Smart Cycle - CS321 Course Project

Group GAP
Anindya Vijayvargeeya - 2001010015
Gunjan Dhanuka - 200101038
Pranjal Singh - 200101084

Table of Contents

Project Idea

Problems Identified

Through our experience of staying on the campus for about three years and talking to other people in our departments and hostel, we were able to identify the following four problems:-

Theft

There are frequent emails on the campus Outlook group almost every day where a cycle is lost or stolen, and the owner of the cycle faces a lot of trouble in trying to find his bicycle among the thousands of bicycles on the large campus. This has been a rampant issue, considering the utility and price of these cycles.

Accidents

Various roads on the campus are pitch dark at night and are very dangerous for cycle riders.

Navigation

It is not very easy to navigate to an unknown place using a cycle due to a lack of features like dashboards in cars, etc.

Our Solution

Anti-theft Alarm

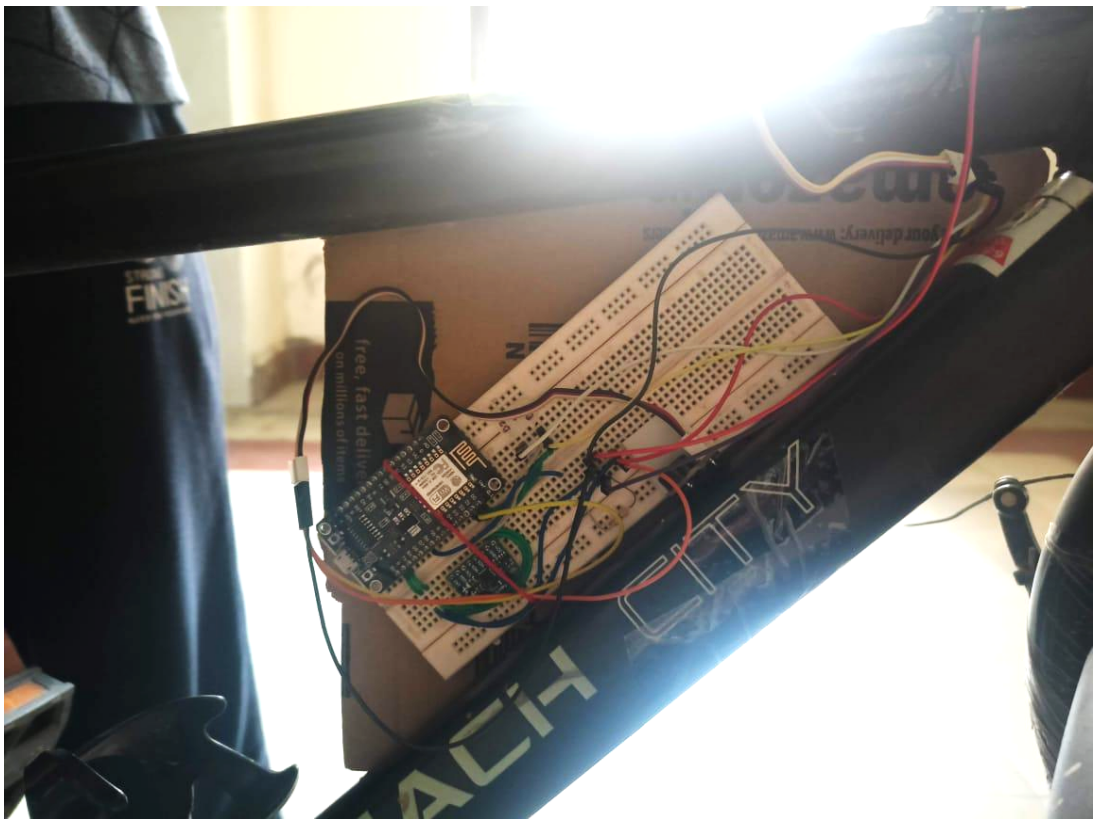
- The theft detection is turned off when the cycle is connected to the owner's phone.
- Otherwise, GPS and MPU6050 modules detect theft and send the cycle's location and a camera that transmits the photo of the thief to the cloud, which the owner can access.
- An alarm starts sounding if the module is tampered with or the system detects cycle is being stolen.

Find your cycle

- When paired with the phone, the user can select to navigate to a particular destination.
- Then the turn-by-turn navigation data would be fed to the controller.
- The controller would turn on/off LED lights attached to the handle based on the distance to the turning point.
- When the point is distant, the LED will blink slowly. When the rider is near the turn, the LEDs will start blinking fast, prompting the user to turn.

Automatic Lights

- We use a Digital Light Sensor to determine the ambient light of the surroundings.
- The headlights will automatically turn on if the ambient lights fall below a certain threshold.
- Also, when we detect deceleration using the MPU6050 sensor, the tail light turns on to indicate that the cycle is about the stop.



Equipment Required

1. Esp8266 NodeMCU controller
2. Mpu6050 accelerometer and gyroscope
3. Grove v1.1 light sensor
4. Gps neo 6m i2c sensor
5. Breadboard
6. Power bank

7. Buzzer
8. LED
9. Resistors
10. Jumper cables
11. Cycle
12. Cardboard
13. Double-sided tape
14. Scissors

Methodology

Hardware

Interfacing the sensors and other devices:-

ESP8266 NodeMCU

Libraries required:-
ESP8266 by ESP8266 community, ESP8266 board manager add-on, CP2102 driver.

How to:-

1. Firstly on the installed Arduino IDE, add the *ESP8266* library by the ESP8266 community.
2. Add an *ESP8266 board manager* to the Arduino IDE, which can be done by following the link.
3. Install the *CP2102 driver* on the laptop, which can be downloaded from this link.
4. Connect the laptop to the NodeMCU using a USB cable; before uploading, we need to select the com port, and the type of the board should be *NodeMCU 1.0(ESP-12E Module)*. Interfacing with NodeMCU is done.

MPU6050

Libraries required:-
MPU6050.h

How to:-

1. Go to the *Sketch* menu in Arduino IDE, in the Include library menu select *add .ZIP library* option. Select the *ESP8266_MPU6050-master.zip* file from the project folder. This contains the *MPU6050.h* file.
2. Connect the GND and VCC pins of the MPU6050 sensor to the GND and VCC pins of the NodeMCU . Connect the SCL and SDA pin of the MPU6050 sensor to D1 and D2 pins of the NodeMCU respectively.
3. A green light should glow indicating the sensor receives power and our interfacing is done.

Grove v1.1 light sensor

Libraries required:-

Digital_Light_TSL2561.h

How to:-

1. Firstly install the *Grove - digital light sensor* module on arduino from the arduino store, it contains the above mentioned library.
2. Connect the GND and VCC pins of the light sensor to the GND and VCC pins of the NodeMCU. Connect the SCL and SDA pin of the light sensor to D1 and D2 pins of the NodeMCU respectively.
3. Interfacing of the light sensor is done.

GPS neo 6m i2c

Libraries required:-
TinyGPS++.h

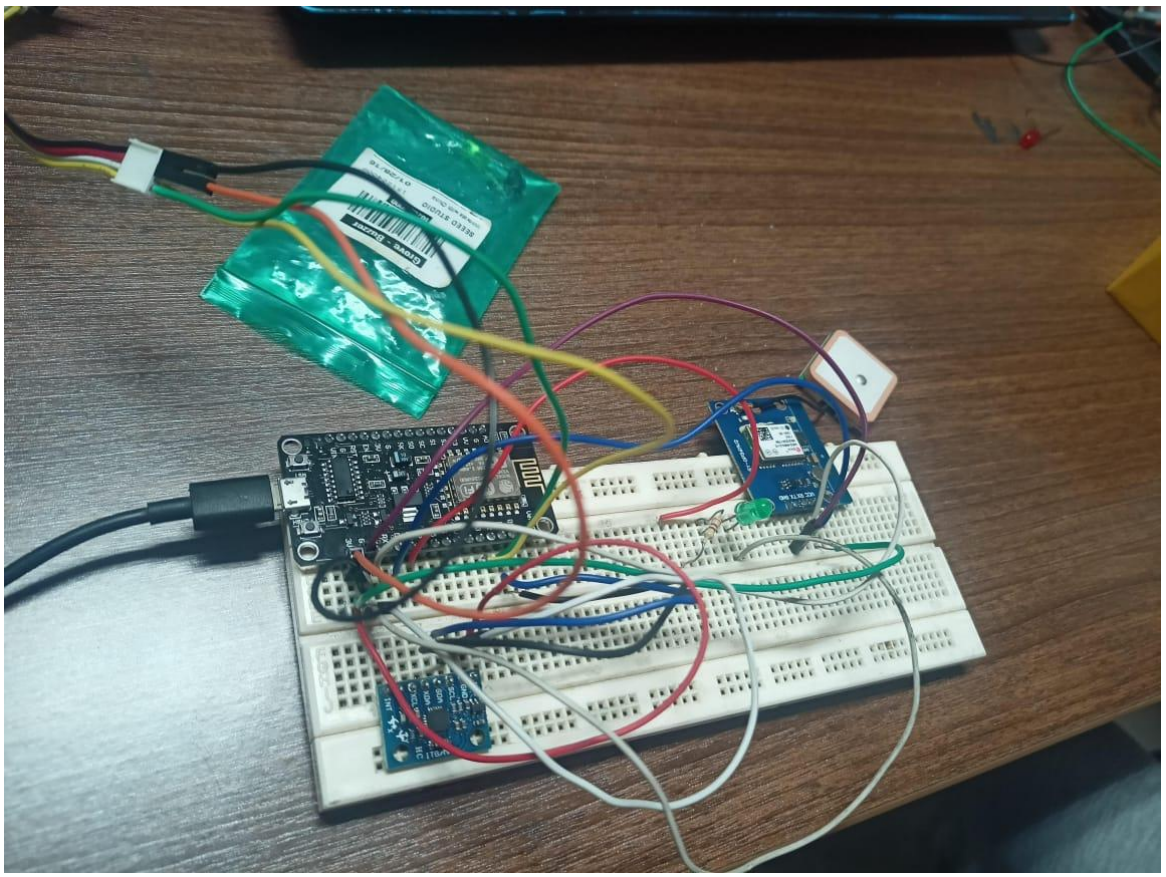
How to:-

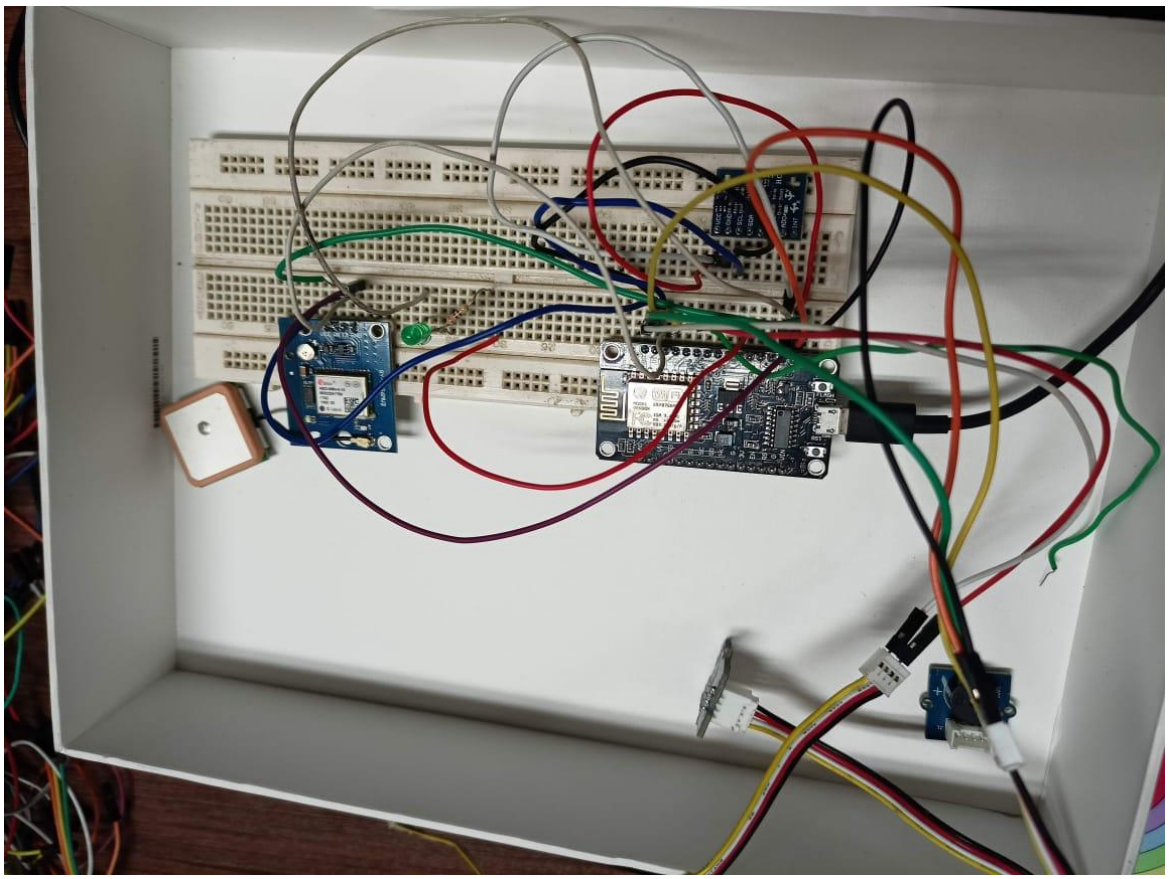
1. Install the *TinyGPSPlus-1.0.2.zip* library in a similar manner as the MPU6050 library.
2. Connect the GND and VCC pins of the light sensor to the GND and VCC pins of the NodeMCU. Connect the Tx and Rx pin of the gps sensor to D1 and D2 pins of the NodeMCU respectively.
3. The gps sensor will take 1-5 minutes to setup. After which a red light will start blinking on the sensor.

Buzzer
Connect the VCC and GND pins of the buzzer to the VCC and GND pins of the NodeMCU. Connect the SIG pin of the buzzer to D0 pin of the NodeMCU.

LED
Connect the GND pin of the LED to the GND pin of the NodeMCU and connect the VCC pin of the LED to the D7 pin of the NodeMCU.

Circuit images:-





Mounting the circuit on a cycle:-

We created a triangular shaped box using cardboard which could easily fit in the space near the bottle stand of a cycle. We fixed our system to the box using a double sided tape along with a power bank to power the system. Then we fixed the box on the cycle using duct tape. The light sensor needed to be exposed to the environment we fixed it on the centre bar of the bicycle with double sided tape and similarly we fixed the headlight LED to the handle bar.

Software

NodeMCU

Our hardware is controlled by the NodeMCU which was programmed with Arduino IDE, containing several scripts to control the MPU sensor, the LDR sensor, the GPS sensor and to handle the MQTT protocol.

The code for the NodeMCU is in this GitHub repository: https://github.com/Fronsto/Smart_Cycle.

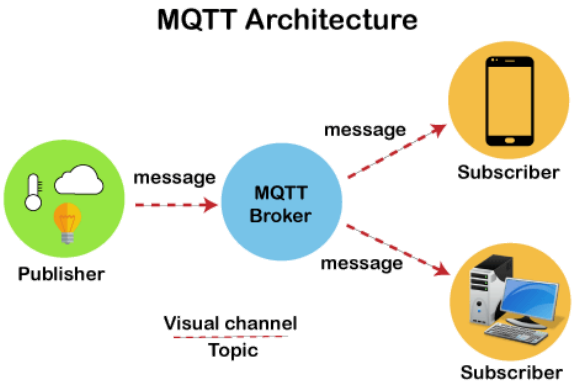
A small explanation of the files in the repository:

- **Cycle.ino:** The main driver code that calls all the helper functions and runs the actual setup and loop functions. It also contains the MQTT commands to send data to the appropriate channels.
- **GPS_Module.ino:** The code to setup the GPS sensor using SerialStream and then parse the data coming from that stream to get the latitude and longitude.
- **MPU_Module.ino:** The code to setup the MPU6050 sensor, and then set the configuration values. It also has the checkMPUSettings function that runs a suite of tests on the MPU and prints the values of various parameters obtained from the Accelerometer and Gyroscope.
- **MQTT.ino:** The code to connect the NodeMCU to an MQTT Broker, and make it subscribe to the required channels to receive data. It also contains the MQTTcallback function that is fired whenever the MQTT client receives a message on one of its subscribed channels. The reconnect function tries to re-connect to the broker automatically incase there is a connection drop.
- **light_sensor.ino:** The code is to setup the Grove Digital Light Sensor using the TSL2561 library. The runLDR() function reads the Ambient Light value in lux and if it falls below a certain threshold (here, 500 lux) then it turns on the LED connected to the headlight of the cycle. When the ambient light is above that threshold, it turns that LED back off.
- **ESP8266_MPU6050-master.zip:** This library is used to interface the MPU6050 sensor with the IDE.
- **TinyGPSPlus-1.0.2.zip:** This library is used to interface the GPU sensor and interpret the GPS values.

MQTT Protocol

MQTT stands for **Message Queueing Telemetry Transport,** and is a machine-to-machine IoT connectivity protocol. It is extremely lightweight and is a publish-subscribe messaging transport protocol. This protocol is useful for the connection with the remote location where the bandwidth is a premium. These characteristics make it useful in various situations, including constant environment such as for communication machine to machine and internet of things contexts. It is a publish and subscribe system where we can publish and

receive the messages as a client. It makes it easy for communication between multiple devices. It is a simple messaging protocol designed for the constrained devices and with low bandwidth, so it's a perfect solution for the internet of things applications.



We have used the following channels to publish and subscribe using either our app or the NodeMCU:-

1. **Movement:** To send alert if there was any movement detected by the accelerometer on the NodeMCU to the application on the phone
2. **Lock:** To send the lock or unlock command from the phone to the NodeMCU
3. **Longitude:** To send the longitude obtained from the GPS to the phone app
4. **Latitude:** To send the latitude obtained from the GPS to the phone app
5. **Lux:** To send the ambient light value obtained from the LDR sensor to the mobile app for data collection purpose
6. **Alive:** An occasional signal sent by the NodeMCU to indicate that it is connected to the phone. Incase this signal stops coming after a certain threshold, then an alert is sent to the user saying that the cycle is disconnected.

Flutter Application

Along with the circuit, we also developed a companion app that the owner of the cycle can use to lock/unlock, using Flutter to develop a cross-platform application that can be used on both Android and iOS. Flutter is an open source framework by Google for building beautiful, natively compiled, multi-platform applications from a single codebase. The code for the Flutter application can be found here: https://github.com/GunjanDhanuka/smart-cycle-app.

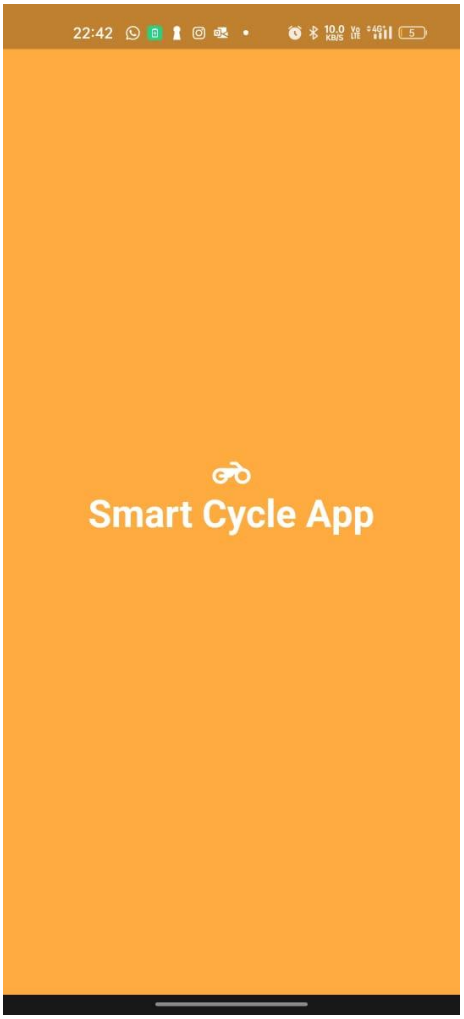
The key files in the above repository are:

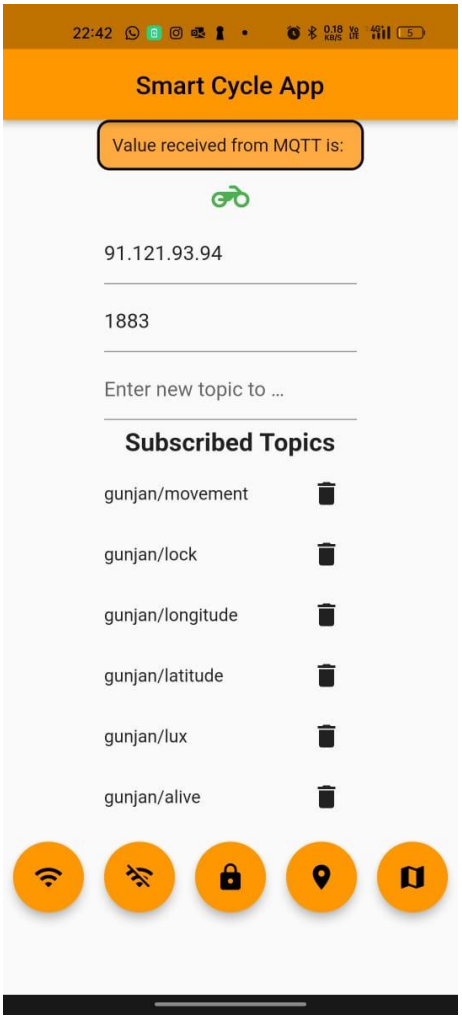
1. **main.dart:** The file from where the application is built and which contains the routing information for the various pages of the Flutter application.
2. **mqtt.dart:** The file contains the code for the MQTT client that connects to the broker and sends/receives messages.
3. **splash.dart:** The splash screen that loads when the app is launched.
4. lock.dart: The file which contains the logic and UI for the lock/unlock widget in the application.
5. **location.dart:** The file which handles displaying the map and the marker using the MapBox API and the coordinates obtained either from the phone or from the cycle.

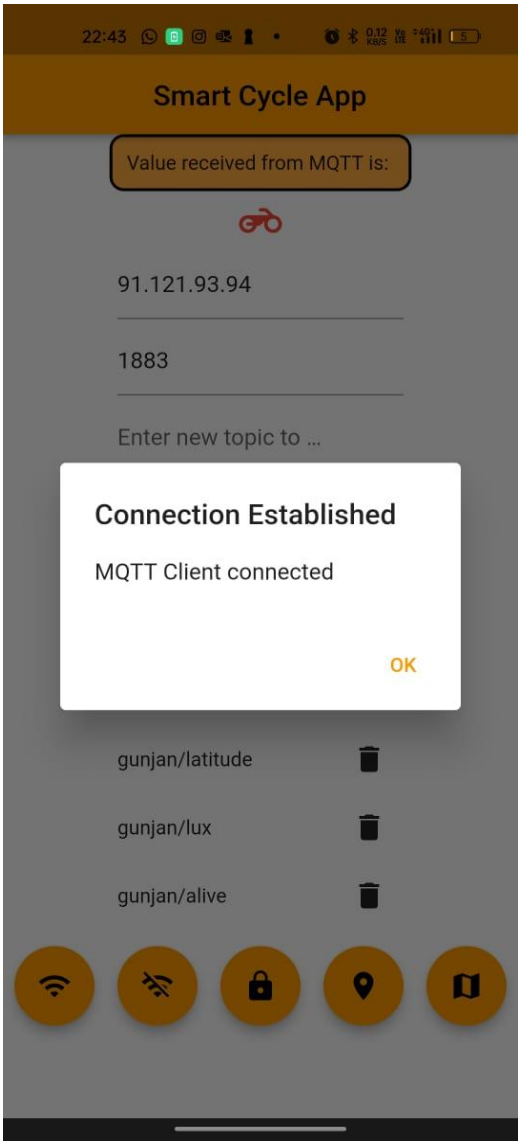
Features implemented in the app:

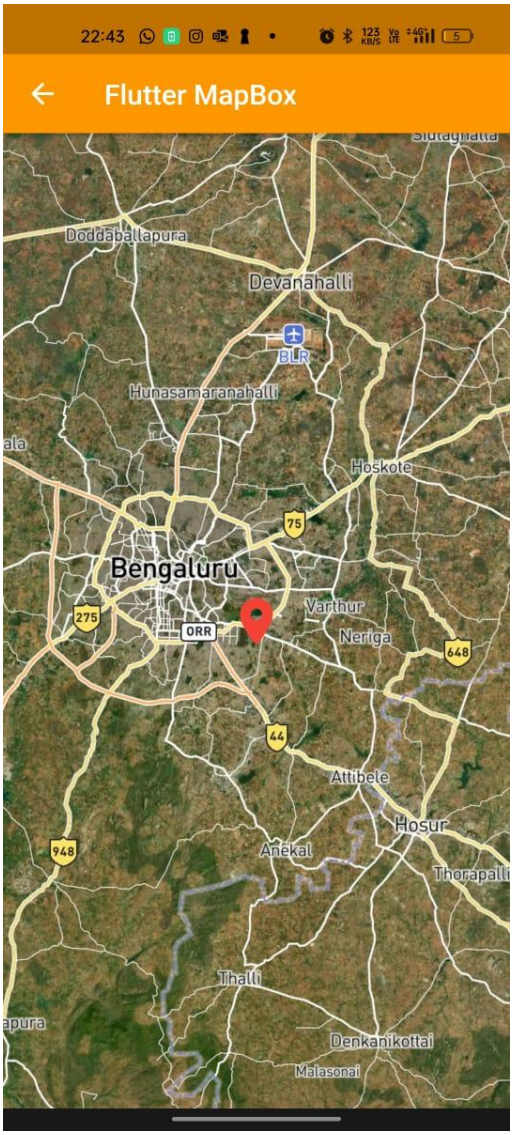
- Connect / disconnect to Cycle Alert.
- Alarm if the cycle moves when locked.
- Ability to tweak and connect to MQTT channels for further usage.
- Show the GPS data of the cycle when the alarm is buzzed.
- Allow to lock or unlock the cycle remotely.

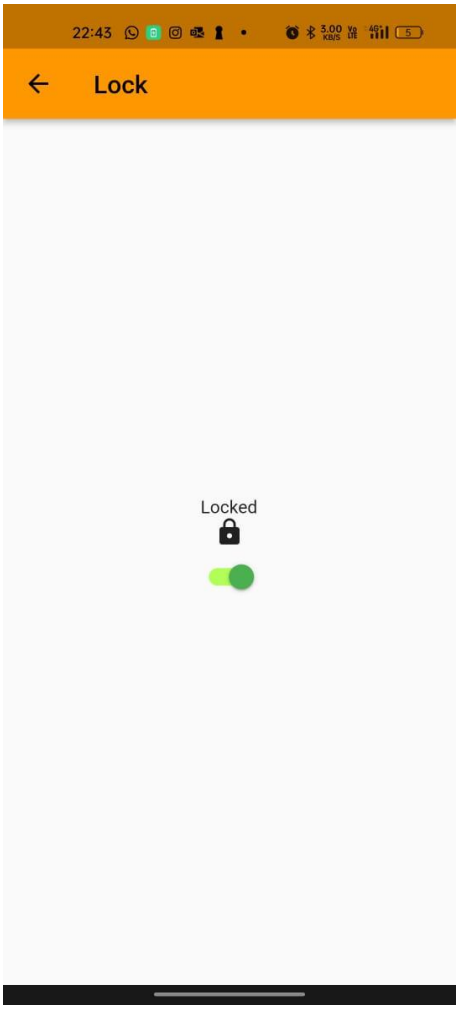
Screenshots of the Flutter Application






22:43
Smart Cycle App
Value received from MQTT is:
91.121.93.94
1883
Enter new topic to ...
Connection Established
MQTT Client connected
OK
gunjan/latitude
gunjan/lux
gunjan/alive


22:43
Flutter MapBox
Doddaballapura
Devanahalli
BLR
Hunasamaranahalli
Hoskote
75
Bengaluru
275
ORR
Varthur
Nerigа
648
44
Attibele
Hosur
948
Anekal
Thorapalli
Thalli
Denkanikottai
Malasonai



Challenges and limitations

1. **Problems with the GPS Sensor:-**
    - The MPU6050 Sensor, The Grove light sensor and the gps neo 6m sensor, all use I2C communication through SCL and SDA pins of the NodeMCU. Various I2C devices can be connected to the same SCL and SDA pins and they can be differentiated on the basis of a unique device address that each device has. When we connected the light and MPU6050 sensors to the D1 and D2 pins(default pins for I2C communication in ESP8266) the sensors perform well, but when we connect the GPS sensor to the same pins the NodeMCU only recognises the GPS sensor and prints garbage value on the serial monitor.
    - We also tried creating 2 sets of SCL and SDA pins using 2 separate sets of pins but that also didn't work because we were still getting garbage values on the serial monitor and weren't able to get meaningful data from the sensors.

- We ruled out the possibility of an overloading of the NodeMCU leading to reduced power since NodeMCU can support upto 128 devices so 3 sensors should be working fine. We were unable to resolve the issue in the end.

2. **Problems with mounting and dismounting of the system**
    - Creating a packing for the system was a major challenge for us.
    - Taping and untaping the system to the cycle again and again was a cumbersome process.
    - The light sensor which is exposed to the environment is also vulnerable to wear and tear.

3. **Problems with multi-threading.**
    - NodeMCU does not support multithreading so we had to run the code in a serial order only.
    - If there is a delay due to one part of the circuit all other parts of the code stop working.
    - For eg if the wifi connection breaks or a sensor gets disconnected due to a loose connection, The system will get stuck at that part.

Future Scope

1) More research could be done on the problem with the GPS sensor and other methods and different libraries could be tried in the future with more knowledge of I2C connection as well as of the working of the neo 6m gps module.
2) Creating a better holding mechanism for the system which is not only more secure but is also more convenient for mounting and dismounting. A glass case could be made to protec the light sensor from the environment.
3) Features like turn by turn navigation can be implemented using the MQTT protocol to get directions from the navigation system of a phone.
4) Battery and charging system can be improved, currently we are using a power bank for powering the system.. A new battery pack can be created for the system with in-place charging.
5) We can create a similar system with a microprocessor like raspberry Pi which supports multithreading so that different parts of the code can be run separately.