UPPSALA
UNIVERSITET

Report for IDT086

Project: Bus Tracking System

Group 18

Jatin Anand
Mohammad Metwally

November 9, 2019

Contents

Abstract

Public transportation system plays a vital role in our daily lives. Sometimes the arrival time is inaccurate. This project aims to display real time tracking of buses to show the exact bus location for the users. Raspberry Pi is the microprocessor used for this system. A GPS module is used for fetching the location of the buses. An internet dongle will be used for an internet connection to send the GPS coordinates to the cloud server after being processed by the Raspberry Pi. An HTML code is used to read the coordinates from the cloud server and display it on Google Maps API on a webpage.

1 Introduction

Buses are the most common form of public transport as it reduces the traffic on roads and it is a pocket-friendly means of transport. People occasionally resort to different types of transportation to reach their destinations, due to the confusion caused by the bus schedules. On the other hand, some passengers wait at the bus stops for longer time prior to the bus arrival so that they do not miss the bus. A solution to this problem is by providing an optimal method to know the arrival time of the bus to avoid this uncertainity. Bus tracking system aims to display the exact location of the bus to the users through a web page or a mobile application. This system will encourage the passengers in taking the buses daily and not seek for other ways to reach their destination.

2 System Components

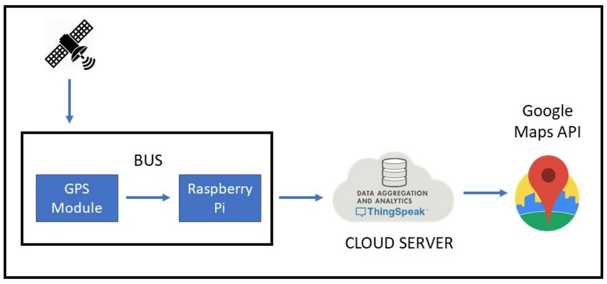


Figure 1: System structure.

The system structure as shown in **Figure 1** has four components which makes the whole bus tracking system. Each component being the GPS module, Raspberry Pi, ThingSpeak cloud server and the Google Maps API, is elaborated in the following sub-sections.

2.1 GPS Module

Mini Gmouse VK-162 GPS Receiver(USB) is the GPS Module used in this project. The GPS Module uses Standard NMEA Protocol format which means the data we receive from the GPS module will be in NMEA format. NMEA package has to be installed to receive the data on Raspberry Pi. pynmea is used as we use a python code to read the data.

2.2 Raspberry Pi 3

The bus tracking system uses a quad core processor in Raspberry Pi 3 Model B. A processor is required in this system to process the collection of data from the GPS module to Raspberry Pi and to send the data to a cloud server. GPS module can also be used with any other processor than Raspberry Pi as the aim is to just get the coordinates from the GPS module and upload that data to the cloud server for further use.

2.3 Cloud Server: ThingSpeak

ThingSpeak cloud server allows us to make fields for uploading different types of data in a graphical representation. We use MQTT API to communicate to the cloud. ThingSpeak has a write API key and a read API key. These keys allow us to write or read the data into the cloud. Program code listing will be more elaborative as it is written in python programming language in **Listing 1**. Thingspeak creates a channel ID and each channel has API keys to communicate with another machine. We use write API key generated by ThingSpeak to upload the location data on ThingSpeak.

2.4 Google Maps API

Google Maps API helps us customize google maps with our own content to display on webpages. Google Maps API is used to display the GPS location data which is saved in the cloud server, to point the exact location of the bus on google maps in real time.

3 Bus Tracking System

The bus tracking system consists of Raspberry Pi 3 and a GPS module. As shown in **Figure 1**, GPS module gets the location from a satellite and sends the data to Raspberry Pi. Raspberry Pi uploads the data to ThingSpeak cloud server. An HTML code is used for Google Maps API to display the data from the cloud server and point it on google maps on a webpage. Python3 is used to program the GPS module to collect the location coordinates and upload them to ThingSpeak cloud server. GPS module is connected to Raspberry Pi via a USB connection.

3.1 System Configuration

The GPS module has to be configured in the Raspberry Pi. To do that, connect the module using the USB to the Raspberry Pi USB port. Use `lsusb` command to list all the devices connected to the Raspberry Pi via USB.

Now, you have to check the port at which the module is connected. Go to the terminal and find the directory `/dev/tty*`

To find the serial port ID for the GPS module use `ls` command then look for tty* files. (Note: Do this once with the GPS module connected and once without the module connected to the Raspberry pi and compare the files to find the serial port ID in which the GPS data is being sent).
Our serial port ID was ttyACM0 at which the GPS module sent its coordinates data. Note the serial port ID which will be in tty**** format.

A package has to be installed in Raspberry Pi to read GPS data and display it on the screen. We use gpsd package.

Install the gpsd package in your Raspberry Pi using the following command:

```
sudo apt-get install gpsd gpsd-clients python-gps
```

Reset the GPS module to collect accurate data from the satellite:

```
sudo systemctl stop gpsd.socket
sudo systemctl disable gpsd.socket
```

After this step you must edit the default gateway by going to the following file and editing the gateway 127.0.0.1 to 0.0.0.0

```
emacs /lib/systemd/system/gpsd.socket
```

Kill all gpsd processes by:

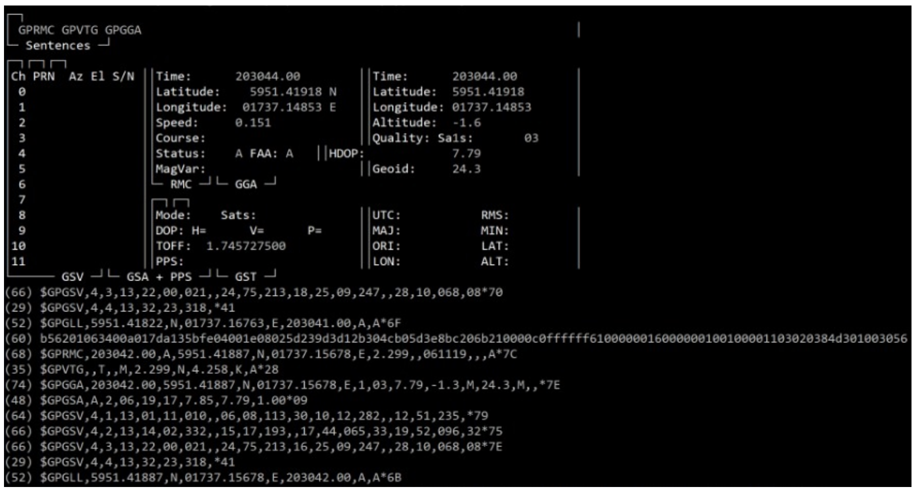
```
sudo killall gpsd
```

Sets the path to serial Port ID to which GPS is sending the data. Include your serial port ID in </tty****>

```
sudo gpsd /dev/tty**** -F /var/run/gpsd.sock
```

To display the GPS values on a screen from gpsd package, use this command:

```
gpsmon
```

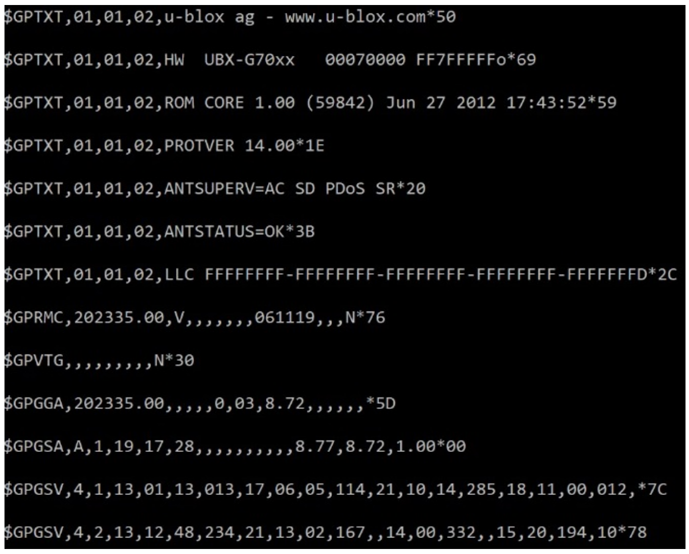


```
┌┐
│ GPRMC GPVTG GPGGA                                                         │
└ Sentences ┘
┌┐┌┐┌┐
Ch PRN  Az El S/N ││Time:      203044.00         ││Time:      203044.00      │
 0                ││Latitude:    5951.41918 N    ││Latitude:  5951.41918     │
 1                ││Longitude:  01737.14853 E    ││Longitude: 01737.14853    │
 2                ││Speed:      0.151            ││Altitude:  -1.6           │
 3                ││Course:                      ││Quality: Sa1s:      03    │
 4                ││Status:     A FAA: A   ││HDOP:           7.79          │
 5                ││MagVar:                      ││Geoid:      24.3          │
 6                │└ RMC ┘└ GGA ┘
 7                │┌┐┌┐
 8                ││Mode:    Sats:               ││UTC:          RMS:        │
 9                ││DOP: H=      V=      P=      ││MAJ:          MIN:        │
10                ││TOFF:  1.745727500           ││ORI:          LAT:        │
11                ││PPS:                         ││LON:          ALT:        │
└────── GSV ┘└ GSA + PPS ┘└ GST ┘
(66) $GPGSV,4,3,13,22,00,021,,24,75,213,18,25,09,247,,28,10,068,08*70
(29) $GPGSV,4,4,13,32,23,318,*41
(52) $GPGLL,5951.41822,N,01737.16763,E,203041.00,A,A*6F
(60) b56201063400a017da135bfe04001e08025d239d3d12b304cb05d3e8bc206b210000c0ffffff61000000160000001001000011030203840301003056
(68) $GPRMC,203042.00,A,5951.41887,N,01737.15678,E,2.299,,061119,,,A*7C
(35) $GPVTG,,T,,M,2.299,N,4.258,K,A*28
(74) $GPGGA,203042.00,5951.41887,N,01737.15678,E,1,03,7.79,-1.3,M,24.3,M,,*7E
(48) $GPGSA,A,2,06,19,17,7.85,7.79,1.00*09
(64) $GPGSV,4,1,13,01,11,010,,06,08,113,30,10,12,282,,12,51,235,*79
(66) $GPGSV,4,2,13,14,02,332,,15,17,193,,17,44,065,33,19,52,096,32*75
(66) $GPGSV,4,3,13,22,00,021,,24,75,213,16,25,09,247,,28,10,068,08*7E
(29) $GPGSV,4,4,13,32,23,318,*41
(52) $GPGLL,5951.41887,N,01737.15678,E,203042.00,A,A*6B
```

Figure 2: Data displayed on gpsmon screen

To display the data being received at the serial port, use this command:

```
cat /dev/ttyACM0
```



```
$GPTXT,01,01,02,u-blox ag - www.u-blox.com*50

$GPTXT,01,01,02,HW  UBX-G70xx   00070000 FF7FFFFFo*69

$GPTXT,01,01,02,ROM CORE 1.00 (59842) Jun 27 2012 17:43:52*59

$GPTXT,01,01,02,PROTVER 14.00*1E

$GPTXT,01,01,02,ANTSUPERV=AC SD PDoS SR*20

$GPTXT,01,01,02,ANTSTATUS=OK*3B

$GPTXT,01,01,02,LLC FFFFFFFF-FFFFFFFF-FFFFFFFF-FFFFFFFF-FFFFFFFD*2C

$GPRMC,202335.00,V,,,,,,,061119,,,N*76

$GPVTG,,,,,,,,,N*30

$GPGGA,202335.00,,,,,0,03,8.72,,,,,,*5D

$GPGSA,A,1,19,17,28,,,,,,,,,,8.77,8.72,1.00*00

$GPGSV,4,1,13,01,13,013,17,06,05,114,21,10,14,285,18,11,00,012,*7C

$GPGSV,4,2,13,12,48,234,21,13,02,167,,14,00,332,,15,20,194,10*78
```

Figure 3: Data received at the serial port

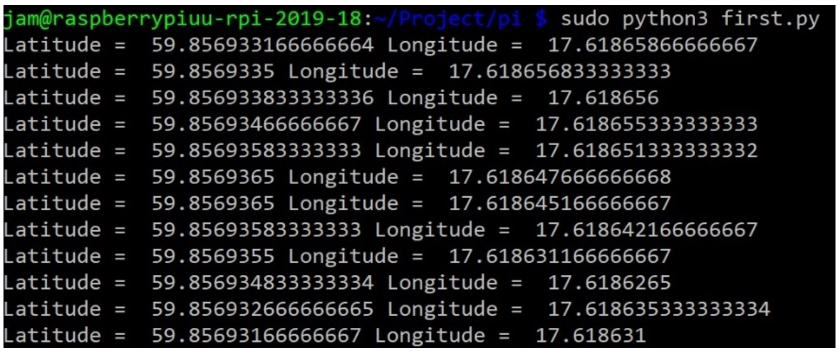
After the GPS is setup and the data is displayed, it needs to be uploaded to the cloud to be read by google maps API. We use the python code in **Listing 1** to upload our live coordinates to ThingSpeak cloud server.

Listing 1: first_gps.py

```
from __future__ import print_function
import serial
import pynmea2
import paho.mqtt.publish as publish
import string
import time

#assigns the serial port to ttyACM0 with 9600 baud rate and a timeout of 0.5
    seconds
serialStream = serial.Serial("/dev/ttyACM0", 9600, timeout=0.5)
while(1):
#Thingspeak channel ID
    channelID = "896656"
#Thingspeak write API key
    writeAPIKey = "HOC815LAIRPODA2H"
#Thingspeak hostname
    mqttHost = "mqtt.thingspeak.com"
    mqttUsername = "Bus tracking data"
#MQTT API key
    mqttAPIKey = "CCBI1R47UJINGCOO"
    tTransport = "websockets"
    tPort = 80
#Assigns Channel ID and the write API key
    topic = "channels/" + channelID  + "/publish/" + writeAPIKey
#reads the GPS data and stores in sentence
    sentence = serialStream.readline().decode()
    if sentence.find('GGA') > 0:
#Converts the data to nmea standards
        data = pynmea2.parse(sentence)
#Uploads the data to the fields created on ThingSpeak
        payload = "field1=" + str(data.latitude) + "&field2=" + str(data.
            longitude)
#Publish/write the data to the Thingspeak cloud server
        publish.single(topic, payload, hostname=mqttHost, transport=\
tTransport, port=tPort,auth={'username':mqttUsername,'password':mqttAPIKey})
#Prints the current Latitude and Longitude values from the GPS module
        print ("Latitude = ", data.latitude,"Longitude = ", data.longitude)
#Sleeps for 15 seconds as Thingspeak free license allows an update every 15s
        time.sleep(15)
```

The ouput of this code is shown in **Figure 4** which prints the Latitude and the Longitude of the current location. **Figure 5** shows the coordinate data that has been uploaded to ThingSpeak cloud server in a graphical representation.



```
jam@raspberrypiuu-rpi-2019-18:~/Project/pi $ sudo python3 first.py
Latitude =  59.856933166666664 Longitude =  17.61865866666667
Latitude =  59.8569335 Longitude =  17.618656833333333
Latitude =  59.856933833333336 Longitude =  17.618656
Latitude =  59.85693466666667 Longitude =  17.618655333333333
Latitude =  59.85693583333333 Longitude =  17.618651333333332
Latitude =  59.8569365 Longitude =  17.618647666666668
Latitude =  59.8569365 Longitude =  17.618645166666667
Latitude =  59.85693583333333 Longitude =  17.618642166666667
Latitude =  59.8569355 Longitude =  17.618631166666667
Latitude =  59.856934833333334 Longitude =  17.6186265
Latitude =  59.856932666666665 Longitude =  17.618635333333334
Latitude =  59.85693166666667 Longitude =  17.618631
```

Figure 4: GPS Module coordinates (Latitude & Longitude).

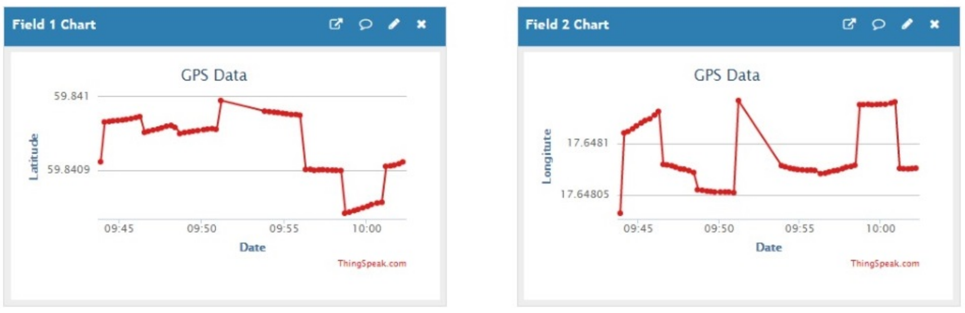


Figure 5: GPS data displayed on ThingSpeak.

Now, the Location data has to be displayed on google maps. For doing this, we use the Google maps API. An HTML code helps us get the data from ThingSpeak cloud using the read API key and point the location coordinates on google maps API on an HTML webpage. This is how the pointer will point to the bus location on google maps on a webpage.

Listing 2: maps_webpage.html

```
<html>
 <head>
   <title>UL</title>
   <meta name="viewport" content="initial-scale=1.0, user-scalable=no">
   <meta charset="utf-8">
   <style>

     /* Map height to define the size of the div
       * element that contains the map. */

      #map {
        height: 100%; }
         html, body {
        height: 100%;
        margin: 0;
        padding: 0;   }
    </style>
    <script src="https://maps.googleapis.com/maps/api/js?key=
         AIzaSyDLVP3OxpFJwl9jDfS2ZkNCiuLTic-cXzg"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.
         min.js"></script>
    <script>
      var map;
        var x;
        function loadmaps(){ $.getJSON("https://api.thingspeak.com/channels
            /896656/fields/1/last.json?api_key=NIA24KP25AC19H05", function(
            result){
            var m = result;
            x=Number(m.field1);
                           //alert(x);          });
$.getJSON("https://api.thingspeak.com/channels/896656/fields/2/last.json?
    api_key=NIA24KP25AC19H05", function(result){
            var m = result;
            y=Number(m.field2);
        }).done(function() {
                initialize();    });      }
        window.setInterval(function(){
        loadmaps();
            }, 9000);
      function initialize() {
          //alert(y);
        var mapOptions = {
          zoom: 18,
          center: {lat: x, lng: y}        };
        map = new google.maps.Map(document.getElementById('map'),
            mapOptions);
        var marker = new google.maps.Marker({
          position: {lat: x, lng: y},
          map: map        });
        var infowindow = new google.maps.InfoWindow({
          content: '<p>Marker Location:' + marker.getPosition() + '</p>'});
        google.maps.event.addListener(marker, 'click', function() {
          infowindow.open(map, marker);        });      }
      google.maps.event.addDomListener(window, 'load', initialize);
    </script>
  </head>
  <body>
    <div id="map"></div>
  </body>
</html>
```

The HTML file will direct to google maps as shown in **Figure 6** and point the location on the webpage.

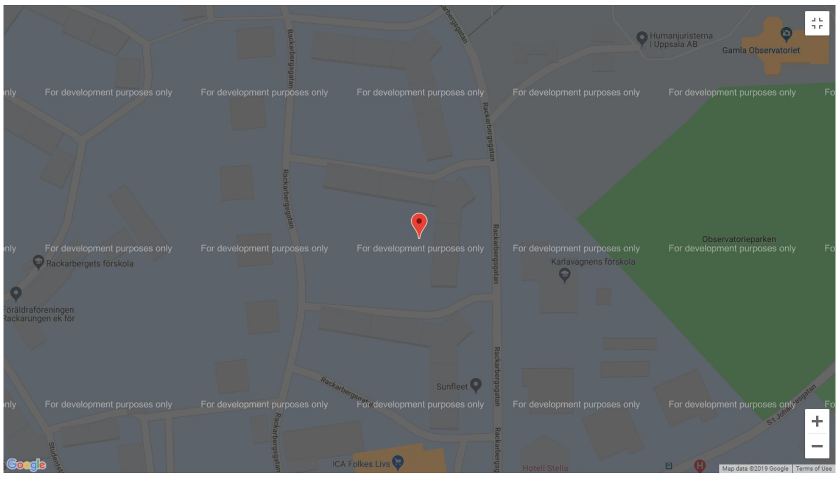


Figure 6: GPS data displayed on Google Maps API

4 Conclusion

Raspberry Pi along with the GPS module will be present inside every bus in the transport department. Public bus transport department can add the link to the webpage in **Figure 6** to their mobile application for sharing the location of each bus. This can help the department to monitor live locations of all the buses in real time.

Additional features can be added to improve the productivity of this system by calculating the speed of the buses or adding another sensor to Raspberry Pi to count the number of passengers travelling in each bus to monitor the population in the bus.

References

[1] GitLab: gpsd,
`https://gpsd.gitlab.io/gpsd/index.htmll`

[2] GitHub: pynmea2,
`https://www.github.com/Knio/pynmea2l`

[3] MathWorks: ThingSpeak,
`https://se.mathworks.com/help/thingspeak/use-raspberry-pi-board-that-runs-python-websockets-to-publish-to-a-channel.html`

[4] GitHub: HTML code,
`https://github.com/vikkey321/Display-Thingspeak-GPS-data-on-Google-Map-from-ESP12E-and-Neo-6M-GPS/blob/master/index.html`

[5] Google Maps API,
`https://developers.google.com/maps/documentation/javascript/tutorial`