Garbage-Bin Tracker

An Application which tracks the capacity of garbage bins using sensors

Group 6 :
Fredrik Hammar
Pedro Diniz
Gee Kiyanage Chamila Dilhani

Department of Computer
and Systems Sciences

Degree project 04 HE credits
Computer and Systems Sciences
Degree project at the bachelor/master level
Autumn/Spring term 2021
Supervisor: Mahbub Alam

Table of Contents

List of Figures

List of Abbreviations

IoT : Internet of Things
SSH - Secure Shell
MQTT - Message Queuing Telemetry Transport

1. Abstract

This project report is for the IOT (Internet of Things) project of smart waste management system called “Garbage-Bin Tracker”, an application which tracks the capacity of garbage bins using sensors. It was developed using proximity sensors and Raspberry Pi and mobile application was developed using Python and android studio. While using MQTT internet protocol to transfer data. The users are allowed to use the mobile application and track the garbage level at any time, and the user will get the notification when the maximum level of the garbage bin is reached. The aim of this project is to manage resources in an effective way and to have a better waste management system.

This report will include the project architecture, description of source code and the application output with the evaluation and the recommendations for the future improvements.

Key words : Garbage-Bin Tracker , IOT, smart waste management

2. Introduction

IoT (Internet of Things), refers to the network of connected physical devices/ objects that facilitate communication or exchange of data between each of them without having human intervention. These systems may be embedded with multiple physical things such as sensors, software and hardware gear, etc. The use of IoT is rapidly increasing around the world. The number of IoT connected devices is projected to increase to 43 billion by 2023, which means threefold increase while comparing with 2018.

This project “Garbage-bin tracker” is an innovative project which will help to maintain cities clean.

2.1 Background

The concept of smart garbage has been part of the trend too. It has become more and more prevalent. Smart garbage aims to solve the issues that regular waste management systems face nowadays. We use sensors to measure the capacity and current fullness level of the containers, monitoring it 24 hours a day, and delivering the information about it to users through a mobile application.

In the regular waste management system, the waste is collected by trucks which follow a predefined route. However, it does not take into consideration the capacity of the garbage bins, whether they are full or not. Because of that, the trucks end up collecting half-full or even empty garbage bins, which is a waste of resources, such as time, fuel consumption, use of roads, financial resources, among others. It also affects the environment in a negative way, due to all the waste and pollution caused by trucks.

These problems can be solved by using a smart garbage system. We envision our solution being used by private individuals and companies at first. This could be due to security issues, since sensors could be more susceptible to be stolen in public places.

3. System requirement

Hardware Requirements:

- Ultrasonic distance sensor - HC-SR04
- Breadboard and connection wires
- Raspberry Pi
- Neo 6M GPS Module

Software Requirements :

- MQTT broker
- Python for scripting
- Android studio for UI (Mobile app)

3.1 Hardware Description

Ultrasonic distance sensor - HC-SR04

The Ultrasonic distance sensor will be used to measure the distance between the top of the lid to the top of the garbage level.

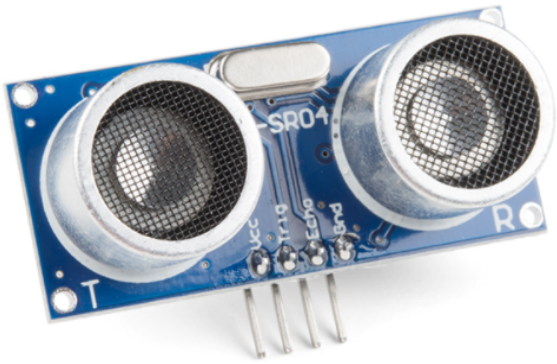


Figure 3.2.0 : Ultrasonic distance sensor - HC-SR04

Features and Characteristics

- HC-SR04 is used to determine the distance to an object. It provide excellent non-contact range detection and high accuracy while giving stable reading and it's easy to use
- It will offer 2cm to 400cm range. There's no any operational issues, but sometimes it is difficult to detect soft materials (like clothes)
- Working Voltage DC +5V and 15mA current
- The measuring angle will be 30 (degrees) and accuracy can reach to 3mm

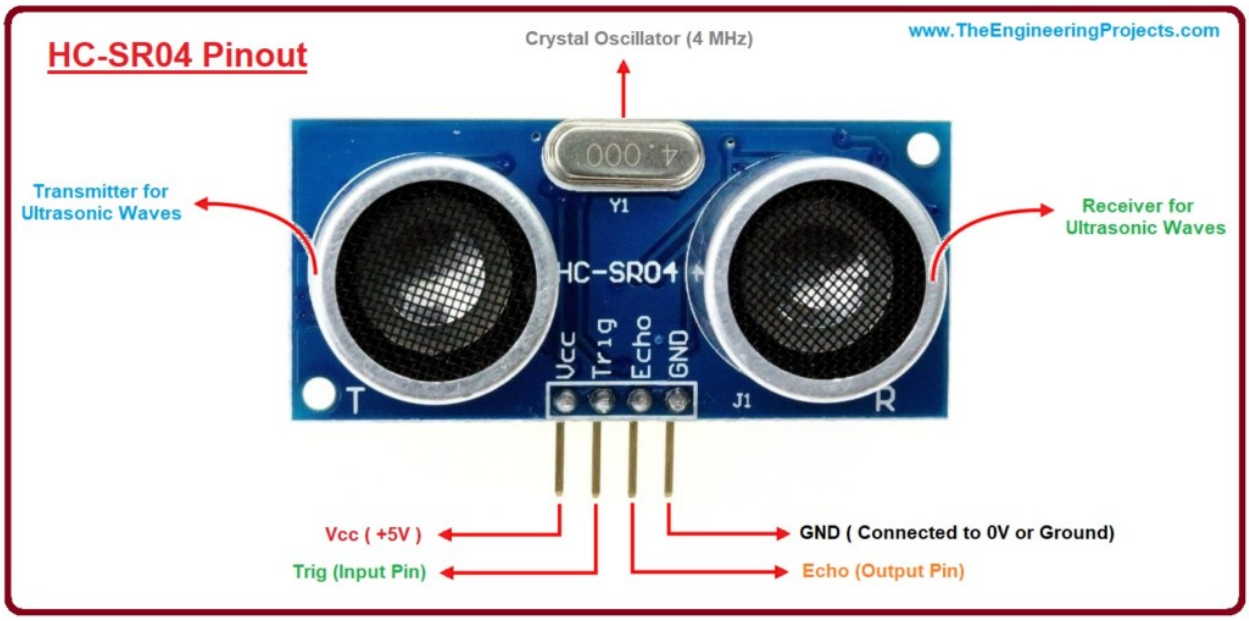


Figure 3.2.1 : HC-SR Pinout

Neo 6M GPS Module

This GPS module will receive the signals from GPS satellite and GPS track the satellite positions (upto 12 positions) in order to calculate the required positions. This GPS module responds to 16-bit Modbus register commands. The accuracy will depend on the numbers of the satellite tracked the positions.

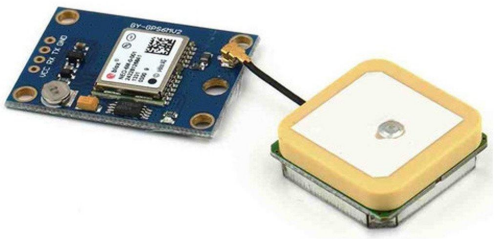


Figure 3.2.2 : Neo 6M GPS Module

Raspberry Pi

We used Raspberry Pi - model 4 B for this project, provided by this course.

The Raspberry Pi is a credit-card-sized single-board computer developed in the United

Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and developing countries. (Livingstone and Adams)

A user can interact with the Raspberry Pi either by connecting peripheral devices such as a mouse, keyboard, and a screen to the board or by connecting to the Raspberry Pi through SSH (Secure shell). (“Raspberry Pi Documentation - Using Linux”).

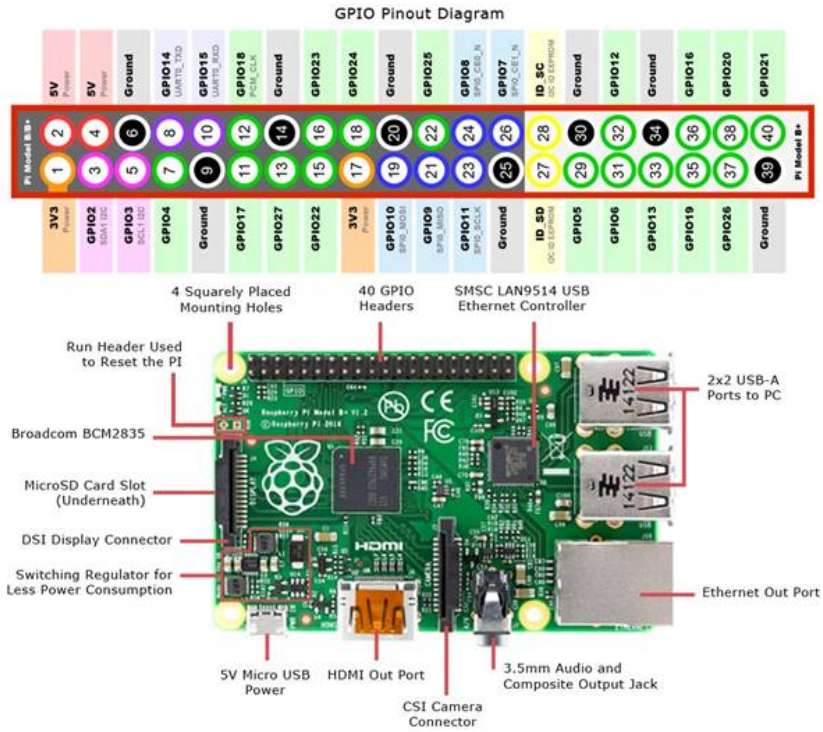


Figure 3.2.3 : GPIO Pinout diagram

3.2 Software Description

MQTT protocol

MQTT is a publish-subscriber based messaging protocol and its asynchronous message delivery protocol and it is lightweight and supports implementation to a wide range of devices. It works based on the publish/subscribe model.

Mosquitto is an open source message broker that implements the MQTT protocol. The MQTT protocol provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for Internet of Things messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers. (“An open source MQTT broker”)

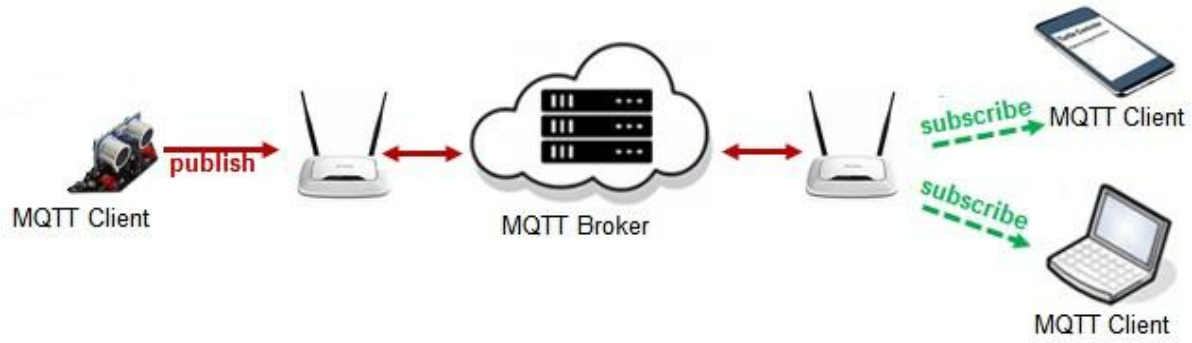


Figure 3.2.4 : MQTT Publish and subscriber

The MQTT internet protocol is used to transfer data from the sensors to the phone application, with the Raspberry Pi being the broker using Mosquitto. Raspberry Pi acts as a middleware, MQTT creates the publish / subscribe pattern, where the sensors are publishing data and the phone application is subscribing to it.

4 . Methodology

The first step after deciding to work on our IoT application - Garbage Tracker was to create a timeline for our project and estimate the time schedules for constructing each part of the Garbage Tracker - starting from the proximity sensor, going through communication with the Raspberry PI, and the Android Application itself with a end user UI. This was the base for the project.

We brainstormed new ideas as well that could further enhance our IoT project. Such ideas were to:

1. Implement a tracking system, through the use of a GPS, together with each sensor and device. The end user would be able to track the location of the trash bin in a map on the phone;
2. Create an “add” feature for the end user, allowing the possibility for him/her to track more than one trash bin at a time.
3. Display information about the history of garbage pick up dates to the end user.

Due to time constraints for the project, we knew there might not be time to implement these ideas, so as the project moved forward we would adjust the timeline, so that if we realized we were ahead of schedule, then we would implement these bonus features.

This decision made sense at the time, because we could not estimate how long it would take to add these features, since we didn’t have significant experience with the technologies involved, such as IoT applications, Python, GPS modules, the sensors, the MQTT protocol, etc.

Firstly, we decided to test the proximity sensor from the Lab, VCNL4010 and whether it could be suitable for the Garbage-Bin Tracker. For that, we made use of the knowledge and the same set up from the Laboratory lessons 1, 2 and 3 from this current course. We realized it would not work, due to the limitation of the range the proximity sensor could measure.

Therefore, we replaced it with the Ultrasonic distance sensor - HC-SR04. By reading online manuals and the datasheet, we understood about the installation of it. We used the breadboard, PuTTY as our interface to interact with the Raspberry Pi using SSH, and set up a script, similar to the one from labs, with MQTT script to publish the values.

Parallel to that, we worked on what the Android App would look like, so we started prototyping the AndroidApp UI, by creating a virtual device. We used the base knowledge from labs and online guides for that.

When we then proceeded to create the logic of the Android Application and connect it to the UI. We used a few simulation values for that and tested it.

Finally, after realizing everything was going ahead from what was expected in the schedule from the timeplan, we decided to implement the bonus features. We made use of different sources of material, ranging from documentation and datasheets for the GPS moduleNeo 6M, to Laboratory, course and online

material for the MQTT based script, as well as online courses for the Android application virtual device UI.

5. Design and Development

This is the section where we explain in greater detail the design of the Garbage-Bin Tracker.. We have described the hardware and software used in previous sections, and now we see how they are assembled and interact with each other, forming the entirety of the IoT application.

5.1 Application Design

5.1.1 The Broker and Subscribing / Publishing data from the sensors

We created a python script inside the Raspberry Pi. It contains the set up for the Raspberry Pi to act as a publisher, sending the values coming from the sensors. This implementation is fully based on the MQTT protocol. The most important aspects to highlight are:

1. The broker - “test.mosquitto.org”. We have a connection from the client to the broker, which publishes the messages to the subscribers from the topic.
2. The topic - All subscribers from it receive contantant data updates from the ultrasonic distance sensor and the GPS sensor.
3. The functions for each sensor - Which return the data from those sensors. When publishing to subscribers, those functions are called and the values are combined and sent in a string.

The Python script then is set up to continuously publish values coming from the sensors.

5.1.2 The Ultrasonic Distance Sensor

Firstly, we set up the ultrasonic distance sensor on the breadboard and created a Python script to get the distance by the sensor. (“Raspberry Pi Distance Sensor using the HC-SR04”)
The value was retrieved from the sensor by using the following piece of code:

```
try:
      GPIO.setmode(GPIO.BOARD)

      PIN_TRIGGER = 7
      PIN_ECHO = 11

      GPIO.setup(PIN_TRIGGER, GPIO.OUT)
      GPIO.setup(PIN_ECHO, GPIO.IN)

      GPIO.output(PIN_TRIGGER, GPIO.LOW)

      print "Waiting for sensor to settle"

      time.sleep(2)

      print "Calculating distance"

      GPIO.output(PIN_TRIGGER, GPIO.HIGH)

      time.sleep(0.00001)

      GPIO.output(PIN_TRIGGER, GPIO.LOW)

      while GPIO.input(PIN_ECHO)==0:
            pulse_start_time = time.time()
      while GPIO.input(PIN_ECHO)==1:
            pulse_end_time = time.time()

      pulse_duration = pulse_end_time - pulse_start_time
      distance = round(pulse_duration * 17150, 2)
      print "Distance:",distance,"cm"
```

Figure 5.2.1.0 Code to retrieve the sensor values

We then proceed to publish the distance through the use MQTT protocol to the broker. Lastly, our Android application received the message containing the distance from the broker, after subscribing.

We proceeded to test this while filming it. We used the prototype of the Android UI we had at the time to display the values coming from the sensor. Moreover, we created the logic to express that distance in centimeters coming from the sensor to express the capacity of a garbage bin, in percentage. The lower the distance value from the sensor the higher is the percentage value seen by the user on the screen from the Android virtual device, which represents the “capacity” attribute of each Garbage-Bin Tracker.

The figures below demonstrate the actual distance received from the sensor and the capacity of the trash bin, as it gets full.

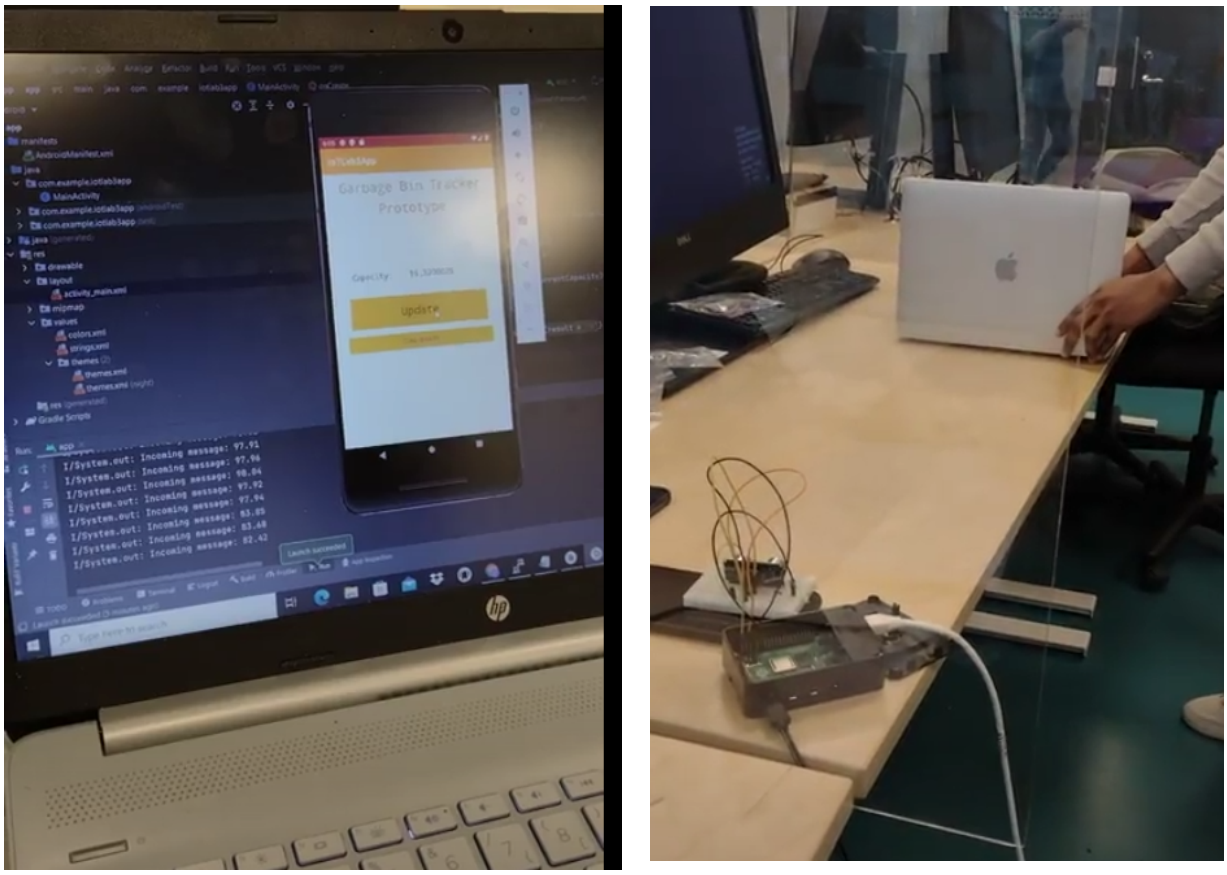


Figure 5.2.1.1 Actual distance received from the sensor and the capacity of the trash bin

After testing the sensor and being able to configure it and then receive the values, we decided to switch to simulation values for later stages. We assumed this was the most logical decision due to time constraints and practicality, as well as to allow a more efficient development of the application on Android Studio.

5.1.3 The GPS Sensor

We installed the Neo 6M GPS Module following an online guide. (Das) After the installation, we set up a Python script to get the value from the sensor and publish it. However, we realized that, especially initially, the GPS sensor does not always pick up signals from the satellites. The sensor tends to function whenever placed outside under a clear sky, which are not conditions we are fortunate enough to have very often.

The sensor eventually was able to pick up signals and we decided to test it by checking the values. The sensor sends data regarding the latitude and longitude.

From that moment onwards we decided to simply simulate those two values, for the same reasons we simulated the values from the ultrasonic sensor.

On the Android Studio side, we created the setup to use the latitude and longitude values from the GPS sensor. When the user clicks on the “location” icon from one of the Garbage-Bin Trackers, a map is opened using those latitude and longitude values. We use the Maps SDK for Android, which is based on the famous Google Maps, and followed a guide for this set up. (“Maps SDK for Android Quickstart”)

The Google Maps implementation is handled in the MapsActivity.java file. The figure below contains a snippet of it:

```
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;
import com.example.GarbageBinTrack.databinding.ActivityMapsBinding;

public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

    private GoogleMap mMap;
    private ActivityMapsBinding binding;

    private float latitude;
    private float longitude;
    private String name;
    MyApplication myApplication = (MyApplication) this.getApplication();


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        binding = ActivityMapsBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        Intent intent = getIntent();
        GarbageBin targetedBin = myApplication.getGarbageList().get((Integer) intent.getExtras().get("id") - 1);

        latitude = targetedBin.getLatitude();
        longitude = targetedBin.getLongitude();
        name = targetedBin.getName();

        // Obtain the SupportMapFragment and get notified when the map is ready to be used.
        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }

    /**
     * Manipulates the map once available.
```

Figure 5.2.1.2 Snippet for google map implementation

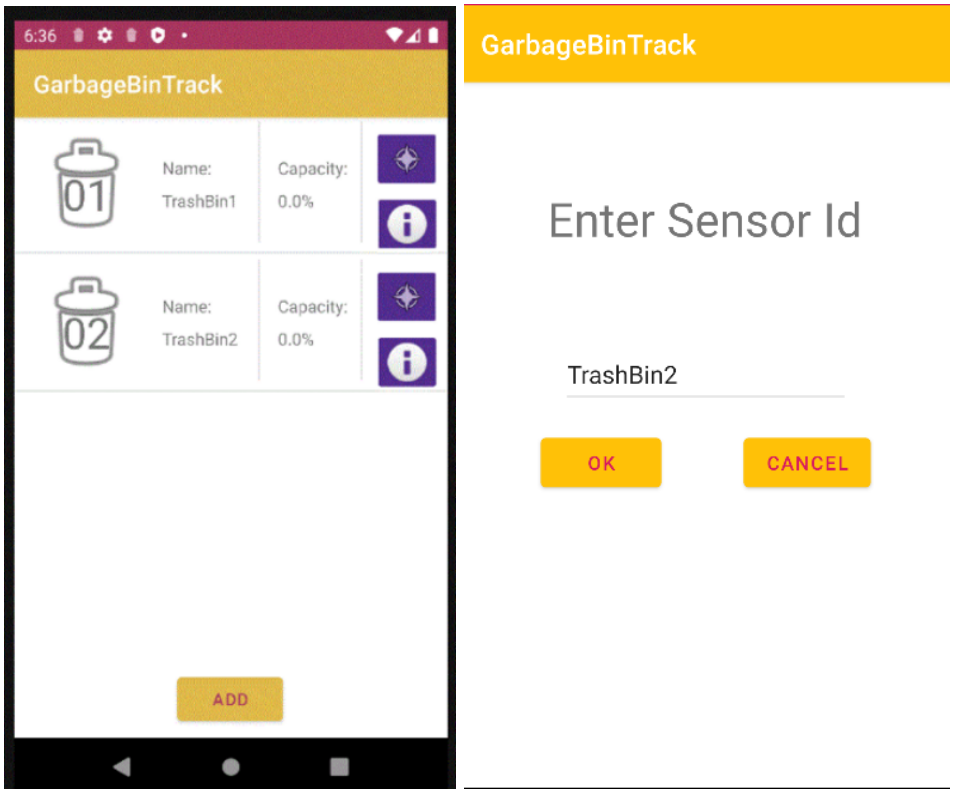
5.2 User Interface

The App is set up so that a user can add Garbage Bins to a recycler view by writing down the topic name they want to subscribe to (Topic is referred to as sensor Id in the app). The recycler view displays the Garbage bins name (which is defaulted as the topic for the bin) and the capacity of the Garbage bin which is continually updated based on the messages we get from the subscribed to topic.

The entry screen can be seen below, on figure. The user from the App can immediately see:

1. Each of the trash bins listed on the Garbage-Bin Tracker App;
2. The capacity of each trash bin, represented in percentage. At 100%, the trash bin is full;
3. The “location button”, to the top right, which displays the location of the trash bin on a map;
4. The “information button”, just below the “location button”. It provides the user with a short history about trash collection.

The entry screen also has an add button at the bottom of it which takes you to a screen that is used to add Garbage bins to the recycler view (See figure next to main screen).

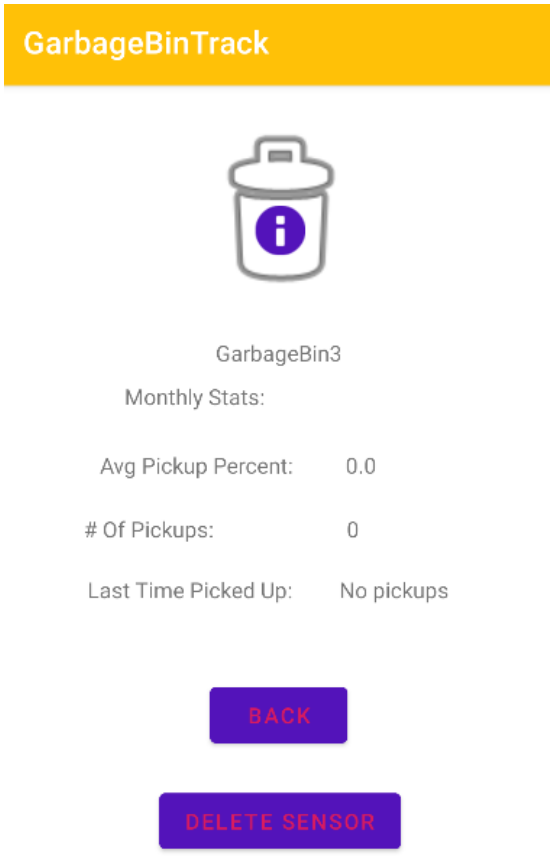


When the user clicks the “location button”, another screen is opened displaying a map and the location of the bin based on the Latitude/Longitude being sent to the subscribed topic the bin is referring to. This was done using the Google Maps API.

The information screen is found below, in figure. It displays some data that might be useful for the user of the application:

1. The average fullness of the trash bin, when trash is picked up, displayed in percentage;
2. The total number of trash pickups in the current month
3. The date when the last pickup occurred.

Here the user can also Delete the sensor from the app by clicking a button.



5.2.1 MQTT in Android studio

In Android Studio, we used the Eclipse Paho Android Service, which is an MQTT client library written in Java for developing applications on Android. (“Eclipse Paho Android Service”)

When the application is started, we set up the virtual device and connect to the Broker, which controls the flow of data from the sensors and publishes to the subscriber of a topic.

After connecting using the Mosquitto server URI, listening through port 1883, we can subscribe to a topic, as the figure below (Code located in MainActivity.java):

```
private void subscribe(String topicToSubscribe) {
    final String topic = topicToSubscribe;
    int qos = 1;
    try {
        IMqttToken subToken = client.subscribe(topic, qos);
        subToken.setActionCallback(new IMqttActionListener() {
            @Override
            public void onSuccess(IMqttToken asyncActionToken) {
                System.out.println("Subscription successful to topic: " + topic);
            }
            @Override
            public void onFailure(IMqttToken asyncActionToken,
                                  Throwable exception) {
                System.out.println("Failed to subscribe to topic: " + topic);
                // The subscription could not be performed, maybe the user was not
                // authorized to subscribe on the specified topic e.g. using wildcards
            }
        });
    } catch (MqttException e) {
        e.printStackTrace();
    }
}
```

The topic is named as in the Python file in the Raspberry Pi. This means that once subscribed to that topic, then the Application is ready and receiving the messages coming from the broker.
We then set a callback function which calls the “Subscribe” method.

In this function we pass the topic as a parameter. Once this is done. The application is subscribed and will receive the messages coming from any published data to that topic.

The messages are received to the client, within, in the “messageArrived” function (Code located in MainActivity.java), Here we look through a list of garbage bins which have been registered in the app. If the message arrived topic matches any of the topics of our Garbage bins registered in our recyclerview we update the Garbage bin based on the data received from the topic. The data that arrives is a string composed of the Ultrasonic sensor range and the longitude/latitude of our gps. The message is as follows:

XX-YY-ZZ (XX being Ultrasonic sensor range, YY Being Latitude, ZZ Being longitude). We split this string of different data into an array of these 3 values.

```
@Override
public void messageArrived(String topic, MqttMessage message) throws
        Exception {
    String newMessage = new String(message.getPayload());
    System.out.println("Incoming message: " + newMessage);
    String[] arrOfStr = newMessage.split( regex: "-",  limit: 3);
    for(GarbageBin gb : MyApplication.getGarbageList()) {
        if(gb.getTopic().equals(topic)) {

            //Callibrate garbage bin
            if(!gb.isCalibrated()){
                gb.setEmptyBinValue(parseFloat(arrOfStr[0]));
                gb.setCalibrated(true);
            }
```

We First check if the Garbage bin is calibrated (See figure above). A newly created garbage bin won't be calibrated by default. Then we take in the value we get from the ultrasonic sensor (arrOfStr[0]) and say that the garbage bin has been calibrated. We do this so we know where the bottom of the garbage bin is.

```
//Check if garbage got picked up
if(gb.getPercent() > 5 && parseFloat(arrOfStr[0]) > 50){
    gb.setPickupNotification(false);

    Date c = Calendar.getInstance().getTime();
    System.out.println("Current time => " + c);

    SimpleDateFormat df = new SimpleDateFormat( pattern: "dd-MMM-yyyy", Locale.getDefault());
    String formattedDate = df.format(c);

    //Check new month
    if(!gb.getLastPickupDate().equals(formattedDate)){
        gb.setAveragePickupProcentThisMonth(0f);
        gb.setTimePickedUpThisMonth(0);
    }

    gb.setTimePickedUpThisMonth(gb.getTimePickedUpThisMonth() + 1);
    gb.setLastPickupDate(formattedDate);
    gb.calculateAndSetAvgPercent( newPercent: (1 - (parseFloat(arrOfStr[0]) / gb.getEmptyBinValue())) * 100);


}
```

Then we check if the Garbage has been picked up by comparing the Garbage bin's current value with the value arrived (See figure above). If it did get picked up we register data that will later get displayed in the information screen.

```
//Send Notification when trash bin is more than 80% full
if(parseFloat(arrOfStr[0]) > 80 && !gb.isPickupNotification()){
    gb.setPickupNotification(true);


    String notMessage = gb.getName() + " is full and ready too be picked up";
    NotificationCompat.Builder builder = new NotificationCompat.Builder( context: MainActivity.this, channelId: "My Notification");
    builder.setContentTitle(gb.getName() + " is full");
    builder.setContentText(notMessage);
    builder.setSmallIcon(android.R.drawable.ic_menu_delete);
    builder.setAutoCancel(true);
    NotificationManagerCompat managerCompat=NotificationManagerCompat.from(MainActivity.this);
    managerCompat.notify( id: 1,builder.build());


}



//Set Garbage bin values
gb.setPercent( (1 - (parseFloat(arrOfStr[0]) / gb.getEmptyBinValue())) * 100);
gb.setLatitude(parseFloat(arrOfStr[1]));
gb.setLongitude(parseFloat(arrOfStr[2]));

myApplication.UpdateDatabaseItem(gb);
```

And lastly before we register the values we got from the topic to the garbage bin we check if the Garbage bin is above a certain capacity with the message arrived. If it is, we send a notification to the user that the Garbage bin is full and ready to be picked up (See figure above).

5.2.2 SQLite Database

All the Garbage bin data gets registered into a database locally on the phone using SQLite. This is all handled by the DataBaseHelper.java class. This makes it so the user won't have to continuously add the sensor every time they open the app. It also helps with saving the data that has accrued over the month on the information screen.

6. Evaluation

We then present the results when using the application and we make an evaluation in this section.

Overall, we have successfully completed the objectives from our project and, because we were ahead in the time schedule, were able to implement the bonus features. It should be noted, however, that this is not a finished product, due to us not actually having tried it inside trash bins, created a login system for the users, been concerned about equipment security or even tested it extensively.

Moreover, the evaluation is based on a simulation of the data coming from the sensors.

Finally, from our own observations, the ultrasonic proximity sensor appears to obtain values that are true only of obstacles facing it directly. When we tested objects to the sides, the values would not correspond to the real distance. This was a limitation from the sensor.

That said, we were still able to materialize our concept and make it functional, within the time constraint that we had.

1. We were able to obtain values from the ultrasonic and GPS sensors and convert them, creating a representation of the capacity of the trash bin that the user from the android application is able to understand. When the user opens the application, he or she will see on the screen the trash bin to the left, with a corresponding capacity on the right, displayed in percentage.

2. We implemented support for more than one sensor, so that now values from the sensors are published by the Raspberry Pi, go through the broker and then are subsequently handled by the android application. The user can see each of the trash bins one above the other, and their own corresponding information to the right.

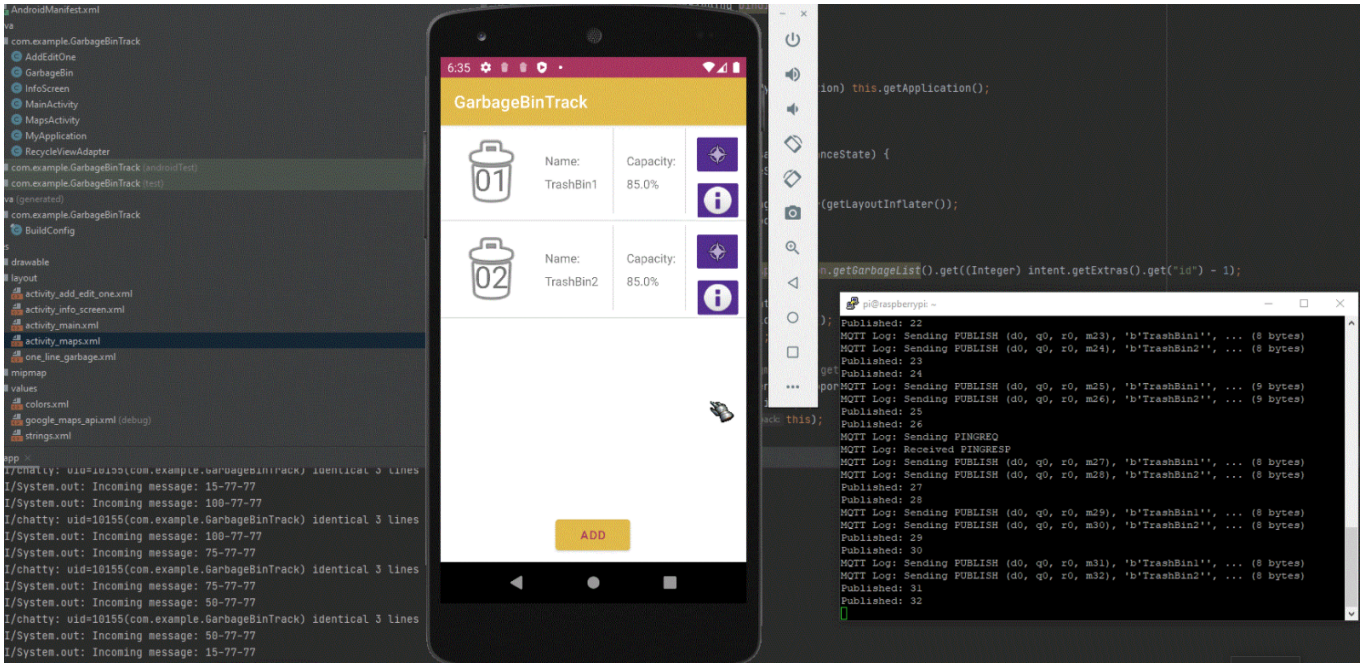
3. We have also been able to integrate a GPS system for the application, allowing the user to see the location of each trash bin on Google Maps. With a click on the “Location” icon, a map is displayed to the user as well as a location sign for the trash bin.

4. The user can check for information regarding the history of trash collection for each trash bin. By clicking on the “information” icon, the user can see the useful stats for that particular trash bin.

In the simulation, we created a few values that should represent the ones coming from the GPS sensor and ultrasonic sensor. They are published in a Python script within the Raspberry Pi, using the Mosquitto MQTT broker.

These values are received by the subscriber and converted into information that the user can understand. In other words, the values are converted into the capacity of the Trash bin (ultrasonic sensor distance value) and a Google Maps location (GPS sensor latitude and longitude).

The figure below illustrates our evaluation:



7. Discussion

The members from our group took interest in different topics within IoT, and creating an application that could be useful for a wide range of different users was our motivation and goal at the same time.

We realized that IoT is an ever growing area, with global spending on Enterprise ioT technologies being increased by 12.1% in 2020, to $128.9 billion. The growth trend continues through 2021, with an estimated 24% compared to 2020. (“Global IoT spending in 2021 to grow 24%, led by investments in IoT software”)

There are several IoT based products already in the market which are used and loved by many, such as smart home appliances, like refrigerators, televisions, security systems, computer related peripherals, such as printers, microphone and webcams, technology made to wear, like Fitbit, speaker devices such as Amazon Echo, among many others.

Our idea was to introduce something new to the market, which may be useful for private individuals, for example, living in a house or apartment complex. They would be able to know more about their own waste, and that could lead to them being more conscious about the impact they might be causing, and thus adopting more environmentally friendly behaviors. Moreover, it could be a time saver for them to know which days, time, and which garbage bin should they be throwing the trash. Even companies could benefit from this as well, especially if they produce a considerable amount of waste and must allocate resources for its management.

The concept can, however, be extended to public authorities who manage waste and the pickup. They would be able to know more about the habits of people in each area concerning amount of trash disposed as well as dates and times, and then adjust their trash pickup schedule accordingly, even in real-time. They could also keep track of measurements from the statistics provided by the Garbage-Bin Tracker.

IoT management systems are indispensable already for many companies for the automation and optimization of processes that they provide. At the same time, waste management is a serious global concern, and it is an issue that could be tackled with the help of IoT applications.

8. Future improvement

The vision for the future regarding this project is to have this working as an application completely ready for the end user. That could range from individuals, to companies or even public authorities.

For this to be possible however, some obstacles must be surpassed and some new features need to be implemented.

A few suggestions would be:

1. Implement an online SQL database which can be viewed from a website and synced between multiple different devices.

2. Currently we check if the user has picked up the garbage by comparing values gotten from the sensor and checking for if we suddenly go from a high value to 0-5% capacity. This method can be prone to error and we would have liked to pair it with another sensor that can for example check when the garbage bin is open for a more accurate result.

3. To implement a user authentication system. That would allow each user to know only about their own Garbage-Bin tracker devices.

4. The device would need adjustments in order to be more compact, fit in different trash bin shapes and models, while at the same time be protected and resistant enough to last.

5. Improve on device security, to protect against attempts to remove it out of the trash bins

6. It appears to us that one sensor might not be enough to cover the whole of the trash bin. Perhaps by implementing a solution with multiple sensors, the value could be more accurate.

7. Companies, private individuals and public authorities all have particular necessities and benefit differently from information provided by the Garbage-Bin Tracker. This means that there is room for improvements and adjustments within the “information” screen, where users are able to see statistics about the collection of each trash bin. A solution tailored for companies and public authorities would probably have to be more complete regarding these statistics, so different versions of the Garbage-Bin Tracker would exist.

Moreover, our project is not focused on the product's appearance or the price range. However, those are important factors to take into consideration when introducing a new product in the market. The design of the product itself, as a compact, safe but also visually appealing solution would be the next step, while also considering production and consumer costs.

9. References

- www.jython.ch. (n.d.). *mqtt*. [online] Available at: https://www.jython.ch/engl/index.php?inhalt_links=navigation.inc.php&inhalt_mitte=mbit/mqtt.inc.php [Accessed 2 Jan. 2022].
- Nevon Projects. (2016). *IOT Garbage Monitoring System Project*. [online] Available at: https://nevonprojects.com/iot-garbage-monitoring-system/ [Accessed 20 Dec. 2021].

- "Build a UI with Layout Editor." *Android Developers*, 25 August 2020, https://developer.android.com/studio/write/layout-editor. Accessed 3 January 2022.
- Das, Arijit. "Use Neo 6M GPS Module with Raspberry Pi and Python." *GitHub Pages*, 10 July 2019, https://sparklers-the-makers.github.io/blog/robotics/use-neo-6m-module-with-raspberry-pi/. Accessed 3 January 2022.
- "Eclipse Paho Android Service." *Eclipse Paho | The Eclipse Foundation*, https://www.eclipse.org/paho/index.php?page=clients/android/index.php. Accessed 4 January 2022.
- "Global IoT spending in 2021 to grow 24%, led by investments in IoT software." *IoT Analytics*, 16 June 2021, https://iot-analytics.com/2021-global-iot-spending-grow-24-percent/. Accessed 2 January 2022.
- Livingstone, Ian, and James Adams. "Raspberry Pi." *Wikipedia*, https://en.wikipedia.org/wiki/Raspberry_Pi. Accessed 3 January 2022.
- "Maps SDK for Android Quickstart." *Google Developers*, https://developers.google.com/maps/documentation/android-sdk/start. Accessed 4 January 2022.
- "An open source MQTT broker." *Eclipse Mosquitto*, https://mosquitto.org/. Accessed 3 January 2022.
- "Raspberry Pi Distance Sensor using the HC-SR04." *Pi My Life Up*, 22 March 2018, https://pimylifeup.com/raspberry-pi-distance-sensor/. Accessed 5 January 2022.
- "Raspberry Pi Documentation - Using Linux." *Raspberry Pi*, https://www.raspberrypi.com/documentation/computers/using_linux.html. Accessed 2 January 2022.