



12/14/2017

Blind Runner

Final Project Report – Embedded
Interface Design



Subhradeep Dutta | Sharat Rattehalli Puttaraju | Sridhar Pavithrapu
UNIVERSITY OF COLORADO BOULDER

Team Members:

- Subhradeep Dutta
- Sharat Ratehalli Puttaraju
- Sridhar Pavithrapu

Introduction:

With the advent of autonomous cars using machine learning and computer vision techniques to navigate their way around busy roads, this seemed like the right moment to try our hands at some kind of remote driven car that combines the usage of sensors and cloud connection protocols.

This project aims to create a cloud-controlled car that has multiple user interfaces for navigation and control. The project leverages three Raspberry Pi devices and establishes various communication protocols for them to communicate among each other.

The project aims at developing a prototype for a cloud controlled car (Car Pi) which can be controlled from two different controllers (Gesture & UI Pi) remotely. Amazon Web Services IoT framework was leveraged for our MQTT communications, Lambda for automated code execution, DynamoDB and Rekognition for user authentication and SNS for user notification.

One apt use case for this kind of car would be when parents want to drive their kids to school but they have to be at work at the same time. In these situations, the parent could ask their kids to get seated on the car and then drive the car remotely and back from their office desk. Apart from this there is the added benefit of collision detection and prevention.

Process Steps:

Three Raspberry Pis will be used for this project. Individual functionalities of each of them are highlighted below:

Car Pi: The first raspberry Pi is placed on a cloud-controlled car. A camera is interfaced to the Pi on the car and placed on top to provide a subjective view of the car's movement. The video from the camera is streamed using TCP to the other Raspberry Pis. Apart from this, the car also has four proximity/ultrasonic sensors which are placed on the four sides of the car for collision warning. If the user tries to navigate in a direction that could result in a collision, the car stops moving ahead and warns the user about a possible collision. This sensor data is sent to the AWS cloud

using MQTT protocol, where it is processed using AWS IoT rules engine and a decision is made whether a collision is likely or not. Once the collision is expected or likely, the notification is again sent to the other Pi's using AWS MQTT protocol. Once the car stops, the sensors stop sending data to the AWS server allowing the user some time to realign his car in the right direction. Once the car is back on track, sensor data is sent to the AWS server again.

UI based control Pi: The second Pi has an interactive user interface that has navigation controls for the car. The navigation controls send commands to the cloud using AWS MQTT protocol, which are received by the Pi on the car. The streamed video can be viewed on this Pi.

Gesture based control Pi: The third Pi will be used for gesture navigation which would translate gestures to commands, which will then be sent to the Car Pi using the MQTT protocol.

System Inputs:

- Sensor data from ultrasonic/proximity sensors
- UI inputs for navigation control
- Gesture commands for navigation control
- Camera video and image input

System Outputs:

- Video output from the camera on the car
- Commands to the Pi placed on the car
- SNS email and SMS notifications

Required Super-Project Content:

Code:

- Python is used for code development for all three Pis
- Node.js is used for Lambda function

Two UIs:

- One Qt interface for the Gesture Pi
- One Qt interface for the UI Pi

Network:

- AWS IoT framework is used for MQTT communication

Use of at least Three Communication Protocols:

- MQTT for communicating between AWS IoT, Lambda, DynamoDB and the Pi
- TCP for video streaming
- UDP for inter Pi communication

At least one Sensor Interface/Connection:

- Ultrasonic/Proximity Sensor connected to Car Pi
- Gesture sensor connected to Gesture Pi

System Diagram:

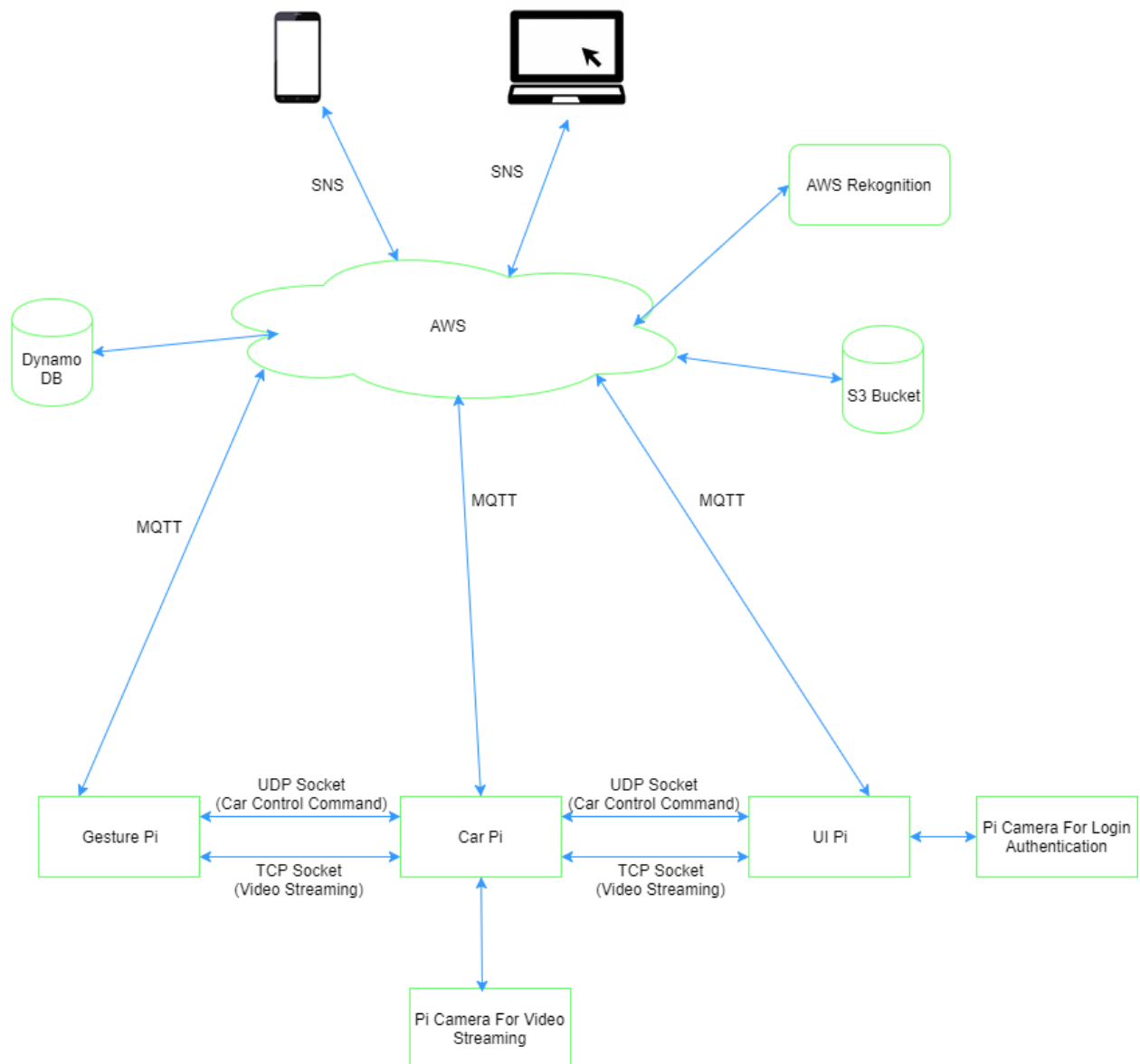


Figure 1: System Diagram

Hardware Setup:

The hardware setup includes the following components:

- Raspberry Pi (3)
- Pimoroni Skywriter HAT Gesture sensor
- HC-SR04 Ultrasonic sensor (4)
- 16000 mAh power bank
- Battery
- Bare bones car model
- Raspberry Pi camera (2)
- L298N motor driver

Hobbyist Car (Extra Credit): To maintain a low budget for the prototype, we reused an existing hobbyist car to suit our application. The motors were connected via a motor driver to the Pi to control them using the GPIO pins.

Car Pi: The Car pi used a bare bones car frame with built in motors that were powered by a battery. A separate power bank was used to power the Pi. Four different ultrasonic sensors were attached to four sides of the car for sensing distance from the nearest obstacle. The Car Pi also has a Raspberry Pi camera that streams the video over the cloud to the UI or Gesture Pi depending upon which one is the current controller.

Gesture Pi: The gesture Pi uses the Skywriter gesture sensor on top of it for sensing gestures in 3D space. The Skywriter uses the I2C protocol to communicate with the Pi. The Gesture Pi has a user interface that allows the user to switch between the two controllers and also a button for enabling gesture commands. An embedded windows displays the video stream from the camera.

UI Pi: The UI Pi has an intuitive user interface where the user can press buttons to control the direction of movement of the car and also see the video stream from the camera in an embedded window.

Software Design:

1. User authentication using Lambda, MQTT, DynamoDB: For enhancing the security of the entire product, a login page was created to authenticate users before providing access to the controls page in the Gesture Pi. It has the following steps
 - a. Initially the user is presented with a login screen. If the user already has an existing credential, the user can directly login and it would redirect the user to the controller page. Else the user will need to sign up to create an account first.
 - b. When the user tries to sign up, the login details are stored in a database in DynamoDB.
 - c. When the user tries to sign in, a Lambda function is triggered which scans through the corresponding DynamoDB database for a match and allows access or displays a dialog box indicating login failure if no match is found in the database. The communication between Lambda and the UI is over MQTT.
2. User authentication using Rekognition and S3 (Extra Credit): For the UI Pi, Amazon Rekognition was used to authenticate users. This was not possible with the Gesture Pi due to the Skywriter covering up the camera port. Once the user hits the login button, an image is captured on the camera and stored locally. Then a copy of the image is uploaded to Amazon S3 bucket. Once the image is uploaded the new image is compared with the image in the collection. The current threshold for matching faces is set to 80%. If the captured image matches atleast 80% with one of the images in the collection, then the user is redirected to the controller page.
3. Gesture Control using AWS IoT framework: For gesture control using the Skywriter, a MQTT topic was created specifically for transmitting commands to the Car Pi. When a certain gesture was identified by the sensor, it published one of the following commands
 - i. Left
 - ii. Right
 - iii. Front
 - iv. Back
 - v. Stop

The Car Pi is subscribed to this same topic and when it receives these commands, the motors are driven accordingly.

4. Navigation control using AWS IoT framework: For controlling the car from UI Pi, when a button is pressed, corresponding commands are published on the command topic. The Car Pi is subscribed to this topic and executes the commands received on this topic.
5. Automatic collision detection and prevention: The Car Pi has four ultrasonic sensors, one on each side. The sensor values are read and transmitted continuously to AWS Lambda using MQTT on another topic. Each time the sensor values are compared to a threshold value by the Lambda function which is written in Node.js. If the sensor value is less than the threshold value, it means that the obstacle is too close and a collision is likely. Then the Lambda function sends a “Stop” command to the Car Pi which causes the car to stop moving. It also displays a popup window showing collision is likely on which side.
6. Video streaming using TCP: The video captured by the Pi camera is transmitted over TCP to the Gesture Pi and UI Pi. This captured video is then displayed within an embedded window.
7. Controller Button: To determine which Pi gets to control the Car Pi, a slider button was created. In the default state, control is with the UI Pi. When the slider is moved to Gesture Pi, a MQTT message is published on a certain topic. The UI and Gesture Pi are subscribed to this topic. On receiving the published message, the UI and Gesture Pis update a global variable that keeps track of which Pi has control. The other Pi’s controls are disabled for the time being.
8. Current status notification using UDP and SNS: Apart from the slider to indicate which Pi has control, Amazon SNS service was used to indicate which Pi currently controls the car. The Pi which currently has control sends a message over UDP to the Car Pi indicating that it has control. The Car Pi receives this message and publishes it to Amazon SNS. Whenever control is switched from one Pi to another, a SMS and email is sent to the subscribed mobile numbers and email addresses telling that control is now changed to the corresponding Pi.
9. Multithreading (Extra Credit): Since there are multiple functionalities that needed to run simultaneously, multithreading was the only option to put everything together. For example, the camera video stream was supposed to run indefinitely and the corresponding navigation buttons are supposed to work simultaneously. So one thread was assigned to the camera stream and another thread was responsible for publishing commands on the MQTT topic.


UI Mockups:

Welcome page

The Blind Runner

User name

Password

New User? Register here 

Sharat RP Sridhar Pavithrapu Subhradeep Dutta

Figure 2 UI Pi Login Page

Main Page

Exit

Front

Video is shown here!!

Left

Right

Back


Gesture  UI

Figure 3: Main Page of the UI Pi

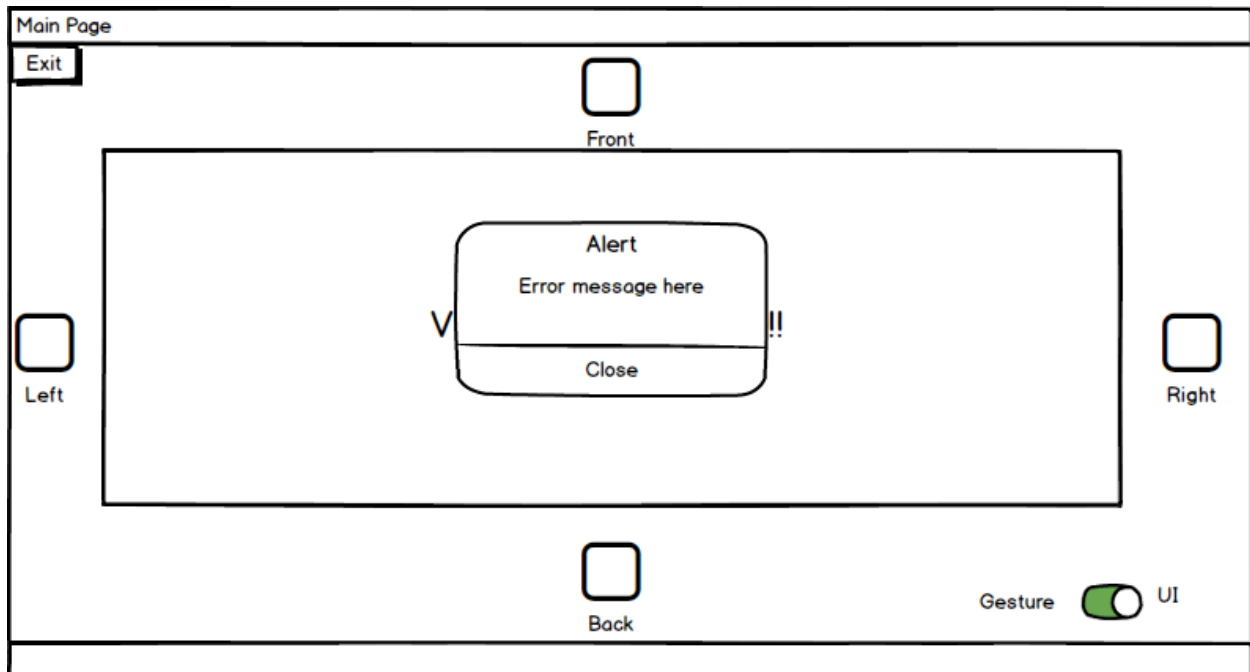


Figure 4: Alert message on the Main Page of the UI Pi



Figure 5: Main Page of the Gesture Pi

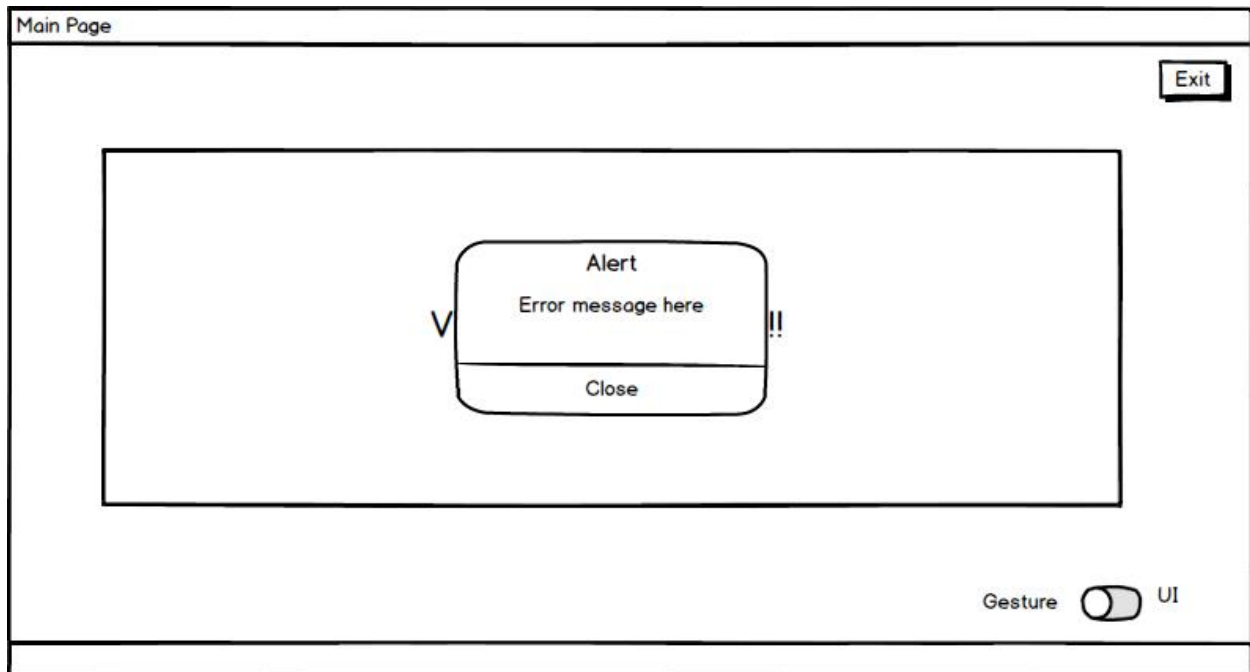


Figure 6: Alert message on the Gesture Pi's Main Page

The image shows a web browser window titled "Registration Page". In the top left corner, there is an "Exit" button. The main content area contains three text input fields. The first field is labeled "Name*", the second is labeled "Password*", and the third is labeled "E-Mail*". Below these fields is a "Register" button.

Figure 7: Registration page on the Gesture Pi

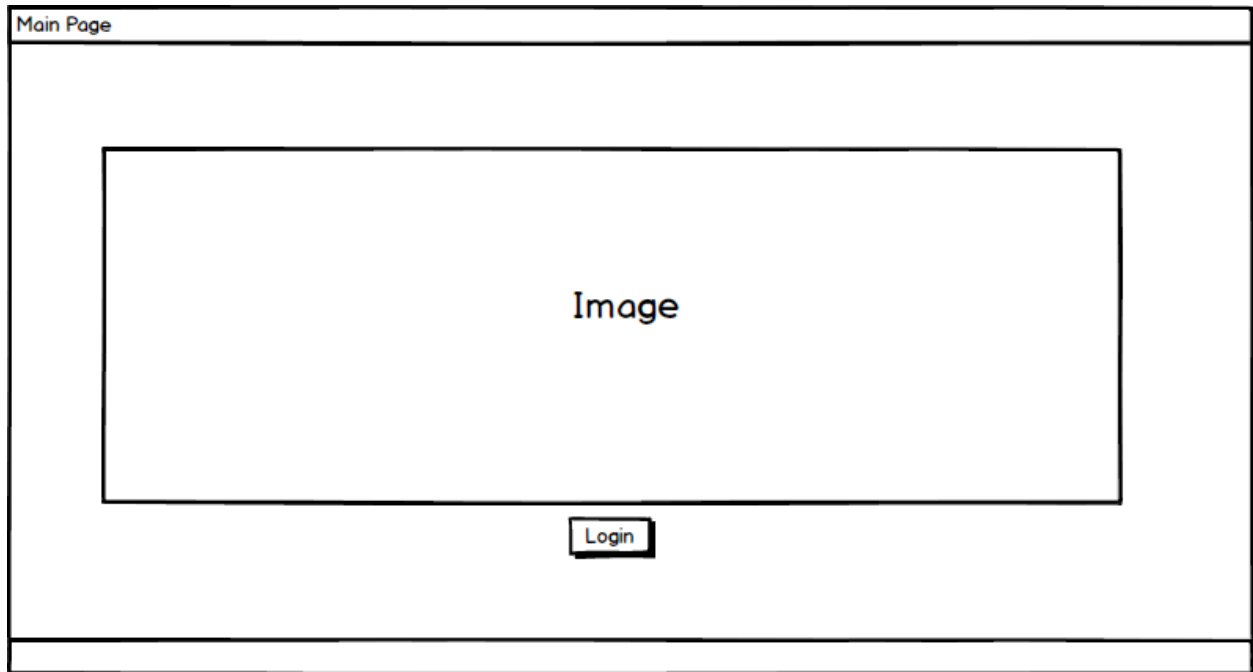


Figure 8: Login Page on the UI Pi

Sequence Diagrams:

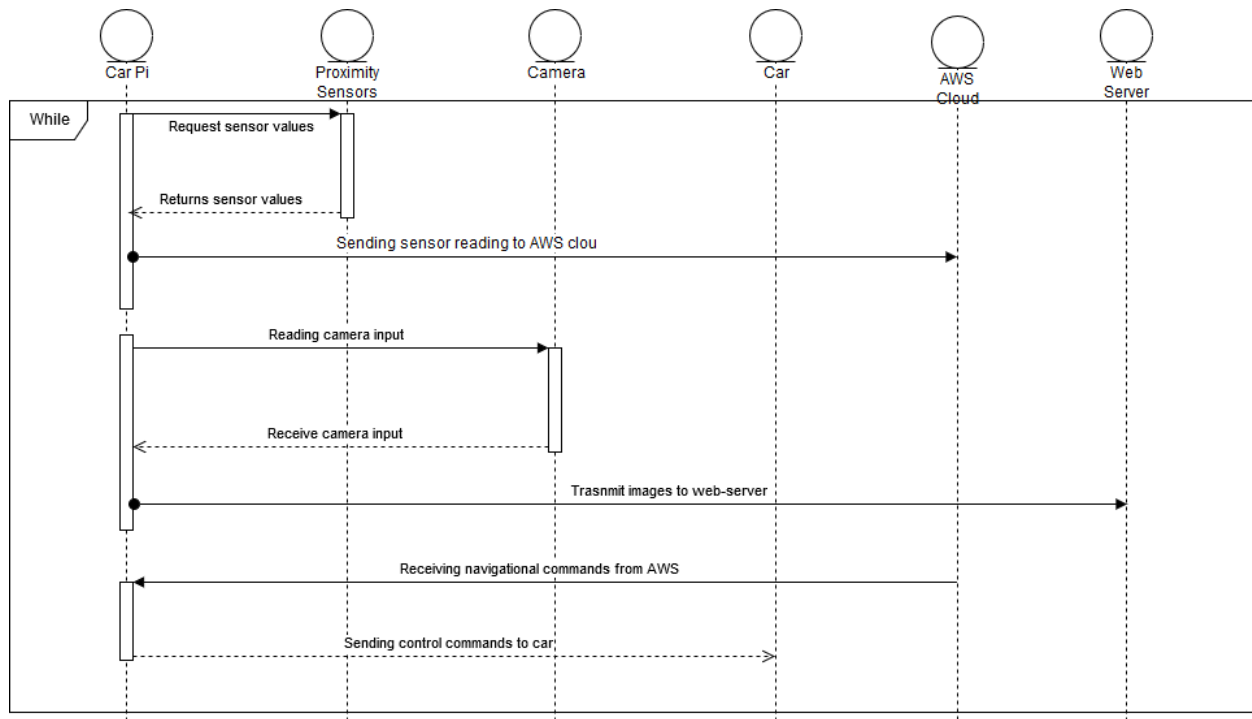


Figure 9: Car Pi Sequence Diagram

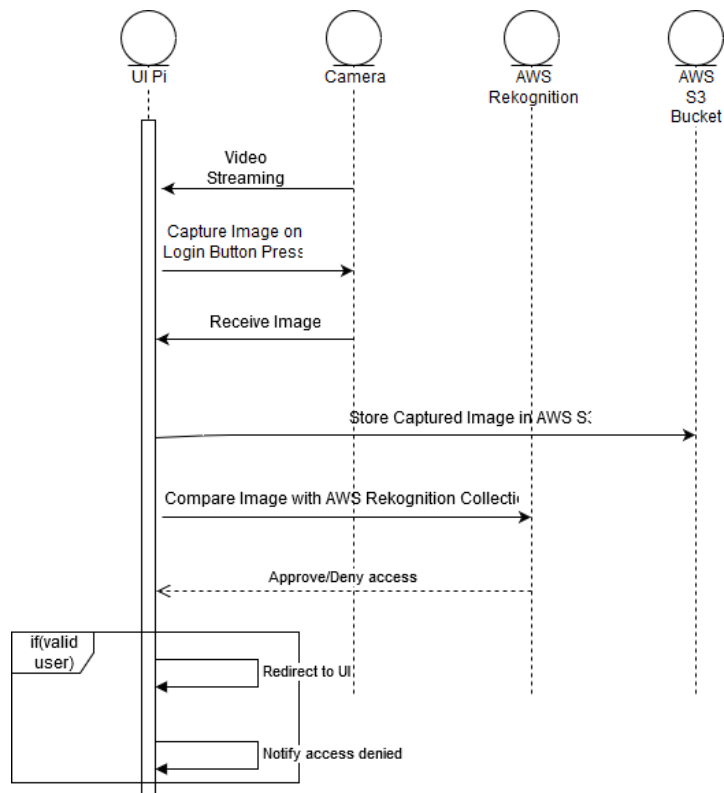


Figure 10: UI Pi Main Sequence Diagram

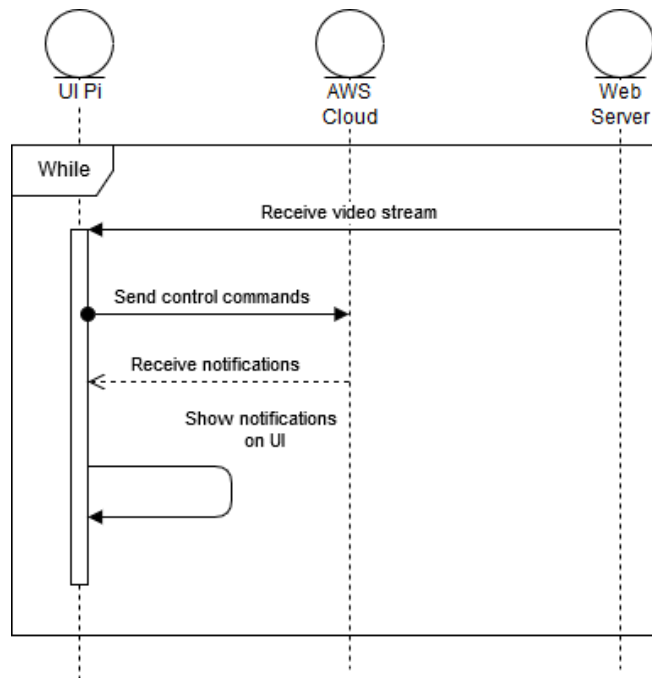


Figure 11: Login Sequence Diagram of the UI Pi

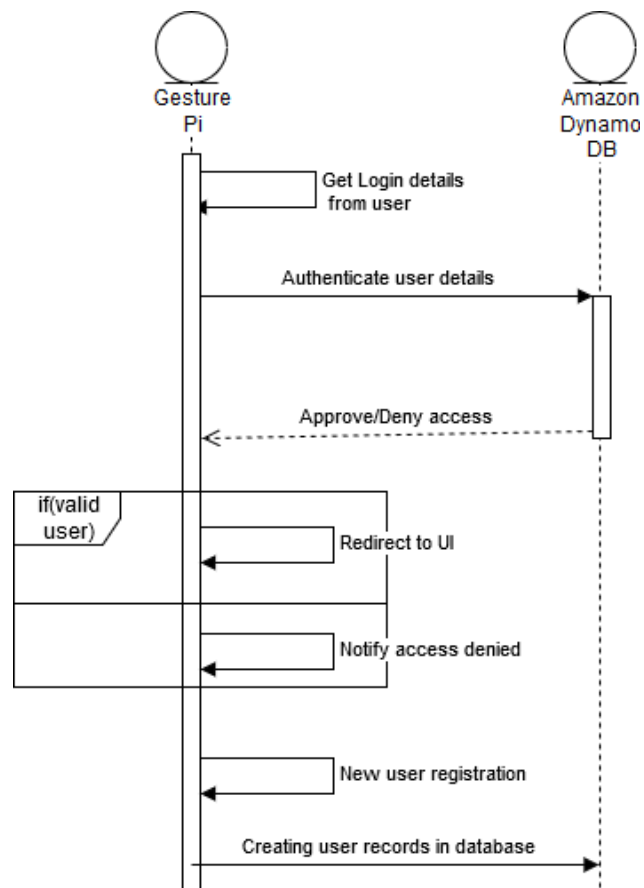
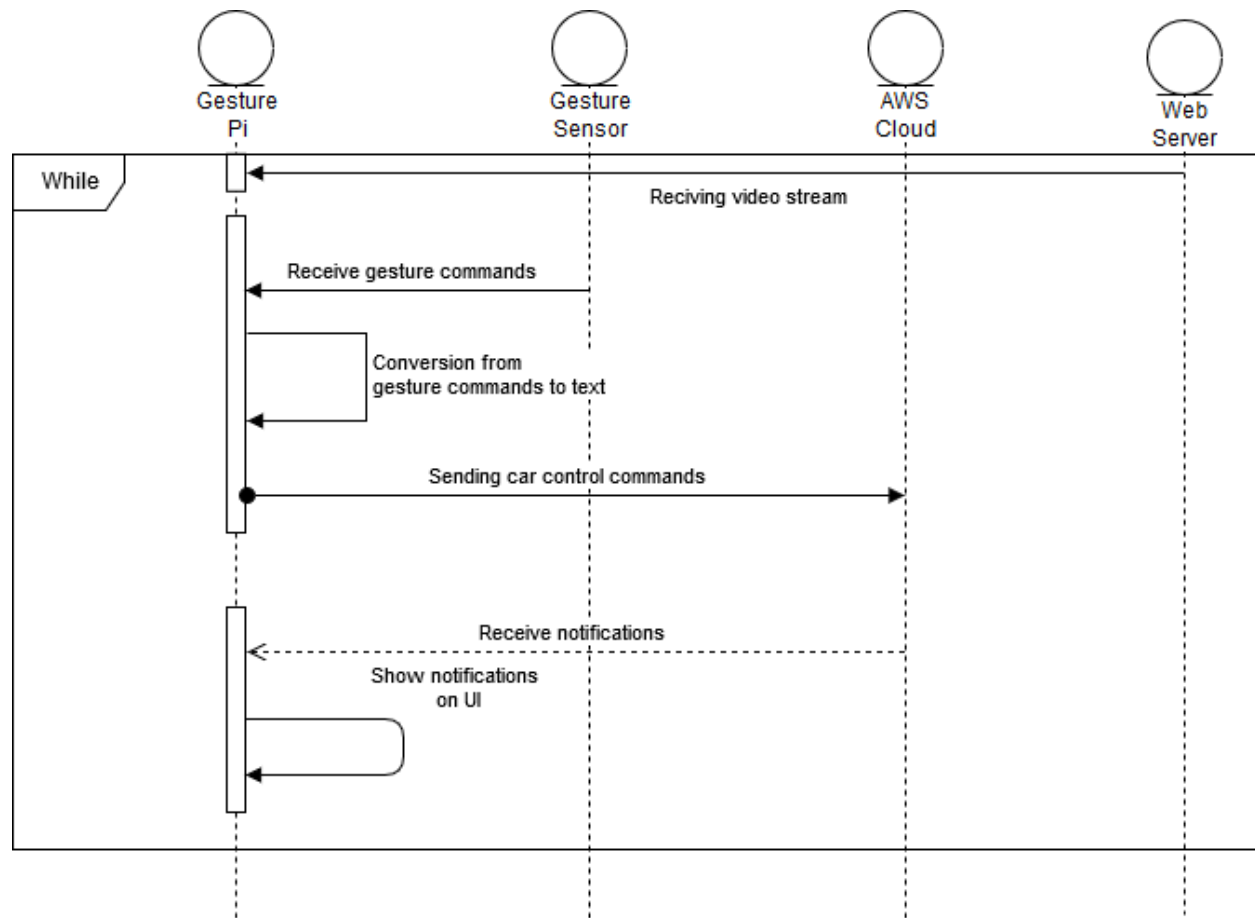
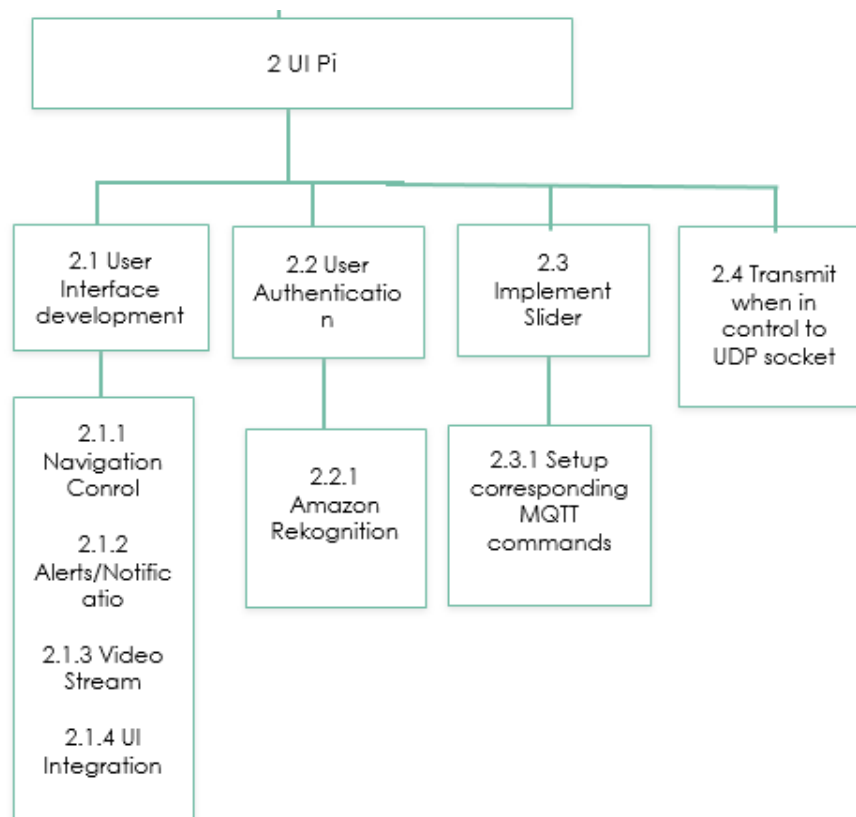
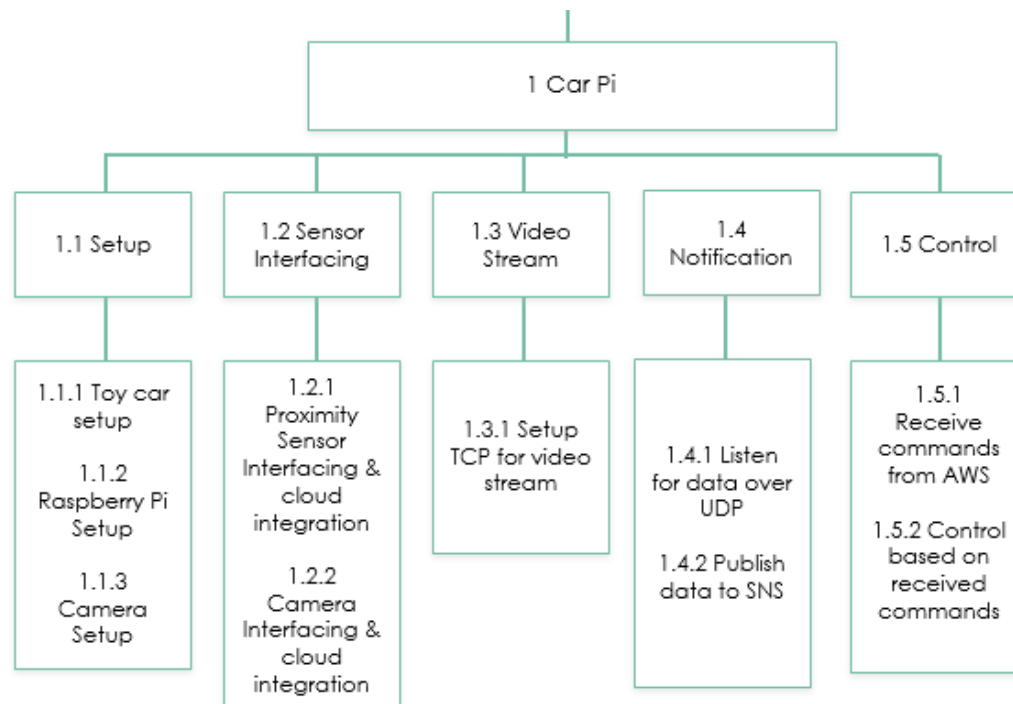


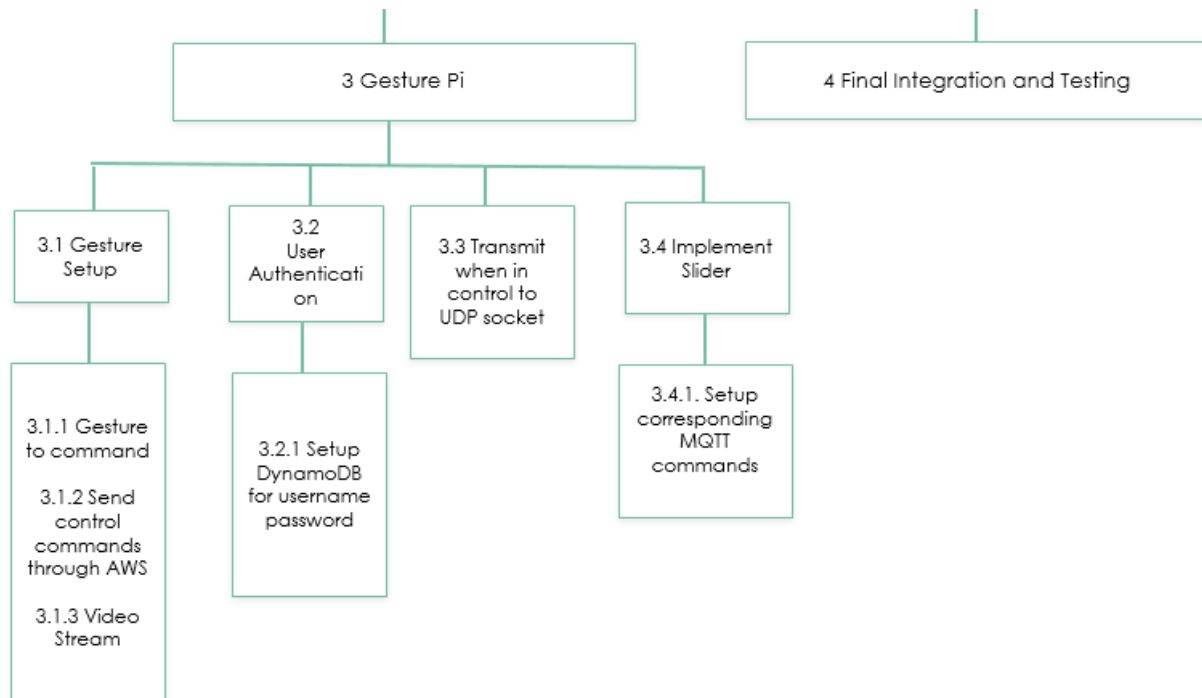
Figure 12: Login Sequence Diagram on the Gesture Pi

*Figure13: Gesture Pi Main Sequence Diagram*

Project Management:

Work Breakdown Structure:



**Distribution of labor:**

PROJECT TITLE			
1	Activity Title	Days of work	Person Incharge
1.1	Car Pi		Sharat
1.1.1	Toy Car setup	2	
1.1.2	Raspberry Pi setup	0.5	
1.1.3	Camera setup	1	
1.2	Sensor Interfacing		Sharat
1.2.1	Proximity sensor Interfacing and Cloud integration	3	
1.2.2	Camera Interfacing and cloud integration	3	
1.3	Video Stream		Sridhar
1.3.1	Setup TCP for video stream	2	
1.4	Notification		Subhradeep
1.4.1	Listen for data over UDP	4	
1.4.2	Publish data to SNS	1	
1.5	Control		Subhradeep
1.5.1	Receive commands from AWS	2	
1.5.2	Control based on received commands	2	

2	UI Pi		
2.1	User Interface development		Subhradeep
2.1.1	Navigation Control	4	
2.1.2	Alerts / Notifications	2	
2.1.3	Video Stream	1	
2.1.4	UI Integration	2	
2.2	User Authentication		Sharat
2.2.1	Amazon Rekognition	3	
2.3	Implement Slider		Sharat
2.3.1	Setup corresponding MQTT commands	3	
2.4	Transmit when in control to UDP socket		Sharat
3	Gesture Pi		
3.1	Gesture Setup		Sridhar
3.1.1	Gesture to command	4	
3.1.2	Send control commands through AWS	2	
3.1.3	Video Stream	1	
3.2	User Authentication		Sridhar
3.2.1	Setup DynamoDB for username password	6	
3.3	Transmit when in control to UDP socket		Sridhar
3.4	Implement Slider		Subhradeep
3.4.1	Setup corresponding MQTT commands	6	
4	Final Iteration and testing	5	Sridhar, Sharat, Subhradeep
	Total	59.5	

Project Overview:

1. Protocols: MQTT, UDP, TCP
2. Sensors: Ultrasonic sensor, camera, gesture sensor
3. AWS Services: IoT framework, S3, Rekognition, Lambda, DynamoDB, SNS

Challenges:

1. The primary challenge was to have all the functionalities run simultaneously. We had to resort to multithreading to overcome this.
2. Initially the idea was to use voice to control the Pi for one of the controllers but the performance for voice recognition was extremely unreliable and the overall user experience was not smooth since the user had to repeat the same commands until the server was able to correctly identify it. Another reason for switching to a gesture based interface was that if a voice command was wrongly identified it could result in a possible collision.
3. The bare bones car model had some preexisting issues such as incorrect wheel alignment as a result of which the car did not move in a straightforward manner.
4. Some time was spent in trying to work with non-functional components such as the microphone and the gesture sensor
5. Rekognition is a fairly new feature and there is very little documentation on how to set it up.
6. The Pi is a single board computer with minimal resource. It is not an ideal candidate for running CPU intensive operations and multithreading. For applications such as cloud driven cars, for a real-time performance more computing power is recommended.

Results & Conclusion:

A prototype of a cloud controlled car was developed with a secure user interface. A video stream allowed the user to view where the car was headed and control the car accordingly. The project was instrumental in getting experience with various cloud protocols and rapid prototyping. AWS cloud infrastructure was leveraged to transmit data from one machine to another via the cloud.

The designed prototype gave us a glimpse of the difficulties faced by autonomous car makers. Nevertheless we were able to navigate the car using the video stream seen through the embedded user interface. The prototype was able to detect possible collisions using the ultrasonic sensors and avoid actual collision as expected. A real world version of this prototype would have more powerful hardware under the hood to achieve real time performance.



Image 1: User Interface of Gesture Pi



Image 2: Gesture Pi Controls Page

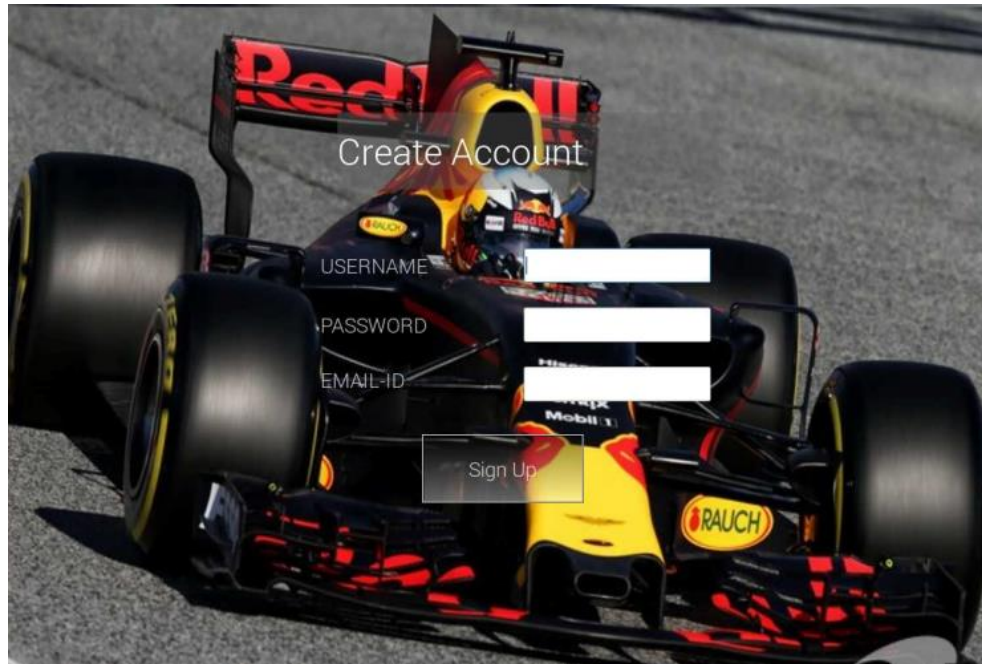


Image 3: Sign Up Page of Gesture Pi

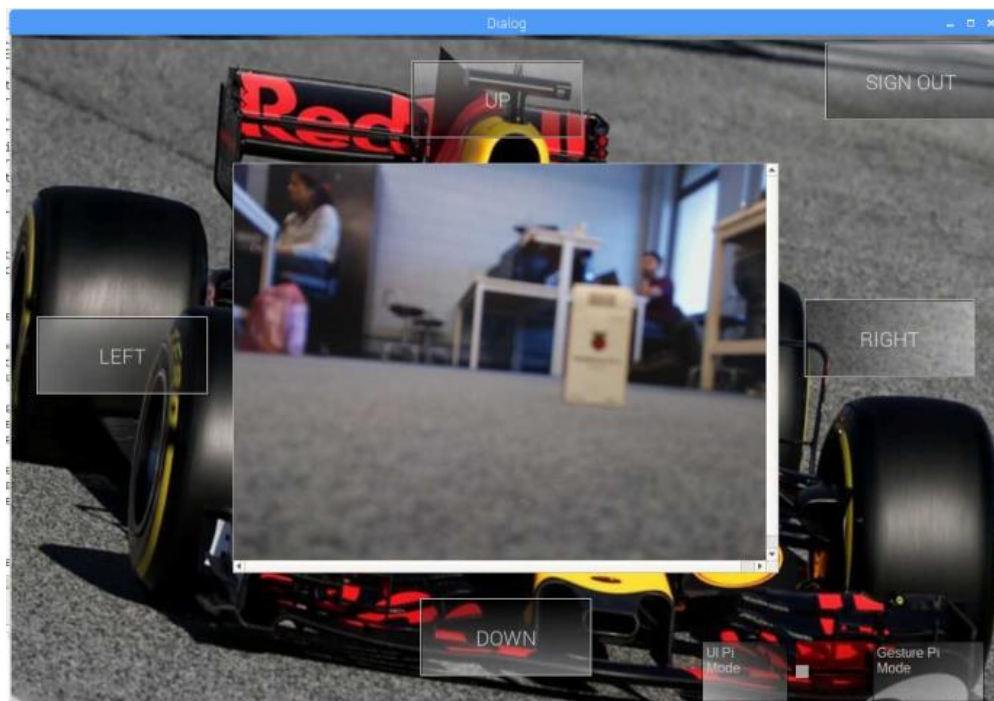


Image 4: UI Pi Controls Page



Image 5: UI Pi Login Page

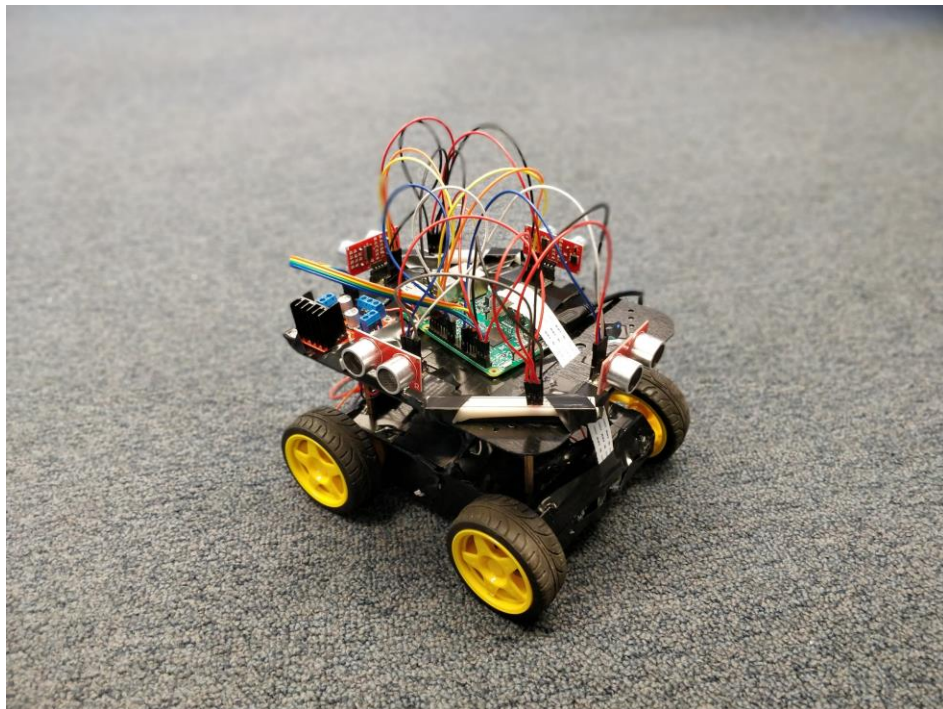


Image 6: Car Pi-1

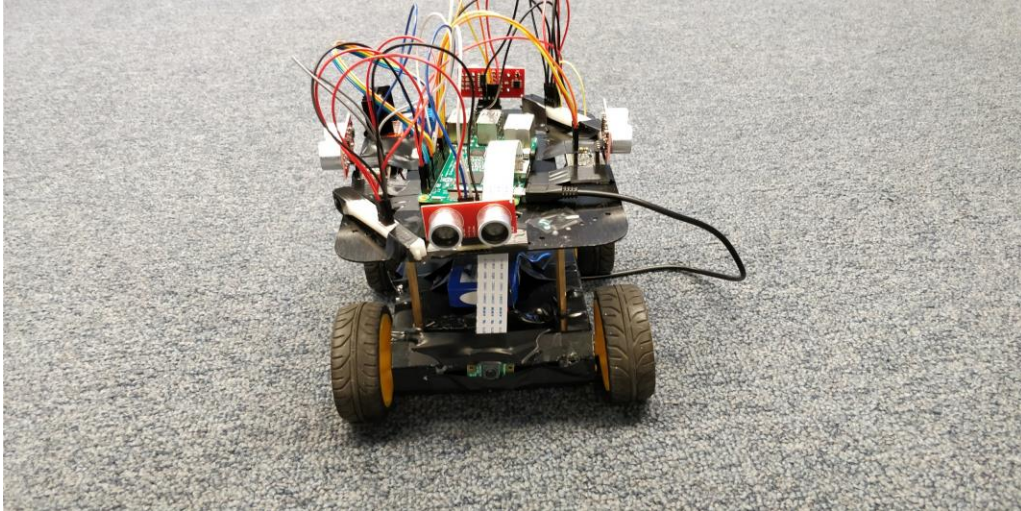


Image 7: Car Pi-2

Acknowledgement:

We would like to thank Professor Bruce Montgomery for introducing us to various cloud infrastructures and their endless applications in building IoT devices. The initial coursework on UX was instrumental in designing better interfaces for our super projects.

References:

- Open source community including stackoverflow
- <https://github.com/dwarcher/amazon-rekognition-example/blob/master/import.js>
- <https://gist.github.com/alexcasalboni/0f21a1889f09760f8981b643326730ff>
- <https://www.raspberrypi.org/documentation/usage/camera/python/README.md>
- AWS Documentation
- <https://www.hackster.io/mariocannistra/python-and-paho-for-mqtt-with-aws-iot-921e41>
- <https://martinfitzpatrick.name/article/multithreading-pyqt-applications-with-qthreadpool/>
- <https://nikolak.com/pyqt-threading-tutorial/>
- <http://doc.qt.io/qt-5/thread-basics.html>
- <https://github.com/pimoroni/skywriter-hat>