

PRACTICAL GROUP – SMART CAR

Nguyễn Nhật Huy
103802911

Huỳnh Nguyễn Quốc Bảo
103804535

Lê Hoàng Hải
103542974

SWE30011 – IOT Programming

—
Dr. Hai Van Ho



TABLE OF CONTENT

INTRODUCTION	3
CONCEPTUAL DESIGN	5
TASKS BREAKDOWN.....	7
IMPLEMENTATION	9
1. Sensing System.....	9
2. Cloud Computing.....	11
3. Communication Protocol.....	13
4. Edge Server	13
5. Mobile Application (User Interface/Control)	14
USER MANUAL	16
LIMITATIONS	18
RESOURCE.....	19



INTRODUCTION

The **Internet of Things** (or IoT) is a network consisting of physical objects or devices that are connected to the Internet and can interact with each other. It is a major milestone in technological advancements in that it opens up endless possibilities for innovative applications of different devices and making way for further development of existing technologies. With that in mind, a car can be outfitted with various IoT-related technologies, such as sensors, processors, ..., to communicate with other cars or infrastructures to optimize performance and improve road safety. This type of car is known as **Smart Car**, which is what we will be doing for this assignment.

Traditional cars, while serving their intended purpose of travel, are a volatile and error-prone piece of technology that contributes to many growing problems in modern society, such as environmental or traffic concerns. Granted that modern cars have improved a lot over the course of history, as long as human input is involved, the risk of accidents is an ever-looming threat hovering over drivers. Thus, the need for a safer, more efficient, and possibly automatic mean of traversal has been a top priority for many car manufacturers.

The rising tide of IoT has led to new technologies being incorporated into car designs that lets it perform tasks previously impossible for traditional cars. With advanced functionalities such as self-driving, object avoidance, path optimization, Smart Cars are both efficient and safe with what it does, being able to intelligently weave in and out of traffic without human input.

The aim of our project is to create a mini smart car that will be able to detect obstacles, track and automatically follow a line through a mounted camera. A full-fledged IoT communication will be established between the IoT node (the car) and the edge device through an MQTT broker. A cloud computing platform is also utilized through use of an AWS (Amazon Web Service) EC2 server.

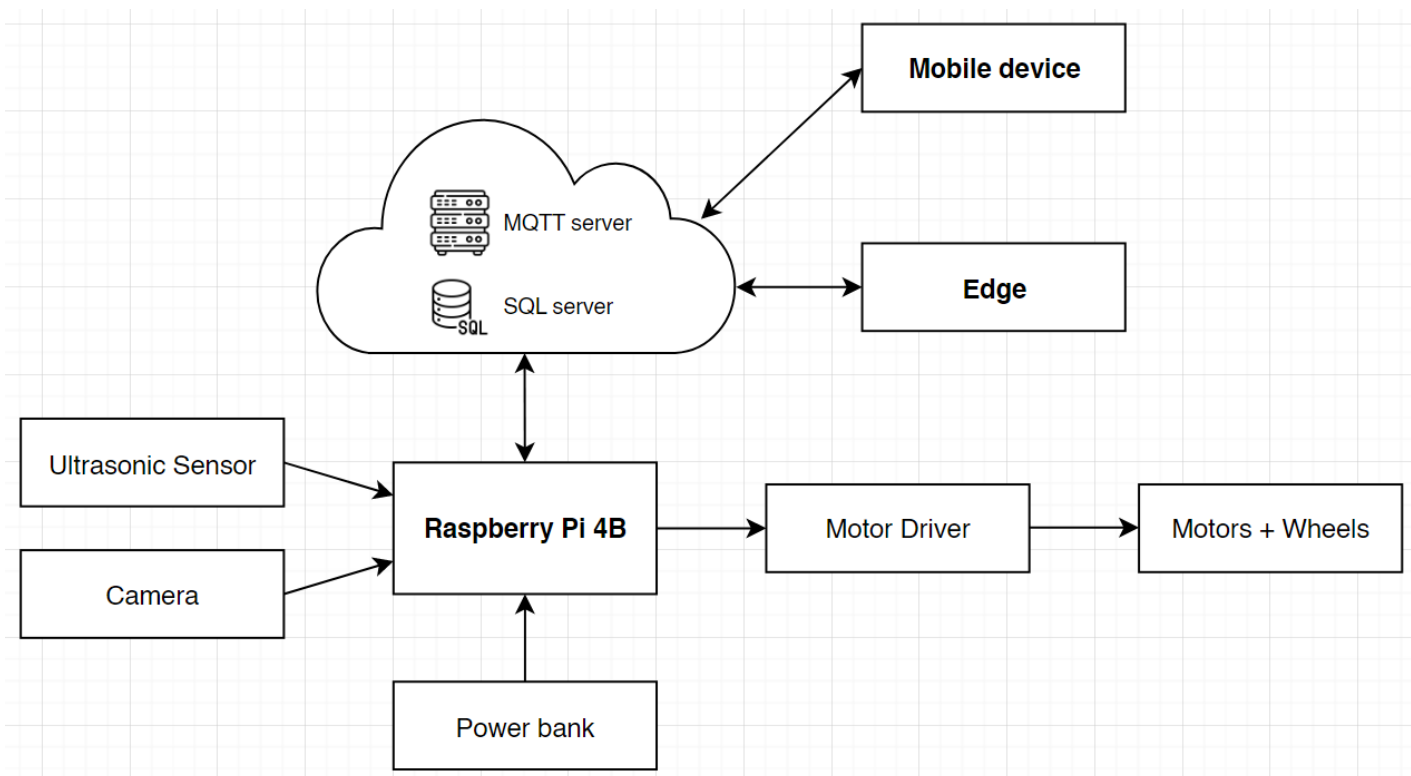
A presentation video link is included in [Appendix](#).

Hardware	Software
<p>Raspberry Pi 4B</p> <p>Camera Raspberry Pi V1</p> <p>HC-SR04 Ultrasonic sensor</p> <p>L293D Motor driver</p> <p>Car chassis (4 wheels + motors)</p> <p>Power bank 5V (two 18650 batteries)</p> <p>Charger</p> <p>Jumper wires</p> <p>USB Type-C cable</p> <p>32GB MicroSD card</p> <p>Breadboard</p> <p>GPIO Breakout extension board</p>	<p>Raspbian OS Buster</p> <p>VMWare</p> <p>PuTTY SSH session</p> <p>OpenCV 3.0</p> <p>Cloud server (Amazon Web Service EC2)</p> <p>Visual Studio Code</p> <p>Python 2.7</p> <p>Cloud ware</p> <p>MySQL server</p> <p>Mosquito MQTT broker</p> <p>Android OS</p>

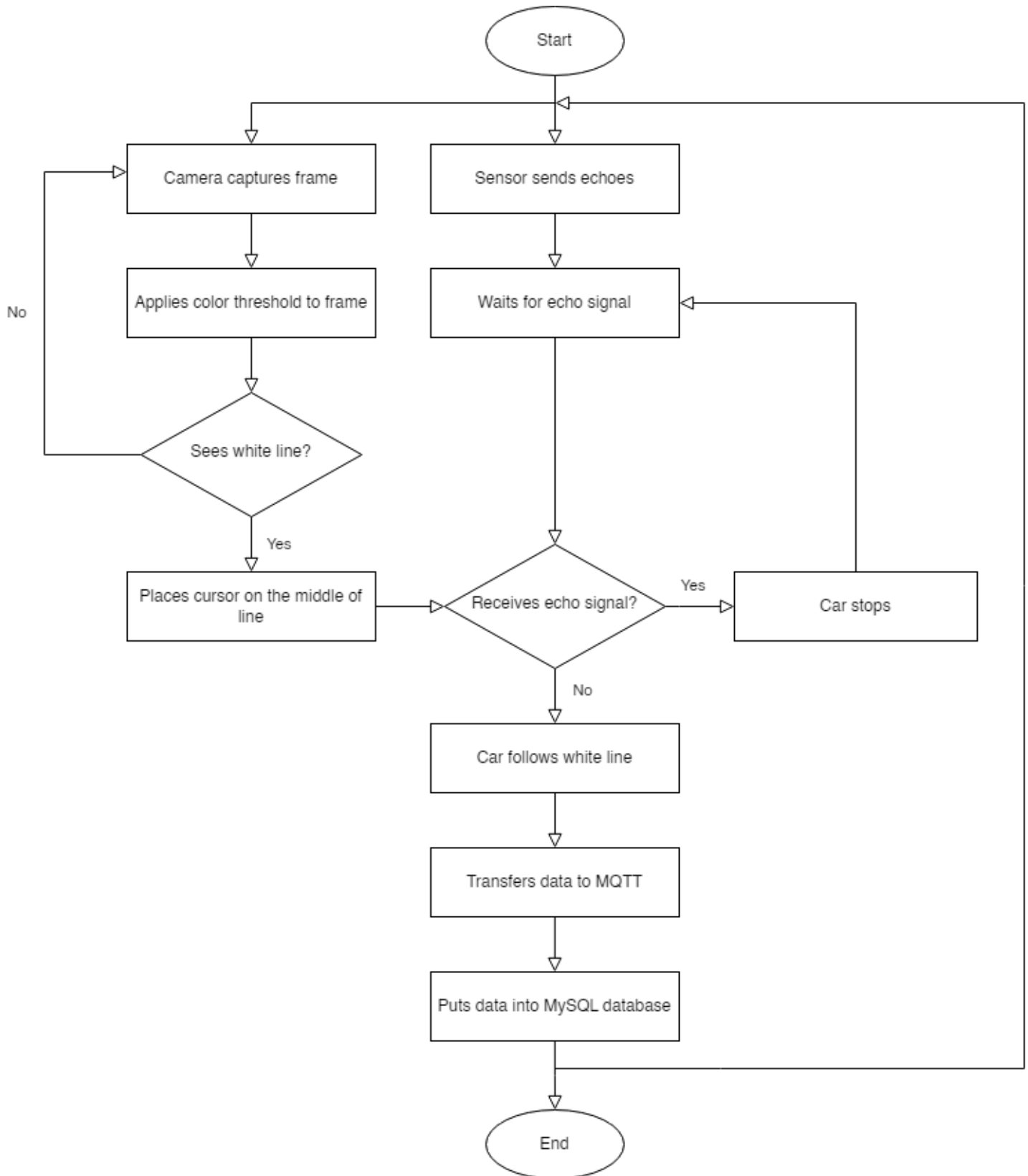
CONCEPTUAL DESIGN

For a car to be considered smart, it needs to employ various IoT technologies such as sensors, processors, actuators, and other communication devices. Smart cars should be able to drive itself automatically while avoiding obstacles on the road as well as being able to connect to other IoT devices within the network. We implemented an ultrasonic sensor on the front to detect obstacles and a camera to implement self-driving. The Raspberry Pi will be the brain of the IoT node, receiving information from the sensors and sending signals to the motor drivers to move the car.

Block diagram:



Operation flowchart:





TASKS BREAKDOWN

	Lê Hoàng Hải	Huỳnh Nguyễn Quốc Bảo	Nguyễn Nhật Huy
Tasks	<i>Gather hardware:</i> List and purchase the required hardware.	<i>Gather hardware:</i> List and purchase the required hardware.	<i>Gather hardware:</i> List and purchase the required hardware.
	<i>Assemble car:</i> Use the gathered materials to put together the car.	<i>Assemble car:</i> Use the gathered materials to put together the car.	<i>Assemble car:</i> Use the gathered materials to put together the car.
	<i>Set up cloud server:</i> Set up an EC2 cloud computing platform on AWS.	<i>Coding:</i> Code for the Raspberry car, Raspbian edge device and mobile application.	<i>Testing:</i> Help test the user interface on mobile devices, physical map and car tracking.
	<i>Establish IoT communication:</i> Set up MQTT broker connection with IoT node and edge server.	<i>Testing:</i> Test the car's ability to move, camera's connectivity, user interface on mobile app and car tracking.	<i>Write report:</i> Log, compile the work done and write report on project.
	<i>Coding:</i> Code the Raspberry IoT node, Raspbian edge device and mobile application to control car.	<i>Video editing:</i> Edit the presentation video.	<i>Presenting:</i> Present the final product.
	<i>Testing:</i> Test the car's ability to move, camera's connectivity, mobile application, and car tracking.		

	<i>Recording:</i> Record the presentation video		
--	---	--	--

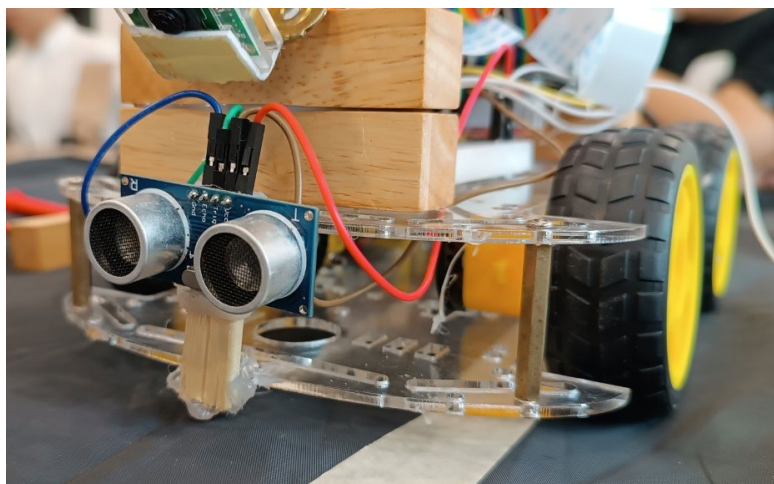
IMPLEMENTATION

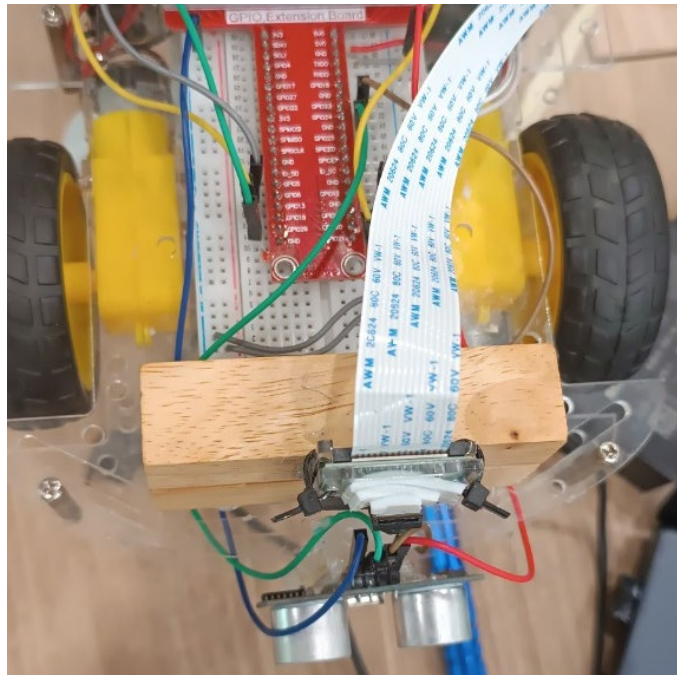
1. Sensing System

This system utilizes two main sensing system: an ultrasonic sensor (model HC-SR04) used to detect objects and a camera (Camera V1) coded to automatically follow a road made up of two lines.

First, the ultrasonic sensor is a type of sensor that sends out ultrasonic waves through a transmitter that bounces off objects and reflects back to the sensor's receiver. The interval between sending and receiving waves can be used to measure the distance between the object and the sensor itself. In this context, the sensor is mounted on the front to detect obstacles on the road, and then relaying that information to the Raspberry for it to process and take appropriate actions.

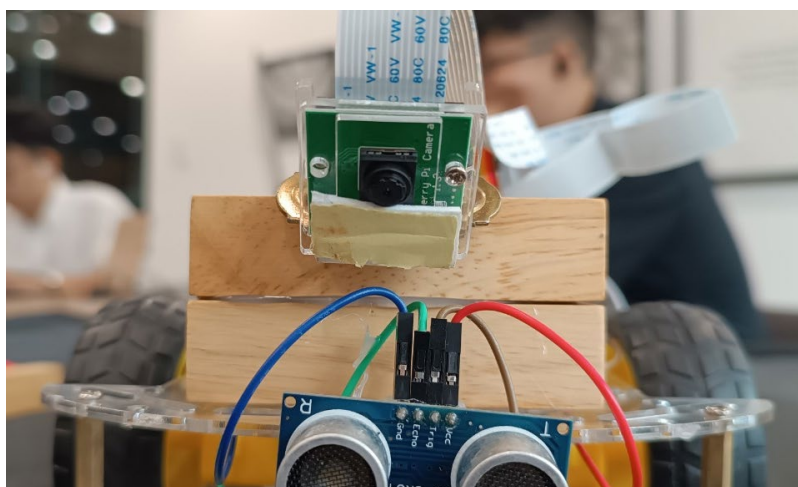
An ultrasonic sensor has 4 pins: VCC, Trig, Echo and GND. The VCC pin is responsible for powering up the sensor itself, thus it is connected to the 5V pin from the GPIO Extension board. The GND pin provides ground connection and is subsequently connected to the GND pin on the Extension board. Trig is the input pin that sends out ultrasonic waves while Echo is the output pin that receives the bounced echo signal and sends that information back to the Raspberry Pi for processing.





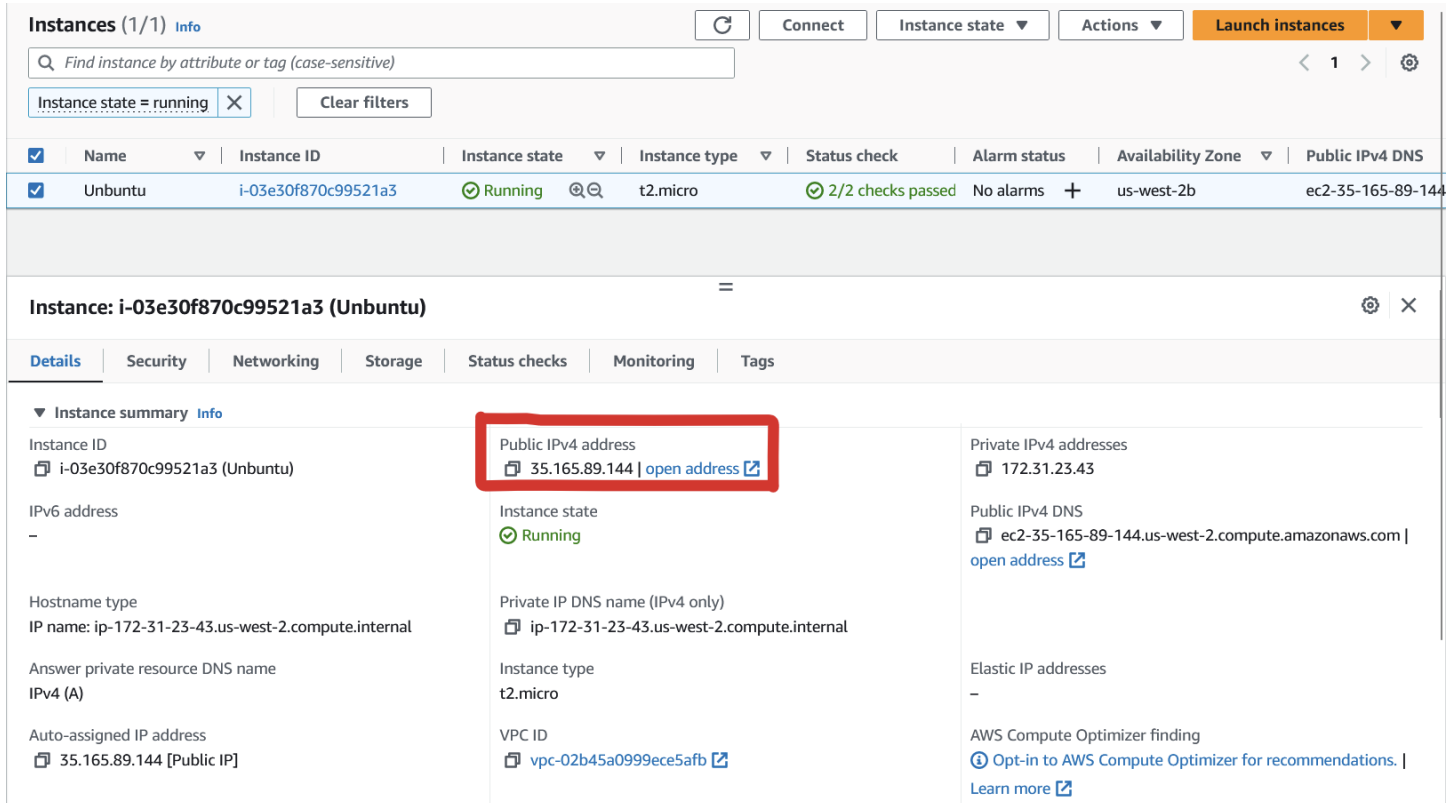
Secondly, a custom camera is set up to capture and process images in real-time. The camera is coded to analyze the image captured and draw relevant lines for the car to track. By implementing a computer vision algorithm known as color thresholding, the car can detect and maintain its center position along the lines drawn.

Color thresholding is an image processing method that changes all pixels within a specified color range to *white* and anything outside of it to *black*. First, we greyscale the captured image for easier processing, then convert it to a binary image using thresholding by setting all pixels to white when above a certain color threshold and anything below it to black. The result is a binary image where the points of interest are painted white while other background items are black. We then identify the connected areas in the binary image to form contours and apply a centroid calculation on them to be able to detect the center of the shapes, or in this case lines, for the car to follow. We also included extra steps to clean up the binary image using *erosion* and *dilation*, formally called *morphological operations*, for more accurate processing.



2. Cloud Computing

This car features a cloud computing platform in form of an EC2 (Elastic Compute Cloud) server hosted on AWS (Amazon Web Service). EC2 is a web service that allows you to host your own virtual servers (called Instances) where you can run your custom applications. The instances themselves are quite configurable in terms of operating systems and software configurations, while being highly scalable to suit the user's needs.



The screenshot displays the AWS Management Console interface for an EC2 instance. At the top, there's a search bar and filters. The instance list shows one instance named 'Ubuntu' with ID 'i-03e30f870c99521a3', in a 'Running' state, of type 't2.micro'. Below this, the 'Instance: i-03e30f870c99521a3 (Ubuntu)' details are shown. The 'Public IPv4 address' is highlighted with a red box, showing '35.165.89.144'. Other details include the instance state 'Running', private IP '172.31.23.43', and public IPv4 DNS 'ec2-35-165-89-144.us-west-2.compute.amazonaws.com'.

On the cloud, we also implemented a MySQL server to store the car's instructions into a database for processing, among other relevant information.

```
*** System restart required ***
Last login: Sun Apr  9 09:00:58 2023 from 115.78.237.195
ubuntu@ip-172-31-23-43:~$ systemctl status mysql.service
● mysql.service - MySQL Community Server
   Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2023-04-07 11:22:08 UTC; 2 days ago
     Process: 1369 ExecStartPre=/usr/share/mysql/mysql-systemd-start pre (code=exited, status=0/SUCCESS)
    Main PID: 1377 (mysqld)
      Status: "Server is operational"
        Tasks: 42 (limit: 1141)
       Memory: 372.7M
          CPU: 13min 26.285s
      CGroup: /system.slice/mysql.service
              └─1377 /usr/sbin/mysqld

Apr 07 11:22:07 ip-172-31-23-43 systemd[1]: Starting MySQL Community Server...
Apr 07 11:22:08 ip-172-31-23-43 systemd[1]: Started MySQL Community Server.
```

An SQL table called MOVEMENT is created with several columns:

- +) No: Item number in the list.
- +) Time: Date and time when the data is recorded.
- +) CarMovement: Status message of the car.

When the car runs, the table is populated with data.

```
mysql> USE Project3;
Database changed
mysql> CREATE TABLE Movement (
  ->   No INT NOT NULL AUTO_INCREMENT,
  ->   time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  ->   CHAR    CarMovement VARCHAR(50),
  ->   PRIMARY KEY (No)
  -> );
Query OK, 0 rows affected (0.05 sec)

mysql> DESCRIBE Movement;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default          | Extra              |
+-----+-----+-----+-----+-----+-----+
| No         | int           | NO   | PRI | NULL             | auto_increment    |
| time       | timestamp     | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED  |
| CarMovement | varchar(50)   | YES  |     | NULL             |                    |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.03 sec)
```

```
mysql> use Project3
Database changed
mysql> select * from Movement
  -> ;
+-----+-----+-----+-----+
| No | time                | CarMovement |
+-----+-----+-----+-----+
| 1 | 2023-04-09 09:13:12 | On Track!   |
| 2 | 2023-04-09 09:13:13 | On Track!   |
| 3 | 2023-04-09 09:13:14 | On Track!   |
| 4 | 2023-04-09 09:13:15 | On Track!   |
| 5 | 2023-04-09 09:13:15 | On Track!   |
| 6 | 2023-04-09 09:13:16 | On Track!   |
| 7 | 2023-04-09 09:13:17 | On Track!   |
| 8 | 2023-04-09 09:13:18 | Stop        |
| 9 | 2023-04-09 09:13:18 | On Track!   |
|10 | 2023-04-09 09:13:19 | On Track!   |
|11 | 2023-04-09 09:13:20 | On Track!   |
|12 | 2023-04-09 09:13:21 | On Track!   |
|13 | 2023-04-09 09:13:22 | On Track!   |
|14 | 2023-04-09 09:13:22 | On Track!   |
|15 | 2023-04-09 09:13:23 | On Track!   |
|16 | 2023-04-09 09:13:24 | Stop        |
|17 | 2023-04-09 09:13:25 | Stop        |
|18 | 2023-04-09 09:13:25 | Stop        |
|19 | 2023-04-09 09:13:26 | Stop        |
|20 | 2023-04-09 09:13:26 | Stop        |
+-----+-----+-----+-----+
```


3. Communication Protocol

For our IoT communication protocol, we have set up a Mosquitto MQTT broker to serve as a central point for communication between the car and the edge device. The car publishes messages in a “topic” to the broker, which is then forwarded to the edge device that is subscribed to said “topic”. The broker itself has administrator role which allows it to access and publish/subscribe data through the internet.

```
ubuntu@ip-172-31-23-43:~$ systemctl status mosquitto.service
● mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2023-04-07 11:09:18 UTC; 2 days ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
  Main PID: 495 (mosquitto)
    Tasks: 1 (limit: 1141)
   Memory: 2.5M
      CPU: 1min 14.861s
   CGroup: /system.slice/mosquitto.service
           └─495 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Apr 07 11:09:17 ip-172-31-23-43 systemd[1]: Starting Mosquitto MQTT Broker...
Apr 07 11:09:18 ip-172-31-23-43 systemd[1]: Started Mosquitto MQTT Broker.
ubuntu@ip-172-31-23-43:~$
```

4. Edge Server

In IoT, an edge server is a server on the edge of a network that provides localized processing and data storage which reduces the need to constantly transport data to and from data centers, resulting in improved performance. For this project, we are hosting a virtual Raspbian OS edge server using VMWare. The edge server receives data collected from the car through the MQTT broker and upload it to the database.

```
1 import paho.mqtt.client as mqtt
2 import pymysql
3
4 broker_address = "35.165.89.144"
5 topic = "linetracking"
6 port = 1883
7
8 # Connect to the database
9 try:
10     connection = pymysql.connect(
11         host='35.165.89.144',
12         user='haile',
13         password='12345678',
14         database='Project3',
15     )
16     print("Connection to the database successful!")
17
18 except pymysql.MySQLError as error:
19     print("Failed to connect to the database: {}".format(error))
```



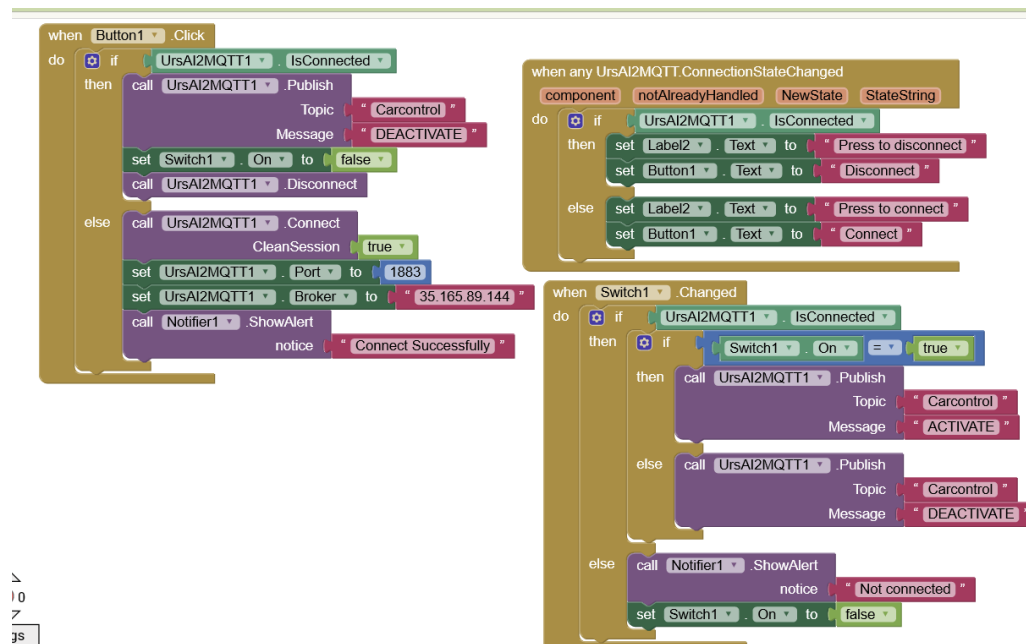
```

7 def on_message(client, userdata, msg):
8     # Insert the message payload into the database
9     try:
10         with connection.cursor() as cursor:
11             sql = "INSERT INTO Movement (CarMovement) VALUES (%s)"
12             cursor.execute(sql, (str(msg.payload.decode()),))
13             connection.commit()
14             print("Message inserted successfully!")
15
16     except pymysql.MySQLError as error:
17         print("Failed to insert message into the database: {}".format(err
18
19     print("Received message: " + str(msg.payload))
20
21 client = mqtt.Client()
22 client.on_connect = on_connect
23 client.on_message = on_message
24 client.connect(broker_address, port, 60)

```

5. Mobile Application (User Interface/Control)

We also included a mobile application to allow control over the car anywhere at any time. By using the online website MIT App Inventor, we have designed a simple app that enables the user to turn the car on and off at will. The app will connect and send instructions to the MQTT broker, which then forwards it to the car to take appropriate actions.





USER MANUAL

1. Getting started

- First, insert two 18650 batteries into the power bank to power up the car.
- Install the *Project3_MQTT.apk* app on your mobile device using the QR code below.



- Alternatively, you can also use:
<https://1drv.ms/f/s!AidtVA4gbYvyhWQl2M6XIUoJ6Xd?e=iHx3LY>

2. Controlling the car

Should any problem arise during operation of the car, refer to the Troubleshooting section below.

- Open the *Project3_MQTT* app and connect to the MQTT server. You can also disconnect to the server this way.
- Use the *Turn on/off* switch to turn the car on or off.



- Place the car on a surface, preferably flat.
- The car will automatically identify a line and follow it.
- If the car detects an object, it will stop in its track until the obstacle is removed.

3. Troubleshooting

- If you are having trouble connecting to the MQTT server, please check your Wi-Fi connection.
- If the car is not running, it is possible the power bank is out of battery. Replace them with new, recharged ones.
- If any of the problems persist, contact the manufacturer for support.



LIMITATIONS

Due to the scope of this project, some limitations have surfaced during the development of the car, including:

1. Hardware limitations

One of the main issues is the limitations of the Camera V1 we implemented for road tracking. Since we opted for a cheap option, the camera wasn't as good as it can be and it suffered from major frame rate issues which led to slow, inaccurate tracking. The car would often go offroad because the camera did not have enough time to process a new frame to change the car's direction.

2. Tracking flaws

The tracking method used is color thresholding, which could result in unintended tracking behavior if the environment is unsuitable.

3. Car design

While this does not pose a big functionality issue, the car does not look very pleasing to the eyes despite our best efforts to optimize and clean up wiring.

4. Mobile app limitations

The app in its current state does not offer much functionality other than being able to turn the car on and off.

The app also only works on Android platform as it is in an *.apk* format, greatly hampering its accessibility.

5. Security concerns

Since we don't have much experience with cyber security, some parts of the servers might not be well-secured.



RESOURCE

1. <https://iot4beginners.com/self-driving-car-based-on-raspberry-pi-and-opencv/>
2. <https://github.com/ishanambike/Self-driving-Car-Using-Raspberry-Pi>
3. <https://www.geeksforgeeks.org/opencv-python-tutorial/>
4. <https://howchoo.com/g/ndy1zte2yjn/how-to-set-up-wifi-on-your-raspberry-pi-without-ethernet>
5. <https://imagemagick.org/script/color-thresholding.php#:~:text=Use%20color%20thresholding%20to%20specify,with%20a%20hyphen%20between%20them.>



APPENDIX

Presentation video: <https://youtu.be/2d2KHnJlTVU>