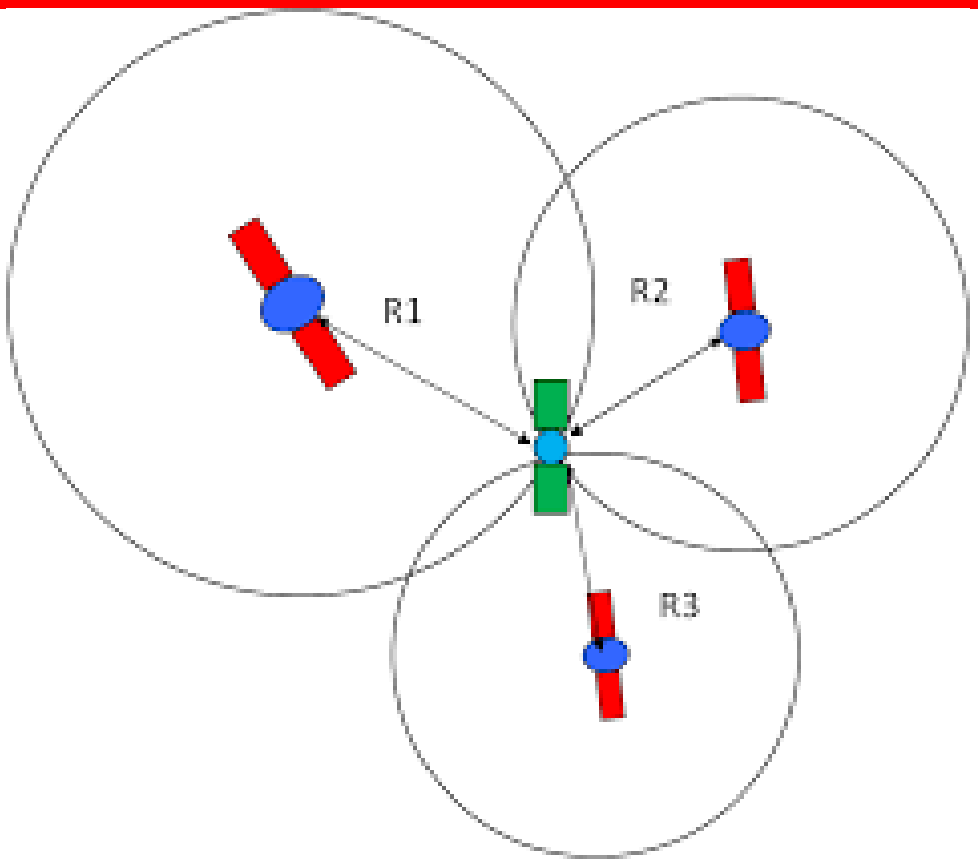
Indian Institute of Technology Madras

Centre For Innovation

Wi-Fi Positioning System with ESP8266



Team Members

Sai Chandrahaas Vadali
Chandrashekhar Dhulipala
Raj Kiriti Velichety
Tarun Annapareddy

Table of Contents

Bibliography

1. Abstract

This project uses WiFi signal strength from various fixed nodes in order to calculate the user's position in real-time. The advantage over GPS is that this can be used for short-range positioning, like in a building/mall. To serve as Wi-Fi nodes, we use ESP8266 modules, a family of Wi-Fi chips with a microcontroller unit inbuilt. One ESP module is used as a station, which measures the signal strength from the other ESPs (Access points). This station ESP uploads the retrieved data on to a web page.
Then we download the data on a computer, process it and pinpoint the location of the station ESP, relative to the access points.
Calibration of signal strength vs distance, Calculations on the data and plotting of the final location are done using Python.

2. Introduction

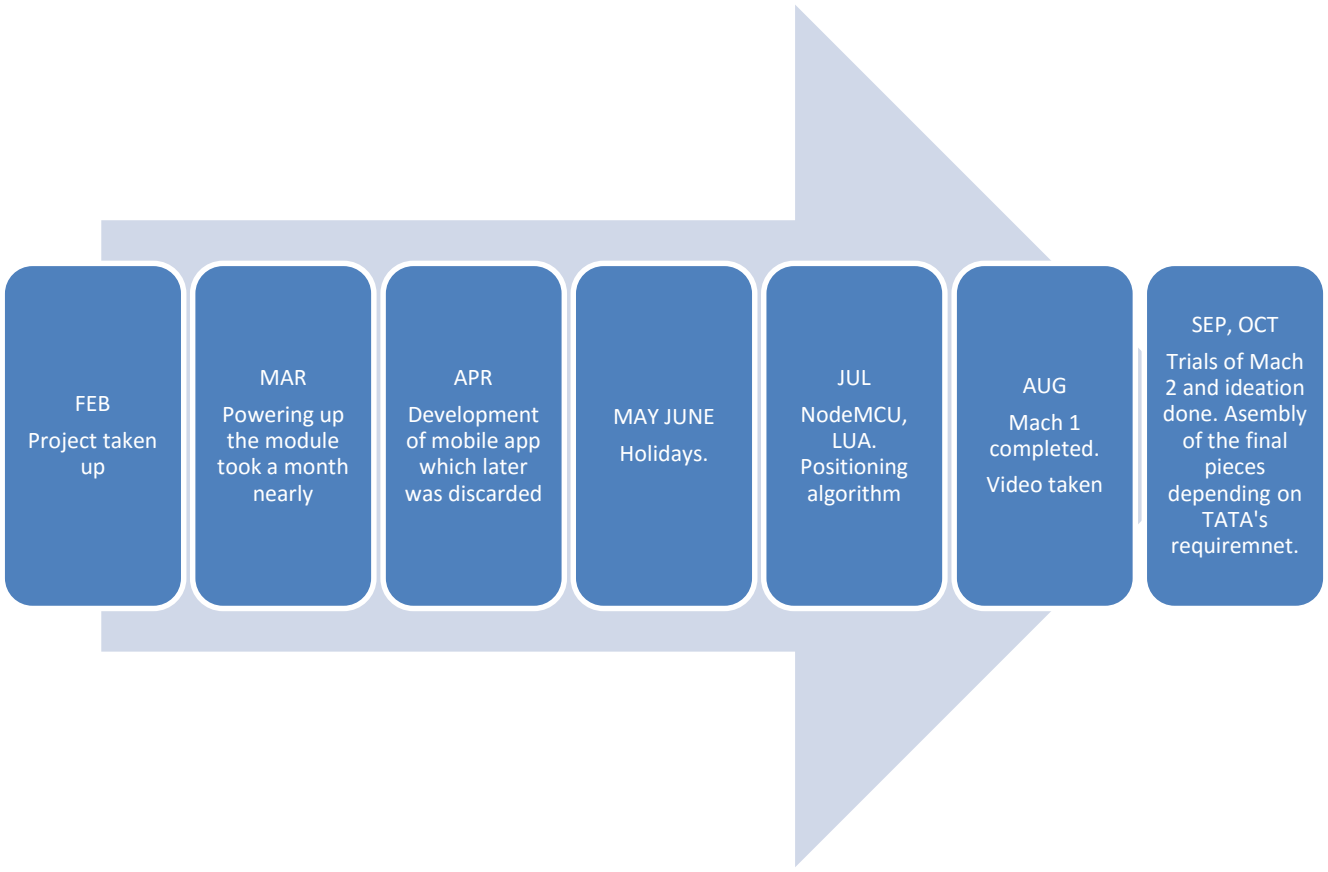
In February 2016, the Electronics club released a list of projects for us to choose from. We found this particular problem to be quite challenging. We would be getting an opportunity to work with ESP8266 modules, which had already created a storm in the IOT world but were relatively new in the Indian markets.

We also concluded that this project would have considerable real-life utility. If a huge building is mapped with ESPs and our positioning system is implemented, the user need not worry about navigating long corridors and confusing stairways/pathways anymore.

3. Research

The first few days were dedicated to finding out about any similar projects. However, we could not find anything related to our project. People have done all kinds of IoT projects using ESP8266 but none seem to have tried something like this. This motivated us greatly to work on this project without giving up. One point to be noted is that – Wi-Fi positioning had been **done** before! The novelty of our idea stems from the fact that we used an ESP8266 to do the positioning. One can find a Wikipedia page on Wi-Fi positioning algorithms. We chose the simplest and least accurate one - RSSI based positioning. In general, one can expect an accuracy of 2 – 4m. We tried our best to achieve a better accuracy. The best accuracy we achieved was 0.4m and an average variation anywhere between 1-2m. One point we observed was that as the size of the field where positioning is done was increased, the accuracy was better.

4. Project Timeline



5. Overview of Tasks

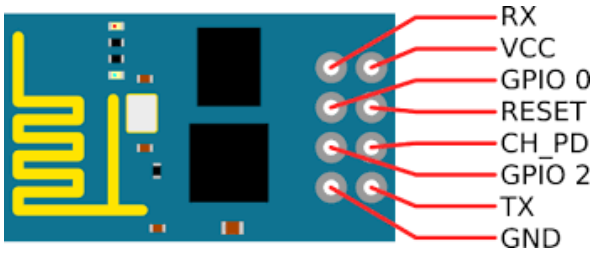
The project revolves around the protagonist – ESP8266. The ESP8266 is a Wi-Fi transceiver, i.e., it can act as web client or a web server. The first step is to get the module up and running. There are many versions of this module. We used the ESP8266–01 version. The ESP8266 uses 3.3V and around 200mA current. An appropriate IC must be used to regulate the voltage. The Arduino's 3.3V output might not provide enough current. If you want to use an Arduino to power, it up then use a proper voltage regulator like LM1117. Once you are able to power it up you have to decide whether you want to use an Arduino and Arduino IDE to code or directly code the ESP8266 using NodeMCU Lua IDE. We started off with Arduino IDE as it was the simpler choice. We later rejected it because we felt it was redundant to have an Arduino connected all the time when the ESP8266 itself could handle all the computations by itself. So, we started working with an IDE called ESPlorer. For the ESPlorer to be able to code, an old programming language called Lua has to be installed. ESPlorer can also be used to code in Micropython, something we have tried in development of version 2. The important aspect is the calibration of signal strength vs distance. This is a painstakingly long process as one has to take close to 8000 values of the haphazardly varying signal strengths at different places. The easier part is the curve-fitting. We chose a least squares power law fit to the obtained graph. Other types of curve fitting can be looked up in Wolfram's site. Now comes the positioning

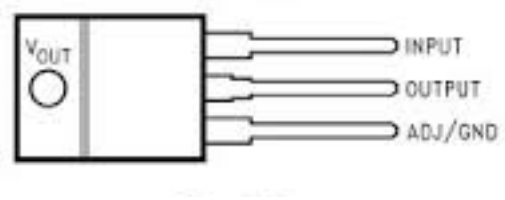
algorithm. Initially we tried a very basic algorithm which assumed a very ideal behavior of the signal strength at a given place. But the erratic behavior of the signal strength at a given place forced to us to seek other options. After some research and modifications, we were able to come up with a pretty decent algorithm. Note that the ESP8266 is capable enough to do these complex calculations. However, we chose to offload the calculations onto a python program. We were mainly concerned with 2D positioning. This requires 3 nodes for tracking the target ESP8266. So, we setup 3 access points in the nodes and a client(station) in the target. The client measures the signal strength of each of the nodes and hosts it in a webpage whose IP address is fixed by us. Note that the client has to connect to another access point (like CFI WIFI). Now, we have a python program which reads data from this page and performs the necessary calculations to display the result. The 3D positioning is similar to the 2D one, except it needs a minimum of 4 target nodes. The final part is soldering the circuit onto a GCB. Note that ESP8266 is not breadboard friendly.

6. Design History & Overview

Mach 1

This prototype is not user friendly. One has to change certain parameters all the time in the code and then run the python code. This would be difficult for someone who hasn't worked in the project. In the above section we have explained how to power up the module and to choose a programming environment. Once this is done, we have a choice of either working with the in-built AT commands or flashing the NodeMCU firmware into it. Here is the pin diagram of an ESP8266–01 and the IC LM1117





To program it without an Arduino we have to use a USB to Serial (TTL level) converter. We used a FTDI FT232RL. Make sure that the FTDI is in 3.3V mode. To flash the NodeMCU firmware we have to first connect the ESP8266 in flash mode. The connections are as follows

From FTDI	To ESP8266
RX	TX
TX	RX
5V to LM1117 input	LM1117 output to VCC
	LM1117 output to CH_PD

GND to LM1117 GND	LM1117 GND to GND
	LM1117 GND to GPIO 0

To flash the firmware into the ESP8266 we used a tool called NodeMCU-flasher-master. We can also set the default baud rate for serial communication. Once flashing is done change the GPIO 0 pin from LOW to HIGH. This puts the module in programmable mode. As mentioned before, we used a programming language called Lua. We setup the three nodes to be access points named Origin, X-axis and Y-axis. The client side or target has a code which does two things – firstly, measure the signal strength of the three nodes, concatenate them as strings and secondly, host them in a webpage on a given IP address in the local network. This has a few limitations. One can send a mere 1460 bits of information from the client at a time. We were initially sending 9 bits of data at a time and then refreshing the page. However, in the python code we could not get it to read from the refreshed page. The code used to try and read the "next" 9 bits from the page. Since, we were refreshing the same 9 bits and there was no "next" 9 bits, we had to change our approach. We concatenated the signal strengths and then used the code to read a string of data. But after around 140 times of positioning, the ESP8266 would abruptly end the execution of the code because the 1460 limit was exceeded. This was NOT solved in Mach 1. Now coming to the positioning algorithm – we used a trilateration algorithm which was roughly based on a concept called "Projection onto Convex Sets". With some help we were able to extend this concept from 2 to 3 circles. This method provides us to find the point of intersection of two circles. If the 3 circles are not intersecting, it gives a

The limitations of Mach 1 –

- The ESP8266 module which would be used as the target was still connected to the laptop, as every time we restarted he module, the setting-up-of-client code wouldn't work. Once the code was uploaded, we could then disconnect it and power it externally using a power bank.
- The 1460 bits limitation.
- The accuracy was not that good when we tested in short ranges like 2-3m. We later realized that testing in such short field was not the point of this project.

Mach 2 (currently in last stages of development)

This version aimed to tackle the problems mentioned above. We found out about the deep sleep mode of the ESP8266. This would serve two purposes. Firstly, we needn't disconnect the power supply when we did not use it. We could just switch it to the deep sleep mode. Note that the deep sleep works only for 5-10 minutes. We need to write a bit of code connecting the GPIO 2 and adding a SPST. The second advantage is that we can now make the ID card version to last for many

days. The ID card module will be explained later. To tackle the 1460 bits limit, we found a website called Thingspeak.com which helps anyone in storing and analyzing data. However, XML or JSON in Lua was a bit difficult. This is when we wanted to shift to micropython. We have tested out the basic stuff in micropython. We started with flashing the ESP8266 with the supporting firmware. We used a software called ESPtool to achieve this. The access point code was easily available in the documentation. So was the case with the code of signal strength measurement. We are currently working on implementing the GET and POST requests from the ESP8266 so that it can upload the information to the website mentioned above. The python code remains almost same, with minor modifications such as, asking it to read it from this current website's page rather than the ESP8266 client's page.

ID CARD module

This is the one which we will be developing for the Tata Innoverse Challenge. As the name suggest, you can fix this module behind an employee's ID card and can track him in the warehouse. The changes include powering the module up with Lithium button batteries and giving the whole circuit a product like finish. The User Interface for the code on the computer would also be developed. Few minor change to the code would be done so as to read the data from their database.

7. What did we learn?

Firstly, we learnt about the wonderful chip that the ESP8266 is. At under $5, it packs a lot more features than expected of a generic microcontroller.
Like any microcontroller, it has a flash memory to store and run programs. We learnt about different ways to program the ESP chip, namely AT commands, through Arduino IDE and
through the LUA language using NodeMCU firmware.
In addition to this, it has Wi-Fi capabilities with a full TCP-IP stack. We learnt how to leverage these capabilities to measure signal strength, as well as connect to a webpage on the local network.
Secondly, we spent considerable time calibrating the ESP's signal strength with distance. We applied a couple of regression models and learnt how to get a best-fit curve for a given data set.
Then we moved on to the positioning algorithm. We studied different trilateration methods and analysed the pitfalls of each one. We finally settled on the method "Projection onto convex sets".
Last but not the least, we learnt how to code efficiently in Python and LUA to achieve the tasks mentioned above.

8. Images, Video of the project in action

Here is the YouTube link to the project

https://www.youtube.com/watch?v=ET937p3KhgM

9. Applications

- The obvious application is tracking things or people indoors. For the best results we can use multiple nodes in a warehouse each covering a field, say of 30*30 sq m area. The project is amazing as it solves this problem at throwaway cost.
- We can apply this tracking to make bots or drones which automatically navigate indoors when given the coordinates of the initial and final positions.

10. Acknowledgements

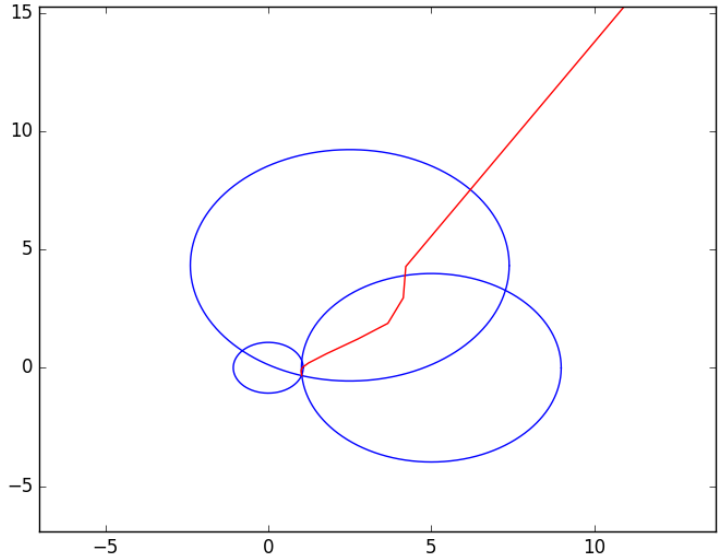
Our team would love to express our gratitude to the Electronics Club CFI for helping us all along. Special mentions to Hari Venkat Kiran and Aboobacker Siddique for supporting us through thick and thin.

Appendix A – Data Sheets & Calculations

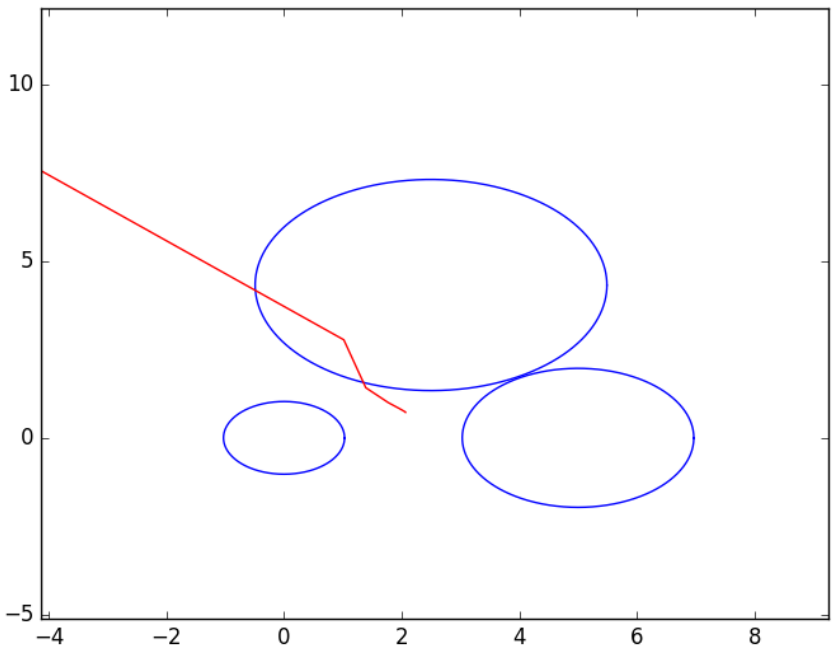
Here is the link to the datasheet of ESP8266-01

http://www.puntoflotante.net/ESP-8266-ESP-01.pdf

Here is picture of the output of the positioning algorithm. This shows how the algorithm calculates the point of intersection of the circles.



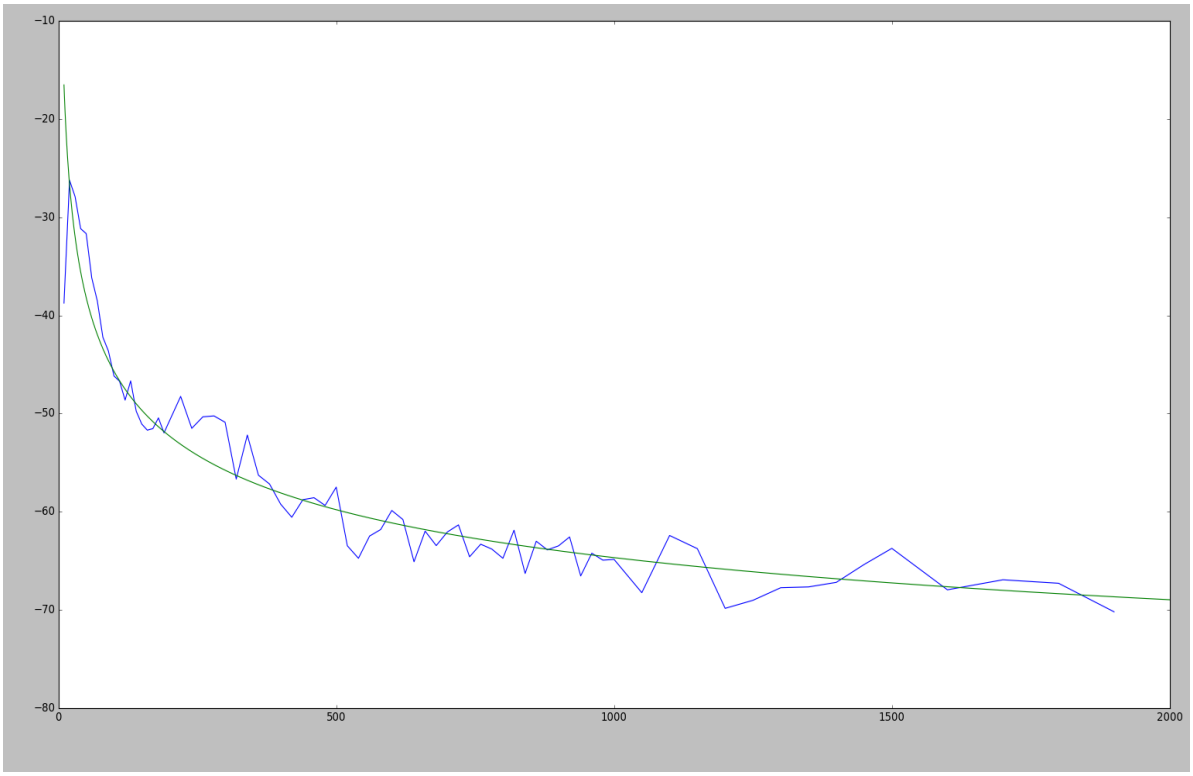
For circles which are not intersecting the figure is shown below. We observe that the point shown by the algorithm is on the locus of those points which we would get if the radii of the circle were kept on increasing till they intersect. Hence, we get a good approximation in real time scenarios even when the circles are not intersecting.



The curve fitting of the obtained values of signal strength vs distance is one of the major calculations involved. Here is the link for the method we applied to achieve this

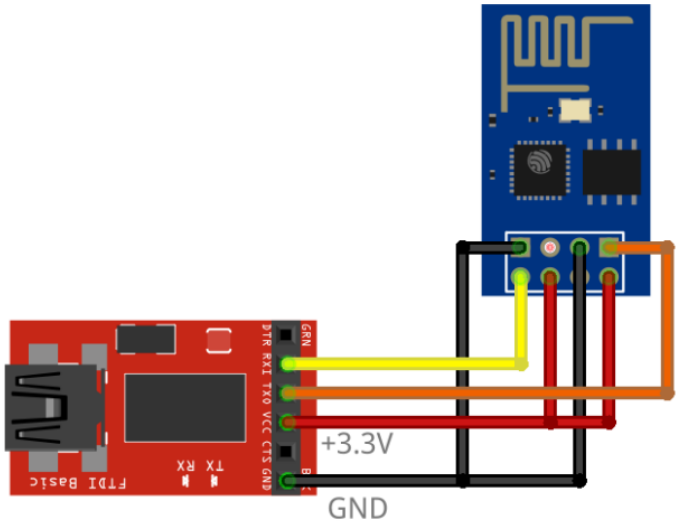
http://mathworld.wolfram.com/LeastSquaresFittingPowerLaw.html

Here is the image of the curve fitting.



Appendix B – Electrical Schematics

Here is the circuit to operate ESP8266 in flash mode



Note that one has to change GPIO 0 (which is connected to Ground here) from GND to 3.3V output to put the ESP8266 in programmable mode.

Appendix C - Parts Drawings

Not applicable.

Appendix D – Source Code

Here is the python code for the positioning and plotting

```
import math
import cmath
import numpy as np
import sys
import random
import matplotlib.pyplot as plt
import time


import socket
```

```
#while True:
#     try:
          #time.sleep(1)



          #time.sleep(1)
          #mysock.close()
          #break
#     except:
#          print('Oops!')

data = ''
i = 0
#fig = plt.figure()
ex = 0
plt.ion()
while True:
     mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
     mysock.connect(('192.168.0.125',80))
     s = 'GET http://192.168.0.125/ HTTP/1.0\n\n'
     b = bytearray()
     b.extend(map(ord,s))
     mysock.send(b)
     #plt.axis([-500,500,-500,500])
     ex = ex + 1
     if ex == 1 or ex == 2:
          continue
     data = mysock.recv(9)
```

```
        data = data.decode('ascii')
        #print (data, type(data))



#d12 = double(raw_input("Enter distance between 1 and 2:"))
#d23 = double(raw_input("Enter distance between 2 and 3:"))
#d13 = double(raw_input("Enter distance between 3 and 1:"))

#cosAngle = ((d12*d12 + d13*d13 - d23*d23)/(2*d12*d13))
#x = d23*cosAngle
#f = math.acos(cosAngle)
#sinAngle = math.sin(f)
#y = d23*sinAngle
        center1 = complex(0., 0.)
        center2 = complex(450., 0.)
        center3 = complex(0., 650.)



        l = data.split("-")
        #l2 = l[1][0:2]
        #l3 = l[2][0:2]
        #l4 = l[3][0:2]
        #cen1 = l[1][2:]
        #cen2 = l[2][2:]
        #cen3 = l[3][2:]
        #print (data)
        l = [int(l[1])*(-1),int(l[2])*(-1),int(l[3])*(-1)]
        #a = (4.022246-math.log(100+l[0]))/0.025231
        #b = (4.022246-math.log(100+l[1]))/0.025231
```

```
#c = (4.022246-math.log(100+l[2]))/0.025231
#if cen1 == 'Barca2':
a = np.power(((100+l[0])/128.363072),(1/(-0.18681484)))
b = np.power(((100+l[1])/128.363072),(1/(-0.18681484)))
c = np.power(((100+l[2])/128.363072),(1/(-0.18681484)))



r1 = a
r2 = b
r3 = c

print(r1,r2,r3)

#if sys.argv[4] == "y":
r1 += random.gauss(0., 0.1)
r2 += random.gauss(0., 0.1)
r3 += random.gauss(0., 0.1)

#print (r1, r2, r3)

guess_list = []
guess = complex(5.,5.)
lim = int(max(abs(guess.real), abs(guess.imag))) + 5
#print ("lim = " + str(lim))
guess_list.append((guess.real, guess.imag))
#print ("guess = " + str(guess))

for i in range(100):
    proj1 = center1 + ((guess-center1) * r1 / abs(guess-center1))
    proj2 = center2 + ((guess-center2) * r2 / abs(guess-center2))
```

```
            proj3 = center3 + ((guess-center3) * r3 / abs(guess-center3))
            guess = (proj1 + proj2 + proj3) / 3.
            guess_list.append((guess.real, guess.imag))

        print (guess)

        theta = [i*cmath.pi*2/1000. for i in range(1000)]
        circ_complex1 = [center1 + cmath.rect(r1, ang) for ang in theta]
        circ_complex2 = [center2 + cmath.rect(r2, ang) for ang in theta]
        circ_complex3 = [center3 + cmath.rect(r3, ang) for ang in theta]
        circ1 = map(lambda x: (x.real, x.imag), circ_complex1)
        circ2 = map(lambda x: (x.real, x.imag), circ_complex2)
        circ3 = map(lambda x: (x.real, x.imag), circ_complex3)

        plt.scatter(center1.real,center1.imag)
        plt.scatter(center2.real,center2.imag)
        plt.scatter(center3.real,center3.imag)
        plt.scatter(guess.real,guess.imag)
        #plt.plot(*zip(*circ1), color='b')
        #plt.plot(*zip(*circ2), color='b')
        #plt.plot(*zip(*circ3), color='b')
        #plt.plot(*zip(*guess_list), color='r')
        #plt.axis([-lim, lim, -lim, lim])
        #fig.canvas.draw_idle()
        plt.draw()
        plt.pause(1)

        plt.clf()
    #s1 = data[0:2]
    #s2 = data[2:4]
```

```
#s3 = data[5:7]

    #time.sleep(6)

    mysock.close()
```

Appendix E – Cost Analysis

Here is an estimate for mach 1 and the ID card module.

Component	Cost
ESP8266	Around 240 (varying) per piece
IC LM1117	40 – 70 per piece
Breadboard, gcb, bergs and wires	180-200
FTDI FT232RL (need not be bought on the user side)	Around 300 per piece
USB cable	40 per piece
Lithium Button batteries	12-15 per 1 piece
Battery Holders	30 per piece
Total amount for a single ID card module	Around 350 (please don't add up the costs here!)

Bibliography

- Here is the link for the basic setup and installation.

http://benlo.com/esp8266/esp8266QuickStart.html

- The micropython introduction and installation

https://docs.micropython.org/en/latest/esp8266/esp8266/quickref.html

- The research paper from which we took the inspiration for the positioning algorithm

http://publications.lib.chalmers.se/records/fulltext/138669.pdf

Positioning Algorithms for Wireless Sensor Networks

Mohammad Reza Gholami

Communication Systems Group Department of Signals and Systems

Chalmers University of Technology

Gothenburg, Sweden 2011.