

Contents

Declaration.....	Error! Bookmark not defined.
Abstract.....	Error! Bookmark not defined.
Acknowledgements.....	Error! Bookmark not defined.
Contents	0
List of Figures	5
1. Introduction.....	0
1.1 Aims and Objectives	0
• Implementation of autonomous navigation:	6
• Incorporation of QR Code Recognition:.....	6
• Integration of Robotic Arm:.....	6
• User Control.....	6
• Effective Project Management.....	6
1.2 Project Deliverables	6
Hardware Prototypes:.....	6
Block Diagrams:	6
Software Code:.....	6
Simulation Results:	7

Project Report:	7
1.3 Project Plan	7
Project Initiation and Research (Week 1 - Week 4)	7
Design and Specification (Week 5 - Week 8).....	7
Procurement of Components and Interim Report (Week 9 - Week 12)	7
Prototype Development - Hardware (Week 13 - Week 16).....	8
Software Development (Week 17 - Week 20).....	8
Integration and Testing (Week 21 - Week 24).....	8
Documentation, Project Review, Submission, and Post-Project Review (Week 25 - Week 27).....	8
1.4 Risk Analysis.....	8
Hardware component failure.....	8
Software bugs and glitches	9
Insufficient technical knowledge	9
Schedule overrun	9
Difficulty in QR code recognition	9
1.5 Legal, Social, Ethical and Professional Issues	10
Social Implications:	10
Legal Implications:	10

Ethical and Professional Issues:.....	10
1.6 Report Structure	11
Section 1.....	11
Section 2.....	11
Section 3.....	11
Section 4.....	11
Section 5.....	11
Section 6.....	12
Section 7.....	12
2. Literature Review.....	14
3. Technical Background	15
3.1 Robotics	15
3.2 Computer Vision.....	15
3.3 Raspberry Pi.....	15
3.4 Robotic Arm Control	15
3.5 GPIO and Motor Control	15
3.6 QR Code Recognition	16
3.7 Future Technologies: Machine Learning and AI.....	16

4. Design	17
4.1 Robotic car hardware:	17
4.2 Robotic Car	20
4.3 Robotic Arm.....	20
4.4 Color Detection	20
4.5 QR Code Recognition	20
4.6 Control System.....	20
4.7 Manual control	21
4.8 Block diagram	21
4.9 Flowchart	22
5. Implementation	23
5.1 Hardware implementation:.....	23
5.2 Software implementation	23
QrCode recognition.....	24
Color Detection.....	25
Servo motor functions.....	27
Dc motor functions	29
Distance detection	31

Delay function.....	32
Check user input function	33
Direction car function	33
Main loop	34
Testing.....	36
Colour detection.....	37
Auto mode.....	39
Manual mode:	48
6. Results and Discussion	51
6.1 Discussion of Results	52
6.2 Project Management and Progress	53
7. Conclusions and Further Work	55
7.1 Conclusions.....	55
7.2 Suggestions for Further Work.....	56
Appendix A.....	57

List of Figures

Figure 1RoboticCar.....	17
Figure 2Raspberry.....	17
Figure 3SmartCardBoard.....	18
Figure 4Servo.....	18
Figure 5DcMotor.....	19
Figure 6UltrasonicSensor.....	19
Figure 7PiCamera.....	19
Figure 8Battery.....	19
Figure 9Block diagram.....	21
Figure 10Flowchart.....	22
Figure 11RedObjectDetection.....	37
Figure 12YellowObjectDetection.....	38
Figure 13GreenObjectDetection.....	39
Figure 14Spinning.....	40
Figure 15GreenBasketQR.....	41
Figure 16CentralQR.....	42
Figure 17LeftQR.....	43
Figure 18RedBasketQR.....	44
Figure 19Yellow basket QR.....	45
Figure 20Collect QR.....	46
Figure 21Distance<10.....	47
Figure 22ManualMode.....	48
Figure 23ManualModeMovements.....	48
Figure 24InvalidChoice.....	49
Figure 25AutoMode.....	49

1.1 Aims and Objectives

- **Implementation of autonomous navigation:** This involved enabling the robotic car to move and navigate autonomously within a given space, utilizing sensory inputs to avoid obstacles.
- **Incorporation of QR Code Recognition:** The objective here was to enable the robotic car to recognize QR codes and perform specific tasks based on the information encoded in the QR codes. This functionality was expected to guide the robotic car's navigation and actions.
- **Integration of Robotic Arm:** The goal was to equip the car with a functioning robotic arm capable of basic interaction with objects, primarily for the purpose of picking up and releasing objects.
- **User Control:** While maintaining the emphasis on autonomy, the project also aimed to provide a user control option, allowing manual intervention and direction when necessary.
- **Effective Project Management:** Lastly, an objective that spanned the entire project was effective project management, involving the careful planning, execution, and adaptation of tasks and goals in a dynamic development environment. This included maintaining a clear project timeline and regularly assessing progress against the initial plan.

1.2 Project Deliverables

Hardware Prototypes: The final, functional model of the autonomous robotic car, fitted with an integrated robotic arm.

Block Diagrams: Comprehensive diagrams outlining the electronic and wiring configuration of the robotic car, showing the connections between the different components such as the motors, sensors, and controller.

Software Code: The complete program code required to control and coordinate the activities of the robotic car. This included modules for distance detection, QR code recognition, robotic arm operation, and user control.

Simulation Results: This would document the results of various simulated scenarios run to test the robotic car's abilities in a controlled environment. These simulations would ideally demonstrate the capabilities of the car in navigation, object interaction, and QR code recognition.

Project Report: This report would document the entire process of the project, from conception and planning through to execution and evaluation. It would include all project details such as methodology, results, and conclusions.

1.3 Project Plan

Project Initiation and Research (Week 1 - Week 4)

Week 1-2: Understanding project requirements and objectives. Researching on autonomous vehicles, robotic arms, QR code recognition, and their applications.

Week 3-4: Identification of necessary technologies, tools, and components.

Design and Specification (Week 5 - Week 8)

Week 5-6: Detailing out system specifications. Creating circuit design for the robotic car and the robotic arm.

Week 7-8: Outlining software requirements for autonomous control and QR code recognition.

Procurement of Components and Interim Report (Week 9 - Week 12)

Week 9: Acquiring all necessary hardware components, including motors, sensors, controller, QR reader, etc.

Week 10-12: Prepare and submit the interim report detailing the project progress up to this point, including project initiation, research, design, and specifications.

Prototype Development - Hardware (Week 13 - Week 16)

Week 13-16: Assembling the robotic car and the robotic arm as per the design. Establishing electrical connections as per the circuit diagram.

Software Development (Week 17 - Week 20)

Week 17-20: Coding the control program for autonomous navigation, QR code recognition, and robotic arm operation. Implementing user interface for manual control. Conducting unit testing and debugging.

Integration and Testing (Week 21 - Week 24)

Week 21-24: Integrating hardware and software components. Conducting system testing, including QR code recognition, and robotic arm functions. Addressing identified issues and refining the system.

Documentation, Project Review, Submission, and Post-Project Review (Week 25 - Week 27)

Week 25-26: Preparing project report, detailing the project process, results, and conclusions.

Week 27: Conducting a final review of project outcomes against the set objectives. Making necessary adjustments and refinements. Submitting the project report and delivering the final presentation.

1.4 Risk Analysis

Main risks associated with this project, the likelihood of their occurrence, and the mitigation strategies used to minimize these risks:

Hardware component failure

- Probability: Medium

- Mitigation Strategy: All components were tested upon procurement to ensure their functionality. Spares were also kept on-hand to quickly replace any faulty parts.

Software bugs and glitches

- Probability: High
- Mitigation Strategy: A robust testing and debugging methodology was employed, ensuring any bugs were promptly identified and fixed. Good coding practices were also followed to minimize the introduction of errors.

Insufficient technical knowledge

- Probability: Medium
- Mitigation Strategy: Detailed research was conducted to thoroughly understand all technologies involved in the project. Regular consultations with a supervisor and experienced colleagues helped in dealing with complex technical issues.

Schedule overrun

- Probability: High
- Mitigation Strategy: A well-structured project plan was implemented, and progress was continuously monitored. Potential bottlenecks were identified early, allowing for proactive schedule adjustments.

Difficulty in QR code recognition

- Probability: Medium
- Mitigation Strategy: Various QR code recognition libraries and tools were evaluated to ensure the most reliable one was used. Additional time was allocated for this task, understanding its complexity and the potential challenges it posed.

1.5 Legal, Social, Ethical and Professional Issues

Social Implications:

The primary social implication of this project lies in its potential to automate manual tasks, reducing physical exertion and risk for human workers. However, while this provides benefits such as increased efficiency and potentially improved safety, there may also be negative implications. These could include job displacement due to automation or privacy concerns if the robotic system is used in public or semi-public spaces.

Legal Implications:

The project stick to various legal considerations. Respect for intellectual property is maintained throughout, with no copyright or patent infringements. All used libraries and software components are either open source or used under appropriate licensing. In terms of hardware, the project complies with the RoHS directive, ensuring that all electronic components used do not contain hazardous substances.

Ethical and Professional Issues:

This project was conducted with high ethical and professional standards. All decisions and actions were guided by respect for privacy, intellectual property, and the overall well-being of individuals who could be affected by the project's outcome.

Furthermore, the potential use of this technology was considered carefully to avoid any applications that could harm individuals or infringe on their rights. For instance, the autonomous system is designed not to collect or store any personal data unless explicitly permitted by all relevant parties.

1.6 Report Structure

The structure of this report has been designed to provide a comprehensive and logical progression through the various stages of the project.

Section 1

An introduction to the project is provided, detailing its aims and objectives, deliverables, project plan, risk analysis, and discussion on the legal, social, ethical and professional issues relevant to the project.

Section 2

Presents a literature review, discussing previous research and developments in the field that have direct relevance to the project.

Section 3

The Technical Background provides a detailed understanding of the technical concepts and theories that underpin the project, which are fundamental to the design and implementation.

Section 4

Details the approach taken to designing the solution, from initial concepts to the finalised design. This includes the considerations and decisions made along the way.

Section 5

Covers the implementation of the project, documenting the process of turning the design into a working, tangible system.

Section 6

Discusses the results obtained from the project and provides an analysis and discussion of these results. It also includes a self-reflection on the project management aspects of the work, discussing progress and any challenges encountered and how they were managed.

Section 7

Presents the conclusions drawn from the work and suggests potential paths for further development and research. This includes a discussion on the project's successes and areas for improvement, and suggestions for future research directions.

1. Introduction

The project aimed to design a robotic car with advanced capabilities in recognizing, grabbing, and sorting colored balls into their corresponding baskets using QR codes as reliable guiding indicators. The underlying concept of this endeavor involved developing an autonomous system capable of efficiently performing color-based sorting tasks, with potential applications in diverse sectors like manufacturing, waste management, and inventory management.

The robotic car was meticulously engineered to discern colored balls and their respective baskets by utilizing an integrated camera that decodes QR codes and identifies colors. Once the car detects a colored ball within its gripper, it swiftly scans for the QR code associated with that color. Upon successfully identifying the correct QR code, the car autonomously navigates to the designated basket to deposit the ball.

Significant progress has been made in the development of the car's color and QR code recognition capabilities, as well as its autonomous navigation. However, the current design necessitates manual intervention for retrieving balls from the collection basket. This aspect becomes the focal point for future enhancements, with the ultimate goal of achieving a fully automated sorting process.

This report comprehensively outlines the design, implementation, and results of the project, while also addressing the challenges encountered during the development process. Furthermore, it discusses potential avenues for improvement and highlights the scope of future work required to fully accomplish the project's objectives.

2. Literature Review

The primary focus of this investigation turns around the automation of tasks conventionally executed by humans, specifically through the deployment of machine-learning algorithms and color recognition technologies. The logic behind this inclination towards automation has been thoroughly debated in numerous studies and industrial reports, with a recurring theme centered around the fear of job displacement by machines.

The Applied Robotics module has played a vital role in both conceptualizing and implementing this project, offering valuable theoretical frameworks and practical applications. The module's coverage of automation and car-like robot manipulation offered a robust foundation for the development of the project. Moreover, the Computer Vision module, which provided comprehensive knowledge on color detection algorithms and QR code recognition, has been a main reference point.

Although the project does not introduce a new concept in automation, it emphasizes an important and often overlooked aspect: automating basic tasks that are still performed manually in certain industries. While there are numerous studies and commercial applications that have explored task automation, the endeavor here is to refine the approach and present a possible solution to a specific problem.

The issue of job displacement, which is a significant concern associated with automation implementation, has been a subject of intense debate in recent years. This debate has encompassed various perspectives, including ethical, economic, and social considerations.

It is important to acknowledge that although the project draws inspiration from existing literature and applies established knowledge, it does not directly address a specific gap in the current body of knowledge. Instead, it utilizes well-established concepts within a specific practical context, reinforcing the relevance of these concepts in real-world applications.

3. Technical Background

This section presents an exploration into the pivotal technology that underpins this innovative project.

3.1 Robotics

The fascinating realm of robotics takes center stage in this project, primarily through the utilization of a versatile mobile robot. This robot, equipped with a robust robotic arm, serves as the core operational entity, capable of grabbing and relocating colored balls with efficiency and precision.

3.2 Computer Vision

Computer vision is another integral aspect of this project. Employed to recognize different colors and identify QR codes, this discipline of artificial intelligence mimics the complexity of human vision using digital images from cameras, videos, and deep learning models. Through OpenCV, a popular computer vision library, the robot can detect and discern the different colored balls and corresponding QR codes associated with each basket.

3.3 Raspberry Pi

Raspberry Pi plays a main role as the project's computing platform. This small, single-board computer runs the Python program responsible for controlling the robot, processing the camera input, and making autonomous decisions based on the processed data.

3.4 Robotic Arm Control

Servo motors, under the control of the Raspberry Pi, drive the movements of the robotic arm. The pulse-width modulation (PWM) technique is used to control the servo motors, providing the precision necessary for the robot to pick up and deposit balls effectively.

3.5 GPIO and Motor Control

The Raspberry Pi General Purpose Input/Output (GPIO) pins are used for interfacing with external electronic devices, like the motors that drive the robot. A motor driver circuit is implemented to amplify the control signals from the GPIO pins, controlling the speed and direction of the robot.

3.6 QR Code Recognition

QR code recognition, an integral part of this project, is achieved using the Pyzbar library. This library is capable of reading QR codes and returning the associated data. In this project, this feature is utilized to identify the appropriate baskets corresponding to the color of the ball that the robot is holding.

3.7 Future Technologies: Machine Learning and AI

Looking ahead, the integration of advanced technologies like machine learning and artificial intelligence (AI) holds great promise for the project. These transformative technologies will play a pivotal role in automating the ball retrieval process from the collection basket, further enhancing the autonomy of the robotic car. By leveraging machine learning algorithms, the robot can learn to handle balls of varying sizes and weights, continuously improving its grasp and manipulation techniques over time. Additionally, AI techniques will enable the robot to make intelligent decisions in complex scenarios, enhancing adaptability and efficiency in real-world sorting applications. The integration of machine learning and AI technologies represents a significant advancement for the project. As these cutting-edge technologies continue to be explored and developed, there is enormous potential for automating sorting processes and expanding the range of applications across various industries.

4. Design

4.1 Robotic car hardware:

The robotic hardware comes from a company called Freenove, it equipped with a plastic car body, 4 wheels, a plastic robotic arm which can go up and down and also a gripper which can close and open the mechanism.

Robotic car picture:



Figure 1RoboticCar

It is equipped with the following electronic components:

- Raspberry Pi3



Figure 2Raspberry

- Tank Smart Card Board (Motors driver)

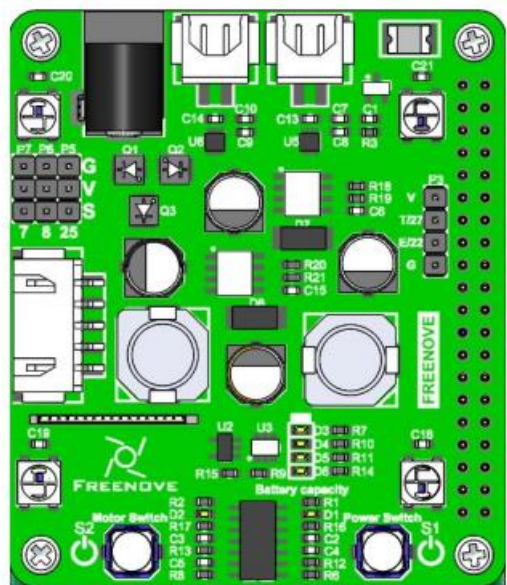


Figure 3SmartCardBoard

- Servo motor SG90 x2



Figure 4Servo

- DC motor x2



Figure 5DcMotor

- Ultrasonic sensor HC-SR04



Figure 6UltrasonicSensor

- Pi camera module



Figure 7PiCamera

- Battery 18650 x2



Figure 8Battery

4.2 Robotic Car

The mobile robot platform forms the base of the design. The robotic car is equipped with wheels and motors for mobility. Each motor is connected to the wheels for locomotion, while the control of the motors is achieved via the Raspberry Pi using a motor driver circuit.

4.3 Robotic Arm

The robotic arm, the primary actor for the sorting task, is situated on the robotic car. The arm has a gripper designed to hold and transport colored balls. The movements of the arm are controlled by servo motors, allowing it to effectively grab the balls.

4.4 Color Detection

Color detection is a central part of the design. A camera module interfaced with the Raspberry Pi is used to identify the colors of the balls. OpenCV, an open-source computer vision library, is used to process the images from the camera and detect the colors of the balls.

4.5 QR Code Recognition

QR code recognition is another critical aspect of the design. This functionality, facilitated by the Pyzbar library, is used to identify the specific baskets into which the balls must be deposited. The camera module captures images of the QR codes placed on or near the baskets, which are then processed to obtain the associated color information.

4.6 Control System

The control system is the brain of the operation. This system, running on the Raspberry Pi, manages the movement of the robotic car, controls the robotic arm, and processes the inputs from the camera module. Based on the color of the ball in

possession and the QR code information, the system determines the basket to which the ball should be transported.

The robotic car it's also able to detect the distance between the basket and it, using an ultrasonic sensor.

4.7 Manual control

The automatic picking up of the ball is currently unavailable, for this reason the initial part of the process, where the car has to pick the ball from the collect basket, at the moment is done manually.

4.8 Block diagram

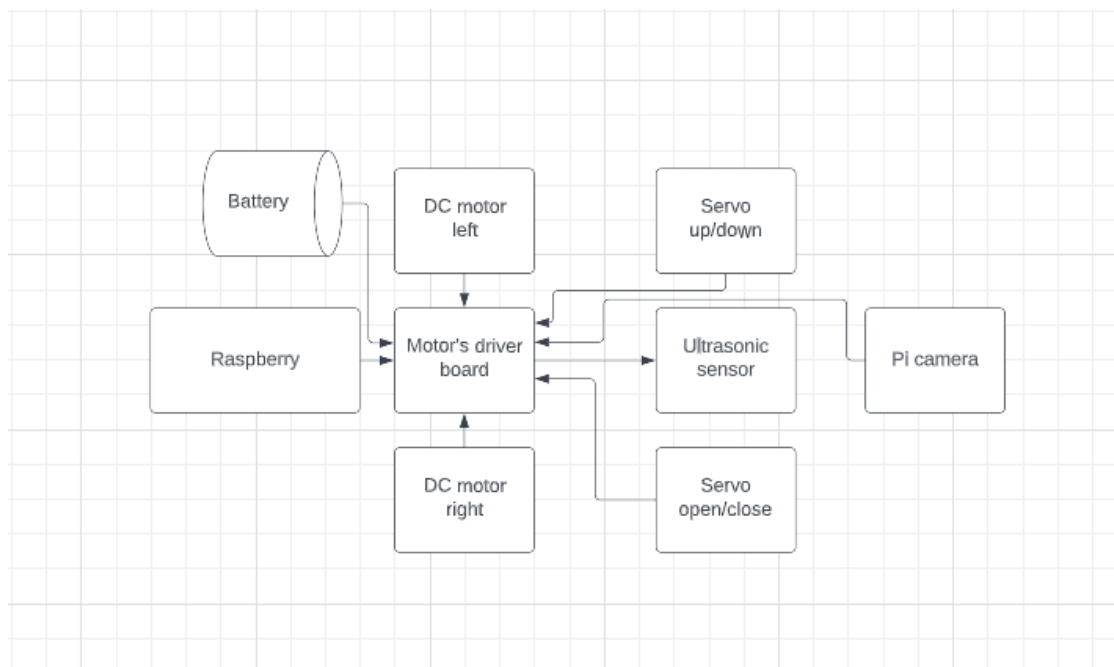


Figure 9Block diagram

4.9 Flowchart

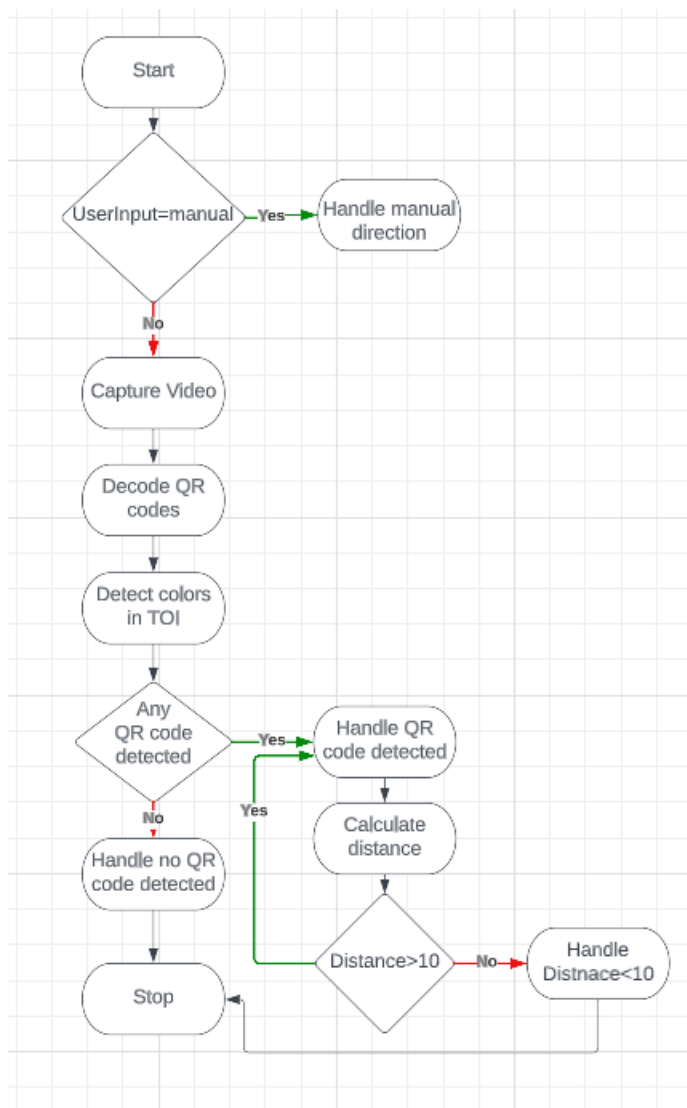


Figure 10Flowchart

5. Implementation

5.1 Hardware implementation:

The hardware implementation phase of this project focused on assembling and configuring the Freenove Robotic Tank. The task involved constructing the chassis, installing and wiring the motors, connecting the sensors, and setting up the Raspberry Pi microcontroller.

The construction of the tank began with the assembly of the chassis, which served as the base structure for all other components. This multi-layered structure was carefully assembled according to the provided instructions, ensuring that each part was correctly aligned.

Next, the motors were installed onto the chassis. This required careful placement and secure fastening to ensure smooth, consistent movement once powered. The wheels were then attached to the motor shafts, again ensuring secure connections to maintain consistent performance during operations.

The next stage of the hardware implementation involved integrating the Raspberry Pi microcontroller onto the chassis. The Raspberry Pi served as the central control unit for the robot, coordinating sensor input and motor output. The controller was affixed to the top layer of the chassis to protect it from potential damage and to allow easy access for wiring and troubleshooting.

The Raspberry Pi was then connected to the Smart card board, which has the function of controlling the various motors and sensors, the various components of the robot were then connected to the Smart board card.

5.2 Software implementation

In the following section the various part of the code will be explained:

QrCode recognition

```
# Start Camera Code
def decode(image):
    # find and decode QR codes in the image
    decoded_objects = pyzbar.decode(image)

    # print the type and data of each QR code
    for obj in decoded_objects:
        print("Type:", obj.type)
        print("Data:", obj.data.decode("utf-8"), "\n")

    return decoded_objects

# Capture video from the webcam
cap = cv2.VideoCapture(0)

def QrCode():
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Rotate the frame 180 degrees
    frame = cv2.rotate(frame, cv2.ROTATE_180)

    # Get the frame width
    frame_width = frame.shape[1]

    # Apply operations on the frame
    decoded_objects = decode(frame)

    position = "None"

    # Display the resulting frame
    for obj in decoded_objects:
        # Convert polygon points to numpy array
        points = np.array(obj.polygon, np.int32)
        points = points.reshape((-1, 1, 2))

        # Get the x-coordinate of the center of the QR code
        qr_center_x = obj.rect.left + obj.rect.width / 2

        # Determine position of QR code
        if qr_center_x < frame_width / 2 - 20: # Margin of 20 pixels
            print("QR code is on the left.")
            position = "Left"
        elif qr_center_x > frame_width / 2 + 20: # Margin of 20 pixels
            print("QR code is on the right.")
            position = "Right"
        elif qr_center_x > frame_width / 2 - 20 and qr_center_x < frame_width / 2 + 20:
            print("QR code is in the center.")
            position = "Center"

        # Check if the QR code data is "Collect"
        qr_data = obj.data.decode("utf-8")
        if qr_data == "Collect":
            print("The QR code says 'Collect'.")

        # Draw the QR code outline on the frame
        cv2.polylines(frame, [points], True, (0, 255, 0), 3)

        # Put the QR code data as text on the frame
        cv2.putText(frame, qr_data, (points[0][0][0], points[0][0][1] - 15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0),
            2)

    detect_color(frame)

    cv2.imshow('frame', frame)
    return position
```

The `decode(image)` function is defined to detect and decode QR codes within an image. The `pyzbar.decode(image)` function is used to find the QR codes in the image. For each QR code found, it prints the type and the decoded data. This function returns the decoded objects.

The `QrCode()` function is defined to process each frame of the video from the webcam. First, it captures a frame from the video using `cap.read()`. The frame is then rotated 180 degrees using `cv2.rotate(frame, cv2.ROTATE_180)`. The width of the frame is obtained using `frame.shape[1]`. The `decode(frame)` function is called on the frame to decode any QR codes present.

For each decoded QR code, the position is determined by examining the x-coordinate of the QR code's center. If the center's x-coordinate is less than half the frame width minus a margin of 20 pixels, it is considered to be on the left. If it's greater than half the frame width plus a margin of 20 pixels, it is considered to be on the right. If it falls within the margins, it is considered to be in the center, whenever a QR code is detected a green outline is drawn around the QR code.

Color Detection

```
def detect_color(frame):
    # Convert the image from BGR to HSV color space. This is done because HSV is more effective at color detection.
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Define the central region of interest (ROI).
    # We are interested in the color at the center of the frame.
    center_x, center_y = frame.shape[1] // 2, frame.shape[0] // 2 # Obtain the coordinates for the center of the image
    radius = 10 # Define the radius of the ROI
    mask_roi = np.zeros(frame.shape[:2], np.uint8) # Create a black image of the same size as the frame
    # Draw a white circle at the center of the mask. This circle is our ROI.
    cv2.circle(mask_roi, (center_x, center_y), radius, 255, -1)
    # Bitwise-AND the mask and the hsv image to isolate the color in the ROI.
    hsv_roi = cv2.bitwise_and(hsv, hsv, mask=mask_roi)

    # Draw the ROI circle on the original frame for visual purposes.
    cv2.circle(frame, (center_x, center_y), radius, (255, 255, 255), 1)

    # Define range for red color in HSV
    red_lower = np.array([0, 152, 76])
    red_upper = np.array([20, 255, 116])
    # Create a mask for the red color
    red_mask = cv2.inRange(hsv_roi, red_lower, red_upper)

    # Define range for green color in HSV
    green_lower = np.array([36, 25, 25])
    green_upper = np.array([86, 255, 255])
    # Create a mask for the green color
    green_mask = cv2.inRange(hsv_roi, green_lower, green_upper)

    # Define range for yellow color in HSV
    yellow_lower = np.array([20, 100, 100])
    yellow_upper = np.array([30, 255, 255])
    # Create a mask for the yellow color
    yellow_mask = cv2.inRange(hsv_roi, yellow_lower, yellow_upper)
```

```

# Check if any red pixels are present in the ROI.
if cv2.countNonZero(red_mask) > 0:
    print("Red")
    return "Red" # If red pixels are detected, return "Red"
# Check if any green pixels are present in the ROI.
elif cv2.countNonZero(green_mask) > 0:
    return "Green" # If green pixels are detected, return "Green"
# Check if any yellow pixels are present in the ROI.
elif cv2.countNonZero(yellow_mask) > 0:
    return "Yellow" # If yellow pixels are detected, return "Yellow"
else:
    # If no red, green or yellow pixels are detected in the ROI, return "None Colour"
    return "None Colour"

```

The detect_color(frame) function takes the frame as argument and detects whether there's a significant presence of red, green, or yellow color in the center of the frame.

The frame is converted from BGR color to HSV color using cv2.cvtColor(frame, cv2.COLOR_BGR2HSV). HSV (Hue, Saturation, Value) color space is often better for color detection because it separates the color and brightness information.

The central region of interest (ROI) is defined.

The HSV image is bitwise-ANDed with the mask to isolate the color in the ROI.

The function checks if there are any non-zero (i.e., white) pixels in each color mask in the ROI. This indicates that the color is present.

If red is present, the function returns "Red".

If green is present, it returns "Green".

If yellow is present, it returns "Yellow".

If none of these colors are detected, the function returns "None Colour".

Servo motor functions

```
# Define the pin number on the Raspberry Pi GPIO to which the first servo is connected.
SERVO_PIN_1 = 8

# Define the pin number on the Raspberry Pi GPIO to which the second servo is connected.
SERVO_PIN_2 = 25

# Connect to the local Raspberry Pi using the pigpio library. This creates an instance of the pigpio.pi class with
pi = pigpio.pi()

# Set the mode of the pins that the servos are connected to as OUTPUT. This means that these pins will be used to
pi.set_mode(SERVO_PIN_1, pigpio.OUTPUT)
pi.set_mode(SERVO_PIN_2, pigpio.OUTPUT)

def set_angle(servo_pin, angle):
    # This function sets the angle of the servo.
    # The Pulse Width Modulation (PWM) value that corresponds to the desired angle is calculated.
    # Note: The calculation used here depends on the specific type of servo. In this case, it is assumed that a
    pwm_value = int((float(angle) * 2000 / 180) + 500)
    # The PWM value is then used to set the pulse width for the specified servo pin.
    pi.set_servo_pulsewidth(servo_pin, pwm_value)

def up():
    # This function makes the first servo move to the 'up' position, which is defined as an angle of 170.
    set_angle(SERVO_PIN_1, 170)

def down():
    # This function makes the first servo move to the 'down' position, which is defined as an angle of 140.
    set_angle(SERVO_PIN_1, 140)

def open():
    # This function makes the second servo move to the 'open' position, which is defined as an angle of 50.
    set_angle(SERVO_PIN_2, 50)

def close():
    # This function makes the second servo move to the 'close' position, which is defined as an angle of 100.
    set_angle(SERVO_PIN_2, 100)
```

This portion of code controls two servo motors using the Raspberry Pi's General Purpose Input Output (GPIO) pins and the pigpio library.

The first two lines define the GPIO pin numbers to which the two servo motors are connected. In this case, servo 1 is connected to GPIO pin 8, and servo 2 is connected to GPIO pin 25. An instance of the pigpio.pi class is created, which represents the local Raspberry Pi. This instance is used to control the GPIO pins. The GPIO mode for the pins connected to the servos is set to OUTPUT.

The function set_angle() is defined. This function takes a servo pin number and an angle as arguments, and sets the servo connected to the specified pin to the specified angle. To do this, it first calculates the corresponding Pulse Width Modulation (PWM) value for the given angle.

The up() and down() functions control the angle of the first servo, which allows the robotic arm to move up and down.

The `open()` and `close()` functions control the angle of the second servo, which open and close the gripper.

Dc motor functions

```
IN1_A = 23 # Forward
IN2_A = 24 # Backward

# Motor B (right motor) GPIO Pin configuration
IN1_B = 6 # Forward
IN2_B = 5 # Backward

# Connect to the local Raspberry Pi using the pigpio library.
pi = pigpio.pi()

# Set up channels
# Motor A
# Setting the GPIO pins for Motor A as OUTPUT.
pi.set_mode(IN1_A, pigpio.OUTPUT)
pi.set_mode(IN2_A, pigpio.OUTPUT)

# Motor B
# Setting the GPIO pins for Motor B as OUTPUT.
pi.set_mode(IN1_B, pigpio.OUTPUT)
pi.set_mode(IN2_B, pigpio.OUTPUT)

# Function to control Motor A's direction and speed
def motor_a(direction, speed):
    if direction == 'forward':
        # Set the backward control pin to low and control speed with the forward control pin
        pi.write(IN2_A, 0) # Set backward to low
        pi.set_PWM_dutycycle(IN1_A, speed) # Set speed on forward control
    elif direction == 'backward':
        # Set the forward control pin to low and control speed with the backward control pin
        pi.write(IN1_A, 0) # Set forward to low
        pi.set_PWM_dutycycle(IN2_A, speed) # Set speed on backward control

# Function to control Motor B's direction and speed
def motor_b(direction, speed):
    if direction == 'forward':
        # Set the backward control pin to low and control speed with the forward control pin
        pi.write(IN2_B, 0) # Set backward to low
        pi.set_PWM_dutycycle(IN1_B, speed) # Set speed on forward control
    elif direction == 'backward':
        # Set the forward control pin to low and control speed with the backward control pin
        pi.write(IN1_B, 0) # Set forward to low
        pi.set_PWM_dutycycle(IN2_B, speed) # Set speed on backward control

# Functions for driving the motors

def forward(speed):
    # Drive both motors in the forward direction
    motor_a('forward', speed)
    motor_b('forward', speed)

def backward(speed):
    # Drive both motors in the backward direction
    motor_a('backward', speed)
    motor_b('backward', speed)

def left(speed):
    # Drive Motor A backwards and Motor B forwards to turn left
    motor_a('backward', speed)
    motor_b('forward', speed)

def right(speed):
    # Drive Motor A forwards and Motor B backwards to turn right
    motor_a('forward', speed)
    motor_b('backward', speed)

def stop():
    # Stop both motors by setting the speed to zero
    motor_a('forward', 0)
    motor_b('backward', 0)
```

This portion of code control two DC motors (Motor A and Motor B) connected to the Driver board via GPIO pins using the pigpio library. For each motor, two pins are defined, one for forward movement (IN1) and one for backward movement (IN2).

Motor Control Functions: Functions `motor_a` and `motor_b` are defined to control the direction and speed of each motor. They take in two parameters, direction and speed. Depending on the direction input, the appropriate pin is set to high and the speed is controlled by setting the Pulse Width Modulation (PWM) duty cycle.

Driving Functions: Functions `forward`, `backward`, `left`, `right` and `stop` are defined to control the movement of the robot. Each function controls the direction of both motors to achieve the desired movement.

- `forward` and `backward` move both motors in the same direction.
- `left` and `right` move the motors in opposite directions to achieve turning.
- `stop` sets the speed of both motors to zero to stop the robot.

Distance detection

```
def Distance():
    # Set the pin numbering mode to use the Broadcom SOC channel numbers.
    GPIO.setmode(GPIO.BCM)

    # Assign the GPIO pin numbers for the trigger and echo pins.
    Trig = 27
    Echo = 22

    # Set the trigger pin as an output and the echo pin as an input.
    GPIO.setup(Trig, GPIO.OUT)
    GPIO.setup(Echo, GPIO.IN)

    # Send a 10us pulse to the trigger pin to start the distance measurement.
    GPIO.output(Trig, True)
    time.sleep(0.00001) # This is the duration of the pulse sent from the trigger pin
    GPIO.output(Trig, False)

    # Record the start time of the pulse.
    while GPIO.input(Echo) == 0:
        pulse_start = time.time()

    # Record the end time of the pulse.
    while GPIO.input(Echo) == 1:
        pulse_end = time.time()

    # Calculate the pulse duration by subtracting the start time from the end time.
    pulse_duration = pulse_end - pulse_start

    # Calculate the distance using the speed of sound and the pulse duration.
    # Speed of sound is approximately 343 m/s, so in cm it is 34300 cm/s.
    # As the sound has to travel back to the sensor, we divide it by 2,
    # so 34300/2 = 17150.
    distance = pulse_duration * 17150

    # Round the distance to two decimal places.
    distance = round(distance, 2)

    # Return the calculated distance.
    return distance
```

This function is used to calculate the distance to an object using an ultrasonic sensor, which operates by sending out a sound wave and measuring the time it takes for the wave to bounce back after hitting an object.

- **GPIO setmode:** The `GPIO.setmode(GPIO.BCM)` function is used to define the pin numbering system. BCM stands for Broadcom SOC channel numbers, which refer to the GPIO (General Purpose Input/Output) numbers on the Broadcom SOC (System on a Chip) that the Raspberry Pi uses.
- **Trigger and Echo Pins:** The trigger and echo pins are defined (pins 27 and 22, respectively, in this case). The trigger pin is used to send out the sound wave, and the echo pin is used to receive the bounced back wave.
- **GPIO Setup:** The trigger pin is set as an output pin (since it sends a signal), and the echo pin is set as an input pin (since it receives a signal).
- **Send Pulse:** A 10-microsecond pulse is sent via the trigger pin. This pulse serves as the sound wave that will bounce back after hitting an object.

- **Record Pulse Times:** The times at which the pulse is sent (pulse_start) and received back (pulse_end) are recorded.
- **Calculate Pulse Duration:** The pulse duration is calculated as the difference between the end and start times. This duration is the round trip time of the sound wave from the sensor to the object and back.
- **Calculate Distance:** The distance to the object is calculated using the formula $\text{distance} = \text{speed} * \text{time}$. Here, the speed is the speed of sound, which is roughly 34300 cm/s, and the time is the pulse duration. Because the sound wave has to travel to the object and back, the time is divided by 2, making the effective speed 17150 cm/s.
- **Round Distance:** The distance is then rounded to two decimal places.
- **Return Distance:** Finally, the calculated distance is returned by the function.

Delay function

```
# Initialize the global variable Time to 0.
Time = 0
# Function to create a delay.
def delay(s):
    # Declare Time as a global variable so that it can be modified within the function.
    global Time

    # Set the duration of the delay.
    delay = s

    # Check if the elapsed time since the last time the delay function was called
    # is greater than or equal to the delay duration.
    if time.time() - Time >= delay:
        # If so, update the Time variable to the current time.
        Time = time.time()

    # Return the time elapsed since the last update of the Time variable.
    return time.time() - Time
```

This is a delay function, which is used to create a delay of a specific number of seconds.

Delay Duration: The variable delay is set to the input argument s. This represents the delay duration in seconds.

Check Time Elapsed: The function checks whether the time that has elapsed since Time was last set is greater than or equal to the desired delay (delay). time.time()

returns the current system time in seconds since the epoch (usually January 1, 1970), so subtracting Time from this gives the number of seconds that have passed since Time was last set.

Update Time: If the elapsed time is greater than or equal to the delay duration, Time is updated to the current time (time.time()).

This function doesn't pause the program execution for s seconds like a traditional sleep function, rather it allows for checking whether s seconds have passed since the last time the check was true.

Check user input function

```
# This function checks if there is any available input from the user in the stdin buffer.
def is_user_input_available():
    # The select function waits until there is some input available or the timeout has passed.
    # In this case, the timeout is set to 0 which means it won't wait and will return immediately.
    # It returns three lists - the first one contains the objects that are ready for reading.
    # We're only interested in the first list, so we compare it to a list containing only sys.stdin.
    return select.select([sys.stdin], [], [], 0) == ([sys.stdin], [], [])
```

This function check if the input has insert something in the raspberry shell.

Direction car function

```
# Function to control the direction of the car based on user's input.
def DirectionCar(choice):
    # The function checks the choice parameter to decide the direction of the car.
    if choice == "w":
        # If the choice is 'w', the car moves forward with speed 230.
        forward(230)
    elif choice == "q":
        # If the choice is 'q', the car stops.
        stop()
    elif choice == "s":
        # If the choice is 's', the car moves backward with speed 230.
        backward(230)
    elif choice == "a":
        # If the choice is 'a', the car turns left with speed 230.
        left(230)
    elif choice == "d":
        # If the choice is 'd', the car turns right with speed 230.
        right(230)
    elif choice == "open":
        # If the choice is 'open', the action of opening (probably a door or an arm) is performed.
        open()
    elif choice == "close":
        # If the choice is 'close', the action of closing (probably a door or an arm) is performed.
        close()
    elif choice == "up":
        # If the choice is 'up', the action of moving up (probably a door or an arm) is performed.
        up()
    elif choice == "down":
        # If the choice is 'down', the action of moving down (probably a door or an arm) is performed.
        down()
    else:
        # If the choice doesn't match any of the above, it is an invalid choice and a message is printed.
        print("Invalid choice")
```

This function is responsible for controlling the actions of a car-like object based on the input choice. The input choice is a string that defines the specific action to be performed.

Main loop

```
# The script is put into an infinite loop.
while True:

    # This if condition checks if there is any available input from the user.
    if is_user_input_available():

        # If there is input, the script reads it.
        user_input = input("Enter your input: ")
        # Then it prints the input for debugging purposes.
        print("You entered:", user_input)
        # If the user input is "manual", it enters a sub-loop for manual control.
        if user_input == "manual":
            while user_input != "auto":
                # The script keeps reading user input and passing it to the DirectionCar function until the user inputs "auto".
                user_input = input("Enter your input: ")
                DirectionCar(user_input)

    else:
        # If there's no input from the user, the script continues to perform other tasks.
        print("Waiting for input...")

        # If the detected QR code is "Left", it turns the car to the left.
        if QrCode()[0] == "Left":
            left(180)

        # If the detected QR code is "Right", it turns the car to the right.
        if QrCode()[0] == "Right":
            right(180)

        # If the detected QR code is "Center", it performs certain actions based on distance.
        if QrCode()[0] == "Center":
            # If the detected distance is greater than 10, it moves the car forward.
            if Distance() > 10:
                forward(180)

            else:
                # If the detected distance is less than or equal to 10, it stops the car and performs certain actions.
                print("Distance less than 10")
                stop()
                time.sleep(1) # Sleep for 1 second
                open() # Perform the open action
                time.sleep(1) # Sleep for another second

        # If no QR code is detected, it spins the car around for 2 seconds and then stops for 2 seconds.
        if QrCode()[0] == "None":
            if delay(4) < 2:
                # Spin the car around
                motor_a('forward', 180)
                motor_b('backward', 180)
            else:
                # Stop the car
                stop()

# This if condition waits for a specific keyboard input to end the program.
if cv2.waitKey(1) & 0xFF == ord('q'):
    # The waitKey function waits for a keyboard event for (1) ms. If the 'q' key is pressed, the condition will be True.

    # It releases the video capture object.
    cap.release()
    # This is important when working with hardware resources.

    # It destroys all windows created by the cv2 (OpenCV) program.
    cv2.destroyAllWindows()
    # This helps to free up system resources.

    # It resets the servos (SERVO_PIN_1 and SERVO_PIN_2) to 0 pulse width, which effectively turns off the servo motors.
    pi.set_servo_pulsewidth(SERVO_PIN_1, 0)
    pi.set_servo_pulsewidth(SERVO_PIN_2, 0)
    # This is done to ensure that the servo motors are not in a powered state when the program ends.

    # It stops the DC motors.
    motor_a('forward', 0)
    motor_b('forward', 0)
    # Setting the speed of the motors to 0 effectively stops the motors.

    # It cleans up all GPIO channels that have been used in the script.
    GPIO.cleanup()
    # It's a good practice to clean up the GPIO at the end of your project. It will reset all GPIOs to their default state (input).

    # It breaks the infinite while loop, effectively ending the program.
    break
```

An infinite loop is started with `while True`: to keep the script running until explicitly stopped.

It first checks if there is any input from the user with the function `is_user_input_available()`. If there is, it reads this input and prints it out for debugging purposes.

If the input is "manual", it enters in a loop where it continues to accept user input and pass it to the `DirectionCar` function. This loop continues until the user enters "auto", at which point it goes back to the main loop.

If there's no user input, it prints "Waiting for input...", then continues to execute the other parts of the script. It checks for a detected QR code using the `QrCode` function. Depending on the QR code's content, it performs various actions. For "Left" and "Right" QR codes, it turns the car left or right respectively.

If the QR code is "Center", it uses an ultrasonic sensor to measure the distance to an object in front of the car which is supposed to be the destination basket. If the distance is greater than 10 cm, the car moves forward. If it's 10 cm or less, the car stops, waits, opens the gripper, then waits again, it should happen when the car reach the destination and it has to release down the object in the gripper.

If no QR code is detected ("None"), it spins the car around for 2 seconds and then stops it for 2 seconds, repeating this action every 4 seconds, this is function helps the car to "look around" and check if any QR code is available.

It waits for a keyboard input 'q' to end the program. If 'q' is pressed, it releases the video capture object with `cap.release()`, destroys all OpenCV-created windows with `cv2.destroyAllWindows()`, resets the servo motors and stops the DC motors, cleans up the GPIO channels with `GPIO.cleanup()`, and finally breaks the infinite loop, effectively ending the program.

Testing

For this section we expect the following function:

- Colour detection
- Auto mode
 - Spin
 - QR code recognition
 - QR code position detection
 - Car moving according to the QR code position
 - Distance <10 cm handling
- Manual mode
 - Enter in manual mode
 - Moving the car in the 4 direction and stop it
 - Event handling when an incorrect input is inserted
 - Exit manual mode

Colour detection

For this section it will be placed three different object of different colour in front of the robotic car, in order to observe if the colour is detected.

Red object:

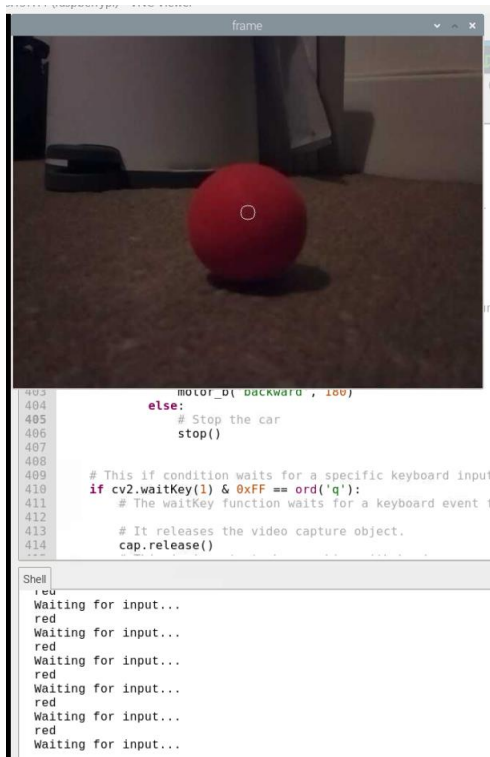


Figure 11RedObjectDetection

We can observe that on the Shell is printed the word red indicating the colour of object.

Yellow object:



Figure 12YellowObjectDetection

We can observe that the colour is identified.

Green object:

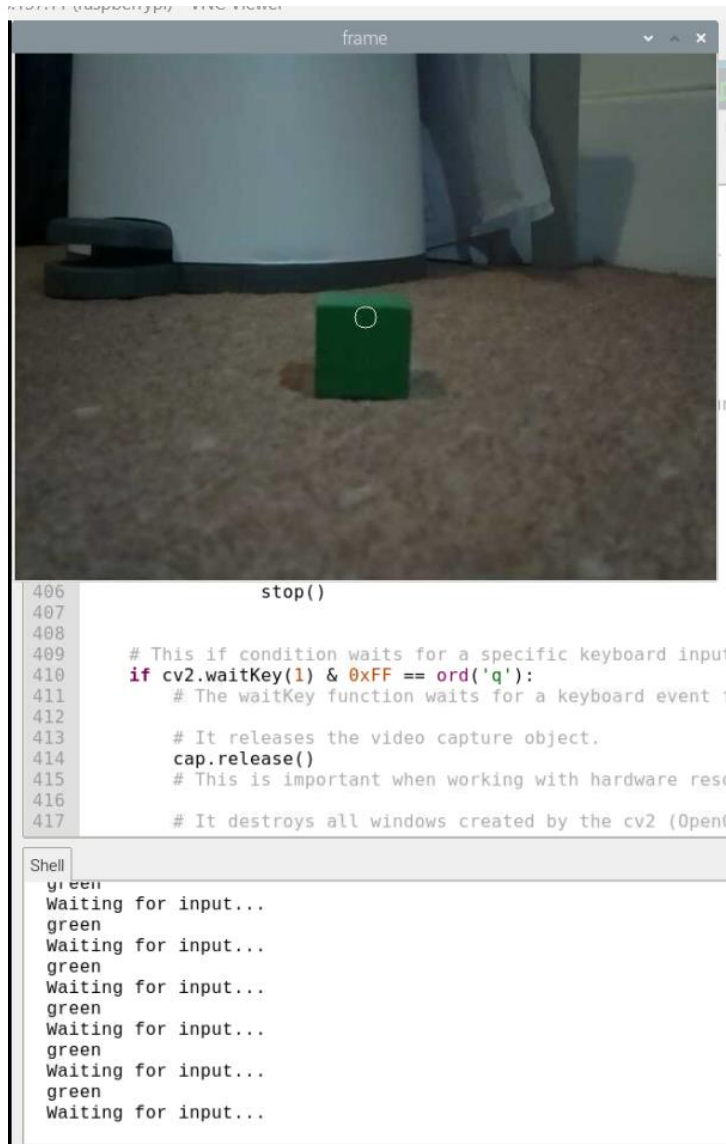


Figure 13GreenObjectDetection

As it can be seen even the green object is recognised.

Auto mode

Spin:

For this test it has been added to the code a print statement in order to evaluate if the car get in the spinning mode.

Code added:

```
# If no QR code is detected, it spir
if QrCode() == "None":
    print("spin")
    if delay(4) < 2:
        # Spin the car around
        motor_a('forward', 180)
        motor_b('backward', 180)
    else:
        # Stop the car
        stop()
```

Test:

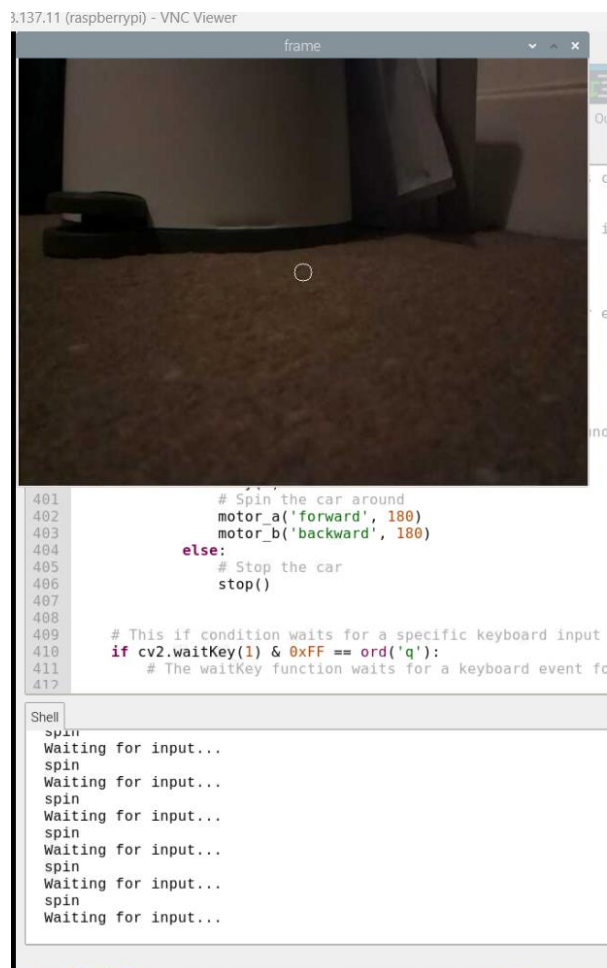


Figure 14 Spinning

It can be observed that there is no code in the front of the car, the raspberry pi is printing “Waiting for input” since is in the auto mode, and it also printed “spin”, indicating that the result is as expected.

QR code recognition and position detection:

Green basket QR code:

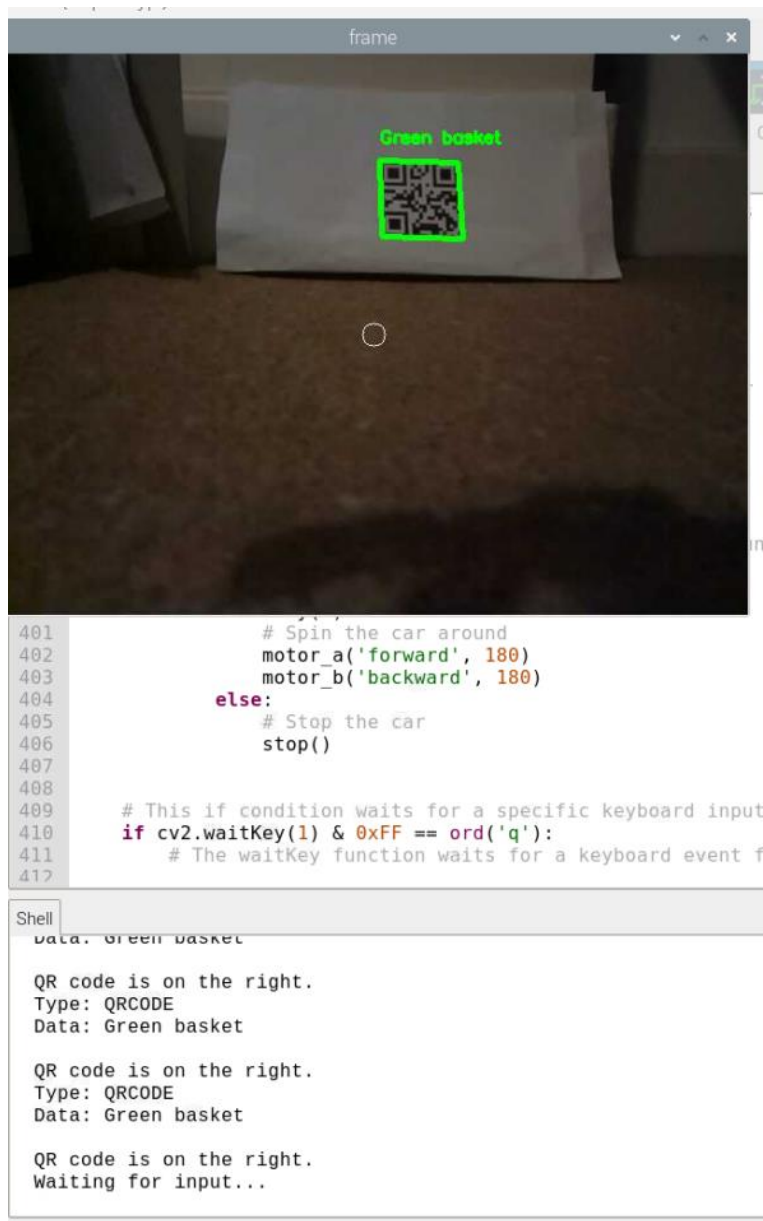


Figure 15GreenBasketQR

It can be observed that the QR code is recognised, the data contained in it is also written next to the green outline.

It can be noticed also that it's saying that the QR code is on the right, indicating the right detection of the position.

Central position:



Figure 16CentralQR

Car moving according to the QR code position:

Since the functions for the different positions of the QR code are working correctly, also the car is moving as expected.

Left position:



Figure 17LeftQR

Red basket QR code:

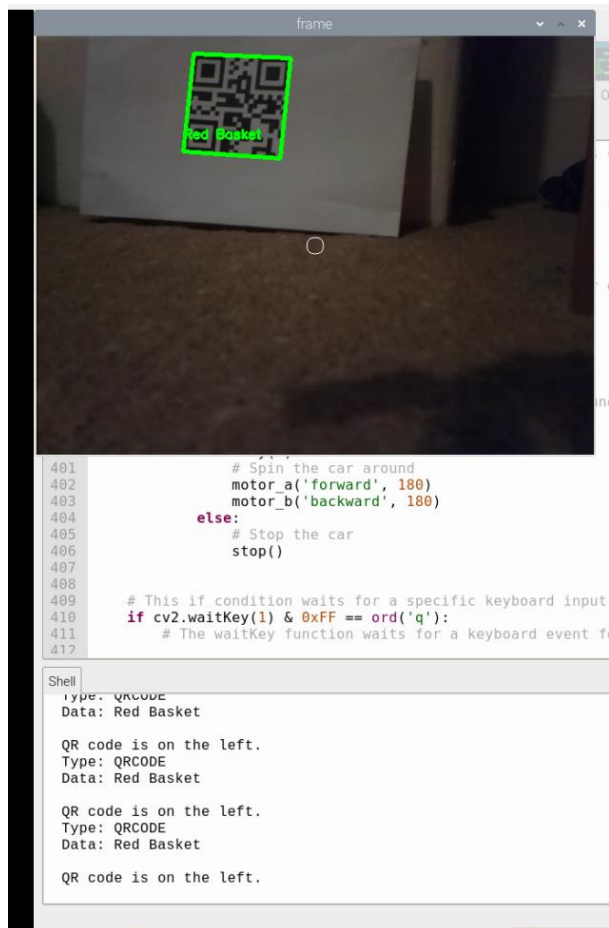


Figure 18RedBasketQR

Yellow basket QR code:



Figure 19 Yellow basket QR

Collect QR code:

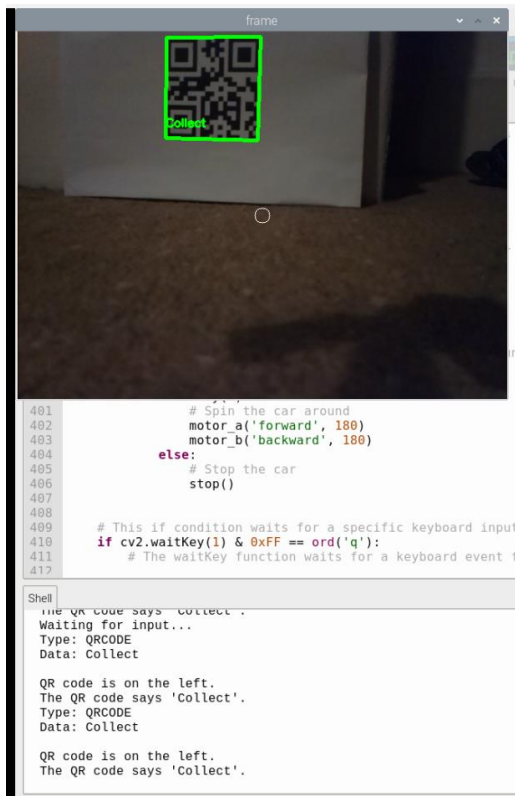


Figure 20Collect QR

Distance <10 cm handling :

Now the QR code will be placed in the center at less than 10 cm of the car, to observe the reaction of the system:

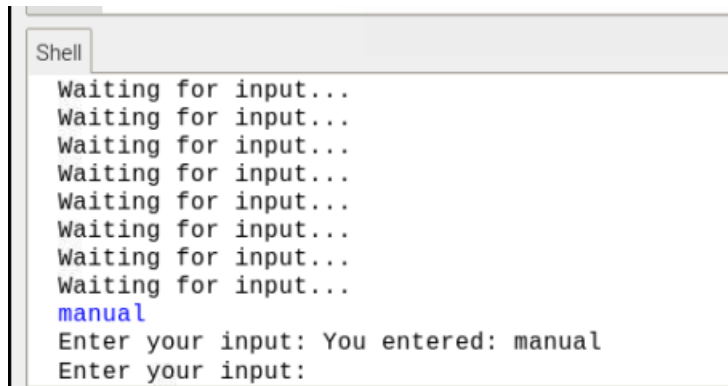


Figure 21Distance<10

Printing the distance less than 10, means that the condition has been satisfied and the function `open()` which open the gripper will also be called.

Manual mode:

Enter in manual mode:



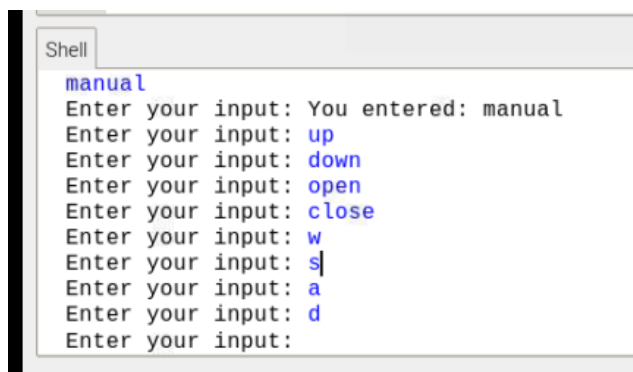
```
Shell
Waiting for input...
Waiting for input...
Waiting for input...
Waiting for input...
Waiting for input...
Waiting for input...
Waiting for input...
Waiting for input...
manual
Enter your input: You entered: manual
Enter your input:
```

Figure 22ManualMode

Now the car is in manual mode.

Checking the up,down,open,close inputs which should move the robotic arm up and down and close and open the gripper, and also checking the function inputs w,s,d,a which should move the car forward(w), backward(s), left(a), right(d).

If no error message will be printed then means that the inputs work correctly.



```
Shell
manual
Enter your input: You entered: manual
Enter your input: up
Enter your input: down
Enter your input: open
Enter your input: close
Enter your input: w
Enter your input: s
Enter your input: a
Enter your input: d
Enter your input:
```

Figure 23ManualModeMovements

As it can be observed the car is moving as expected.

Introducing an invalid input:

```
Shell
Enter your input: up
Enter your input: down
Enter your input: open
Enter your input: close
Enter your input: w
Enter your input: s
Enter your input: a
Enter your input: d
Enter your input: open the arm
Invalid choice
Enter your input:
```

Figure 24InvalidChoice

In case an invalid input is inserted, a error message will be printed and no action will be taken.

Exit manual mode:

```
31 frame = cv2.rotate(frame, cv2.ROTATE_180)
32
Shell
Enter your input: open the arm
Invalid choice
Enter your input: auto
Invalid choice
Waiting for input...
Waiting for input...
Waiting for input...
Waiting for input...
Waiting for input...
Waiting for input...
```

Figure 25AutoMode

When auto is inserted it printed first Invalid choice, this is due to the set of DirectionCar conditions which don't include a condition for the word auto, it can be solved by adding in the DirectionCar() function the option of the auto choice, and for example printing, "Car in auto mode".

6. Results and Discussion

The project's key goals were to design a system that is capable of autonomous navigation and QR code identification, and the implementation and testing phases were centered on validating these functionalities.

The testing procedure involved a systematic approach to verify each component and functionality. Initial tests were dedicated to individual component validation. The DC motors were evaluated for their speed and response to commands, while the servo motor was tested for its positional accuracy and holding capacity as it can be noticed in the figures: as it can be noticed in the figures: [Figure 23ManualModeMovements.](#)

Following the components testing, the integrated system was subjected to several scenarios designed to emulate potential real-world situations. These scenarios involved navigating the car based on QR code instructions figures:

[Figure 15GreenBasketQR](#)

[Figure 16CentralQR](#)

[Figure 17LeftQR](#)

[Figure 18RedBasketQR](#)

[Figure 19Yellow basket QR](#)

, maintaining the car's positioning when no QR code was detected, and commanding the robotic arm to perform specific tasks, figure: [Figure 14Spinning.](#)

The results from these tests were used to fine-tune the system and identify areas requiring attention. Particularly, the inability of the DC motor to facilitate smooth and slow spinning and the struggles of the servo motor were identified as areas of improvement.

To correlate the final implementation with the initial specification, each functional requirement was cross-referenced with the operational capabilities of the system. The autonomous navigation and QR code identification demonstrated in the final implementation stand in alignment with the initial requirements.

However, the system's ability to pick up items autonomously did not meet the project's initial specification. Due to time constraints, this feature could not be fully implemented, although the other requirements were successfully achieved.

Overall, a significant portion of the solution was correctly implemented and working. The autonomous navigation and QR code identification capabilities are evident. However, the results also illustrate the limitations of the current implementation, including the issues with the DC motor and servo motor, and the incomplete autonomous pick-up feature.

These results highlight the successful creation of a system with a strong foundation, which can be further enhanced in the future.

6.1 Discussion of Results

An overall review of the project provides a satisfactory picture of the work undertaken and the consequent results. The design and functionality of the robotic car display a decent response to directional commands.

However, the constraints of time brought limitations. In particular, the autonomous pickup feature could not be fully realized. While this function was intended to be a core aspect of the project, its absence does not negate the other accomplishments.

Similarly, another lack was seen in the operations of the DC motor, responsible for the spinning movement of the car during QR code detection. Ideally, this movement should be smooth and slow, enabling effective scanning. However, the current motor's performance has room for improvement.

The servo motor operating the robotic arm also demonstrated issues. Ideally, the arm should maintain an upright position during the car moving. However, the servo motor's limitations became evident when it struggled to hold the arm up, suggesting the need for a more powerful component.

Currently, the developed system demonstrates effectiveness under specific, controlled circumstances. Yet, without a fully functioning pickup feature and a refined QR code detection mechanism, the system's performance in more variable real-world scenarios may be limited. Hence, while the findings present a considerable scope, there are restrictions to bear in mind.

6.2 Project Management and Progress

Comparing the project's development timeline with the initial Gantt chart provides a revealing look at the planning accuracy and adjustments made during the implementation phase.

The initial Gantt chart provided a useful structure for the project. Some parts of the project followed the original schedule closely, particularly in the early stages involving design and prototyping. However, the reality of the development process, due to challenges and iterations, led to deviations from the plan.

One of the key changes to the plan was an extension of the testing and debugging phase. Initially, it was underestimated how integral this phase was to the project. The challenges faced with the DC motor's speed control and the servo motor's performance demanded more time for problem-solving and testing than initially allocated.

These obstacles were largely overcome through iterative testing, system adjustments. For instance, the issue with the DC motor's speed led to exploring different ways to control the motor, resulting in a more refined speed control. As for the servo motor, acknowledging its limitations under the current setup led to a decision to upgrade the component in the future.

Reflecting on the project's progress, it's fair to say that a significant portion of the original tasks were completed. Key functionalities like autonomous navigation and QR code recognition were successfully implemented, marking important features in the project. However, the autonomous pick-up functionality could not be realized within the given timeframe.

The revised Gantt chart includes extensions in the testing phase and also denotes the part of the project that remains incomplete. It serves as a more accurate representation of the project's lifecycle, bearing the adjustments made to integrate unexpected challenges and delays.

Managing the progress of the project was a continual learning process. Regular check-ins and progress reviews were crucial for staying on track and making necessary adjustments. Recognizing when the project was veering off the planned path and adapting the strategy accordingly was a key part of this process. In this respect, the Gantt chart served not just as a planning tool, but also as a dynamic document that evolved with the project's progression.

7. Conclusions and Further Work

7.1 Conclusions

The aim to design and develop an autonomous robotic car has resulted in notable achievements, simultaneously discovering aspects that demand additional focus and enhancement. The completed system demonstrated a decent execution of autonomous navigation and QR code recognition, proving the feasibility of developing an intelligent car capable of following basic directional instructions. These achievements stand as evidence of the immense potential inherent in such a project and represent substantial progress towards the intended goal.

Despite these positive strides, the development process was not without its challenges, facing a few critical limitations. One challenge involved the control of the DC motor, a crucial component influencing the car's movement. The DC motor's speed proved difficult to control finely, resulting in challenges executing slow, controlled spins when scanning for QR codes. An improvement in this aspect would considerably enhance the effectiveness of the car's scanning capabilities and consequently its navigation prowess.

Another significant issue encountered pertained to the servo motor and the stability of the robotic arm. The servo motor was observed to struggle in maintaining the robotic arm in an elevated position. This revealed an insufficient torque for the task, indicating that a more powerful servo motor would be required for the successful execution of such tasks.

In terms of the goals set out at the beginning, the ambition to implement autonomous pick-up functionality could not be realized within the given timeframe. Although it was an anticipated feature, the complexities involved in its integration necessitated an extension beyond the available time for the project.

7.2 Suggestions for Further Work

The current project presents a multitude of opportunities for further development and refinement. Immediate improvements could involve enhancements to the motor.

Replacing the DC motor with a more refined motor, would allow smoother and more controlled movements, thus improving the effectiveness of QR code scanning.

Replacing the current servo motor with a more powerful model would be another immediate solution. This improvement could resolve the issue related to the stability of the robotic arm and further expand the potential capabilities of the robotic car.

For long-term development, the ambitious task of autonomous pick-up functionality could be adopted as the primary focus. For example the QR code associated with the Collect basket which will be the basket containing the unsorted coloured balls, it could be the first QR code taken in consideration by the car since it has to pick first a ball from the basket and then bring it to the basket associated with the colour of the ball.

In terms of project management, should this project be undertaken again, it would be advised to allocate more time for the testing and debugging phases. This recommendation stems from the observed significance of these stages in hardware-intensive projects and the intricate issues that can arise within them. A more thorough initial assessment of component capabilities against project requirements would also be beneficial, helping to preempt and avoid some of the hardware-related issues encountered.

As it stands, the project serves as a significant stepping stone for any future efforts in the realm of autonomous robotic systems. The experience gained, the foundation laid, and the obstacles identified all contribute to a richer understanding of the challenges and possibilities in this field. The remaining challenges offer exciting opportunities for further exploration and innovation, and any advancements made on this groundwork would undoubtedly contribute to the ongoing progress in autonomous robotic systems.

Appendix A

```
# Import required libraries
import pigpio
import time
import cv2
from pyzbar import pyzbar
import numpy as np
import RPi.GPIO as GPIO
import select
import sys

# Start Camera Code
def decode(image):
    # find and decode QR codes in the image
    decoded_objects = pyzbar.decode(image)

    # print the type and data of each QR code
    for obj in decoded_objects:
        print("Type:", obj.type)
        print("Data:", obj.data.decode("utf-8"), "\n")

    return decoded_objects

# Capture video from the webcam
cap = cv2.VideoCapture(0)

def QrCode():
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Rotate the frame 180 degrees
    frame = cv2.rotate(frame, cv2.ROTATE_180)

    # Get the frame width
    frame_width = frame.shape[1]

    # Apply operations on the frame
    decoded_objects = decode(frame)

    position = "None"

    # Display the resulting frame
    for obj in decoded_objects:
        # Convert polygon points to numpy array
        points = np.array(obj.polygon, np.int32)
        points = points.reshape((-1, 1, 2))

        # Get the x-coordinate of the center of the QR code
        qr_center_x = obj.rect.left + obj.rect.width / 2

        # Determine position of QR code
        if qr_center_x < frame_width / 2 - 20: # Margin of 20 pixels
            print("QR code is on the left.")
            position = "Left"
        elif qr_center_x > frame_width / 2 + 20: # Margin of 20
pixels
            print("QR code is on the right.")
            position = "Right"
        elif qr_center_x > frame_width / 2 - 20 and qr_center_x <
```

```

frame_width / 2 + 20:
    print("QR code is in the center.")
    position = "Center"

    # Check if the QR code data is "Collect"
    qr_data = obj.data.decode("utf-8")
    if qr_data == "Collect":
        print("The QR code says 'Collect'.")

    # Draw the QR code outline on the frame
    cv2.polylines(frame, [points], True, (0, 255, 0), 3)

    # Put the QR code data as text on the frame
    cv2.putText(frame, qr_data, (points[0][0][0], points[0][0][1]
- 15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0),
                2)

    detect_color(frame)

    cv2.imshow('frame', frame)
    return position

def detect_color(frame):
    # Convert the image from BGR to HSV color space. This is done
    because HSV is more effective at color detection.
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Define the central region of interest (ROI).
    # We are interested in the color at the center of the frame.
    center_x, center_y = frame.shape[1] // 2, frame.shape[0] // 2 #
Obtain the coordinates for the center of the image
    radius = 10 # Define the radius of the ROI
    mask_roi = np.zeros(frame.shape[:2], np.uint8) # Create a black
image of the same size as the frame
    # Draw a white circle at the center of the mask. This circle is
our ROI.
    cv2.circle(mask_roi, (center_x, center_y), radius, 255, -1)
    # Bitwise-AND the mask and the hsv image to isolate the color in
the ROI.
    hsv_roi = cv2.bitwise_and(hsv, hsv, mask=mask_roi)

    # Draw the ROI circle on the original frame for visual purposes.
    cv2.circle(frame, (center_x, center_y), radius, (255, 255, 255),
1)

    # Define range for red color in HSV
    red_lower = np.array([0, 152, 76])
    red_upper = np.array([20, 255, 116])
    # Create a mask for the red color
    red_mask = cv2.inRange(hsv_roi, red_lower, red_upper)

    # Define range for green color in HSV
    green_lower = np.array([36, 25, 25])
    green_upper = np.array([86, 255, 255])
    # Create a mask for the green color
    green_mask = cv2.inRange(hsv_roi, green_lower, green_upper)

    # Define range for yellow color in HSV
    yellow_lower = np.array([20, 100, 100])
    yellow_upper = np.array([30, 255, 255])
    # Create a mask for the yellow color

```

```

yellow_mask = cv2.inRange(hsv_roi, yellow_lower, yellow_upper)

# Check if any red pixels are present in the ROI.
if cv2.countNonZero(red_mask) > 0:
    print("Red")
    return "Red" # If red pixels are detected, return "Red"
# Check if any green pixels are present in the ROI.
elif cv2.countNonZero(green_mask) > 0:
    return "Green" # If green pixels are detected, return
"Green"
# Check if any yellow pixels are present in the ROI.
elif cv2.countNonZero(yellow_mask) > 0:
    return "Yellow" # If yellow pixels are detected, return
"Yellow"
else:
    # If no red, green or yellow pixels are detected in the ROI,
    return "None Colour"
    return "None Colour"
# Define the pin number on the Raspberry Pi GPIO to which the first
servo is connected.
SERVO_PIN_1 = 8

# Define the pin number on the Raspberry Pi GPIO to which the second
servo is connected.
SERVO_PIN_2 = 25

# Connect to the local Raspberry Pi using the pigpio library. This
creates an instance of the pigpio.pi class which represents the local
Pi.
pi = pigpio.pi()

# Set the mode of the pins that the servos are connected to as
OUTPUT. This means that these pins will be used to send data out to
the servos.
pi.set_mode(SERVO_PIN_1, pigpio.OUTPUT)
pi.set_mode(SERVO_PIN_2, pigpio.OUTPUT)

def set_angle(servo_pin, angle):
    # This function sets the angle of the servo.
    # The Pulse Width Modulation (PWM) value that corresponds to the
desired angle is calculated.
    # Note: The calculation used here depends on the specific type of
servo. In this case, it is assumed that an angle of 180 corresponds
to a PWM value of 2000, and an angle of 0 corresponds to a PWM value
of 500.
    pwm_value = int((float(angle) * 2000 / 180) + 500)
    # The PWM value is then used to set the pulse width for the
specified servo pin.
    pi.set_servo_pulsewidth(servo_pin, pwm_value)

def up():
    # This function makes the first servo move to the 'up' position,
which is defined as an angle of 170.
    set_angle(SERVO_PIN_1, 170)

def down():
    # This function makes the first servo move to the 'down'
position, which is defined as an angle of 140.
    set_angle(SERVO_PIN_1, 140)

def open():

```

```

    # This function makes the second servo move to the 'open'
    position, which is defined as an angle of 50.
    set_angle(SERVO_PIN_2, 50)

def close():
    # This function makes the second servo move to the 'close'
    position, which is defined as an angle of 100.
    set_angle(SERVO_PIN_2, 100)
# Motor A (left motor) GPIO Pin configuration
IN1_A = 23 # Forward
IN2_A = 24 # Backward

# Motor B (right motor) GPIO Pin configuration
IN1_B = 6 # Forward
IN2_B = 5 # Backward

# Connect to the local Raspberry Pi using the pigpio library.
pi = pigpio.pi()

# Set up channels
# Motor A
# Setting the GPIO pins for Motor A as OUTPUT.
pi.set_mode(IN1_A, pigpio.OUTPUT)
pi.set_mode(IN2_A, pigpio.OUTPUT)

# Motor B
# Setting the GPIO pins for Motor B as OUTPUT.
pi.set_mode(IN1_B, pigpio.OUTPUT)
pi.set_mode(IN2_B, pigpio.OUTPUT)

# Function to control Motor A's direction and speed
def motor_a(direction, speed):
    if direction == 'forward':
        # Set the backward control pin to low and control speed with
        the forward control pin
        pi.write(IN2_A, 0) # Set backward to low
        pi.set_PWM_dutycycle(IN1_A, speed) # Set speed on forward
        control
    elif direction == 'backward':
        # Set the forward control pin to low and control speed with
        the backward control pin
        pi.write(IN1_A, 0) # Set forward to low
        pi.set_PWM_dutycycle(IN2_A, speed) # Set speed on backward
        control

# Function to control Motor B's direction and speed
def motor_b(direction, speed):
    if direction == 'forward':
        # Set the backward control pin to low and control speed with
        the forward control pin
        pi.write(IN2_B, 0) # Set backward to low
        pi.set_PWM_dutycycle(IN1_B, speed) # Set speed on forward
        control
    elif direction == 'backward':
        # Set the forward control pin to low and control speed with
        the backward control pin
        pi.write(IN1_B, 0) # Set forward to low
        pi.set_PWM_dutycycle(IN2_B, speed) # Set speed on backward
        control

```

```

# Functions for driving the motors

def forward(speed):
    # Drive both motors in the forward direction
    motor_a('forward', speed)
    motor_b('forward', speed)

def backward(speed):
    # Drive both motors in the backward direction
    motor_a('backward', speed)
    motor_b('backward', speed)

def left(speed):
    # Drive Motor A backwards and Motor B forwards to turn left
    motor_a('backward', speed)
    motor_b('forward', speed)

def right(speed):
    # Drive Motor A forwards and Motor B backwards to turn right
    motor_a('forward', speed)
    motor_b('backward', speed)

def stop():
    # Stop both motors by setting the speed to zero
    motor_a('forward', 0)
    motor_b('backward', 0)
# Function to calculate distance using an ultrasonic sensor.
def Distance():
    # Set the pin numbering mode to use the Broadcom SOC channel
    numbers.
    GPIO.setmode(GPIO.BCM)

    # Assign the GPIO pin numbers for the trigger and echo pins.
    Trig = 27
    Echo = 22

    # Set the trigger pin as an output and the echo pin as an input.
    GPIO.setup(Trig, GPIO.OUT)
    GPIO.setup(Echo, GPIO.IN)

    # Send a 10us pulse to the trigger pin to start the distance
    measurement.
    GPIO.output(Trig, True)
    time.sleep(0.00001) # This is the duration of the pulse sent
    from the trigger pin
    GPIO.output(Trig, False)

    # Record the start time of the pulse.
    while GPIO.input(Echo) == 0:
        pulse_start = time.time()

    # Record the end time of the pulse.
    while GPIO.input(Echo) == 1:
        pulse_end = time.time()

```

```

    # Calculate the pulse duration by subtracting the start time from
the end time.
    pulse_duration = pulse_end - pulse_start

    # Calculate the distance using the speed of sound and the pulse
duration.
    # Speed of sound is approximately 343 m/s, so in cm it is 34300
cm/s.
    # As the sound has to travel back to the sensor, we divide it by
2,
    # so 34300/2 = 17150.
    distance = pulse_duration * 17150

    # Round the distance to two decimal places.
    distance = round(distance, 2)

    # Return the calculated distance.
    return distance

    # Reset all ports that have been set up by this program to INPUT
mode.
    GPIO.cleanup()
# Initialize the global variable Time to 0.
Time = 0

# Function to create a delay.
def delay(s):
    # Declare Time as a global variable so that it can be modified
within the function.
    global Time

    # Set the duration of the delay.
    delay = s

    # Check if the elapsed time since the last time the delay
function was called
    # is greater than or equal to the delay duration.
    if time.time() - Time >= delay:
        # If so, update the Time variable to the current time.
        Time = time.time()

    # Return the time elapsed since the last update of the Time
variable.
    return time.time() - Time
# This function checks if there is any available input from the user
in the stdin buffer.
def is_user_input_available():
    # The select function waits until there is some input available
or the timeout has passed.
    # In this case, the timeout is set to 0 which means it won't wait
and will return immediately.
    # It returns three lists - the first one contains the objects
that are ready for reading.
    # We're only interested in the first list, so we compare it to a
list containing only sys.stdin.
    return select.select([sys.stdin], [], [], 0) == ([sys.stdin], [],
[])

# Function to control the direction of the car based on user's input.
def DirectionCar(choice):
    # The function checks the choice parameter to decide the

```

```

direction of the car.
    if choice == "w":
        # If the choice is 'w', the car moves forward with speed 230.
        forward(230)
    elif choice == "q":
        # If the choice is 'q', the car stops.
        stop()
    elif choice == "s":
        # If the choice is 's', the car moves backward with speed
230.
        backward(230)
    elif choice == "a":
        # If the choice is 'a', the car turns left with speed 230.
        left(230)
    elif choice == "d":
        # If the choice is 'd', the car turns right with speed 230.
        right(230)
    elif choice == "open":
        # If the choice is 'open', the action of opening (probably a
door or an arm) is performed.
        open()
    elif choice == "close":
        # If the choice is 'close', the action of closing (probably a
door or an arm) is performed.
        close()
    elif choice == "up":
        # If the choice is 'up', the action of moving up (probably a
door or an arm) is performed.
        up()
    elif choice == "down":
        # If the choice is 'down', the action of moving down
(probably a door or an arm) is performed.
        down()
    else:
        # If the choice doesn't match any of the above, it is an
invalid choice and a message is printed.
        print("Invalid choice")
# The script is put into an infinite loop.
while True:

    # This if condition checks if there is any available input from
the user.
    if is_user_input_available():

        # If there is input, the script reads it.
        user_input = input("Enter your input: ")
        # Then it prints the input for debugging purposes.
        print("You entered:", user_input)
        # If the user input is "manual", it enters a sub-loop for
manual control.
        if user_input == "manual":
            while user_input != "auto":
                # The script keeps reading user input and passing it
to the DirectionCar function until the user inputs "auto".
                user_input = input("Enter your input: ")
                DirectionCar(user_input)

            else:
                # If there's no input from the user, the script continues to
perform other tasks.
                print("Waiting for input...")

```



```

        # If the detected QR code is "Left", it turns the car to the
left.
        if QrCode()[0] == "Left":
            left(180)

        # If the detected QR code is "Right", it turns the car to the
right.
        if QrCode()[0] == "Right":
            right(180)

        # If the detected QR code is "Center", it performs certain
actions based on distance.
        if QrCode()[0] == "Center":

            # If the detected distance is greater than 10, it moves
the car forward.
            if Distance() > 10:
                forward(180)

            else:
                # If the detected distance is less than or equal to
10, it stops the car and performs certain actions.
                print("Distance less than 10")
                stop()
                time.sleep(1) # Sleep for 1 second
                open() # Perform the open action
                time.sleep(1) # Sleep for another second

        # If no QR code is detected, it spins the car around for 2
seconds and then stops for 2 seconds.
        if QrCode()[0] == "None":
            if delay(4) < 2:
                # Spin the car around
                motor_a('forward', 180)
                motor_b('backward', 180)
            else:
                # Stop the car
                stop()

        # This if condition waits for a specific keyboard input to end
the program.
        if cv2.waitKey(1) & 0xFF == ord('q'):
            # The waitKey function waits for a keyboard event for (1) ms.
            If the 'q' key is pressed, the condition will be True.

            # It releases the video capture object.
            cap.release()
            # This is important when working with hardware resources.

            # It destroys all windows created by the cv2 (OpenCV)
program.
            cv2.destroyAllWindows()
            # This helps to free up system resources.

            # It resets the servos (SERVO_PIN_1 and SERVO_PIN_2) to 0
pulse width, which effectively turns off the servo motors.
            pi.set_servo_pulsewidth(SERVO_PIN_1, 0)
            pi.set_servo_pulsewidth(SERVO_PIN_2, 0)
            # This is done to ensure that the servo motors are not in a
powered state when the program ends.

```

```
# It stops the DC motors.
motor_a('forward', 0)
motor_b('forward', 0)
# Setting the speed of the motors to 0 effectively stops the
motors.

# It cleans up all GPIO channels that have been used in the
script.
GPIO.cleanup()
# It's a good practice to clean up the GPIO at the end of
your project. It will reset all GPIOs to their default state (input).

# It breaks the infinite while loop, effectively ending the
program.
break
```