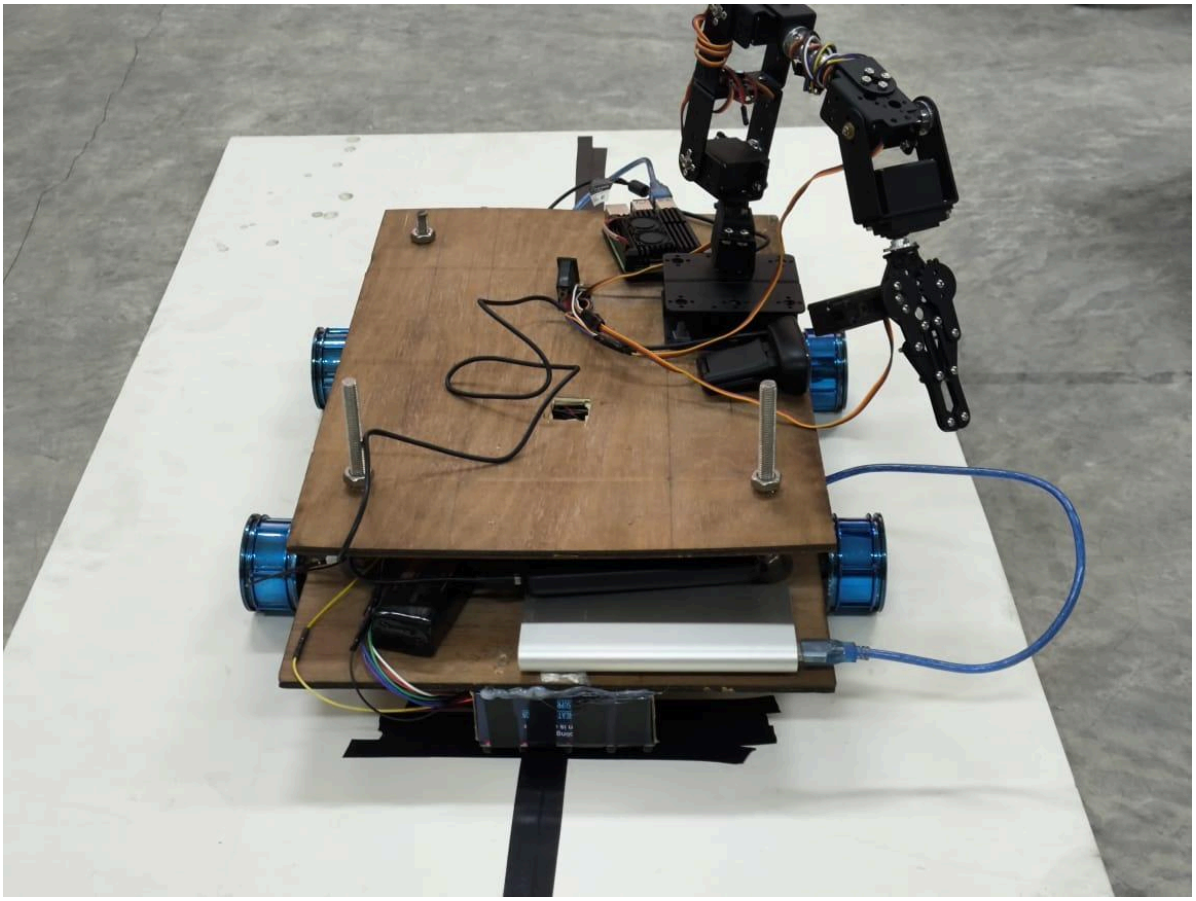
Possible Solutions:

- Use **video streaming protocols** like RTSP or WebRTC for real-time streaming.

3.2 Design Method (PO(a))



3.3 CAD/Hardware Design



3.4 Full Source Code of Firmware

Code for streaming video over a network section:

```
import io
import socket
import struct
import time
import cv2
import serial

# Setup serial connection to Arduino
serial_port = '/dev/ttyACM0' #'/dev/ttyACM0'
host = '192.168.0.111'
#arduino_serial = serial.Serial(serial_port, 115200, timeout=1)

# Network setup
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((host, 8000))
connection = client_socket.makefile('wb')

# Create a separate socket for receiving commands
command_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
command_socket.connect((host, 8001))  # Different port for commands
command_recv = command_socket.makefile('rb')
# cv2.CAP_V4L2
try:
    cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
    if not cap.isOpened():
        print("Error: Could not open video device")
        exit(1)

    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)  # Set width
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240) # Set height
    cap.set(cv2.CAP_PROP_FPS, 15)

    time.sleep(2)
    start = time.time()
    stream = io.BytesIO()

    # Separate thread or process would be better for real-world use
    import threading
    def receive_commands():
        while True:
            try:
                command =
command_recv.readline().decode('utf-8').strip()
                if command != 'x':
```

```
                    print(f"Received command: {command}")
                    # arduino_serial.write(command.encode())
            except Exception as e:
                print(f"Error receiving command: {e}")
                break

    # Start command receiving in a thread
    command_thread = threading.Thread(target=receive_commands)
    command_thread.daemon = True
    command_thread.start()

    while True:
        ret, frame = cap.read()  # Capture a frame from the webcam
        if not ret:
            print("Failed to grab frame")
            break

        # Encode the frame as JPEG
        ret, jpg = cv2.imencode('.jpg', frame)
        if not ret:
            print("Failed to encode image")
            break

        # Send the JPEG image over the network
        connection.write(struct.pack('<L', len(jpg)) )  # Send
length of the image
        connection.flush()
        connection.write(jpg.tobytes())  # Send the actual image
data

        # if time.time() - start > 600:  # Run for 10 minutes (600
seconds)
        #     break

    # Send the end of stream signal (0 length)
    connection.write(struct.pack('<L', 0))
finally:
    connection.close()
    client_socket.close()
    command_socket.close()
    arduino_serial.close()
```

Object Detection:

```
import socket
import struct
import cv2
import numpy as np
import threading
import torch
import cv2
import pandas
import torch
import speech_recognition as sr
import pyttsx3  # Optional: for voice feedback

# # Load the YOLOv5 model (assuming yolov5s is the model)
# model = torch.hub.load('ultralytics/yolov5', 'yolov5s',
pretrained=True)

# Initialize the speech recognizer
```

```
recognizer = sr.Recognizer()

# Optional: Initialize the text-to-speech engine
engine = pyttsx3.init()
engine.setProperty('rate', 150)  # Set speech rate
engine.setProperty('volume', 1)  # Set volume to max

# Setup sockets for video and commands
VIDEO_PORT = 8000
COMMAND_PORT = 8001
CLASS_PORT = 8002
hostIP = '192.168.43.175'
# Video receiver setup
server_socket = socket.socket() #socket.AF_INET, socket.SOCK_STREAM
server_socket.bind((hostIP, VIDEO_PORT))  # Accept connections on
all interfaces
server_socket.listen(0)
print(f"Waiting for video connection on port {hostIP} and
{VIDEO_PORT}...")
video_client, addr = server_socket.accept()
print(f"Video connection established with {addr}")
print('hi')

# Class Receiver setup
class_socket = socket.socket() #socket.AF_INET, socket.SOCK_STREAM
class_socket.bind((hostIP, CLASS_PORT))  # Accept connections on
all interfaces
class_socket.listen(0)
print(f"Waiting for Class name sending connection on port
{CLASS_PORT}...")
class_client, addr = class_socket.accept()
print(f"Class name sending connection established with {addr}")


# Command sender setup
command_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
command_socket.bind((hostIP, COMMAND_PORT))
command_socket.listen(0)
print(f"Waiting for command connection on port {COMMAND_PORT}...")
command_client, addr = command_socket.accept()
print(f"Command connection established with {addr}")



def listen_for_command():
    """Function to listen for a voice command."""
    with sr.Microphone() as source:
        print("Listening for command...")
```

```
        audio = recognizer.listen(source)

        try:
            command = recognizer.recognize_google(audio).lower()
            print(f"Recognized command: {command}")
            return command
        except sr.UnknownValueError:
            print("Could not understand the audio")
            return None
        except sr.RequestError:
            print("Error with the Speech Recognition service")
            return None


# Load the YOLOv5 model (you can use 'yolov5s', 'yolov5m',
'yolov5l', or 'yolov5x')
model = torch.hub.load('ultralytics/yolov5', 'yolov5s',
pretrained=True)
def detect_objects(frame, my_class):

    # Perform inference
    results = model(frame)
    print("frame paise model")
    print(f'shape of each frame {frame.shape}')
    df = results.pandas().xyxy[0]  # xyxy format

    # Extract bounding box coordinates and class labels
    for index, row in df.iterrows():
        x1, y1, x2, y2 = row['xmin'], row['ymin'], row['xmax'],
row['ymax']
        xc = (x1+x2)/2
        yc = (y1+y2)/2

        class_name = row['name']

        confidence = row['confidence']

        print(f"Detected {class_name} with confidence
{confidence:.2f}")
        print(f"Bounding box: ({x1}, {y1}), ({x2}, {y2})")


        if(class_name == my_class):
            print(f"{my_class} paise")

print(f'****************************************{my_class}
paisi at center {xc,yc}**************************************')
```

```
            command_client.sendall('gotit'.encode('utf-8'))  # For
strings
        #     if(240-50 < xc < 240+50):  #and 320-30 < yc < 320+30
        #         return  xc,yc,class_name


    # Render results on the frame
    frame_with_boxes = results.render()[0]

    # Display the frame with detections



    # cv2.imshow('YOLOv5 Object Detection', frame_with_boxes)

    return frame_with_boxes #,xc,yc,class_name


# Function to send commands to Raspberry Pi
def send_commands(status):
    while True:
        try:
            command = status
            if command.lower() == 'exit':
                print("Closing connections...")
                break
            command_client.sendall((command +
'\n').encode('utf-8'))
        except Exception as e:
            print(f"Error sending command: {e}")
            break

    command_client.close()



# # Start command thread
# command_thread = threading.Thread(target=send_commands)
# command_thread.start()


# Class name receiving loop
# my_class = ''
# try:
#     data = b""
#     payload_size = struct.calcsize("<L")

#     while True:
```

```
#         # Receive message size (4 bytes)
#         while len(data) < payload_size:
#             packet = class_client.recv(4096)
#             if not packet:
#                 break
#             data += packet
#         if data is None:
#             print("error receivving class name data")
#             break
#         my_class = data.decode('utf-8')
#         print(f"received class name {my_class}")
#         break

# except Exception as e:
#     print(f"Error receiving class name: {e}")



if __name__ == "__main__":

    user_input = ""
    while user_input.lower() != "start":
        user_input = input("Type 'start' to begin the program: ")
    print("Program started...")

    command = Noneststa
    while(command==None):
        command = listen_for_command()
        print(command)

        if(command is not None):
            # engine.say(f"You said {command}. Please Confirm")
            object_class = command.split(' ')[-1]
            break
        #     send_commands('command_sent')
        #     # Start command thread
        #     command_thread =
threading.Thread(target=send_commands)
        #     command_thread.start()
        #     break


    # Video receiving loop
    try:
        data = b""
        payload_size = struct.calcsize("<L")

        while True:
```

```
            # Receive message size (4 bytes)
            while len(data) < payload_size:
                packet = video_client.recv(4096)
                if not packet:
                    break
                data += packet
            packed_size = data[:payload_size]
            data = data[payload_size:]
            msg_size = struct.unpack("<L", packed_size)[0]

            if msg_size == 0:  # End of stream signal
                print("End of video stream received.")
                break

            # Receive the actual image data
            while len(data) < msg_size:
                data += video_client.recv(4096)
            frame_data = data[:msg_size]
            data = data[msg_size:]

            # Decode the frame
            frame = cv2.imdecode(np.frombuffer(frame_data,
dtype=np.uint8), cv2.IMREAD_COLOR)
            frame_with_boxes = detect_objects(frame, object_class)

            if frame_with_boxes is not None:
                cv2.imshow('Video Stream', frame_with_boxes)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
            else:
                print("Failed to decode frame")

    except Exception as e:
        print(f"Error receiving video stream: {e}")
    finally:
        print("Closing connections...")
        video_client.close()
        command_client.close()
        server_socket.close()
        command_socket.close()
        cv2.destroyAllWindows()

    # C:\Users\Victus\yolov5
```

Robotic Arm:

```
#include <Servo.h> // Include the Servo library

Servo Servo1; // Create a Servo object
Servo Servo2;
```

```
Servo Servo3;
Servo Servo4;
Servo Servo5;
Servo Servo6;
Servo Servo7;

const int pin13 = 13;
void setup() {
  pinMode(pin13, OUTPUT);
  Servo1.attach(2); // Attach the servo to pin 9
  Servo2.attach(3);
  Servo3.attach(4);
  Servo4.attach(5);
  Servo5.attach(6);
  Servo6.attach(7);

  Serial.begin(9600); // Start Serial communication
  Serial.println("Servo Arm Control Initialized");

  // Move to stretched position
  Serial.println("Moving to stretched position...");
  //myServo.write(0); // 0 degrees (stretched)
  delay(2000); // Hold stretched position for 2 seconds

  // Move to hold position
  Serial.println("Moving to hold position...");
  //myServo.write(180); // 90 degrees (holding position)
  Servo2.write(150);
  delay(2000); // Hold the object
}

void claw_open()
{
  Servo1.write(0);
}


void claw_close()
{
  Servo1.write(50);
}


void servo2()
{
  Servo2.write(50);
}

void servo3()
{
  Servo3.write(140);
}

void servo4()
{
  Servo4.write(130);
}
```

```
void up()
{
  Servo5.write(140);
}
void down()
{
  Servo5.write(40);
}

void front()
{
  Servo6.write(100);
}

void left()
{
  Servo6.write(180);
}

void initial()
{
    Servo5.write(140);
    delay(100);
    Servo4.write(130);
    delay(100);
    Servo3.write(50);
    delay(100);
}
int flag = 0;
void loop() {
  // Stop the servo (it remains in the last position)
  // You can add further code here if needed.

  digitalWrite(pin13,LOW);
  int d = 1000;
  int dd = 2000;
  digitalWrite(5,LOW);
  if(Serial.available()>0){
    String msg = Serial.readString();
    if(msg=="gotit"){
      digitalWrite(13,HIGH);
      delay(100);
      Servo6.write(100);
      delay(d);
      Servo3.write(140);
      delay(d);
      Servo1.write(0);
      delay(dd);
      Servo1.write(50);
      delay(d);
      Servo3.write(50);
      delay(d);
      Servo6.write(0);
      delay(dd);
      Servo3.write(140);
      delay(d);
```

```
      Servo1.write(0);
      delay(dd);
      Servo1.write(50);
      delay(d);
      delay(3000);
      digitalWrite(5,HIGH);
    }
  }

}
```

LFR Code:

```
// For 4 wheels -> en1,en2,in1,in2,in3,in4
int stp = 2;
int f = 0;
int r_en1 = 12;
int l_en1  = 9;
int rpwm1 = 11;
int lpwm1 = 10;
int r_en2=2;
int l_en2=3;
int rpwm2=5;
int lpwm2=6;
int out1 = A1;

int out2 =A2;
int out3 =A3;
int out4 =A4;
int out5 =A5;
int speed=20;

int blackThreshold=750;

long timeInMicro;
long distanceInCm;



void moveForward()
{

  analogWrite(rpwm1,speed);
  analogWrite(rpwm2,0);
  analogWrite(lpwm1,0);
  analogWrite(lpwm2,speed);

}
void moveBackward()
{

  analogWrite(rpwm1,0);
  analogWrite(rpwm2,speed);
  analogWrite(lpwm1,speed);
  analogWrite(lpwm2,0);

}
void turnLeft()
```

```
{
  analogWrite(rpwm1,0);
  analogWrite(rpwm2,0);
  analogWrite(lpwm1,speed);
  analogWrite(lpwm2,speed);
  delay(80);

}

void turnRight()
{
  analogWrite(rpwm1,speed);
  analogWrite(rpwm2,speed);
  analogWrite(lpwm1,0);
  analogWrite(lpwm2,0);
  delay(80);
}
void stop()
{
  analogWrite(rpwm1,0);
  analogWrite(rpwm2,0);
  analogWrite(lpwm1,0);
  analogWrite(lpwm2,0);


}
void setup() {
  // put your setup code here, to run once:
  //pinMode(trig,OUTPUT);
  //pinMode(echo,INPUT);
  pinMode(r_en1, OUTPUT);
  pinMode(l_en1, OUTPUT);
  pinMode(rpwm1, OUTPUT );
  pinMode(lpwm1, OUTPUT);
  pinMode(r_en2, OUTPUT);
  pinMode(l_en2, OUTPUT);
  pinMode(rpwm2, OUTPUT);
  pinMode(lpwm2, OUTPUT);
  digitalWrite(r_en1, HIGH);
  digitalWrite(l_en1,HIGH);
  digitalWrite(r_en2,HIGH);
  digitalWrite(l_en2,HIGH);
  Serial.begin(9600);
  analogWrite(rpwm1,0);
  analogWrite(rpwm2,0);
  analogWrite(lpwm1,0);
  analogWrite(lpwm2,0);

}

void loop() {


int s1 = 0;
int s2 = 0;
int s3= 0;
int s4 = 0;
```

```
int s5 = 0;


  // put your main code here, to run repeatedly:
  int sensor1Value = analogRead(A1);
  int sensor2Value = analogRead(A2);
  int sensor3Value = analogRead(A3);
  int sensor4Value = analogRead(A4);
  int sensor5Value = analogRead(A5);

if(sensor2Value > blackThreshold)
{

  s2 = 1;

}

if(sensor4Value > blackThreshold)
{

  s4 = 1;

}
if(sensor3Value > blackThreshold)
{

  s3 = 1;

}
if(sensor5Value > blackThreshold)
{
  s5 = 1;

}
if(sensor1Value > blackThreshold)
{
  s1 = 1;

}

//digitalWrite(trig,LOW);

//digitalWrite(trig,HIGH);

//digitalWrite(trig,LOW);
//timeInMicro= pulseIn(echo,HIGH);
/*distanceInCm = ((timeInMicro/29)/2);
if(distanceInCm <= 10 )
  {
  analogWrite(rpwm1,0);
  analogWrite(rpwm2,0);
  analogWrite(lpwm1,0);
  analogWrite(lpwm2,0);


   }*/
//else{
```

```
if((s1==0)&&(s2==0)&&(s3==0)&&(s4==0)&&(s5==0))
{
  Serial.print("I am robot 7");
  if(stp !=  0)
  {
    stop();
    delay(5000);
    stp = stp - 1;
  }
  else
  {
    moveBackward();
    f = 1;
  }

}
else if((s2==0))
{
  Serial.print("I am robot 1");

  if(s1==0)
  {
    Serial.print("I am robot 1-1");
    int j=2;
    while(j!=0)
    {
      turnLeft();
      stop();
      delay(500);
      j = j-1 ;
    }
  }
  else
  {
    Serial.print("I am robot 1-2");
    turnLeft();
    stop();
    delay(500);

  }
}

else if(s4==0)
{
  Serial.print("I am robot 2");
  if(s5==0)
  {
    int j = 2;
    Serial.print("I am robot 2-1");
    while(j!=0)
    {
      turnRight();
      delay(500);
      stop();

      j = j-1 ;
```

```
    }
  }
  else
  {
    Serial.print("I am robot 2-2");
    turnRight();
    delay(500);
    stop();

  }
}

else if((s1==1)&&(s2==1)&&(s3==0)&&(s4==1)&&(s5==1))
{
  Serial.print("I am robot 3");
  if(!f)
  moveForward();
  else
  moveBackward();
  delay(500);
}

else if((s1==1)&&(s2==1)&&(s3==0)&&(s4==0)&&(s5==1))
{
  Serial.print("I am robot 4");
  if(!f)
  moveForward();
  else
  moveBackward();
  delay(500);
}
else if((s1==1)&&(s2==0)&&(s3==0)&&(s4==1)&&(s5==1))
{
  Serial.print("I am robot 5");
  if(!f)
  moveForward();
  else
  moveBackward();
  delay(500);
}
else if((s1==1)&&(s2==0)&&(s3==0)&&(s4==0)&&(s5==1))
{
  Serial.print("I am robot 6");
  if(!f)
  moveForward();
  else
  moveBackward();
  delay(500);
}

else
{
  stop();
}

//}
```

```
/*else
{
 if((s1==0)&&(s2==0))
  {
    turnRight();
  }
 else if((s4==0)&&(s5==0))
  {
    turnRight();
  }
 else
 {
   moveForward();
 }
}*/

//moveForward();
Serial.println("----");
Serial.print(sensor1Value);
Serial.print("----");
Serial.print(sensor2Value);
Serial.print("----");
Serial.print(sensor3Value);
Serial.print("----");
Serial.print(sensor4Value);
Serial.print("----");
Serial.print(sensor5Value);
Serial.println("----");
Serial.print(s1);
Serial.print("-");
Serial.print(s2);
Serial.print("-");
Serial.print(s3);
Serial.print("-");
Serial.print(s4);
Serial.print("-");
Serial.println(s5);
delay(300);


}
```

We Wrote all the codes from scratch	3.5

4 Implementation

4.1 Description

The code you provided is a system that integrates video streaming, object detection, and

voice command control. It sets up server-client socket communication where the server receives video frames, performs object detection using a pretrained YOLOv5 model, and listens for voice commands using the Google Speech API. The system captures objects from the video stream, identifies them based on the user's spoken command, and sends corresponding responses to control a robotic system or trigger other actions. It includes error handling, multithreading for command listening, and uses text-to-speech for user feedback. The setup also involves separate sockets for video, command, and class name reception.

5 Design Analysis and Evaluation

5.1 Novelty

Here's a summary of the novel features this system brings compared to traditional setups:

1. **Integration of Video Streaming and Object Detection**: Combines real-time video streaming with AI-driven object detection using YOLOv5, allowing for seamless interaction between visual input and decision-making processes.
2. **Voice Command Control**: Incorporates speech recognition (Google Speech API) for hands-free operation, enabling users to control the system using spoken commands instead of traditional manual inputs.
3. **Dynamic Object Detection Based on Voice Commands**: The system can detect and respond to specific objects identified in the video stream, triggered by user-defined spoken commands, enhancing flexibility compared to static object detection.

5.2 Design Considerations (PO(c))

Some key design considerations for this system:

- **Latency and Real-time Performance**
- Error Handling and Robustness
- Scalability and Flexibility
- User Interaction and Accessibility

5.3 Limitations of Tools (PO(e))

1. Object Detection (YOLOv5) Limitations:

- **Environmental Factors**: Object detection models, like YOLOv5, may struggle in poor lighting conditions or with low-quality video input. Low resolution, high noise, or obstructions can affect detection accuracy.

- **False Positives/Negatives**: Object detection models can sometimes generate false positives (incorrectly identifying objects) or false negatives (failing to identify objects), especially in cluttered or dynamic environments.

2. Voice Recognition (Google Speech API and pyttsx3) Limitations:

- **Noise Sensitivity**: Speech recognition is highly sensitive to background noise. In noisy environments, the accuracy of voice commands may degrade, leading to misinterpretation or failure to recognize commands.

3. Robotic Control (Arduino and Raspberry Pi Integration):

- **Limited Computational Power**: The Arduino and Raspberry Pi have limited computational capabilities compared to high-end computing systems. Complex processing tasks, such as running AI models or handling large datasets, may require offloading to more powerful systems or cloud services.
- **Precision and Dexterity**: While a 6-DOF robotic hand can perform complex movements, its precision and dexterity may be limited by the quality of the hardware and control algorithms. This might not be ideal for tasks that require highly refined movements or delicate handling.

4. Hardware Limitations:

- **Component Compatibility**: Ensuring that all hardware components (Arduino, Raspberry Pi, cameras, sensors) are compatible and can work together seamlessly may be a challenge. Limited integration between hardware could result in issues with performance or reliability.
- **Environmental Robustness**: Hardware components may not be robust enough for certain environments. For instance, outdoor or industrial environments with dust, moisture, or extreme temperatures might require specialized hardware that is more resistant to environmental challenges.

5.4 Impact Assessment (PO(f))

5.4.1 Assessment of Societal and Cultural Issues

Implementable in any society and culture as the market place management system is same at pretty much everywhere in the world.

5.4.2 Assessment of Health and Safety Issues

Power sources i.e. batteries used are the safest ones available with no effects on environment whatsoever. So, it's safe to say the robot has no health and safety issues.

5.4.3 Assessment of Legal Issues

No laws were broken when making and using the robot. So, no legal issues.

5.5 Stainability Evaluation (PO(g))

Our robot's most major problem is its sustainability as the robotic arm uses a huge amount of power from the battery and thus drains it completely. Hopefully replacement with a reliable power source will make it more sustainable.

6 Reflection on Individual and Team work (PO(i))

6.1 Individual Contribution of Each Member

Anik Biswas – Robotic arm

Mahid Mostafa – LFR

Shourav Joarder – Raspberry Pi codes and Pi to Arduino communication

Zahin Tazwar – Codes and software

Monideepa Kundu – Hardware assembly and integration

6.2 Mode of TeamWork

All test runs after assembly, problem identification and solution of those problems were teamwork.

7 Communication to External Stakeholders (PO(j))

7.1 Executive Summary

Smart Inventory Management System helps the inventory's sake by restocking front shop products by the voice command of the store keeper or the manager. The product will be identified by image processing and cross matching the class.

7.2 User Manual

1. Always keep the robot on the edge of black line.

   Warning – Otherwise the LFR will not move at first.

2. Make sure the name of the desired object is the last word of the voice command.

8 Project Management and Cost Analysis (PO(k))

8.1 Bill of Materials

Component	Quantity	Cost
Raspberry Pi	1	18,595 Tk
Arduino Uno	1	950 Tk
Webcam	1	2300 Tk
Li-Po Battery	1	1350 Tk
IR sensor array	1	370 Tk
Buck Converter	1	220 Tk
Robotic Arm Set	1	4120 Tk
Servo Motor (MG996)	6	2100 Tk
Jumper Wires	4 set	240 Tk
Miscellaneous		300 Tk
Total		30,545 Tk

8.2 Calculation of Per Unit Cost of Prototype

The per unit cost of the prototype is 30,545 taka.

9 Future Work (PO(l))

This project mainly aims in making a simple Smart Inventory Management Robot. Although we have made a low cost robot capable of doing simple task there are scope of improvements such as,

1. Using a better microcontroller will give computational advantage to the robot making its performance more smooth and fast.
2. Developing a user friendly interface to control the system.
3. Adding more features such as object sorting, expired product detection and removal etc.
4. Making a more compact design will allow smoother operation.
5. Using more advanced and precise sensors.