"LoRa Based Assert Tracking"
Team Project Report

Tallinn 2018

TABLE OF CONTENTS

1 PROJECT OVERVIEW

The following report belongs to a group project held in the course “Navigation and Positioning” during Spring Semester 2018 at Tallinn University of Technology. The project accounts for 20% of the final grade and was undertaken by three students of the class.

The main goal of the practical lab is to develop a working device that uses real-time GNSS data as an input and produces any kind of output on a basic actuator such as LED, buzzer, LCD etc. The characteristics and type of both the working device and the actuator could be chosen freely by the students. A list of available material was also provided, but additional devices could be used, too.

The chosen project should fulfill some requirements to be accepted. It has to be challenging, feasible, innovative, original, evaluable, observable and presentable to ensure high quality. The project took place from March 06 until May 15, i.e. 10 weeks.

2LEARNING GOALS

As this project was a group work, one of the learning goals was to enhance teamworking abilities of the students in order to develop the device. Furthermore, since the type of device and its implementation could be chosen freely by the students, the project was supporting creativity, but also forced the students to present a working device in the end.

In addition to that, the main learning goal was to get more acquainted with the receiving and extracting of GNSS data and to provide ways to display and/or process the information. Again, the complexity of the system was chosen by the students as long as the requirements presented in 1 were met.

To develop the project, the students had to familiarise themselves with the Arduino MEGA controllers and software to set up the technical componens for data processing. Depending on their individual topic, several other applications could be used during development. In this project, LoPy, The Things Network Cloud and MIT App Inventor were used to process data, upload and save it in a cloud and to develop a smartphone app over which the positioning data can be seen by the user.

3 THE PROJECT AND ITS COMPONENTS

In this project, we decided to implement a pet collar that enables tracking via GNSS. The module itself is still in its early design phase and cannot be integrated into a commercially available collar yet, but if additional work is done this may change.

The overall functionality of our system is shown in the following graph. Each component will be briefly explained in the subsequent paragraphs and some will be described in more detail afterwards.

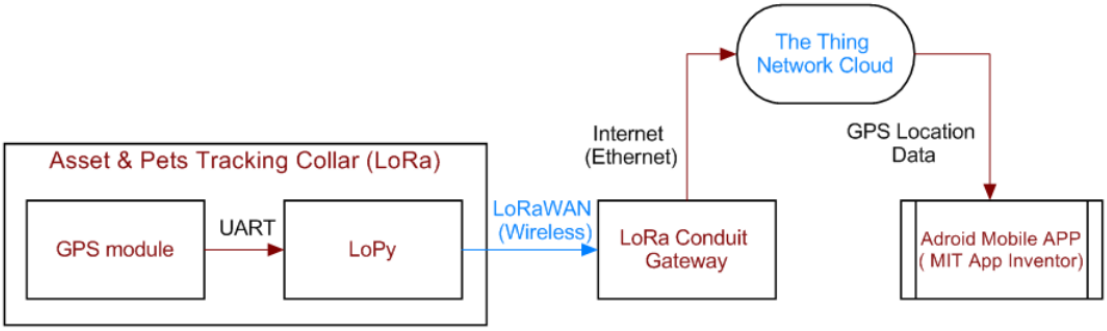

Figure 1: Project Overview

The collar itself consists of two parts: The GPS (GNSS) module and a LoPy microcontroller. The GPS module receives positioning data of the collar/pet and is connected over UART (Universal Asynchronous Receiver-Transmitter) to a programmable LoPy microcontroller. It is programmed via a laptop to send the data in an adequate and processible format to a LoRa Conduit Gateway over a wireless channel.

The Gateway processes the data and uploads the position of the collar into The Things Network Cloud. The user can then download an Android-based Mobile App on their smartphone (which was created using MIT App Inventor) to access the data and see the location of the collar. This requires Internet connection.

All in all, this is how our project works. To provide additional information on our work, we are now describing some key components more extensively.

3.1 RoyalTek REB-4216 GPS module

In this project, we are using the ITEAD RoyalTek REB-4216 GPS module to get the positioning data. It transmits the data over the L1 frequency band for GPS communication (1575.42MHz) and can be operated in either 3.3V or 5V mode. In our project, we used 3.3V mode.

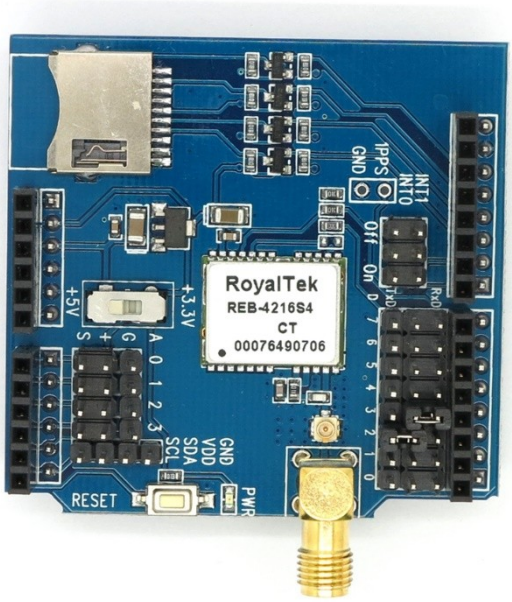

Figure 2: The ITEAD RoyalTek GPS module

3.2 LoPy microcontroller

We are using a Pycom LoPy controller to connect the GPS module and receive the positioning data. The LoPy is connected to an Expansion Board to provide additional features such as USB and SD slots for data saving. The pinout diagram for this board is shown below.

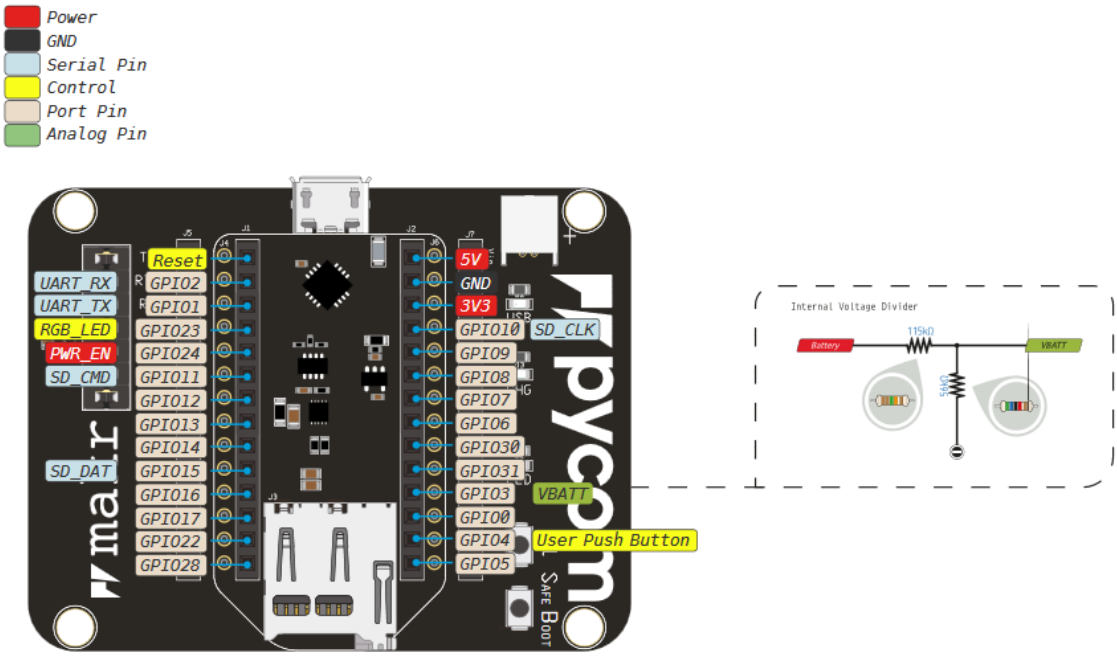

Figure 3: Pycom LoPy microcontroller

The LoPy module itself is connected with its pins on the middle of the board. The GPS module is linked to the LoPy using the following pins (this setup was recommended by the manufacturer):

LoPy	REB GPS Module
GPIO22/P11	Rx
GPIO21/P12 (GPIO28 on board)	Tx
3.3V	3.3V
GND	GND

In order to configure the LoPy, the MicroPython programming language is used. The code sets up the pin location and declaration, defines the structure of the positioning data, converts the received NMEA messages from the GPS module into position coordinates and handles unexpected exceptions such as no received data. The code can be examined in the appendix of this report. Note that for converting the NMEA messages, we used the already existing open-source micropyGPS sentence parser ().

Script on LoPy

```
from micropyGPS import MicropyGPS

from machine import UART

import time

from network import LoRa

import socket

import binascii

import struct


# Initialize GPS

#com = UART(1,pins=('P20','P21'),  baudrate=9600) #config.TX, config.RX

com = UART(1,pins=('P12','P11'), baudrate=9600)

my_gps = MicropyGPS()

data=[] # 1st 4 val is of latitude and last 4 packet is longitude

dev_addrstr = '26 01 1B 6C'

nwk_swkeystr = '9E 47 98 83 D1 D7 EE 7A 93 E2 F6 85 AD 8E 13 2F'

app_swkeystr = 'FE B8 6A 5E B8 8C 46 AD C4 04 E2 AC D0 08 5E C9'


def lora_init():

        global s

        # Initialize LoRa in LORAWAN mode.
```

```
    lora = LoRa(mode=LoRa.LORAWAN)

    # create an ABP authentication params
    dev_addr = struct.unpack(">l", binascii.unhexlify(dev_addrstr.replace(' ','')))[0]
    nwk_swkey = binascii.unhexlify(nwk_swkeystr.replace(' ',''))
    app_swkey = binascii.unhexlify(app_swkeystr.replace(' ',''))

    # join a network using ABP (Activation By Personalization)
    lora.join(activation=LoRa.ABP, auth=(dev_addr, nwk_swkey, app_swkey))

    # create a LoRa socket
    s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)

    # set the LoRaWAN data rate
    s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)
    print ("LORA Initialized")

def sendlorapack():
    global s
    if (s):
        s.setblocking(True)
        s.send(bytes(data))
        # make the socket non-blocking
        # (because if there's no data received it will block forever...)
        s.setblocking(False)
        rcv_data = s.recv(64)          # get any data received (if any...)
        if (rcv_data):
            print('Received lora packet:',rcv_data)
        return 0
    else:
        print ("Error: LORA Initialized")
        return -1

def clear_data():
    list_index = len(data)-1
```

```
            while (list_index >= 0 ):
                    data.pop(list_index)
                    list_index = len(data)-1
            #print ('Clear data : {}'.format(data))

def set_data(la1,la2,lo1,lo2):
    if(la1<0):
        la=int( (((-1)*la1)*1000000) + ((la2/60)*1000000) )
        data.insert( 0, (0xf0 | ((la & 0x0f000000)>>24)) )
    else:
        la=int( (la1*1000000) + ((la2/60)*1000000) )
        data.insert( 0, ((la & 0x0f000000)>>24) )
    data.insert( 1,((la & 0x00ff0000)>>16) )
    data.insert( 2,((la & 0x0000ff00)>>8) )
    data.insert( 3,(la & 0x000000ff) )

    if(lo1<0):
        lo=int( (((-1)*lo1)*1000000) + ((lo2/60)*1000000) )
        data.insert(4, (0xf0 | ((lo & 0x0f000000)>>24)) )
    else:
        lo=int( (lo1*1000000) + ((lo2/60)*1000000) )
        data.insert(4, ((lo & 0x0f000000)>>24) )
    data.insert( 5,((lo & 0x00ff0000)>>16) )
    data.insert( 6,((lo & 0x0000ff00)>>8) )
    data.insert( 7,(lo & 0x000000ff) )

def get_data():
    if com.any():
        print("Got data")
        my_sentence = com.readall()
        print(my_sentence)
        for x in my_sentence:
            my_gps.update(chr(x))
        print('Longitude', my_gps.longitude);
        print('Latitude', my_gps.latitude);
```

```
    if (my_gps.latitude[0] != 0 or my_gps.longitude[0] != 0):
                        set_data( my_gps.latitude[0] , my_gps.latitude[1], my_gps.longitude[0]
,my_gps.longitude[1])
    else:
                        print('Invalid Packet');
                        return -1
    return 0
  else:
    return -1


lora_init()
while 1:
        count = 0
        if( get_data() != -1):
                print('Data Pack : {}'.format(data));
                if( sendlorapack() != -1):
                        print('Packet Sent');
                else:
                        print('Failed to send packet');
                        lora_init()
        else:
                print("No data");


        clear_data()                                    #clearing buffer for next cycle.
        print("\n")
        time.sleep(20)
```

3.3 Multitech MultiConnect Conduit Gateway

The MultiConnect Conduit is a configurable, scalable cellular communications gateway for industrial IoT applications. The conduit allows users to plug in two MultiConnect mCard accessory cards supporting wired or wireless interfaces. There are two versions of the Conduit LoRa Accessory Kits, one for 868 MHz applications and one for 915 MHz applications, designed for operation in the European Union and North American market places, respectively. Each requires the MultiConnect Conduit (available separately) for internet connectivity.

Figure 4: Multitech MultiConnect Conduit Gateway

Multitech MultiConnect Conduit Gateway is configured in LoRa ABP (Authentic ion by personalization) mode for unacknowledged reception of LoRa packet from broadcasting devices. The gateway forwards these packets to TTN cloud.

3.4 The Things Network Cloud

The Thing Network cloud service is free of charge and is specially designed for LoRa. In order to use TTN cloud we registered gateway and application device as shown in Figure 5 and 6. The network and device key generated by registration is used at LoPy and Gateway for authentication.

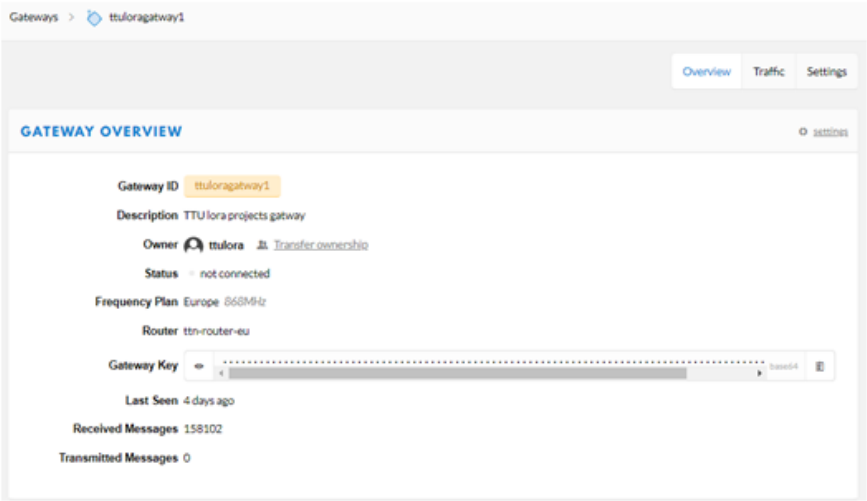

Figure 5: Gateway registration on TTN

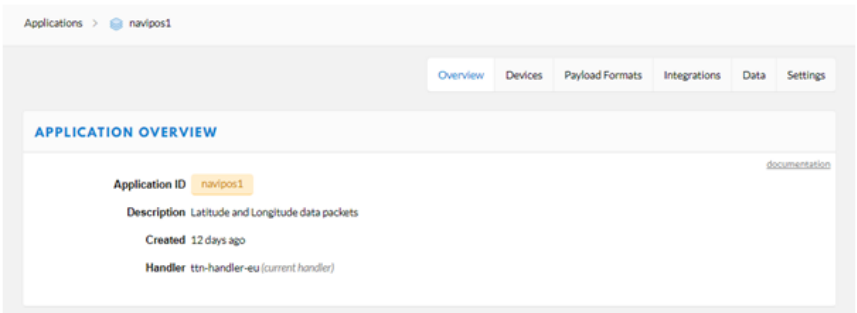

Figure 6: Application device registration on TTN

The data sent from LoPy is encoded to reduced packet size. In the cloud we have to decode the packet into an understandable format. Figure 7 shows the received packet and script shows the procedure to decode the information.

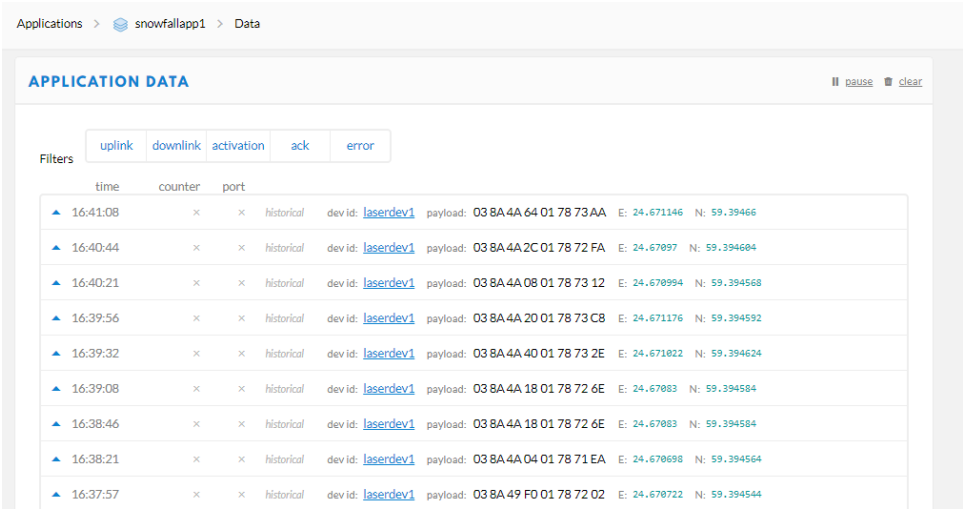

Figure 7: Received packet at TTN

Script on TTN for packet decoding:

```
function Decoder(bytes, port) {
  var la = ( ((bytes[0]&0x0F)<<24) | (bytes[1] << 16) | (bytes[2] << 8) | bytes[3])
  var lo = ( ((bytes[4]&0x0F)<<24) | (bytes[5] << 16) | (bytes[6] << 8) | bytes[7])

  if( bytes[0]&0xF0 )
    la = (-1)*la;
  if( bytes[4]&0xF0 )
    lo = (-1)*lo;

  return {
  N  : la / 1000000.0,
  E  : lo / 1000000.0
  };
}
```

3.5 The App

For the last part of the system, we have developed an Android application which makes a query to the cloud and gather the latitude and longitude of the device. These coordinates are updated in a map with a maker as shown in Figure 8. For more details, a description of the development of the application is given below.

3.5.1 Platform choice

To develop the application, we first analysed the different tools that are available on the market which are free of charge. After the analysis, we found out that MIT App Inventor was our best option due to the fact that any member of the group has sufficient experience to program in Android. MIT App Inventor is an online plaftform which only requires to install drivers on your PC and connect to phone via USB or Wi-Fi.

In addition to that, the platform is a drag and drop tool which does not need to be coded. It is more about knowing how to write the algorithm with logic. In Figure 8, a picture of the main windows of the app development is presented. In Figure 9, the drag and drop code option is shown.

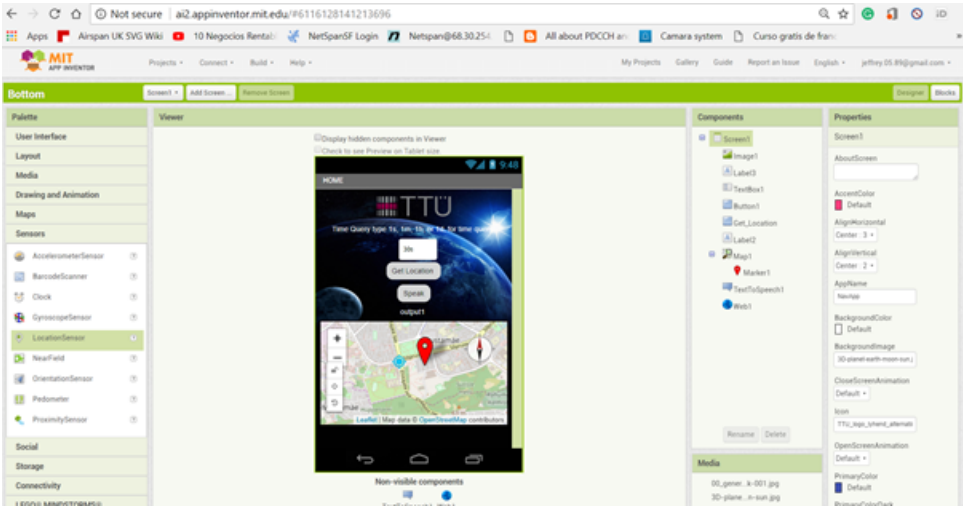

Figure 8: MIT App Inventor dashboard.

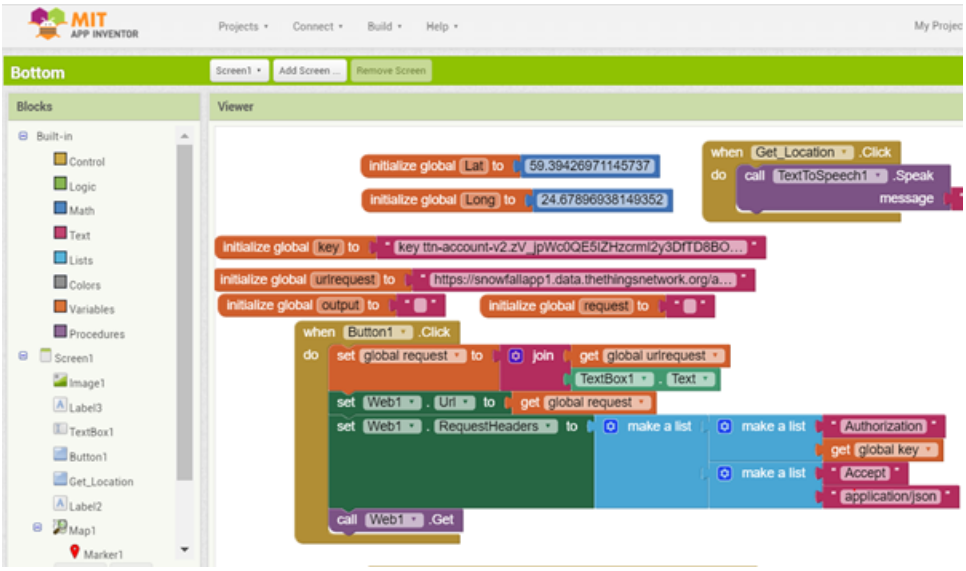

Figure 9: MIT App Inventor code window.

3.5.2 Learning of MIT App Inventor

After deciding to select MIT App Inventor as our development platform, we start playing with it and create some example to understand and get familiar with it. An interesting option we found was the text to speech.

3.5.3 Query to The Cloud

Continuing with the development of the app and after getting familiar with the platforms, we started trying out the connectivity to the cloud. We faced some problems regarding the HTTP request due to some harder formating that we could overcome. In Figure 10, the algorithm for the query is displayed. For instance, the global variable key is for the authentication to the cloud. The global variable urlrequest is the url where the data is stored.

As mentioned before, the more difficult issue, the best way to approach was to create a list with the two headers required such as Autorization and Accept as shown in Figure 10.

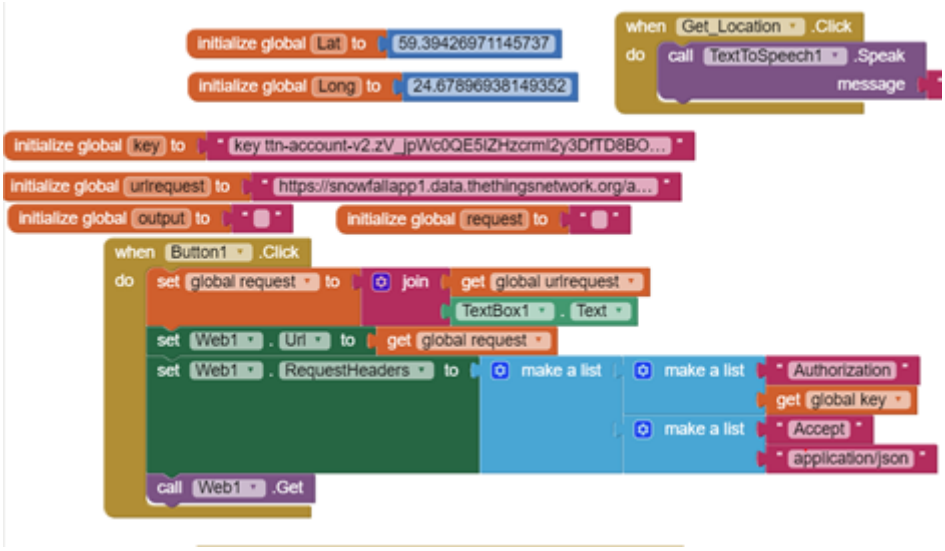

Figure 10: MIT App Inventor header query.

3.5.4 Interpretation of the query

After succesfully obtaining the data from the cloud, we need to interpret and extract the data that is needed to set the location on the application.

In Figure 11, it can be seen how the data is coming from the cloud. The data structure is devided by commas, it has quotes and square brakers. Thus, to extract the data we have a code which looks for the first position and the location of the second letter, then we give a length to extract the latitude and longitude. This can be seen in Figure 12.

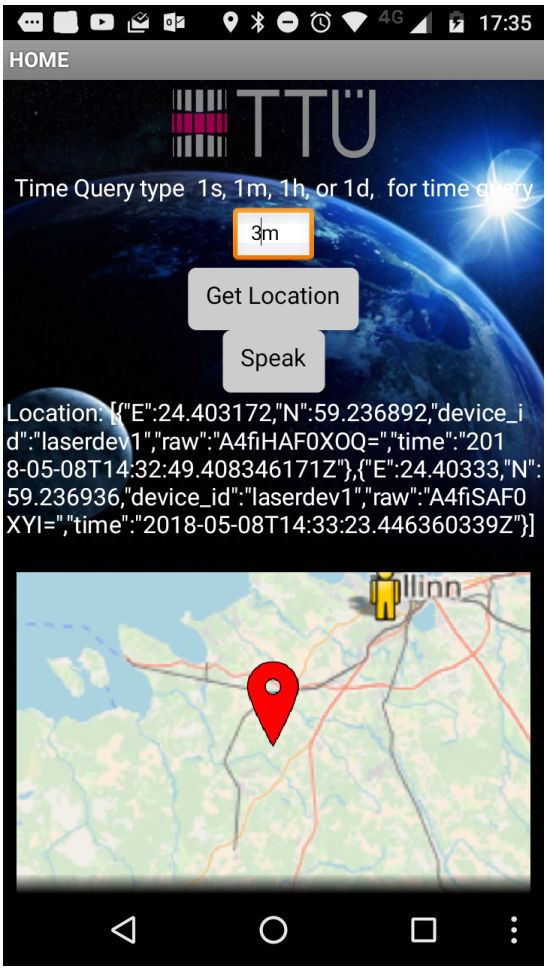

Figure 11: App Interface

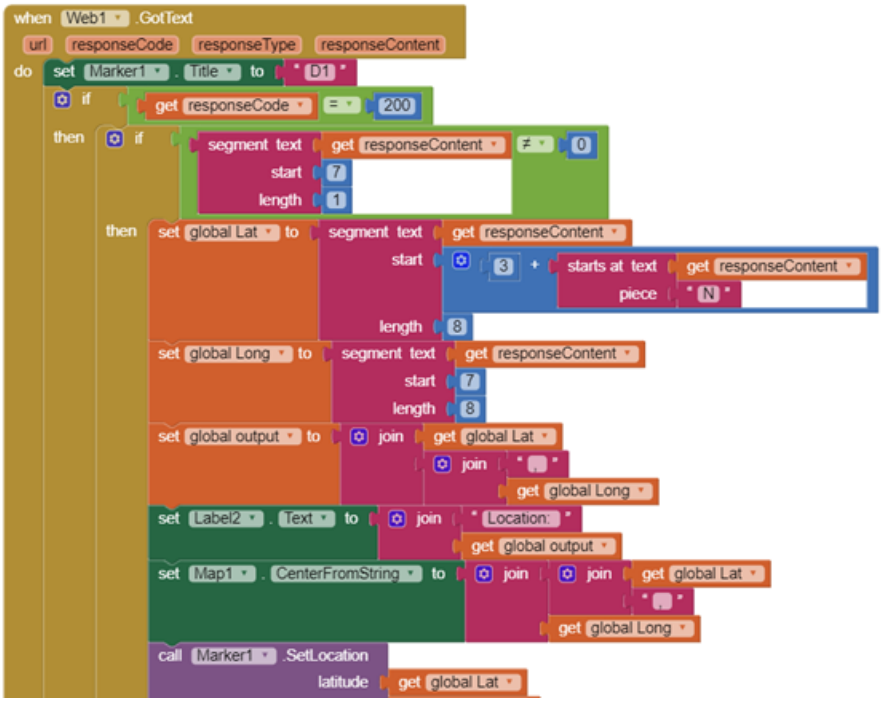

Figure 12: Data Structure from the cloud

This part of the algorithm could be improved for further development.

3.5.5 Visibility of device's location

As final stage, the position gathered from the device and cloud is located on the map as shown in Figure 13. For this prototype we are gathering the data by user request which means that the user is able to select from when the data is required, for example 1 second, 1, minute, 1 hour, or 1 day. Then the user needs to click or press the button to send the query and get the position of the device. The position is shown in text and on the map and the setup of the devices can be seen in Figure 14.

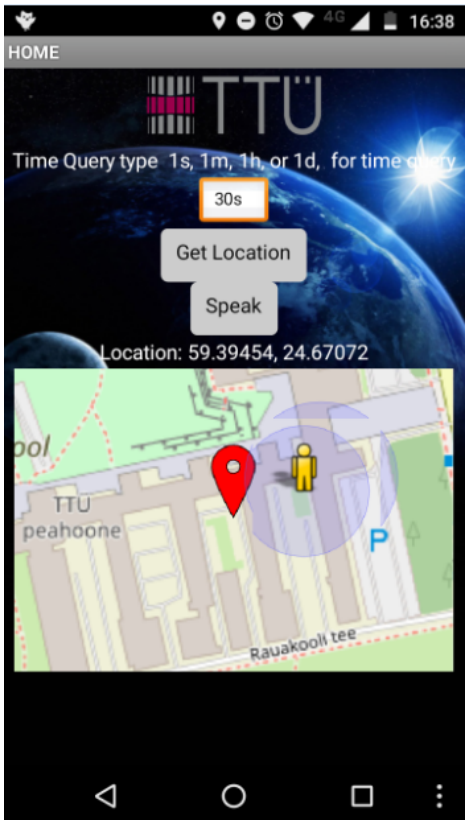

Figure 13: Location Display

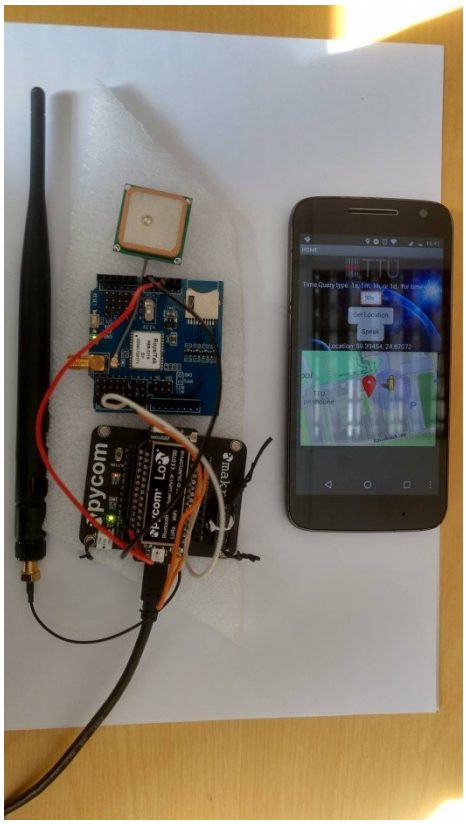

Figure 14: Setup

4 SELF-EVALUATION OF EACH TEAM MEMBER

4.1 Self-Evaluation by Harish

I architect the hardware and software modules of the project. I learned about MIT App Inventor platform and REB-4216 GPS module, which was very new to me. In past I had used LoPy and Conduit gatway, so I shared my knowledge with teammates.

Initially positioning value calculated using “RoyalTek REB-4216” sensor data was not accurate and we have deviation of around 500km. With some brainstorming with team we figure out that we missed to covert minutes parts of latitude and longitude to degree. After doing the necessary correction the deviation reduced to 20m, which is pretty accurate.

At the moment we were using our laptop to power the LoPy, but in order to make this project portable we can move to batteries. LoRa transmission consumes a significant power, so by increasing the interval between consecutive transmission and keeping the module in lower power mode we can make this project more power efficient. In future IMU sensor can also be added to this project for motion detection and can be used as a commercial assert tracking module.

4.2 Self-Evaluation by Jeffrey

I learned a little more about working in group, deviding the tasks depending of team expertice. I was able to understand the used of LoPy module connectivity to the GPS module. Also, I learned the use of the cloud for IoT devices. Apart from this, I enjoyed learning and developing the Android App using MIT App Inventor. It is a tool with a lot of potential for beginners and prototying making this easier and faster. The disavantage of this tool is the loayout but this is more about designing, thus, for my point of view it is an excellent tool for prototyping. Something to highlight is when we were debuging a error in the positioning data. We solved the issue in 15 minutes working together as an integration team, asking about the formatting packet and where is sent in the code. It was a great experience.

Overall, I was able to learn, contribute and transmit to the team what I knew in a proper way thus they could understand my ideas. I realized the importance of location and position of any particular thing. It is essential nowadays for tracking purpose and it is getting easier due to the IoT trends.

4.3 Self-Evaluation by Tobias

Before this project I have neither worked with LoPy, MicroPython or the MIT App Inventor. As a result, I have learned a lot using these tools on how to collect and process positioning data. Overall, I contributed to the project by testing the applications and collecting the data with the GPS module. In addition to that, I provided the layout and design for the report.

I gained a lot of experience in group teamwork and communication while understanding the functionality of the devices with the help of my teammates and learned a lot in terms of navigation and positioning data. When we had a problem with the accuracy of the positioning device, we were thinking about the origin of the problem and found out that our message decoding method was wrong. This was an important experience for me during our project work.

I liked the project of this course because it gave us a lot of free choices in our work so that we could use our own skills and creativity. The only problem for me was that I did not have a lot of knowledge of LoPy and MIT App Inventor so that I did not always know how each step of the project was executed. However, this gave me the opportunity to learn a lot about these technologies.

5 FEEDBACK

In general, we liked the project because we were able to freely choose our topic and the devices for its implementation. In addition to that, we are confident that we achieved good results as we were able to develop a functional prototype that can be upgraded to a professional tracking system.

The amount of time available for the project was limited, but nevertheless it provided us with a lot of knowledge about GPS data acquisition and processing. As a result, it was a helpful part of the course with which we learned a lot.