

# **FA17IS590- Internet of Things**

## **GPS tracker device using Raspberry Pi.**



**Submitted by**

**Pratik Shrivastava and Niteesh Kanungo**

**Instructor – Mark Moran**

## Abstract

We have found about a number of incidence about stolen bicycles at the campus area at University of Illinois, Urbana Champaign. This issue has been raised to the police department but the problem with the system of tracing this bikes is that there is no provision after they went missing. The only resource that can help us track the stolen bikes are the surveillance cameras that are installed at various location around the campus. Also, the places where most of the bicycles stands are at the corners and out of the reach of the cameras. Considering this problem we are aiming to develop a compact and smart Geo-positioning Tracking sensory device using Raspberry-Pie and integrating it with a GPS tracker. The device will be aware of the surrounding scanning for the communication protocols and it will be sending the real time location-data at a custom user defined Intervals. The implementation of the device can be in a number of projects depending upon the requirement. It can be used to track stolen Bicycles at University campus, Tracking of Cargos at scale for accurate shipment and ETA as well as tracking Pets or children who are prone to go missing. This system can have viable application for implementation within many system. In this report we have implemented and discussed the process of developing an IOT system that can help us in Live-tracking, Geo-fencing with using cloud storage services such as Amazon Web Services to keep a track of Geo-location as well as doing Analytics from the Data generated.

## Introduction

The pace of development of Internet has open new channels for networking opportunities. Networking protocols given rise to number of options to create nodes on the networks. This networks serves us as roadways for efficient communication of data and information from one place to another. This connected mesh of network plays a very crucial part in our day to day life, it helps us to assess the environment, to make decision and to act upon the predicted outcomes [1]. For the past decade few this networks have seen a rise of special nodes and infrastructure which we call as Internet of things.

Internet of Things is a buzz word or we can say- it's a network of interconnected devices, appliances or embedded devices which have sensors and actuators. This nodes acts as data collection points and works standalone or can work as a group to help in assessment of the situations or conditions [2]. Experts have estimated the rise of IOT devices to reach up to 30 billion in the next few years. The financial aspects of such devices are also on the rise which is estimated to reach about \$7.1 Trillion by 2020 [3].

The aim of developing such technologies on top of the Internet infrastructure creates opportunities for more direct integrations of the physical networks, sensors and actuators. This helps in reducing human intervention and creating a self-driven autonomous environment. IoT devices have seen their implementation from medical devices such as heart monitoring implants, sensors placed on Airplanes for checking problems on the run, DNA analysis devices to live feed of wild animals in coastal waters [4]. This systems provide a wealth of information in form of data and can help in analyzing existing systems or testing developing systems before the implementation.

In the recent years the use of IoT devices has converge into multiple technologies. This is fueled by use of Internet ubiquitously. Wireless communication, real time analytics, machine learning artificial intelligence and automation has played a vital role in the development of infrastructure one over the other.

We have taken up one such scenario where using IoT devices would really help to work on a problem that the people at campus town usually finds difficult. The problem of Bicycle theft is not new and prone to some major challenges. This bicycles do not hold a good monetary value

for which the already existing system cannot help and track each and every individual cases of bicycle theft. If we can find a way on tracking the Bicycles using the existing IoT infrastructure, and can locate the geographic location of the same in real time we can help in reducing the problem of bicycle theft to a much greater extent. It can also help us to infuse fear in people who are involved in such kind of activities.

In developing such a system we came across many challenges, problems and the viable solution










#### Main objective for the project

- Designing and Working of the IOT system from scratch. This will help us in understanding the development cycle of any product.
- Working with Raspberry Pi, Sensors and AWS Cloud Technology to understand how the implementation is done at the software level and how each part within the ecosystem works as a network.
- Understanding Application of such system as per Business needs, developing business models that can be used in many other scenarios.

#### Main motivation

- We can track stolen Goods, Cargos, kids and Dogs everything with the help of one single product.
- We are trying to make this project as cost effective as possible.
- Creating a compact product is not feasible, as we will be working on Raspberry Pi which do have various IO terminals attached. In the final product most of the parts from the main board can be eliminated to reduce the size of the product to half the size of matchbox
- We are aiming that the product will have very low Power Consumption and will be smart enough to change communication protocols as available to increase the life on single charge.
- Also, the product will be IP67 Weather and dust protected, will be built with industrial design in mind to be performant in any type of scenario.

## Business canvas model

<b>Key Partners</b>  <ul style="list-style-type: none"><li>• AWS services</li><li>• Telecom Provider</li><li>• Cycle Owners</li><li>• Retail Marketers</li></ul>	<b>Key Activities</b>  <ul style="list-style-type: none"><li>• Development of Software</li><li>• Connectivity with AWS and other services</li></ul>	<b>Value Propositions</b>  <ul style="list-style-type: none"><li>• Safety and security of product</li><li>• Live Tracking</li><li>• Cost effective</li><li>• Small and compact</li></ul>	<b>Customer Relationships</b>  <ul style="list-style-type: none"><li>• Customized reports</li><li>• Free demo and trials</li><li>• Free upgrades</li><li>• Referral bonus</li></ul>	<b>Customer Segments</b>  <ul style="list-style-type: none"><li>• Students Cycle Owners</li><li>• Cycle Stores</li></ul>
	<b>Key Resources</b>  <ul style="list-style-type: none"><li>• AWS</li><li>• Hardware</li><li>• Developers</li><li>• Data analyst</li></ul>		<b>Channels</b>  <ul style="list-style-type: none"><li>• Ecommerce</li><li>• Social media</li><li>• Newspaper</li><li>• University mailing list</li></ul>	
<b>Cost Structure</b>  <ul style="list-style-type: none"><li>• Hardware cost</li><li>• Cost of Software development</li><li>• Amazon Web Services</li></ul>			<b>Revenue Streams</b>  <ul style="list-style-type: none"><li>• Sales</li><li>• Subscription based services</li><li>• Cycle Rent</li><li>• Data analytics</li></ul>	

## Hardware Setup

The list of all the equipment used in this project with purchase links from Amazon and their current prices as of December 17, 2017.

### Equipment's and their cost

- Raspberry Pi 3 - \$34.49 – [Link](#)
- Power Bank/Battery pack - \$12.95
- 32 GB Micro SD Card - \$7.95
- Female/Female Jumper Wires - \$4.99
- GPS Module U-Blox Neo-6M with active antenna - \$25.99

Total: ~\$90 (including tax/shipping)

The total cost for one device came close to be about \$100. In the long run and depending upon the used cases this cost can also be brought down if we scale up the total production units and develop various monetization channels within the system like – customized analytics on users and selling anonymous data to for modelling and predictive analytics

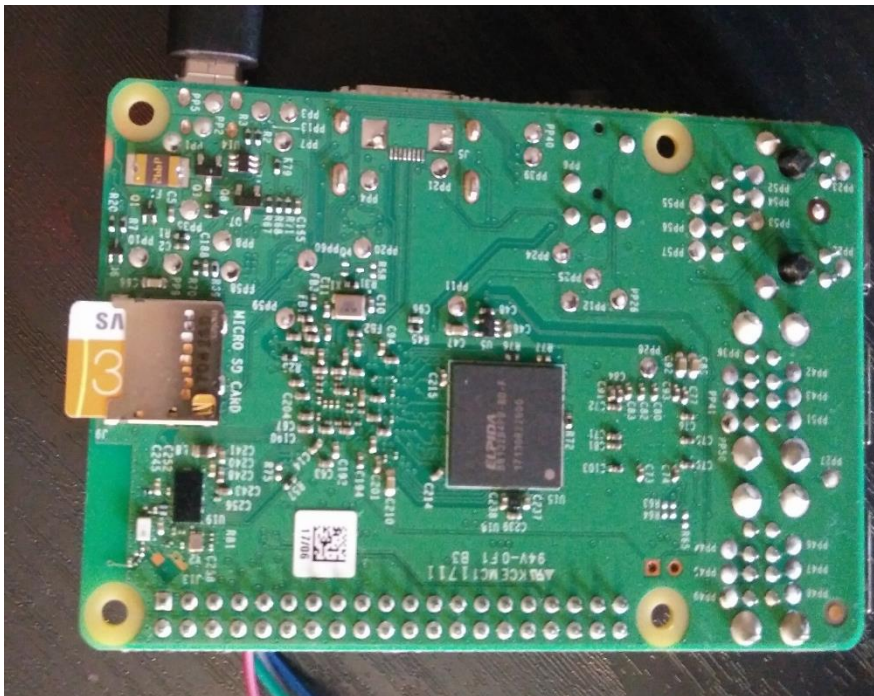
## Hardware Configuration

### Raspberry Pi Configuration

CPU	1.2 GHz 64-bit quad-core ARM Cortex-A53
Memory (SDRAM)	1 GB (shared with GPU)
USB 2.0 ports	4 (via the on-board 5-port USB hub)
Video input	15-pin MIPI camera interface (CSI) connector, used with the Raspberry Pi camera or Raspberry Pi <u>NoIR</u> camera[51]
Video outputs	HDMI (rev 1.3), composite video (3.5 mm TRRS jack)
On-board storage	<u>MicroSDHC</u> slot
On-board network	10/100 Mbit/s Ethernet, 802.11n wireless, Bluetooth 4.1
Power source	5 V via <u>MicroUSB</u> or GPIO header
Weight	45 g (1.6 oz)
Size	85.60 mm × 56.5 mm (3.370 in × 2.224 in), not including protruding connectors

### Pi Setup

- Installing the MicroSD card



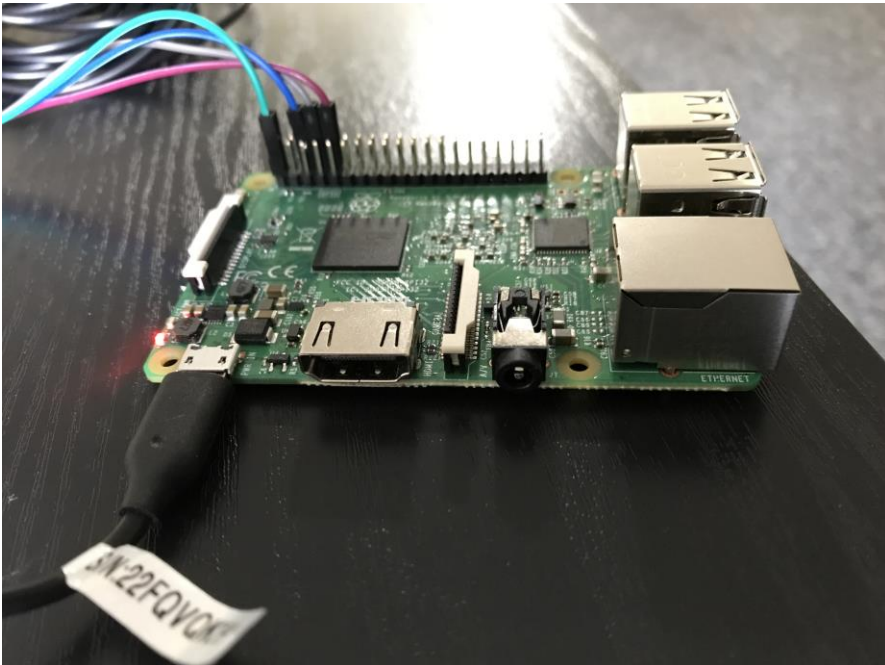
- Power supply



- Battery Power



- Display output via HDMI



We HDMI cable is used to connect it to the monitor for the initial setup of the Raspberry Pi

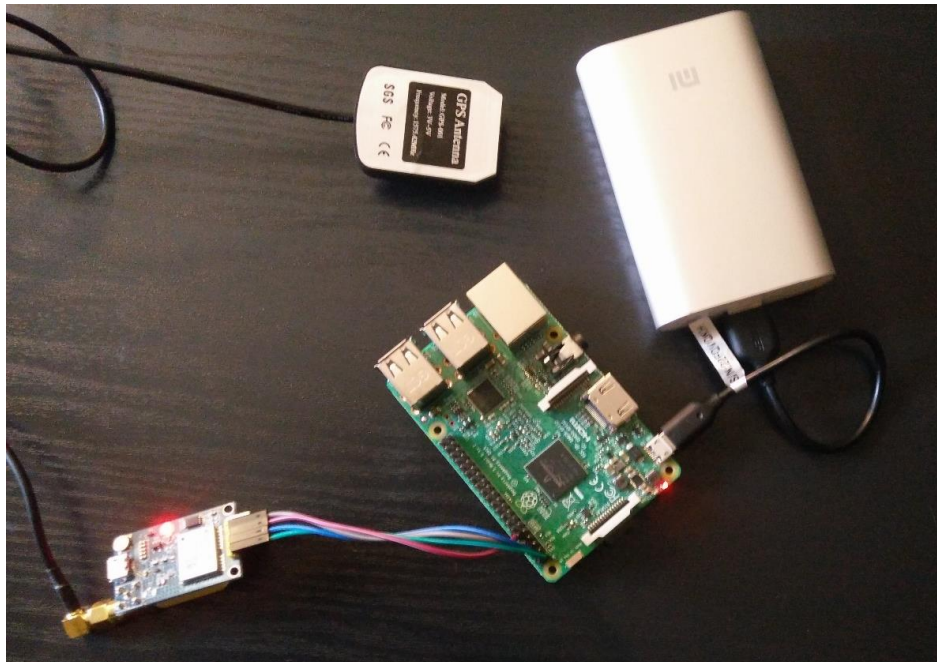
## GPS and Raspberry

### Step 1: Electrical Connection

The first step is to connect the GPS module to the Raspberry PI. There are only 4 wires (F to F), so it's a simple connection.

#### Neo-6M RPI

VCC to Pin 1	which is 3.3v
TX to Pin 10	which is RX (GPIO15)
RX to Pin 8	Which is TX (GPIO14)
Gnd to Pin 6	which is Gnd





## Software setup/Development

### Raspberry Pi Setup:

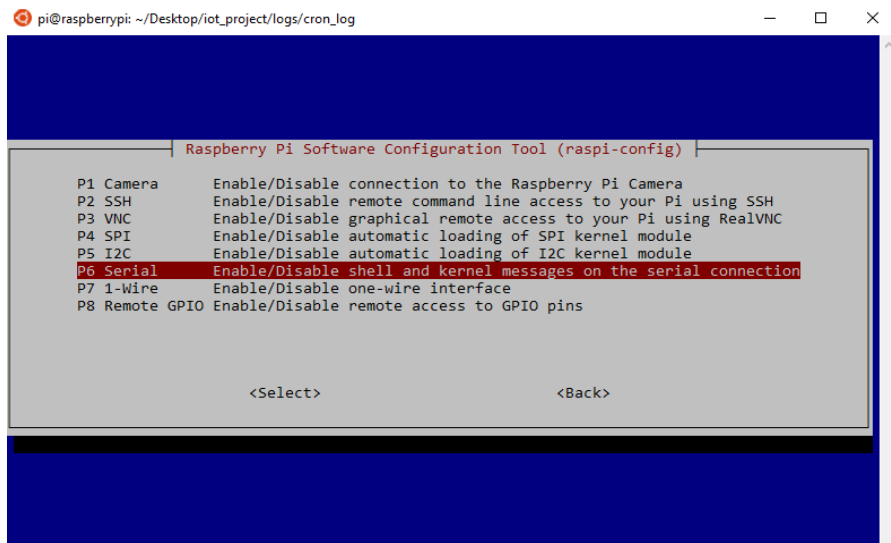
For using the raspberry pi, we need to install an operating system on it. There are many OS available for it, but the default is NOOBS (New Out Of Box Software). We used the steps mentioned in the [documentation](#) for installing the software on the raspberrypi support page and performed the below steps:

- Formatting SDcard - Fat32
- Download NOOBS on local machine
- Copy the NOOBS on the SD card, and plug it in raspberry pi
- Once, pi detects the software a green led light flashes on the board and it starts the process of the installing the software by providing useful prompt.
- Updating packages and Installing Libraries

```
pi@raspberrypi:~/Desktop/iot_project/logs/cron_log $ sudo apt-get upgrade
```

- Enabling SSH and Serial Port: We enabled the ssh and serial port on the raspberry pi using the below command. The ssh enabled us the flexibility to connect the device without connecting it to a monitor and execute our commands remotely.

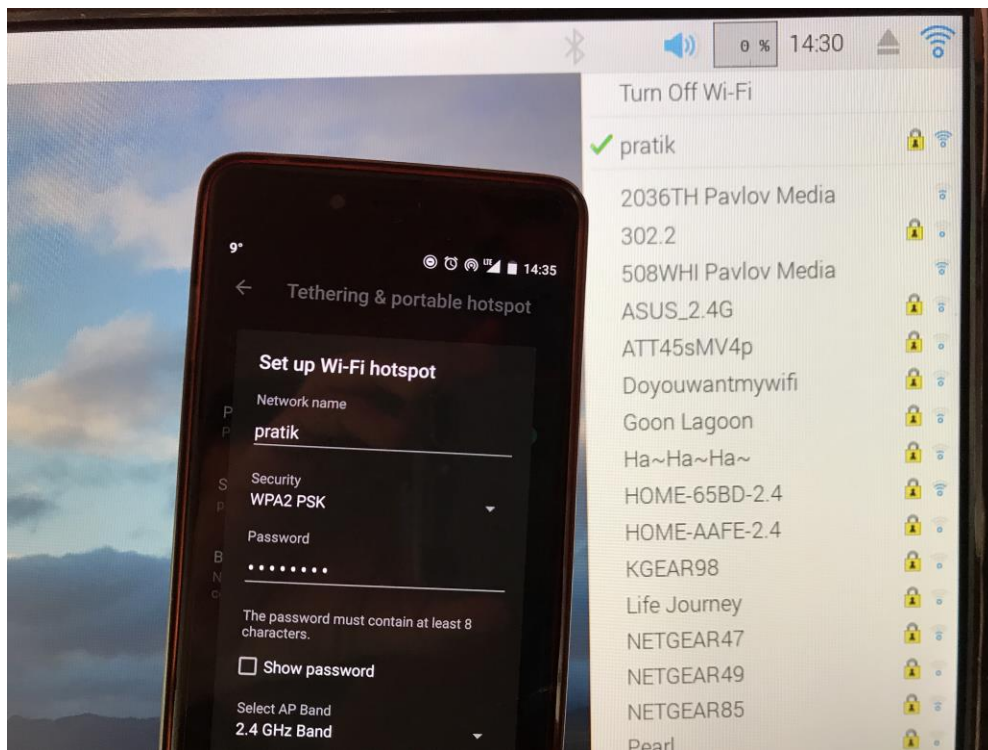
```
pi@raspberrypi:~/Desktop/iot_project/logs/cron_log $ sudo raspi-config
```



```
pi@raspberrypi: ~  
pi@raspberrypi:~$ id  
uid=1000(pi) gid=1000(pi) groups=1000(pi),4(adm),20(dialout),24(cdrom),27(sudo),29(audio),44(video),46(plugdev),60(games),100(users),101(input),108(netdev),997(gpio),998(i2c),999 spi)  
pi@raspberrypi:~$ uname -a  
Linux raspberrypi 4.9.59-v7+ #1047 SMP Sun Oct 29 12:19:23 GMT 2017 armv7l GNU/Linux  
pi@raspberrypi:~$ ^C  
pi@raspberrypi:~$ ^C  
pi@raspberrypi:~$ logout  
Connection to 10.0.0.36 closed.  
root@Wandrer:~# ssh pi@10.0.0.36  
pi@10.0.0.36's password:  
Linux raspberrypi 4.9.59-v7+ #1047 SMP Sun Oct 29 12:19:23 GMT 2017 armv7l  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Sun Dec 17 13:20:19 2017 from 10.0.0.19  
  
SSH is enabled and the default password for the 'pi' user has not been changed.  
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.  
  
pi@raspberrypi:~$ uname -a  
Linux raspberrypi 4.9.59-v7+ #1047 SMP Sun Oct 29 12:19:23 GMT 2017 armv7l GNU/Linux  
pi@raspberrypi:~$ id  
uid=1000(pi) gid=1000(pi) groups=1000(pi),4(adm),20(dialout),24(cdrom),27(sudo),29(audio),44(video),46(plugdev),60(games),100(users),101(input),108(netdev),997(gpio),998(i2c),999 spi)  
pi@raspberrypi:~$
```

Wi-Fi setup with Mobile Hotspot:

Pairing with the Mobile devices Hotspot



## GPS setup –

Below are the required steps for setting up the GPS on the serial port for reading the data.

- Edit cmdline.txt file.

```
pi@raspberrypi:~/Desktop/iot_project/logs/cron_log $ sudo nano /boot/cmdline.txt
```

Remove console=ttyAMA0,115200 from the config.

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 console=tty1 root=/dev/mmcblk0p6  
rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait
```

Edited:

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p6 rootfstype=ext4  
elevator=deadline fsck.repair=yes rootwait
```

- Disable serial-getty on ttyAMA0. This OS service interferes while reading the data from the serial port using the script. We created a logon script which stops and disables this service and reboots.

```
9  ## Stop and disable the getty service.  
10  
11  
12  sudo systemctl stop getty@ttyAMA0.service  
13  sudo systemctl disable getty@ttyAMA0.service  
14  
15
```

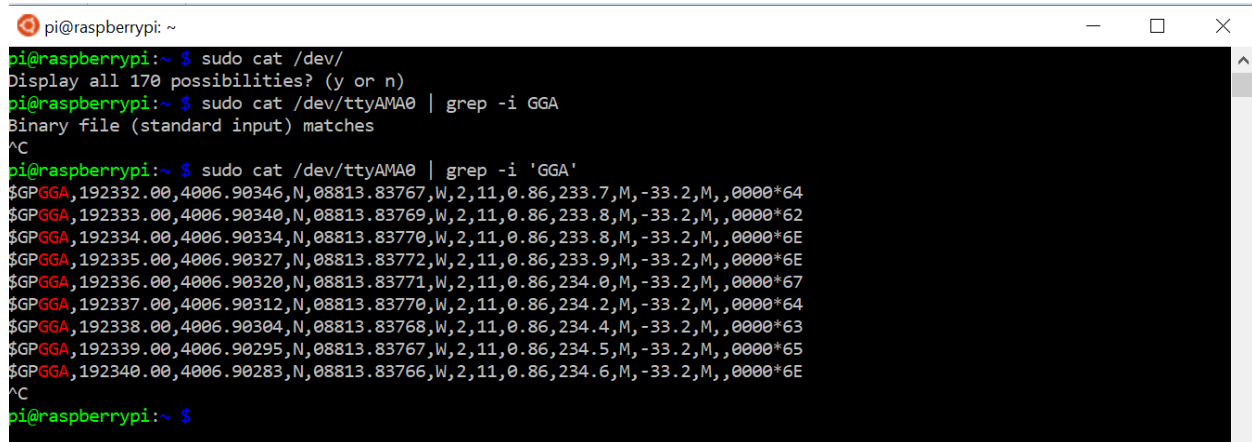
## 3. Reboot

```
pi@raspberrypi:~ $ sudo reboot now_
```

## Testing Serial Port with GPS

For initial testing we wanted to see if the serial port is receiving data from the GPS or not. The fresh GPS takes about 30 min or more to get the fix on the satellites before it starts writing the data. We had to wait for 2 days to get the fix, it has some technical and configuration issues

which lead to this problem. The GPS was configured with the raspberry pi at port ttyAMA0, so it was writing to this port. We used the below command to test whether raspberry pi is receiving the raw data or not.



```
pi@raspberrypi: ~  
pi@raspberrypi:~$ sudo cat /dev/  
Display all 170 possibilities? (y or n)  
pi@raspberrypi:~$ sudo cat /dev/ttyAMA0 | grep -i GGA  
Binary file (standard input) matches  
^C  
pi@raspberrypi:~$ sudo cat /dev/ttyAMA0 | grep -i 'GGA'  
$GPGGA,192332.00,4006.90346,N,08813.83767,W,2,11,0.86,233.7,M,-33.2,M,,0000*64  
$GPGGA,192333.00,4006.90340,N,08813.83769,W,2,11,0.86,233.8,M,-33.2,M,,0000*62  
$GPGGA,192334.00,4006.90334,N,08813.83770,W,2,11,0.86,233.8,M,-33.2,M,,0000*6E  
$GPGGA,192335.00,4006.90327,N,08813.83772,W,2,11,0.86,233.9,M,-33.2,M,,0000*6E  
$GPGGA,192336.00,4006.90320,N,08813.83771,W,2,11,0.86,234.0,M,-33.2,M,,0000*67  
$GPGGA,192337.00,4006.90312,N,08813.83770,W,2,11,0.86,234.2,M,-33.2,M,,0000*64  
$GPGGA,192338.00,4006.90304,N,08813.83768,W,2,11,0.86,234.4,M,-33.2,M,,0000*63  
$GPGGA,192339.00,4006.90295,N,08813.83767,W,2,11,0.86,234.5,M,-33.2,M,,0000*65  
$GPGGA,192340.00,4006.90283,N,08813.83766,W,2,11,0.86,234.6,M,-33.2,M,,0000*6E  
^C  
pi@raspberrypi:~$
```

## AWS (Amazon Web Services)

The AWS IoT provides a great and efficient way for bi-directional communication between the AWS cloud and the interconnected sensors or devices connected to the internet.

### AWS/IOT setup

We have followed the [AWS documentation](#) for setting up the AWS IoT and setting up our device for communicating it with the AWS services. Below are some of the steps that we followed for the steps.

- Create an account on AWS
- Create a Thing
- Creating Certificates
- Policies
- Attaching certificate policies to a thing
- Installing AWS-IOT SDK Python kit on Pi

After installation, the required credentials for working of the AWS IoT should be saved into the device. The below screenshot gives the detail about the creds:

```
pi@raspberrypi:~/Desktop/iot_project $ ls -l
total 36
-rw-r--r-- 1 pi pi 1758 Dec 17 03:54 root-CA.crt
-rw-r--r-- 1 pi pi 451 Dec 17 03:54 iot_pi.public.key
-rw-r--r-- 1 pi pi 1679 Dec 17 03:54 iot_pi.private.key
-rw-r--r-- 1 pi pi 1220 Dec 17 03:54 iot_pi.cert.pem
-rw-r--r-- 1 pi pi 4010 Dec 17 04:18 new_app.py
```

Subscribing for topics:

The image shows a Raspberry Pi terminal window on the left and an AWS IoT console window on the right.

**Terminal Window:**

```
pi@raspberrypi:~/Desktop/iot_project $ ls -l
total 36
-rw-r--r-- 1 pi pi 1758 Dec 17 03:54 root-CA.crt
-rw-r--r-- 1 pi pi 451 Dec 17 03:54 iot_pi.public.key
-rw-r--r-- 1 pi pi 1679 Dec 17 03:54 iot_pi.private.key
-rw-r--r-- 1 pi pi 1220 Dec 17 03:54 iot_pi.cert.pem
-rw-r--r-- 1 pi pi 4010 Dec 17 04:18 new_app.py
-rw-r--r-- 1 pi pi 248 Dec 17 04:23 tracker_app.sh
-rwxr-xr-x 1 pi pi 3977 Dec 17 04:29 gps_tracker_app.py
drwxr-xr-x 3 pi pi 4096 Dec 17 04:42 logs
-rw-r--r-- 1 root root 1017 Dec 17 04:43 test_data_gps.csv
pi@raspberrypi:~/Desktop/iot_project $ sudo python gps_tracker_app.py &
[1] 11931
pi@raspberrypi:~/Desktop/iot_project $ 2017-12-17 13:26:46.217337,40.1150023333,-88.2306148333
2017-12-17 13:26:51.279580,40.1150016667,-88.230614
Published topic gps_location: {'lat': 40.11500166666666, 'timestamp': datetime.datetime(2017, 12, 17, 13, 26, 51, 281117), 'lon': -88.230614}
2017-12-17 13:26:56.352362,40.1150016667,-88.2306148333
Published topic gps_location: {'lat': 40.11500166666666, 'timestamp': datetime.datetime(2017, 12, 17, 13, 26, 56, 353792), 'lon': -88.23061483333333}
pi@raspberrypi:~/Desktop/iot_project $ 2017-12-17 13:27:01.422838,40.1150025,-88.2306163333
2017-12-17 13:27:06.492033,40.115003,-88.2306178333
Published topic gps_location: {'lat': 40.115003, 'timestamp': datetime.datetime(2017, 12, 17, 13, 27, 6, 93461), 'lon': -88.23061783333333}
2017-12-17 13:27:11.561084,40.1150031667,-88.2306193333
Published topic gps_location: {'lat': 40.11500316666667, 'timestamp': datetime.datetime(2017, 12, 17, 13, 27, 11, 562974), 'lon': -88.23061933333334}
2017-12-17 13:27:16.631537,40.1150033333,-88.2306215
Published topic gps_location: {'lat': 40.11500333333333, 'timestamp': datetime.datetime(2017, 12, 17, 13, 27, 16, 632945), 'lon': -88.2306215}
```

**AWS IoT Console:**

The console shows the 'gps\_location' topic. A message is published with the following content:

```
{
  "message": "Hello from AWS IoT console"
}
```

The console also displays a list of messages published to the 'gps\_location' topic, including the following JSON data:

```
{
  "lat": 40.11500333333333, "timestamp": datetime.datetime(2017, 12, 17, 13, 27, 16, 632945), "lon": -88.2306215
}
```



## Application development

Creation of script for running on reboot

```
1  #!/bin/sh
2
3
4  ## Read the data from the GPS
5  echo "started on: " `date`
6
7
8
9
10 ## Stop and disable the getty service.
11
12
13 sudo systemctl stop getty@ttyAMA0.service
14 sudo systemctl disable getty@ttyAMA0.service
15
16 ### sleep for 2 min, so that hte other services like wifi etc are up.
17
18 sleep 120
19
20 ## Execute the gps tracker script.
21
22
23 cd /home/pi/Desktop/iot_project/
24
25 sudo python gps_tracker_app.py > logs\`date.log
```

Creation of a cronjob for executing the above script: This cronjob helps in execution of the gps\_tracker.py script at reboot. So whenever the device is switched on, this cronjob will get executed and will start our script after 2 min.

```
#@reboot echo "hi" > /home/pi/reboot.txt 2>&1
#@reboot /home/pi/Desktop/iot_project/tracker_app.sh > /home/pi/Desktop/iot_project/logs/cron_log/tracker_app.log 2>&1
~
~
~
"/tmp/crontab.XC3Ldk/crontab" 25L, 1054C
```

## Reading and parsing Data from serial port

```
#### Function for reading the serial data from the GPS
#### using the serial ports

def read_serial_data():

    serialStream = serial.Serial("/dev/ttyAMA0", 9600, timeout=5)
    cnt = 0
    while True:
        ### Read the data from serial port line by line.
        sentence = serialStream.readline()

        ### Check for the NMEA GPS fix.
        ### if found , then parse the latitude and longitude values.
        if(sentence.find('GGA') > 0):
            data = pynmea2.parse(sentence)

            lat,lon = data.latitude, data.longitude
            data = str(datetime.now())+ ',' + str(lat) + ',' + str(lon) + '\n'
            # print(data)

            ### Open a file and save this data into it.
            f = open('logs/test_new_app.txt',"a+")
            f.write(data)

            ### Check for the starting position of the GPS.
            ### If cnt == 0 then only starting position or else it would be the next position.
            ###

            if(cnt > 0):
                new_position = [lat,lon]

                ### Pass the starting and new position to the main function for further processing.
                main_func(start_position,new_position)
            else:
                start_position = [lat,lon]
                cnt = 100
            ### wait for sometime before reading the next values.
            time.sleep(delay)
            f.close()

    return
```

## Setting up AWS connection/MQTT connection

```
### AWS credential setup with the MQTT Client

myAWSIoTMQTTClient = AWSIoTMQTTClient(clientId)
myAWSIoTMQTTClient.configureEndpoint(host, 8883)
myAWSIoTMQTTClient.configureCredentials(rootCAPath, privateKeyPath, certificatePath)

# AWSIoTMQTTClient connection configuration
myAWSIoTMQTTClient.configureAutoReconnectBackoffTime(1, 32, 20)
myAWSIoTMQTTClient.configureOfflinePublishQueueing(-1) # Infinite offline Publish queueing
myAWSIoTMQTTClient.configureDrainingFrequency(2) # Draining: 2 Hz
myAWSIoTMQTTClient.configureConnectDisconnectTimeout(10) # 10 sec
myAWSIoTMQTTClient.configureMQTTOperationTimeout(5) # 5 sec

# Connect and subscribe to AWS IoT
myAWSIoTMQTTClient.connect()
```

Checking for the displacement & Pushing Data to AWS and local log file.

```
### Check whether there is a change in position above the
### threshold if yes, then push the data to AWS and into a file.
if(chk_move(change_pos)):

    #print('In check_movement')
    ### Publish data to AWS
    myAWSIoTClient.publish(topic,str(create_data_json(datetime.now(), new_position)), 0)
    print('Published topic %s: %s\n' % (topic,str(create_data_json(datetime.now(),new_position))))

    ### Also log the dat into a file
    log_data_file(filename, str(create_data_csv(datetime.now(), new_position)))
```

## Testing

Testing is very important in building up a good application. It helps and ensures that the application is working fine as per the expectation to have the gulf of evaluation. We also did testing at different stages in our development process and faced different issues which we were able to resolve successfully.

### Functional testing:

In this phase we build up our small modules and tested them at regular intervals but with small amount of data. The functional testing helped us to understand that our code is behaving as per our expectation.

**Reading data from serial port:** We used cat command to check whether we are receiving the data from the GPS or not, and whether raspberry pi is able to read the data from the I/O port or not. This worked fine and we moved ahead.

**Parsing data using serial and pynmea2 libraries:** We read the data from the serial port using serial and checked for the gps fix data. This was then pared using the pynmea2 for getting the latitude and longitude values. This also worked fine and we were able to continuously read data for small amount of time and get the new latitude and longitude values from the gps.

**Geo-fencing:** For geofencing, we used haversine function to check the displacement of the device. The haversine function returns the azimuthal distance between two latitude, longitude pairs. We passed the start position and the new position to the function and calculated the result in meters.

This also worked fine and based on this we checked whether our condition for a threshold movement is met or not.

**Pushing data to MQTT client:** In this step, we used the AWSIoTSDKpython library to communicate with the AWS service. As explained in earlier steps we used the credentials required to communicate and created a communication between the Raspberry Pi and AWS and published the data to a topic. This step also worked successfully without any issues and we were able to subscribe the topic on AWS and receive the data on AWS.

## Integration Testing

This is the most important steps in the development of the product or application and majority of the issues are unearthed during this phase. We also faced a blocking issue in our application development as well.

We were able to read the data from the serial port with the help of the GPS module attached to the Raspberry Pi. The parsing the data to latitude and longitude values and then applying the geo-fencing logic also worked. The connection with the AWS using the MQTT library was also working fine, but it didn't work for a large period of time. We tried to work on those issue but we failed multiple times to get the root cause of it. Below are the issues that we encountered.

1. The GPS didn't write data to the port after a certain amount of time. This was random as it may happen when the application has read 10-15 lines of the data or more.
2. The GPS doesn't writes the data to port, it only had CRLF written to the port, which was read by the serial.Serial() function and raised the exception.

We took a different approach for finding the root cause and completion of the project.

1. Read the data from GPS using the cat command and write it to a file and see whether it terminates or not.
2. Try to handle the exception in the case the CRLF was written by the GPS on the serial port.

In the first step, we were able to read the GPS data but it also get terminates after writing some amount of data. We looked for the errors and found that an OS process [serial-](#)

[getty@ttyAMA0.servic](#) also reads the port and gets started after every reboot. We disabled this process and wrote a logon script which stops it and disabled this service.

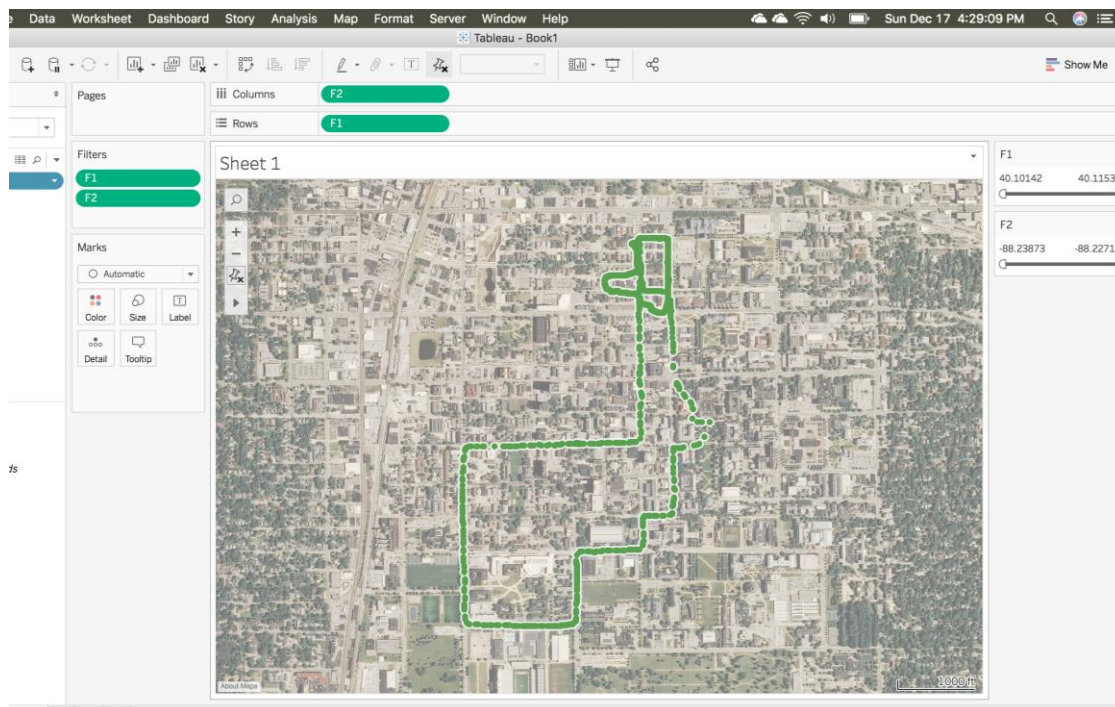
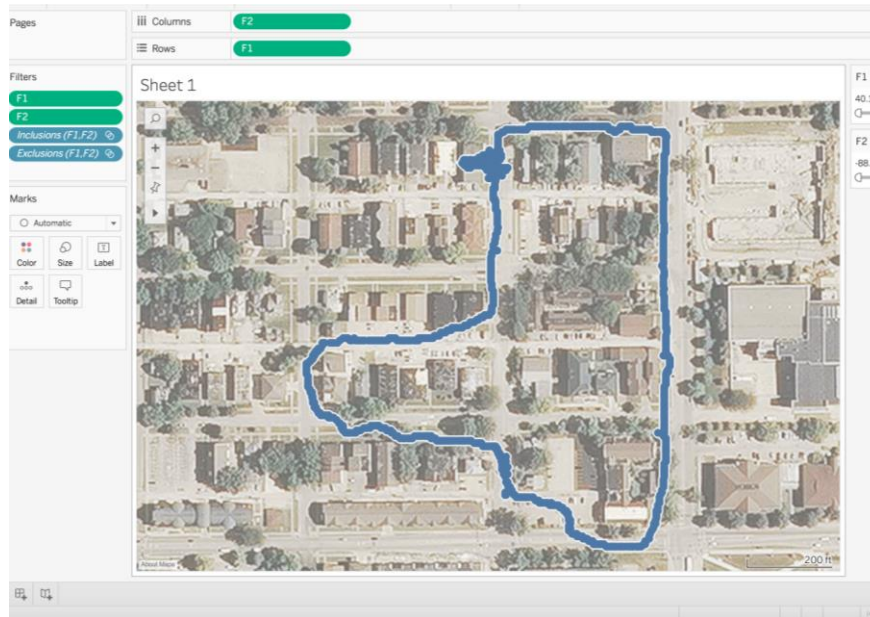
But we still faced the second issue and our application didn't work. Hence, we collected the GPS data without sending it to the AWS and stored it in the file. Since, the data was in raw format, we used our code to parse it by reading the file and then store the values and send it to AWS, which worked successfully.

For the second issue, we again looked for the errors and found that a crude method is disable the condition in the serial code which raises this exception "'device reports readiness to read but returned no data (device disconnected?)' ". In this case, the exception was not raised and our code didn't parses this information as it didn't had the GPS fix data. Hence, we modified the code and it worked fine. But due to the time constraint we didn't get it tested to generate a large amount of data for visualization.



## Visualization

We have captured 2 different data set from the module, the first data is collected while walking. It was collected to test the working efficiency of the device with respect to accuracy of the positioning.



The second data was collected while riding the bicycle. For the second test we measured with respect to accuracy by changing the latency for the data collected. We took a round around the University Campus by connected the device to the Mobile Hotspot.

## Conclusion

We were able to complete the working module of a GPS tracking device that is completely automated and works in accordance to the goal we have set in the main objectives. It is capable of locking itself with the satellite co-ordinates and notify user for the live geo-location. The Data is collected on the AWS services and can be further used to analyze the data. We can also added the functionality of Geo-fencing which will notify the end user if the tracking device moves out of the user defined/customized geometric diameter from the center point.

The Tracking device that we build can be optimized in both performance as well as with size. We can remove most of the IO – HDMI and USB ports, as well as add small battery to accompany with its flexibility. The size can be further optimized based on following industrial design principles in integrating most of the parts on a compact integrated circuit. It could be attached to the collar leash of Cats and Dogs, or we can simply be put it into the bags of the children's or can be attached within the bicycles or cars for additional security.

## References

### GitHub Repository link:

- [https://github.com/pratikshrivastava/IOT\\_FALL17\\_PROJECT](https://github.com/pratikshrivastava/IOT_FALL17_PROJECT)

### References:

- <http://docs.aws.amazon.com/iot/latest/developerguide/iot-sdk-setup.html>
- <http://docs.aws.amazon.com/iot/latest/developerguide/iot-ddb-rule.html>
- <https://punchthrough.com/blog/posts/setting-up-aws-iot-with-lightblue-cloud-connect>
- <http://techblog.calvinboey.com/raspberrypi-aws-iot-python/>
- <http://www.gpsinformation.org/dale/nmea.htm>
- <https://github.com/aws/aws-iot-device-sdk-python#id3>
- <https://raspberrypi.stackexchange.com/questions/45570/how-do-i-make-serial-work-on-the-raspberry-pi3/45571#45571>
- <https://www.linux.com/news/who-needs-internet-things>
- <http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx>
- <http://cordis.europa.eu/fp7/ict/enet/documents/publications/iot-between-the-internet-revolution.pdf>
- <http://www.gartner.com/newsroom/id/3165317>
- [https://en.wikipedia.org/wiki/Internet\\_of\\_things](https://en.wikipedia.org/wiki/Internet_of_things)