Advanced

Mechatronics Project

Report

Advanced Mobile Robotic object sorter

Sotomi Oluwadamilola - ots2014

El Tannir, Omar - oe2104

Ravi, Yukteshwar - yr2364

Ragupathy,Sai Nishit - sr6931

TABLE OF CONTENT

1.0. ABSTRACT

This document presents the conceptualisation and creation of a Mobile Robotic Object Sorter, a cutting-edge venture that utilises Arduino technology to transform sorting operations across manufacturing, logistics, and warehouse management sectors. This project integrates sophisticated robotics and automation to forge a system capable of efficiently categorising and organising objects according to specific criteria with unmatched precision. It amalgamates features such as path-following capabilities, a comprehensive sensor array, a modular gripping mechanism, and wireless connectivity for enhanced monitoring and control, establishing a novel benchmark in automated sorting solutions.

2.0. INTRODUCTION

The dynamic shift towards the integration of robotics and automation within various sectors underscores the necessity for improved efficiency and productivity. The Mobile Robotic Object Sorter project is at the forefront of this innovation, offering a multifaceted and effective approach to sorting and organising challenges. By harnessing the Arduino microcontroller platform, the endeavour seeks to craft an advanced yet user-friendly system tailored to meet the practical demands of automation challenges. This report delves into the project's developmental blueprint, spotlighting the amalgamation

of essential functions such as accurate path navigation, object recognition, adaptable object manipulation, and enhanced remote operational features.

3.0. PARTS AND EXPLANATION

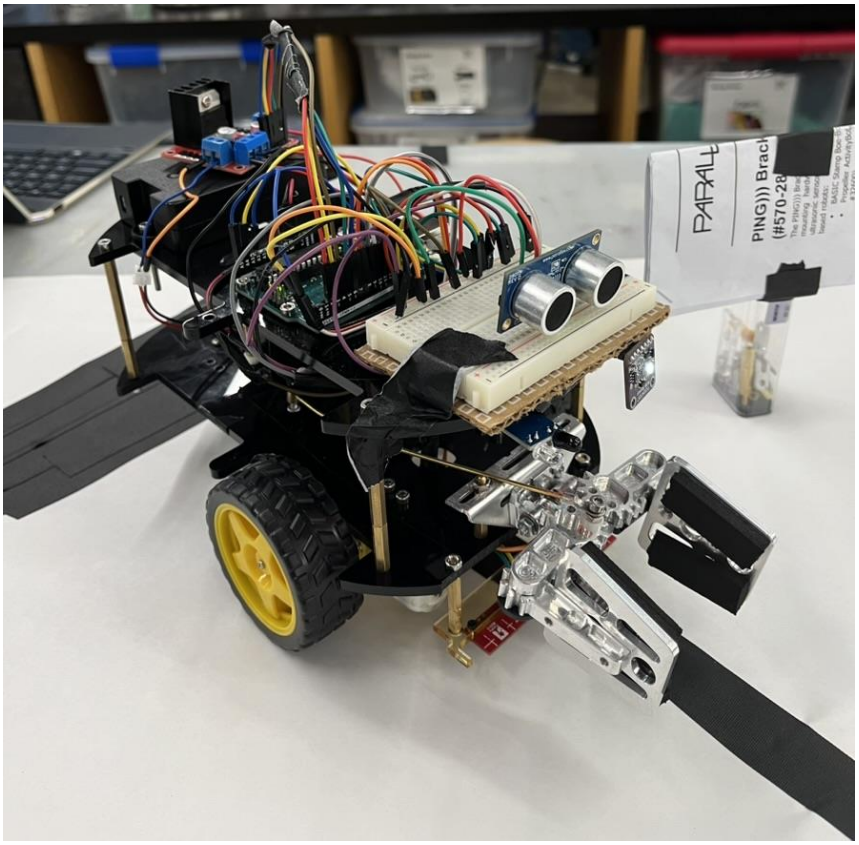

Fig 3.1 An assembled Sorter Bot

3.1 Arduino Microcontroller:

At the heart of the project lies the Arduino board, a pivotal tool in the creation of digital and interactive ventures. Equipped with versatile input/output (I/O) pins, it facilitates connectivity with an array of sensors and actuators, supporting a broad spectrum of applications. The Arduino Integrated Development Environment (IDE) allows for straightforward programming, offering an accessible platform for both novices and experts engaged in prototyping and educational initiatives.

3.2. Gripper Mechanism

A key feature of the sorter is its claw-like gripper mechanism, engineered for the precise acquisition, retention, and release of items. This modular system is driven by actuators such as servo motors and is adept at handling a diverse array of object sizes and materials, showcasing its utility in various settings from production lines to complex environments like surgical procedures.

3.3. RGB Color Sensor:

The inclusion of an RGB color sensor allows for the real-time identification and differentiation of colors by measuring the intensity of reflected light in red, green, and blue spectrums. Its application is critical in quality control, sorting processes, and robotic tasks that require accurate color detection.

3.4 DC Motors

DC motors translate direct electrical energy into mechanical motion and are instrumental across a plethora of applications, from miniature models to substantial industrial equipment. Their operation is based on electromagnetic principles, with a design that offers high torque at low speeds, essential for robotics and automated machinery.

3.5 QTR 8RC Sensor:

This sensor array comprises eight infrared reflectance sensors, optimized for line-following and edge-detection tasks within robotic frameworks. It determines line presence or edges by measuring infrared light reflection, providing essential feedback for precise navigation.

3.6 L298N H-Bridge Motor Driver:

The L298N Dual H-Bridge Motor Driver module facilitates the independent operation of two DC motors, supporting both forward and backward movements. Its H-Bridge configuration is key for reversing motor supply voltage polarity, crucial for powering diverse automated and robotic applications.

3.7 IR Reflective Sensor:

Utilising infrared light, the IR Reflective Sensor identifies object presence and gauges distances by analysing the intensity of reflected IR light. Its non-contact detection capability renders it an optimal choice for proximity sensing, obstacle avoidance, and line-following roles in automation and robotics.

3.8 Ultrasonic Sensor:

The Ultrasonic Sensor quantifies object distances by emitting ultrasonic waves and timing the echo return, essential for precise, non-intrusive measurements. It's a staple in robotics, vehicle parking aids, and obstacle detection systems, offering reliable performance under various conditions.

3.9 Li-ion Battery:

Characterised by their high energy density and longevity, Lithium-ion batteries are the project's power source, facilitating the movement of lithium ions between electrodes through an electrolyte. Their widespread use in portable electronics, electric vehicles, and energy storage solutions speaks to their efficiency and robustness.

4.0. CODE EXPLANATION

The project's functionality hinges on a well-structured code that controls a line-following robotic sorter through an Arduino microcontroller. This system integrates QTR sensors for precise path tracking, alongside an ultrasonic sensor for effective obstacle detection, ensuring the robot navigates seamlessly within its operational framework. Upon encountering an object, a servo-actuated gripper is engaged for object retrieval. Object presence is verified using an infrared sensor, and color classification is determined through the TCS34725 color sensor. This classification dictates the sorting process, wherein objects identified as red are sorted to one side, while objects without color are sorted to the opposite side. The sorting logic is meticulously executed through movements controlled by L298N motor drivers, showcasing a sophisticated integration of sensor technology

and control algorithms. This approach not only enhances the efficiency of the sorting process but also demonstrates the potential of integrating advanced robotics into automation systems, contributing significantly to the field of industrial automation.

13. APPENDIX

```
// linefollower
#include <QTRSensors.h>
#include <PID_v1.h>
double Setpoint;
double Input;
double Output;
QTRSensors qtr;
uint16_t position;

const uint8_t SensorCount = 4;
uint16_t sensorValues[SensorCount];

float Kp = 0.9;
float Ki = 0.025;
float Kd = 0.425;
```

```
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);

float Pvalue;
float Ivalue;
float Dvalue;

int counter;
int speedAdjustment;
int totalError = 0;
int previousError = 0;
int P, D, I, PIDvalue, error;
int lsp, rsp;
int lfspeed = 90;


// dc motors
#include <L298N.h>


#define AIN1 2  //Assign the motor pins
#define BIN1 4
#define AIN2 3
#define BIN2 5
#define PWMA 6
#define PWMB 11

const int offsetA = 1;
const int offsetB = 1;

L298N motor1(PWMA, AIN1, AIN2);
L298N motor2(PWMB, BIN1, BIN2);

// Gripper section
#include <Servo.h>
```

```
Servo servo;

// ultrasonic sensor
long duration, distance;
#define SENSOR_PIN 12 // Define the pin connected to the sensor's SIG pin

// IR sensor
const int irPin = A0;
int obstacleState = 0;
int confirm=0;

// color sensor
#include <Wire.h>
#include "Adafruit_TCS34725.h"

#define TCS34725_ADDR (0x29) // I2C address
#define SCL_PIN A5
#define SDA_PIN A4

Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_ADDR);
int colorValue;


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

// servo setup
  servo.attach(A2);  // Attach gripper servo to pin 6. This line isnt making the robot move

  // ultrasonic setup
  pinMode(SENSOR_PIN, OUTPUT); // Set the sensor pin as an output to send the pulse

  // ir sensor setup
  pinMode(irPin, INPUT); // Sets the irPin as an Input for IR sensor
```

```
 // configure the sensors for line following
 Setpoint = 0;
 myPID.SetMode(AUTOMATIC);
 myPID.SetTunings(Kp, Ki, Kd);
 myPID.SetOutputLimits(-90, 90);
 qtr.setTypeRC();
 // qtr.setTypeAnalog();
 qtr.setSensorPins((const uint8_t[]){7,8,9,10}, SensorCount);

 delay(500);
 pinMode(LED_BUILTIN, OUTPUT);
 digitalWrite(LED_BUILTIN, HIGH); // turn on Arduino's LED to indicate we are in calibration mode

 // for (uint16_t i = 0; i < 400; i++)
 // {
 //   qtr.calibrate();
 // }

   delay(2000);
for (int i = 0; i < 5; i++) { // Repeat the turning and calibrating process
   // Turn gradually to the right and calibrate
   motor_drive(100, -95); // Adjust these values as needed for your robot
   delay(250); // Adjust delay as needed for the turn amount
   motor1.stop();
   motor2.stop();
   qtr.calibrate();

   // Turn back to center and calibrate
   motor_drive(-100, 95); // Reverse direction to return to center
   delay(250); // Adjust delay as needed for the turn amount
   motor1.stop();
   motor2.stop();
   qtr.calibrate();
```

```
  // Turn gradually to the left and calibrate
  motor_drive(-100, 95); // Adjust for a left turn
  delay(250); // Adjust delay as needed for the turn amount
  motor1.stop();
  motor2.stop();
  qtr.calibrate();

  // Turn back to center and calibrate
  motor_drive(100, -95); // Reverse direction to return to center
  delay(250); // Adjust delay as needed for the turn amount
  motor1.stop();
  motor2.stop();
  qtr.calibrate();
 }

 digitalWrite(LED_BUILTIN, LOW); // turn off Arduino's LED to indicate we are through with calibration
 Serial.println();
 Serial.print("Calibration Complete!");
 Serial.println();
 delay(2000);
 openGripper();
}

void loop() {
 ultraSonicObstacleCheck();

 counter = 0;
 confirm=0;

int checkDistance=0;
 while (distance > 4 && checkDistance < 400 ) {
    robot_control();
    ultraSonicObstacleCheck();
```

```
    checkDistance +=1;
    Serial.println(checkDistance);
    if(checkDistance >= 400){
      motor1.stop();
      motor2.stop();
    while(1){
      // infinite loop to stop the car
    }
    }

  }

  pickItem();
  delay(1000);

  confirmItem();
  delay(1000);

if(confirm){
turn90DegreesRight();
  delay(1000);

  // Keep turning until a black line is detected
    bool lineDetected = false;
    while (!lineDetected) {
      lineDetected = keepturning(); // This will stop turning once a black line is detected
      delay(100); // Adjust delay as needed for responsiveness
    }

    while (counter < 5){
      robot_control();
      if (sensorValues[0]>=900 && sensorValues[1]>=900 && sensorValues[2]>=900 && sensorValues[3]>=900){
        counter += 1;
        delay(50);
```

```
    }
    Serial.println(counter);
  }
  dropItem();
}
else {
  openGripper();
  delay(1000); // Delay to ensure the gripper has enough time to open

  while (checkDistance < 400) {
    robot_control();
    ultraSonicObstacleCheck();
    checkDistance += 1;
    Serial.println(checkDistance);

    if (checkDistance >= 400) {
      motor1.stop();
      motor2.stop();
      while(1) {
        // Infinite loop to halt further execution
      }
    }
  }
 }
}



void pickItem(){
  motor1.stop();
  motor2.stop();
  delay(1000);
  closeGripper();
  delay(1000);
```

```
}

void confirmItem(){
   int detectionCount = 0;
   const int totalReadings = 10; // Take 5 readings
   const int detectionThreshold = 7; // Require at least 3 detections to confirm

   for (int i = 0; i < totalReadings; i++) {
      IRSensorObstacleCheck(); // Update obstacleState
      if (obstacleState == LOW) { // Assuming LOW means an object is detected
         detectionCount++;
      }
      delay(200); // Wait a bit before the next reading
   }

   if (detectionCount >= detectionThreshold) {
      confirm = 1;
      Serial.println("Object confirmed.");
   } else {
      confirm = 0;
      Serial.println("No object detected.");
   }
}

void dropItem(){
  motor1.stop();
  motor2.stop();
  delay(1000);
  colorCheck();
  if(colorValue==0){
    //green
    turn90DegreesRight();
    delay(1000);
    moveForwardShortDistance();
```

```
  delay(1000);
  openGripper();
  delay(1000);
  moveBackwardShortDistance();
  delay(1000);
  turn90DegreesRight();
  delay(1000);
  moveForwardShorterDistance();
  delay(1000);
 }
 else{
  //red
  turn90DegreesLeft();
  delay(1000);
  moveForwardShortDistance();
  delay(1000);
  openGripper();
  delay(1000);
  moveBackwardShortDistance();
  delay(1000);
  turn90DegreesLeft();
  delay(1000);
  moveForwardShorterDistance();
  delay(1000);
 }
}


void moveForwardShortDistance() {
 motor_drive(120, 120); // Move forward
 delay(350); // Short delay to move a small distance, adjust as needed
 motor1.stop(); // Stop motors
 motor2.stop();
}
```

```
void moveForwardShorterDistance() {
  motor_drive(110, 110); // Move forward
  delay(350); // Short delay to move a small distance, adjust as needed
  motor1.stop(); // Stop motors
  motor2.stop();
}

void moveBackwardShortDistance() {
  motor_drive(-120, -120); // Move Backward
  delay(350); // Short delay to move a small distance, adjust as needed
  motor1.stop(); // Stop motors
  motor2.stop();
}

bool keepturning() {
  motor_drive(100, 0); // Adjust speed as needed
  delay(200); // Adjust timing for smoother operation
  position = qtr.readLineBlack(sensorValues);
  motor1.stop();
  motor2.stop();
  delay(50); // Short delay to stabilize sensor readings

  // Check if any sensor reads above a threshold, indicating a black line
  for(int i = 0; i < SensorCount; i++) {
    if (sensorValues[i] > 600) { // Threshold value, adjust based on your sensor calibration
      return true; // Line detected
    }
  }
  return false; // Line not detected, continue turning
}

void turn90DegreesRight() {
  motor_drive(125, -125);
```

```
  delay(350); // Adjust timing based on the speed and turning radius of your robot
  motor1.stop();
  motor2.stop();
}

void turn90DegreesLeft() {
  motor_drive(-115, 115);
  delay(350); // Adjust timing based on the speed and turning radius of your robot
  motor1.stop();
  motor2.stop();
}



// gripper close function
void closeGripper(){
  servo.write(0);
}

// gripper Open function
void openGripper(){
  servo.write(180);
}

void ultraSonicObstacleCheck() {
  digitalWrite(SENSOR_PIN, LOW);
  delayMicroseconds(2);

  // Send a 10-microsecond pulse to start the measurement
  digitalWrite(SENSOR_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(SENSOR_PIN, LOW);

  // Change the pin to input to receive the echo
  pinMode(SENSOR_PIN, INPUT);
```

```
  // Measure the time for the echo to return
  duration = pulseIn(SENSOR_PIN, HIGH);

  // Calculate the distance
  distance = duration / 58.2; // Convert the time to distance in cm

   // Print the distance to the Serial Monitor
  // Serial.print("Distance: ");
  // Serial.print(distance);
  // Serial.println(" cm");

  // Reset the pin to output after reading
  pinMode(SENSOR_PIN, OUTPUT);

}

// IR sensor routine
void IRSensorObstacleCheck(){
  obstacleState = digitalRead(irPin);
  //low means object detected
  Serial.println(obstacleState);
}
//
void robot_control(){
  // read calibrated sensor values and obtain a measure of the line position
  // from 0 to 4000 (for a white line, use readLineWhite() instead)
  delay(15);
  position = qtr.readLineBlack(sensorValues);
  // Serial.print(position);
  // Serial.print("    ");
  // error = 1500 - position;

  Input = map(position, 0, 3000, -90, 90);
```

```
  // Serial.print(Input);
  // Serial.print("    ");

  myPID.Compute();
  speedAdjustment = Output;

  lsp = lfspeed + speedAdjustment;
  rsp = lfspeed - speedAdjustment;
  // PID_Linefollow(error);
  // Serial.print(speedAdjustment);
  // Serial.print("    ");

  if (lsp > 90) {
    lsp = 90;
  }
if (lsp < -90) {
    lsp = -90;
  }
  if (rsp > 90) {
    rsp = 90;
  }
  if (rsp < -90) {
    rsp = -90;
  }
  motor_drive(lsp,rsp);
  Serial.print(lsp);
  Serial.print("    ");
  Serial.print(rsp);
  Serial.println();

}


void motor_drive(int left, int right){
```

```
  motor1.setSpeed(abs(left)); // Set speed for left motor
  motor2.setSpeed(abs(right)); // Set speed for right motor

  if(right>0)
  {
    motor2.forward();
  }
  else
  {
    motor2.backward();
  }


  if(left>0)
  {
    motor1.forward();
  }
  else
  {
    motor1.backward();}
}

void colorCheck(){
   uint16_t clear, red, green, blue;

  tcs.getRawData(&red, &green, &blue, &clear);

  int colorTemp = tcs.calculateColorTemperature(red,green,blue);

if(colorTemp <= 3000) {
  colorValue=1; //indicating red
  // Serial.print(colorValue);
  // Serial.println("Red");
}
```

```
else {
  colorValue=0; //indicating yellow
  // Serial.print(colorValue);
  // Serial.println("Green");
  }
}
```