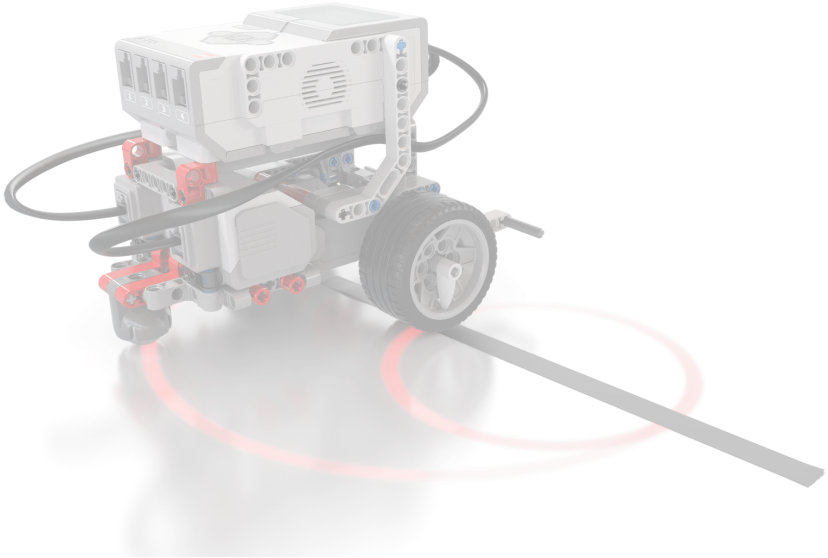
Lego Mindstorms EV3 Autonomous Vehicle Network



Academic Year 2022-23

Intelligent Systems and Robotics Laboratory - DT0201

Members:
Aswin Palathumveetil Jagadeesan (288721)
Rahul Pankajakshan (288722)
Sushmitha Ramaneedi (288723)

Table of contents

1. Introduction

The objective of this project is to design, implement, and evaluate an autonomous vehicular network using the Lego Mindstorms EV3 bricks. The proposed network aims to enable communication between vehicles in the network by allowing them to share information such as their speed, state, and the detection of any obstacles and emergencies via the server. We have used 2 lego kits to create the vehicles in the network and used a PC as a client to simulate the third vehicle in the network.

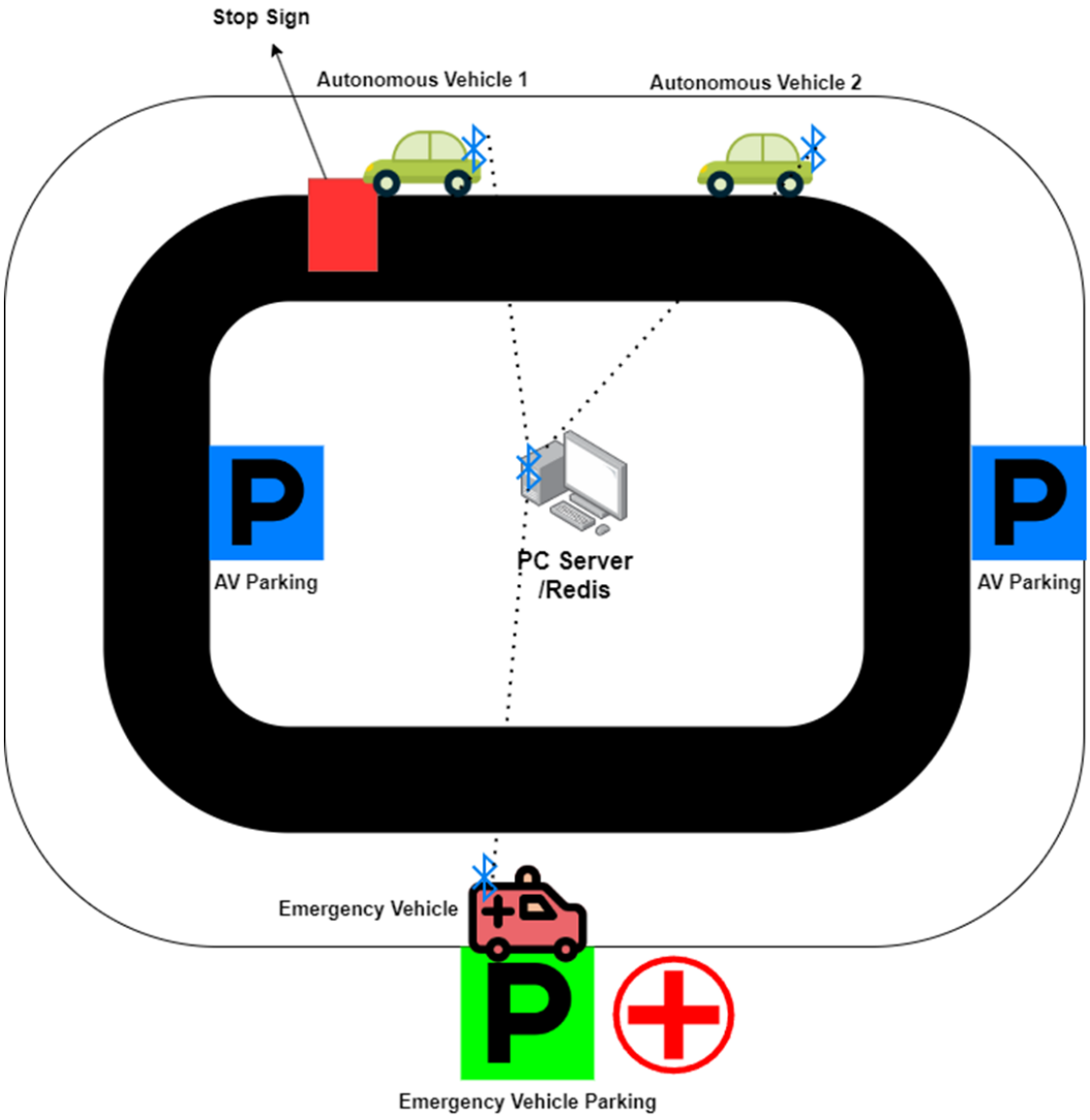


Fig 1: Layout of the robot's environment.

The environment layout is shown in Figure 1, where vehicles communicate with a PC server through Bluetooth using the pybricks Python package. Red coloured strips are used to simulate the traffic stop signs. The vehicles have designated parking spots, with normal AVs assigned blue spots on both lanes, while emergency vehicles have yellow spots located adjacent to a hospital. In the simulation using Coppelia Sim, the server and its communication is handled by a Redis message broker, but rest of the functionalities remain the same.

The system will be able to do the following:

1) Staying within the lane.
2) Alert other vehicles about emergencies.
3) Switching the lane if the presence of a roadblock is detected.
4) Moderating the speed of the vehicle based on the situation.
5) Parking the car on the designated parking spots.
6) Stopping the vehicle when the stop sign (red colour strip) is detected on the lane.

The capabilities and tasks of the networks are further elaborated in section 4.

2. Lego Mindstorms EV3

2.1 Sensors

The Lego Mindstorms EV3 kit comes with the following sensors to monitor the physical world around it:

Sensors	Need
Ultrasonic Sensor	Used to measure the distance between the vehicle and the object/vehicle in front of it
Colour Sensor	The information from the colour sensor can be used for ● Lane following ● Parking ● Stopping
Touch Sensor	The collision of the vehicle can be simulated using a touch sensor.

2.2 Actuator

An actuator receives a signal and performs an action, often in the form of movement in a mechanical machine.

Actuator	Need
Servo Motors	Used to move the robot in 2 dimensional plane

2.3 Body Shape

The robot's body will comprise a ev3 brick along with two motors that support two attached wheels. Positioned downward, there will be a colour sensor for colour detection and also the intensity of reflected light. The front-mounted touch sensor will serve to identify collisions. Additionally, an ultrasonic sensor will be located at the vehicle's base, directed forward to measure the distance from objects ahead. The image below is the representation of autonomous vehicles and emergency vehicles.

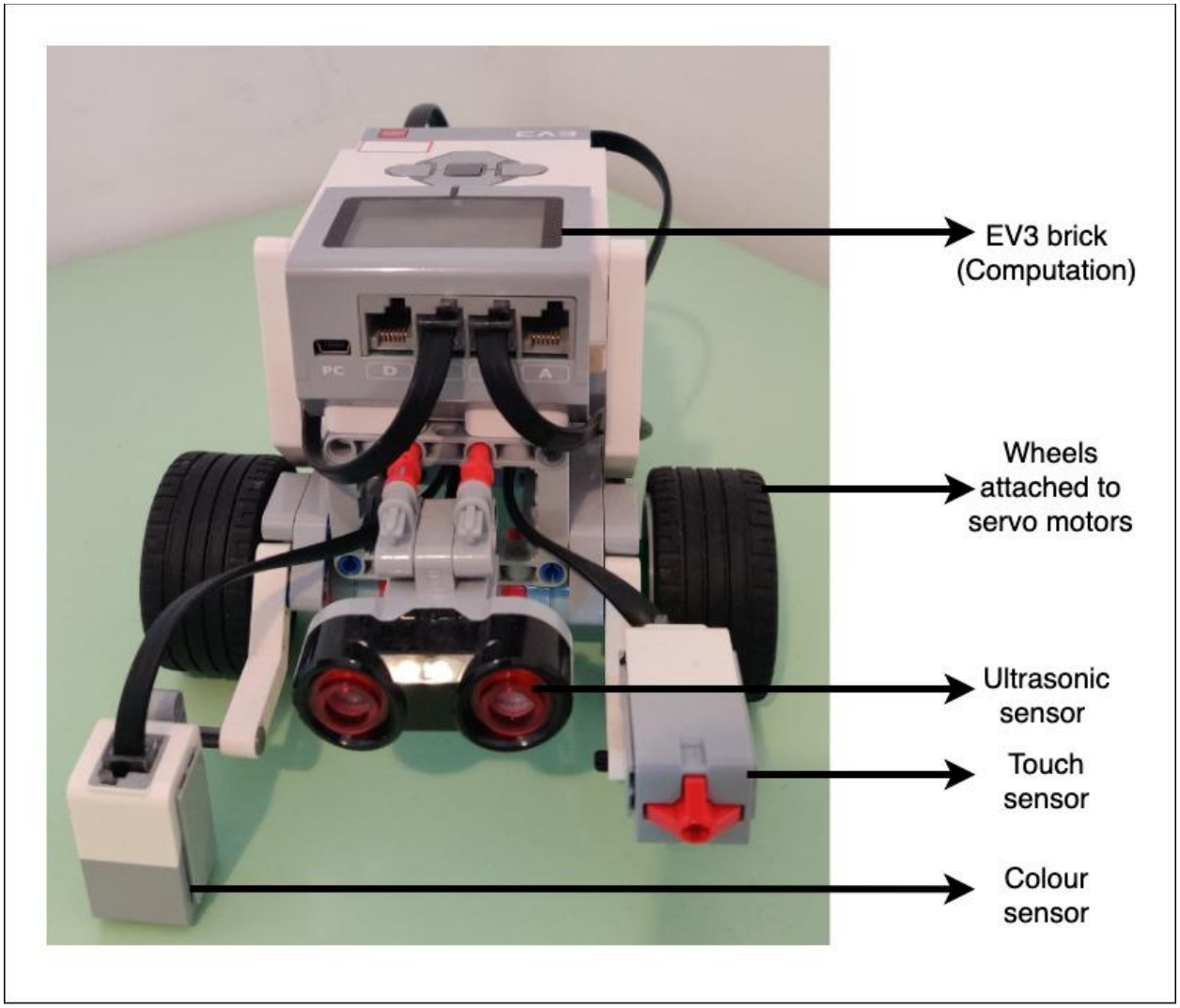


Fig 2: Body shape of the autonomous vehicle

2.4 Degree of freedom (DOF)

Robots operate within a two-dimensional plane. Since there are 3 robots (2 Normal AVs and 1 emergency AV), the DOF is 6.

3. Simulator - Coppelia Sim

The mobile robot chosen is LineTracer. Python3 is used to run the simulation. The sensors and actuators used in the virtual robot are similar. The only differences are:

1) 3 colour sensors are used instead of 1.
2) Touch sensor is not available in Coppelia Sim. Crash/ Accident is simulated by sending a message to the relevant topic.
3) The ultrasonic sensor is not available in the LineTracer robot and a proximity sensor of disc type is attached to the robot.

The figure 3 represents the LineTracer robot with proximity sensor and figure 4 represents the track created in the simulator.

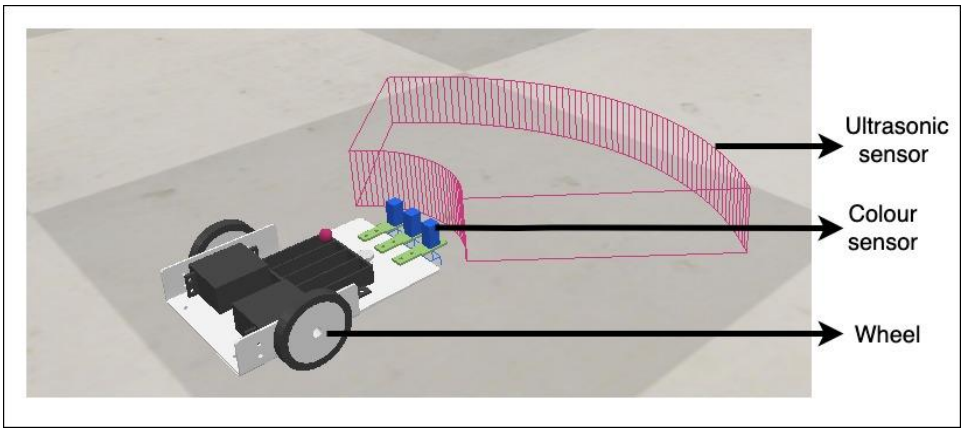


Fig 3: LineTracer mobile robot

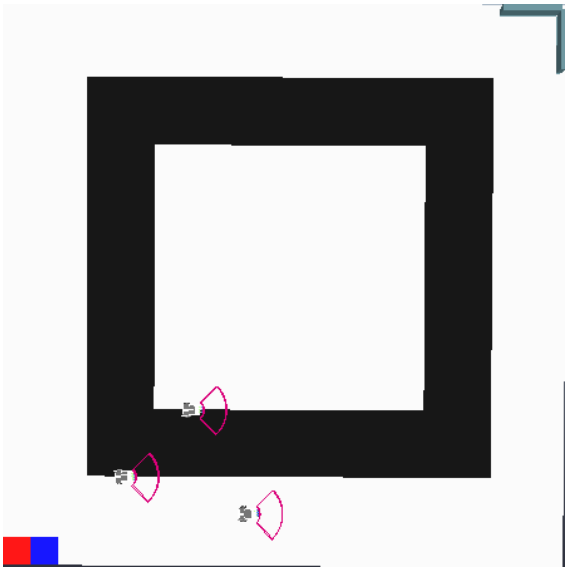


Fig 4: Track created in the simulator

4. Tasks and Capabilities

This section provides an overview of the tasks and capabilities of all actors in the EV3 vehicular network, in both physical and virtual settings.

4.1 Server

All client/AV in this system is connected to Server.

- Monitors the state information of all vehicles..
- The instructions received from the PC client are analysed and communicated to the specified AV. The server is capable of communicating parking and emergency simulation instructions received from a PC client.
- When a collision occurs to a normal AV and it reports an emergency situation to the server, the server must notify all other AVs in the network about the emergency situation. The emergency AV is then alerted. After the emergency AV reports the emergency resolved status to the server, all other normal AVs are notified to return to their normal state.

4.2 Normal Autonomous Vehicle

This vehicle will act as a normal passenger/commercial vehicle on road.

- *Stop Simulation*: When the vehicle detects a red coloured strip on the lane, it will come to a stop and remain stationary until the strip is removed, at which point it will resume movement.
- *Obstacle detection*: The ultrasonic sensors on each vehicle will detect obstacles in their respective lanes. If an obstacle is detected, the vehicle will inform the server and switch to the free lane, coming to a stop.
- *Accident/Crash simulation*: To simulate a vehicle crash, the touch sensor on the vehicle can be pressed, causing it to come to a stop and emit a beeping sound. The server is then notified of the crash, and the beeping sound stops when the server acknowledges the message.
- *Emergency response*: Upon receiving information from the server about an emergency or crash involving another AV, the vehicle will switch to alert mode and move to the safe lane. If the vehicle is already in the safe lane, it will remain in that lane but in alert mode until the emergency is resolved.

- *Parking the vehicle*: Upon receiving a parking command, the vehicle will search for available parking spots, which are identified by blue coloured strips. The UV sensor is then used to check if the parking spot is occupied. If the spot is occupied, the vehicle will continue in the lane until it finds an unoccupied parking spot.
- *Return from parking*: Upon receiving return from parking command to a vehicle which is already parked, the vehicle will make its way back to the track and resume following the lane.

4.3 Emergency Autonomous Vehicle

This vehicle acts as an ambulance.

- *Emergency Alert*: When alerted by the server about an emergency, the AV will switch from a parking state to an emergency state and move to the lane where the accident occurred.
- *Emergency Response Action*: The AV follows the lane with a yellow beacon light and halts upon encountering an obstacle in the lane (assuming the obstacle is the accident vehicle). After reaching the location, it waits for some time and returns back to the emergency parking spot. Vehicle will take a U turn if the distance travelled is less than half of perimeter of track otherwise it will continue in the same direction.
- *Back to the Emergency AV parking spot*: Upon returning to the Emergency parking spot, the AV will take a right turn and park in its initial position when the colour sensor detects the GREEN colour on the lane. It will then alert the server that the emergency has ended.

4.4 Admin/PC Client

The PC client is used to simulate the presence of a third vehicle or to instruct the server.

- *Crash of third vehicle - simulation*: To simulate a crash, the PC client sends a request to the server for assistance, prompting the emergency vehicle to take action and other vehicles to switch to a safer lane.
- *Admin capabilities*: The PC client can act as an administrator, allowing it to instruct the server to park the AVs or declare that the emergency situation has been resolved.

5. State transition diagram

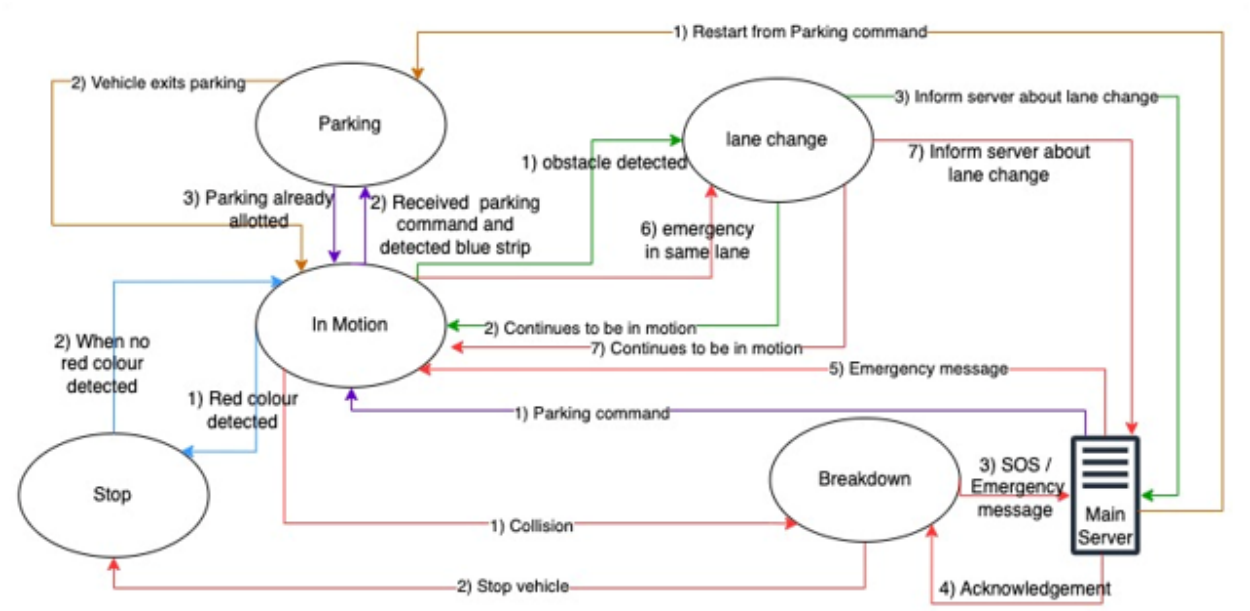


Fig 5: Autonomous vehicle state transition diagram

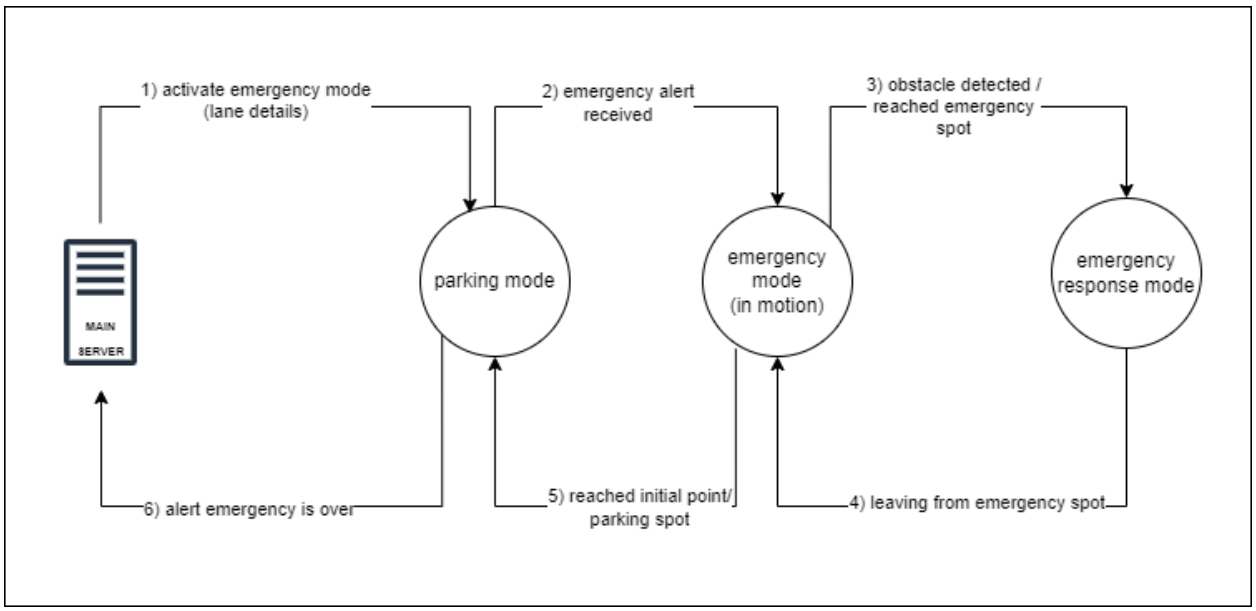


Fig 6 : Emergency vehicle state transition diagram

6. Communication

The communication between the actors in the network is done using bluetooth as it was the only viable option. The PC server essentially acts as a message broker to the connected vehicles and other clients. The documentation to establish connection between multiple EV3 bricks and PC client/server can be found here.

6.1 Message format

The following are the various types of message categories, along with their respective status codes.

Message type	**Message Direction**	**Description**	**Status code**
Crash/SOS	Crashed AV => Server	Crash/SOS alert to server.	10+lane number (0 or 2)
Crash ACK	Server => Crashed AV	Crash ACK from Server to crashed AV when the 'SOS' alert is received.	200
Emergency Alert to Emergency Vehicle	Server => Emergency AV	Emergency alert to Emergency vehicle from server.	10+lane number (0 or 2)
Emergency Alert ACK	Normal AV, Emergency AV => Server	Emergency Alert ACK from both AVs and emergency vehicle to server.	100
Emergency Alert to Normal AVs	Server => Normal AV	Alert from server to Normal AVs on emergency.	10+lane number (0 or 2)
Emergency Over	Emergency AV => Server	Emergency vehicle reached parking after emergency resolution.	0
Emergency Over ACK	Server => Emergency AV	ACK from server to emergency AV when 'Emergency Over' message is received.	200

Emergency Over alert to Normal AV	Server => Normal AV	Alert to Normal AVs when emergency is resolved.	0
Emergency Over ACK - Normal AV	Normal AV => Server	Emergency over ACK from AVs to the server, when server sends ‘Emergency Over’	300
Parking Instruction	Server => Normal AV	Server command to AV for parking	999
Parking Instruction ACK	Normal AV => Server	ACK from AV to server for Parking Instruction.	200
Return from parking instruction	Server => Normal AV	Server command to AV to exit the parking	666
Exit parking instruction ACK	Normal AV => Server	ACK from AV to the server for Return from parking instruction	222

6.2 Sequence Diagrams

Sequence diagrams showing the communication in different scenarios are depicted below.

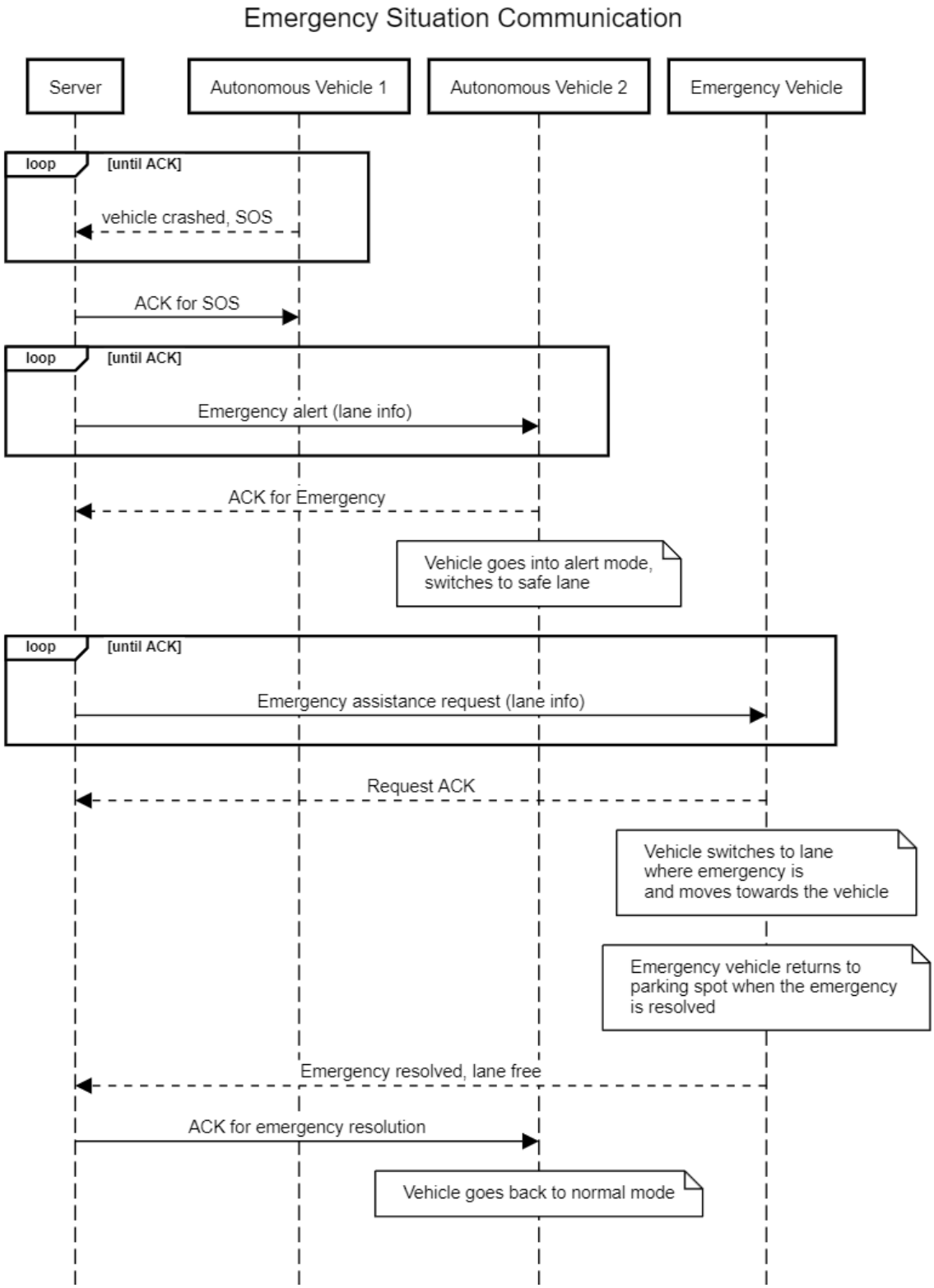


Fig 7 : Emergency situation sequence diagram

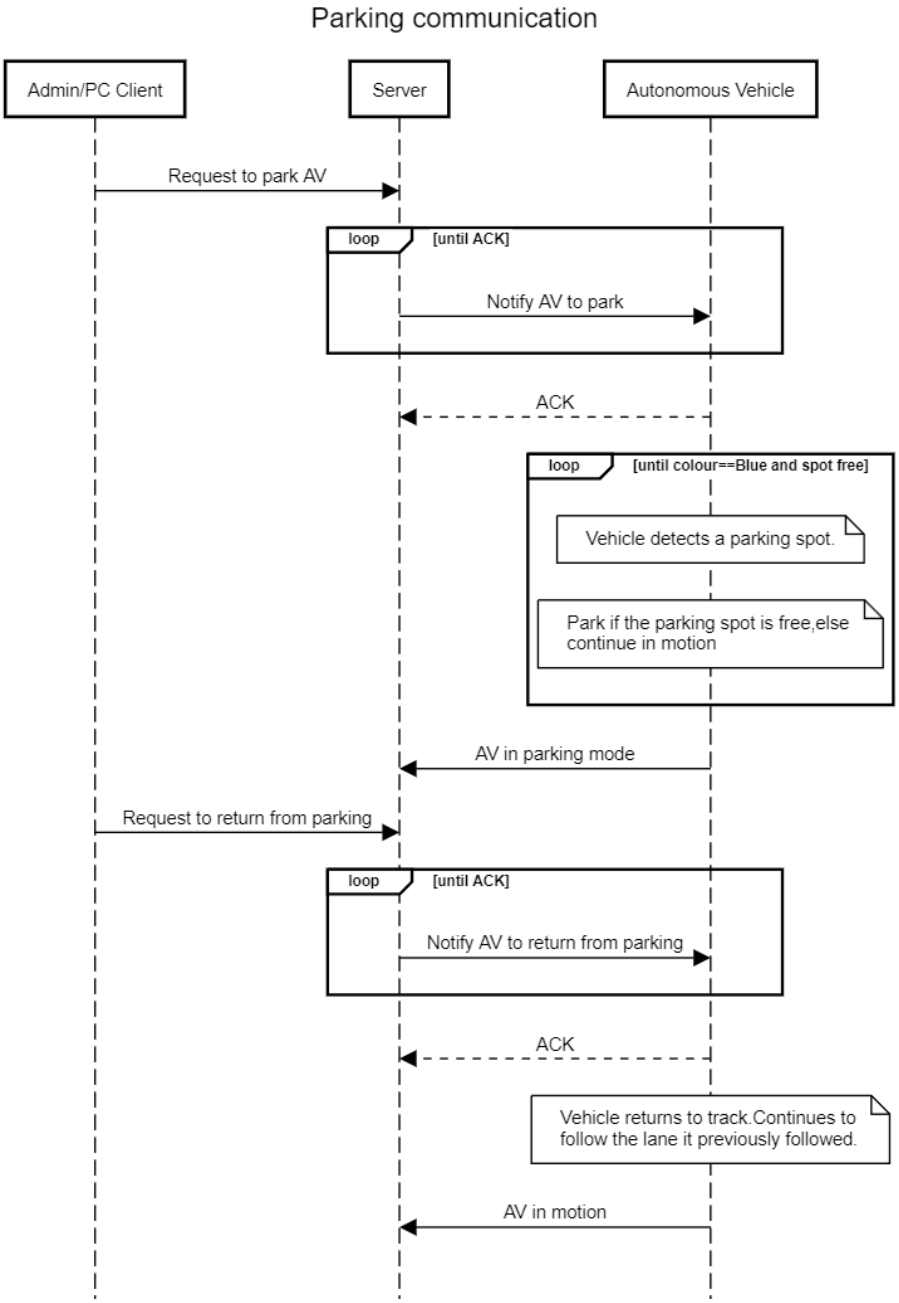
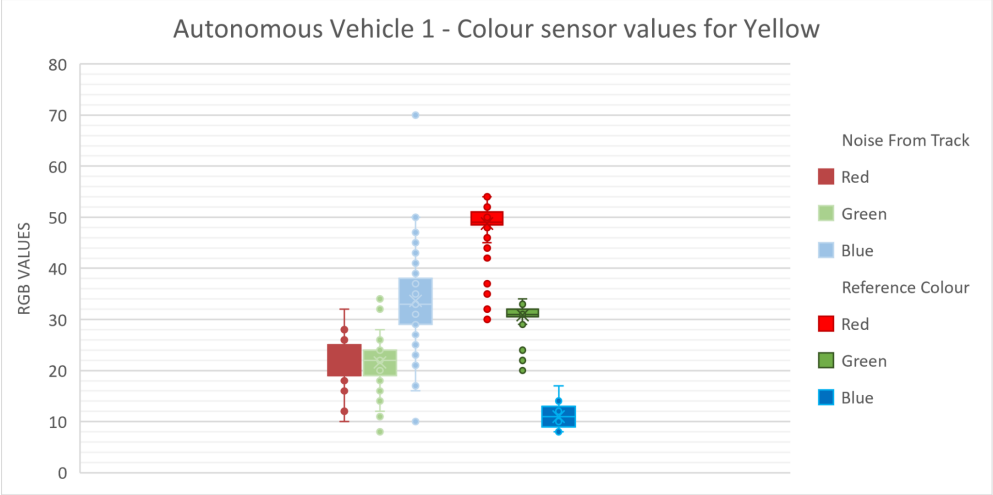


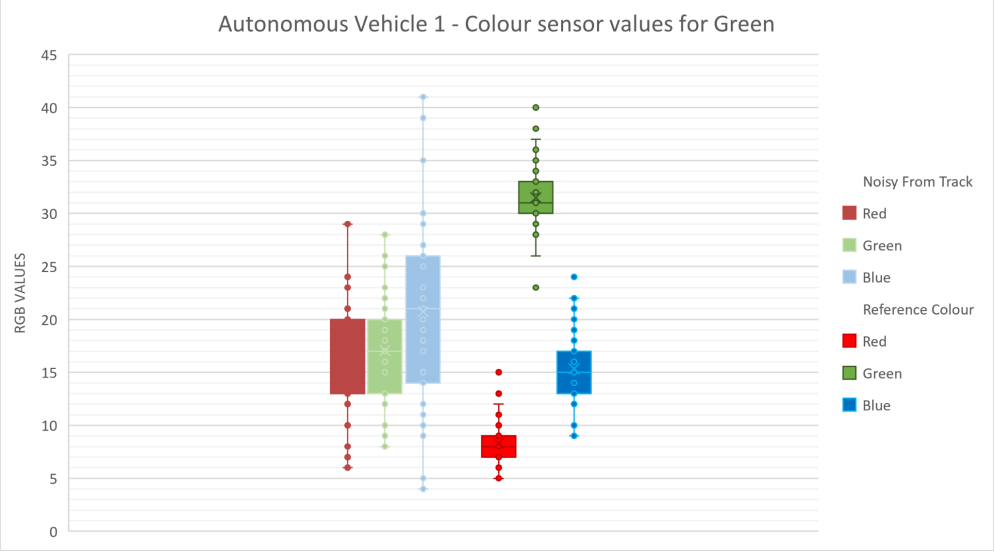
Fig 8 : Parking communication sequence diagram

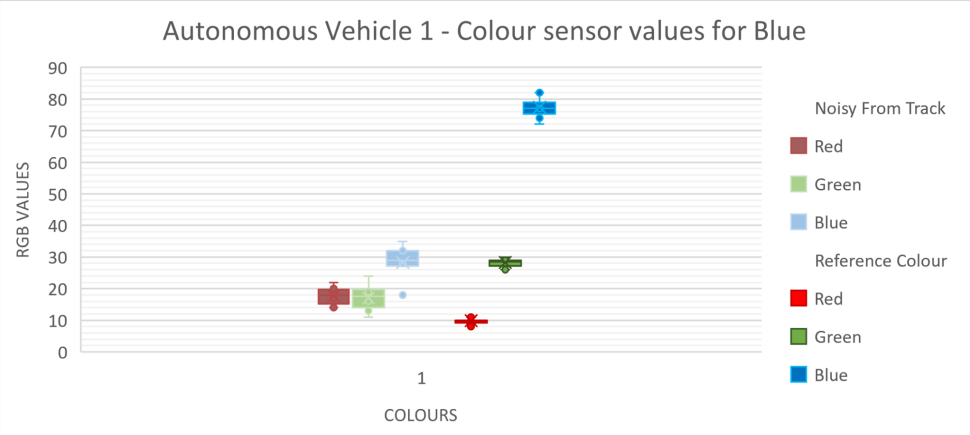
7. Problems and Challenges

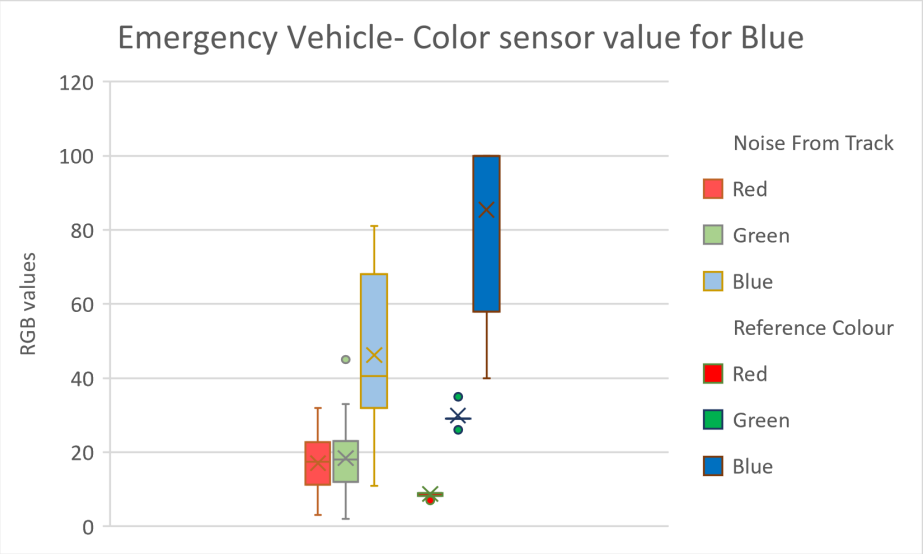
7.1 Colour sensor issue and Sensor data analysis

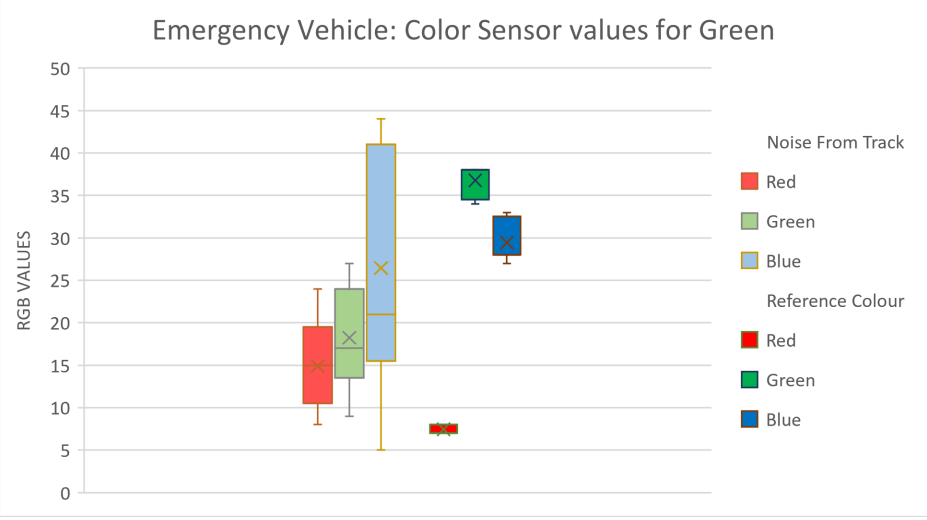
During the vehicle's traversal around the track, it was observed that the sensors detected a significant number of colours, specifically yellow, green, and blue, in addition to white and black which are the colours of the track. This faulty colour detection resulted in the vehicle making erroneous decisions, as each colour held different purposes.The recorded colour sensor values of the track were compared and evaluated against the colour sensor values of the intended reference colours (strips of colour papers) for our project. The vehicle was programmed to take action only if the colours detected matched the reference colours.

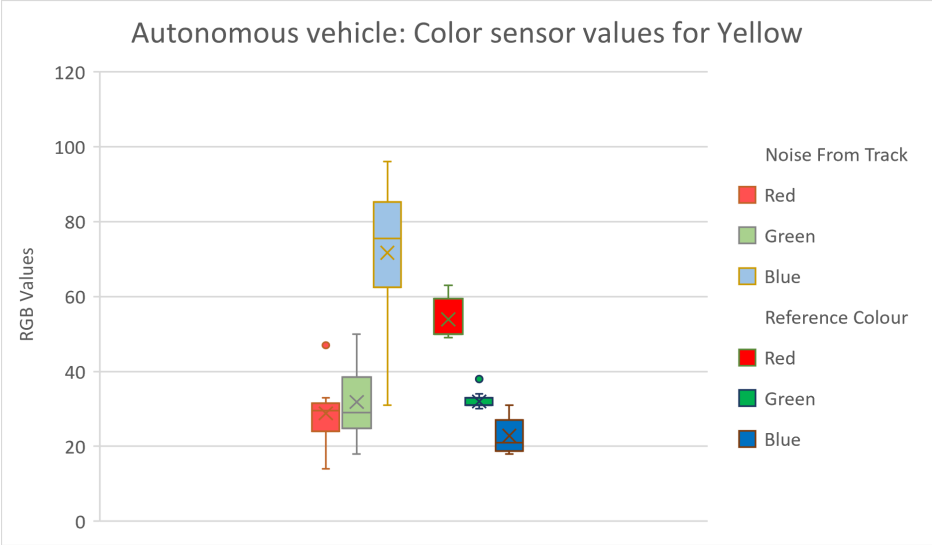





Autonomous Vehicle 1 - Colour sensor values for Blue
RGB VALUES
90
80
70
60
50
40
30
20
10
0
1
COLOURS
Noisy From Track
Red
Green
Blue
Reference Colour
Red
Green
Blue


Emergency Vehicle- Color sensor value for Blue
RGB values
120
100
80
60
40
20
0
Noise From Track
Red
Green
Blue
Reference Colour
Red
Green
Blue


Emergency Vehicle: Color Sensor values for Green
RGB VALUES
50
45
40
35
30
25
20
15
10
5
0
Noise From Track
Red
Green
Blue
Reference Colour
Red
Green
Blue



7.2 Issues with the physical track of the vehicle

We had two options for creating the vehicle track: printing it or using black garbage bags, which were cheaper and more environmentally friendly. Initially, we chose to use the garbage bags, but we soon discovered that their shiny surface caused problems with lane following. Additionally, the uneven material of the garbage bags affected the vehicle's movement. Therefore, we switched to a printed track, and to reduce paper usage, we reduced the size of the track.

Although the current track is performing much better than the garbage bag track, the vehicle's lane following abilities are often disrupted by sensor noise induced by changes in the room's lighting.

Fig 9: The garbage bag based track vs. printed track.

7.3 Compatibility/performance issues

- Bluetooth-based server communication using Pybricks was not supported on both Mac and Windows platforms. To resolve this problem, a USB bootable Ubuntu environment was set up as a workaround.
- The initial choice of simulator was Webots. It requires an NVIDIA or AMD OpenGL. Since the mac didn’t meet this requirement we had to choose Coppelia sim simulator.