



**Design of a discreet building access management system using wireless indoor location tracking and fuzzy logic**

**By**

**Aisling Lee**  
**BEng, BEngTech, MIEI, MIET**  
[https://github.com/ashification/ee580\\_meng](https://github.com/ashification/ee580_meng)

This Report is submitted in partial fulfilment of the requirements of the master's degree in Electronic and Computer Engineering (MECE) of Dublin City University

22<sup>nd</sup> August 2022

Supervisor: Dr. Derek Molloy

## Declaration

I understand that the University regards breaches of academic integrity and plagiarism as grave and serious.

I have read and understood the DCU Academic Integrity and Plagiarism Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach this policy.

I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the sources cited are identified in the assignment references.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

I have used the DCU library referencing guidelines (available at: <http://www.library.dcu.ie/LibraryGuides/Citing&ReferencingGuide/player.html>) and/or the appropriate referencing system recommended in the assignment guidelines and/or programme documentation.

By signing this form or by submitting this material online I confirm that this assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

By signing this form or by submitting material for assessment online I confirm that I have read and understood DCU Academic Integrity and Plagiarism Policy (available at: <http://www.dcu.ie/registry/examinations/index.shtml>).

Name: 

Date: 22/08/2022

## Design and Implementation

Obj 3 - Implement and test a physical installation of the application and its architecture.

Functional and technical design

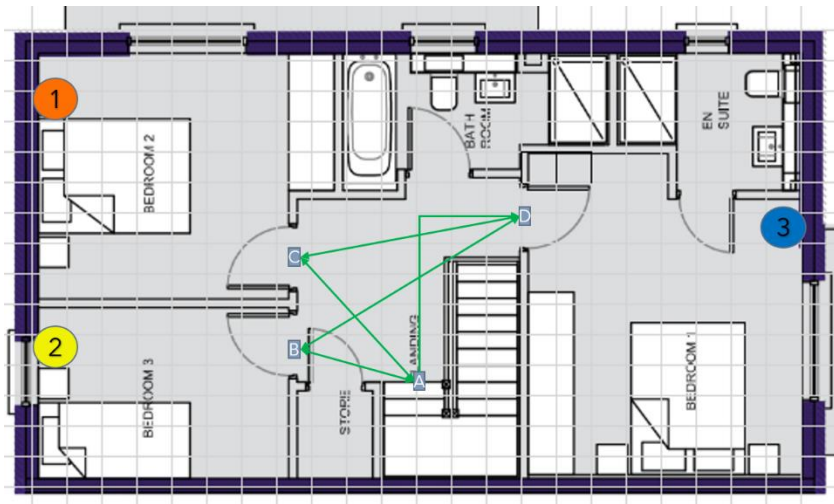
Methd 4 - Implement a wireless transceiver design and integrate with system to detect and interact with end users.

Methd 5 - Build a door locking mechanism system with fail-safes for trigger testing purposes.

## Hardware

### Decawave network

The Decawave hardware was set up within the given test environment as follows:



### Anchor nodes

The anchor nodes were positioned at locations giving the doorways line of sight at roughly the same height off the ground. Node 4 imaged here correlates to node 3 in the image above. The device originally defined as node 3 had to be repurposed which will be covered later on.



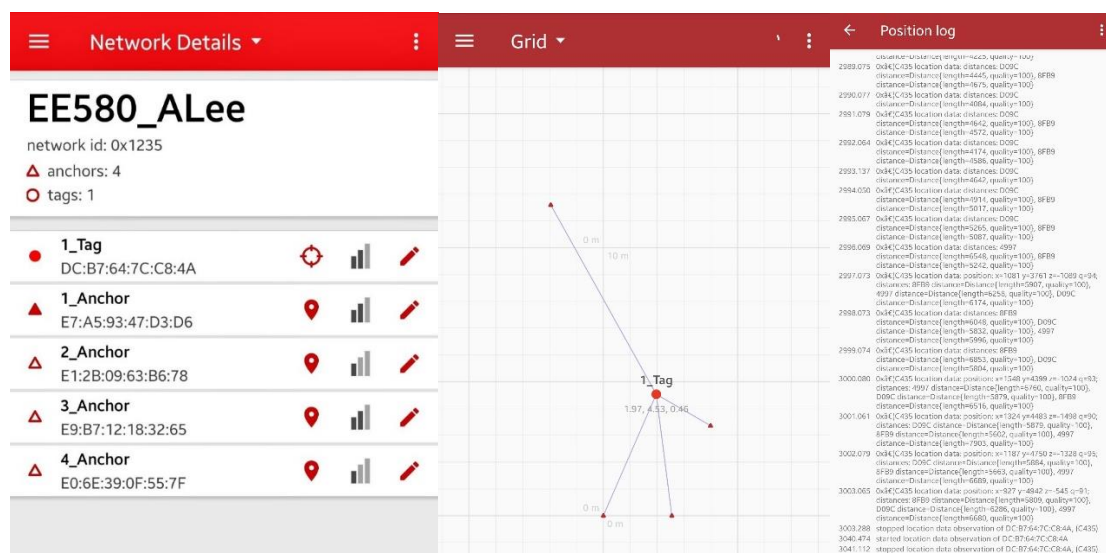
### Tag node

The tag node was initially connected to an external battery pack but found that the power consumption of it to be so low that the battery pack models tested with would switch off and thus interfere with testing, as a result an appropriate battery model was used.



### Android App

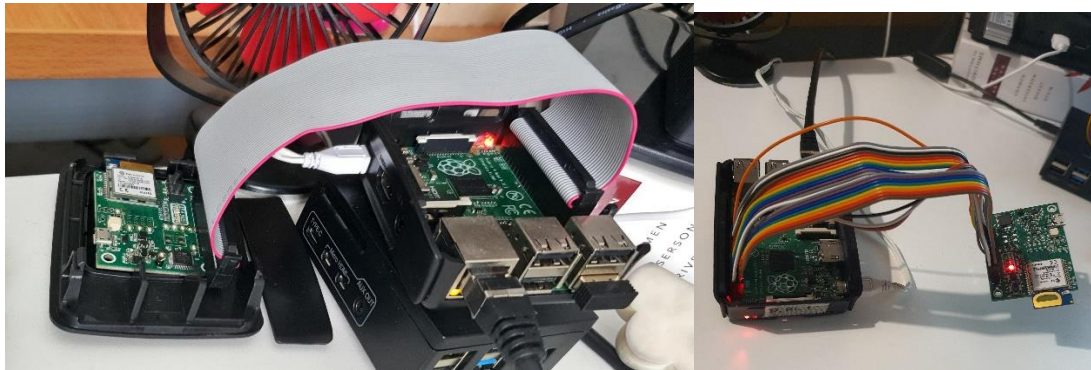
Once the devices were all powered on the android app installed on the phone (the apk needed to be built from the provided project files) below shows the devices after having been added to the network.



Once the devices were connected, I was able to start reviewing the data produced from the tags. It was noted at this point the devices were communicating via Bluetooth to the app and this was notably slow with significant delay. It was also noted that moving to the floor below provided better resolution as the floor acted as a filter. This is common with RSSI signals to alternate values within a wide range even when devices are not moving.

*Bridge node*

The next stage was to connect up the Bridge node. The image on the left is the initial connection. It was noted though a lengthy investigation and testing that the UART connectivity was failing and the anchor number 3 was re-purposed to become the new bridge node ( see image on the right) included in the trouble shooting was ruling out cable connector issues hence the individual jumper cables on the right.



The bridge device was formatted as follows (see next screen shot)

```
Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on Aug 13 2017, 15:25:34.
Port ttyACM0, 19:30:37

Press CTRL-A Z for help on special keys

DWM1001 TWR Real Time Location System

Copyright : 2016-2019 LEAPS and Decawave
License : Please visit https://decawave.com/dwm1001_license
Compiled : Jun 7 2019 18:00:03

Help : ? or help

dwm>
dwm>
dwm> si
[000175.920 INF] sys: fw2 fw_ver=x01030001 cfg_ver=x00010700
[000175.920 INF] uwb0: panid=x1235 addr=xDECA22304422C21E
[000175.930 INF] mode: bn (act,-)
[000175.930 INF] uwbmac: connected
[000175.930 INF] enc: off
[000175.940 INF] ble: addr=E9:B7:12:18:32:65
dwm> nis 0x1235
nis: ok
dwm> nmb

DWM1001 TWR Real Time Location System

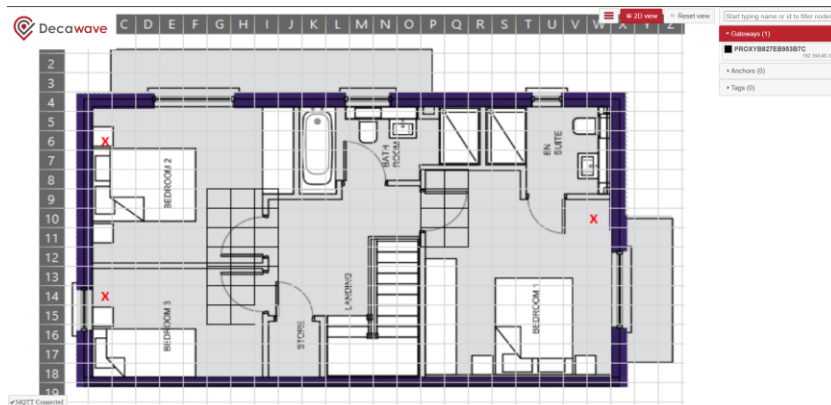
Copyright : 2016-2019 LEAPS and Decawave
License : Please visit https://decawave.com/dwm1001_license
Compiled : Jun 7 2019 18:00:03

Help : ? or help

dwm> si
[000011.360 INF] sys: fw2 fw_ver=x01030001 cfg_ver=x00010700
[000011.360 INF] uwb0: panid=x1235 addr=xDECA22304422C21E
[000011.370 INF] mode: bn (act,-)
[000011.370 INF] uwbmac: connected
[000011.370 INF] enc: off
[000011.380 INF] ble: addr=E9:B7:12:18:32:65
dwm> la
[000376.410 INF] AN: cnt=2 seq=x04
[000376.410 INF] 0) id=DECA55C596B48FB9 seat=0 seems=167 rssi=-80
[000376.420 INF] 1) id=DECA6C8D4D2D09C seat=1 seems=161 rssi=-83
[000376.420 INF]
```

### Gateway

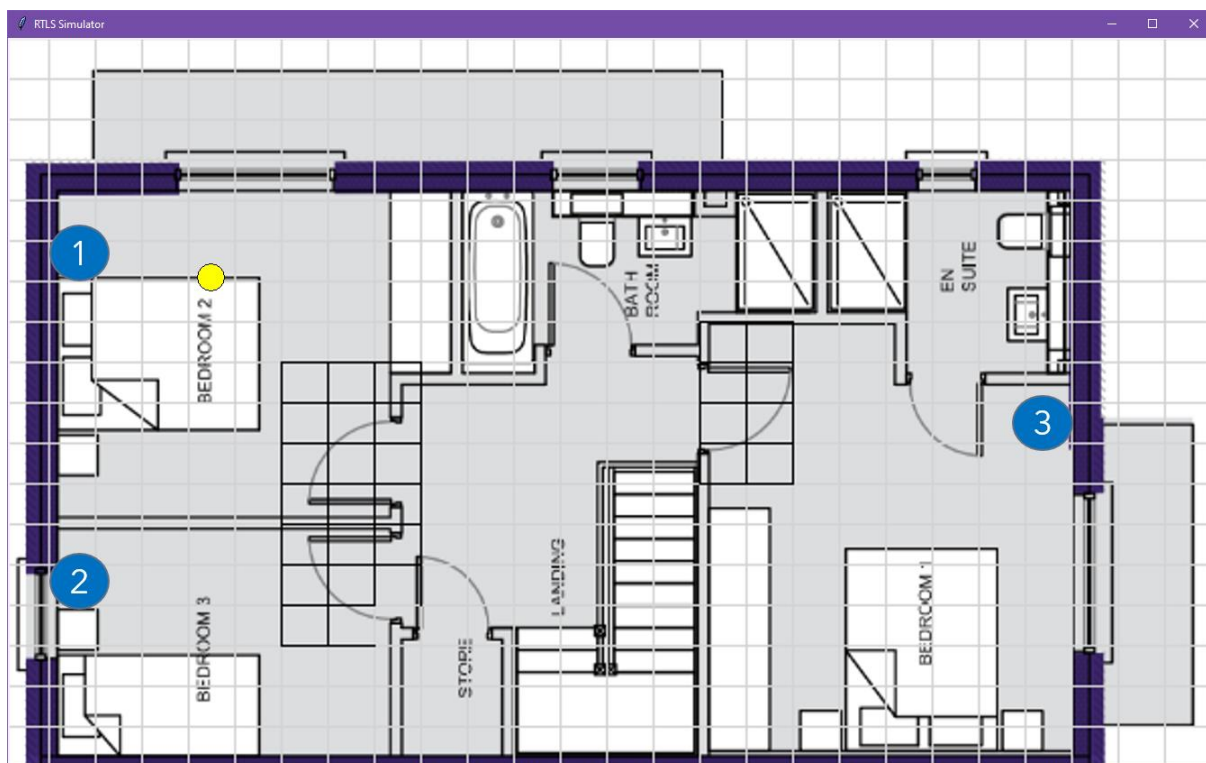
Once the bridge nodes were connected hitting the RPI gateway nodes IP brought us to the DRTL5 web manager as seen below. The RPI is represented as the proxy device however as seen below the gateway device is not present. I was unable to achieve connectivity between the RPI and the bridge node and thus built a simple simulator.



## Software

### UWB Node Simulator

After the failing of the UWB network, I devised the below captured RTL5 simulator. The tag (see yellow dot) was controlled by the arrow keys on a key board and the application generated the XY data. However, the rest of the architecture is still valid should the simulator be removed and replaced with an operational USB network



## Broker

The Broker code used MQTT Paho and the GUI is MQTT.fx

### Installing MQTT.fx

Adapted from here:

<https://console.hivemq.cloud/clients/mqttx?uuid=3a7064825d8d4ce68f4d17cdf59b41e1>

### Python code for broker

Adapted from here:

<https://console.hivemq.cloud/clients/python-paho?uuid=3a7064825d8d4ce68f4d17cdf59b41e1>

### Enabling paho packages to run

Err: ImportError: No module named paho.mqtt.client

Soln from here :

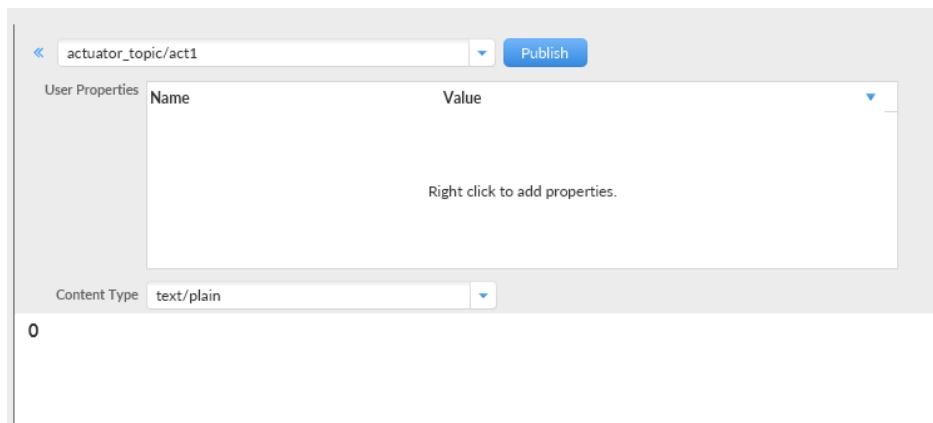
<https://stackoverflow.com/questions/48752469/cannot-run-paho-mqtt-client-importerror-no-module-named-paho-mqtt-client>

The problem is that the library "paho" has been installed (for default) in the folder "/home/pi/.local/lib/python2.7/site-packages" but "sudo python" search this library in the folder "/usr/local/lib/python2.7/dist-packages". I have solved with one link:

```
cd /usr/lib/python2.7/dist-packages
```

```
sudo ln -s /home/pi/.local/lib/python2.7/site-packages/paho
```

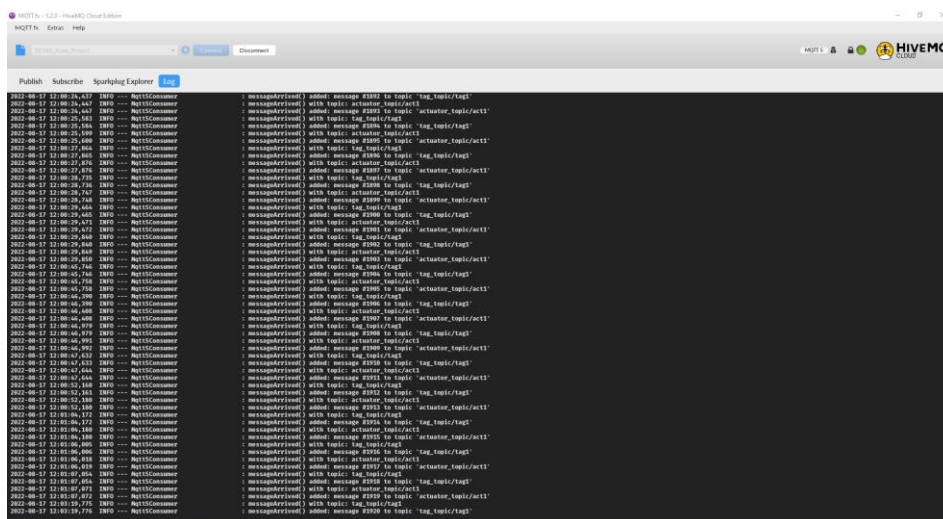
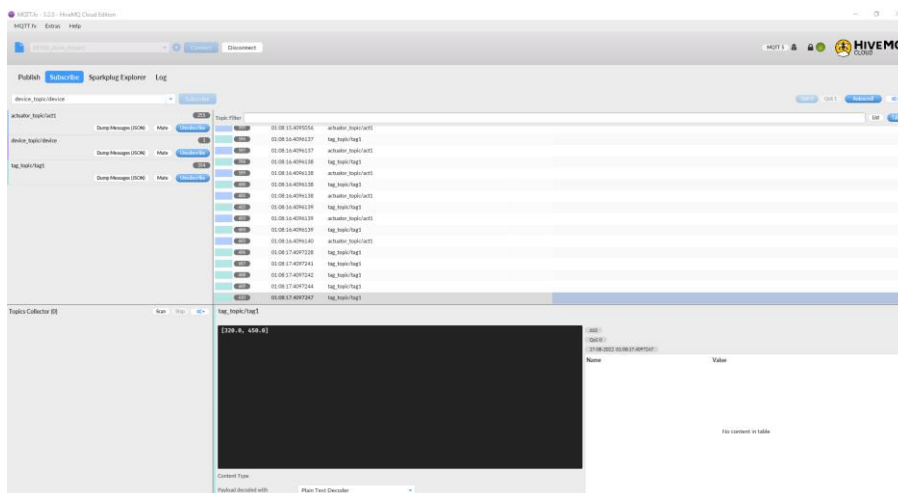
## MQTT.fx operational



The image shows the MQTT.fx web interface for publishing a message. At the top, there is a dropdown menu set to 'actuator\_topic/act1' and a blue 'Publish' button. Below this is a 'User Properties' section with a table that has two columns: 'Name' and 'Value'. The table is currently empty, with a message 'Right click to add properties.' in the center. At the bottom, there is a 'Content Type' dropdown menu set to 'text/plain'. Below the form, the number '0' is displayed.

```
pi@RPIGCIOT:~/Documents/EE580_Actuator $ sudo python actuator_script.py
CONNACK received with code Success.
Subscribed: 1 [<paho.mqtt.reasoncodes.ReasonCodes instance at 0xb668cc10>]
mid: 2
actuator_topic/act3 0 1
1
Turn ON Blue LED
actuator_topic/act2 0 1
1
Turn ON Yellow LED
actuator_topic/act2 0 1
1
Turn ON Yellow LED
actuator_topic/act1 0 1
1
Turn ON Orange LED
actuator_topic/act3 0 1
1
Turn ON Blue LED
actuator_topic/act3 0 1
1
Turn ON Blue LED
actuator_topic/act3 0 0
0
Turn Off Blue LED
actuator_topic/act2 0 0
0
Turn Off Yellow LED
actuator_topic/act1 0 0
0
Turn Off Orange LED
```





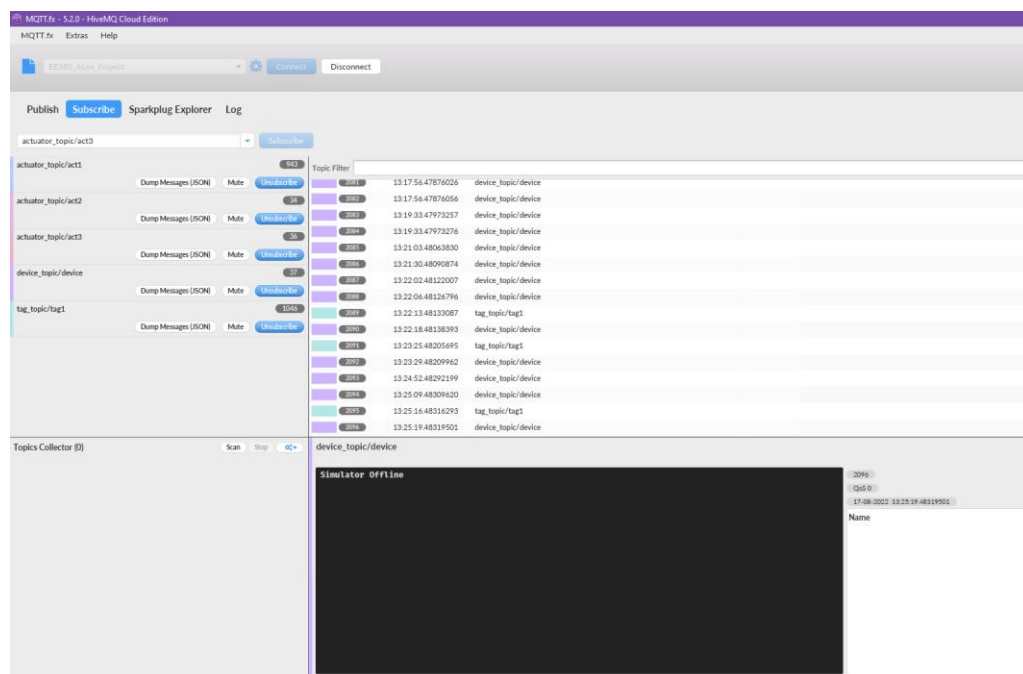
*Last will and testament*

The Last will and testament is used as part of a fail safe design with the intention if parts of the system go offline then the actuator would be notified and from there logic could be developed to lock/unlock all doors but enabling manual control over them

Adapted from here: <http://www.steves-internet-guide.com/mqtt-last-will-example/#:~:text=Python%20MQTT%20Client%20Notes%20To%20set%20the%20last,a%20code%20snippet%20taken%20for%20my%20example%20scripts.>

```
# LWT Message set up before connection est
lwt="Simulator Offline" # Last will message
lwt_post_topic="device_topic/device"
print("Setting Last will message=",lwt,"topic is",lwt_post_topic )
client.will_set(lwt_post_topic, lwt,1,retain=False)
```

demonstrated working here:



```
pi@RPiGCIOT:~/Documents/EE580_Actuator $ sudo python actuator_script.py
('Setting Last will message=', 'Actuator Offline', 'topic is', 'device_topic/device')
CONNACK received with code Success.
Subscribed: 1 [<paho.mqtt.reasoncodes.ReasonCodes instance at 0xb65e8da0>]
Subscribed: 2 [<paho.mqtt.reasoncodes.ReasonCodes instance at 0xb65e8d78>]
device_topic/device 1 Actuator Online
Actuator Online
mid: 3
device_topic/device 1 Simulator Offline
Simulator Offline
Turn OFF Red LED
```

## Databases

The purpose of the database is to store previous location data and time stamps for use in later calculations. Below is the code used to create, insert and query the database:

```
# Create the DB
db_connection = sqlite3.connect('user_location_tracking.db')
# Create Table
db_cursor.execute("""CREATE TABLE tag_location (
    tag_id text,
    x_coord int,
    y_coord int,
    time_stamp int
) """)
db_connection.commit()
db_connection.close()

def submit_todb(x1_coord, y1_coord, time1):
    # Connect to the DB
    db_connection = sqlite3.connect('user_location_tracking.db')
    # Create a Cursor
    db_cursor = db_connection.cursor()
    db_cursor.execute("INSERT INTO tag_location VALUES (:tagid, :x1_coord,
:y1_coord, :date_time)",
    {
        'tagid': "tag_1",
        'x1_coord': x1_coord,
        'y1_coord': y1_coord,
        'date_time': time1
    })

    db_connection.commit()
    db_connection.close()
```

The query works by querying the tables last entries OID value (row index number), then subtracting n from it (which is defined elsewhere in this case its 5) since they are sequentially incrementing this is the safest way to get the 5<sup>th</sup> most recent record submitted. From there we then have the OID value the function then queries for specific values (x and y co-ordinates, and time stamp) finally returning those values back from the function to be used in the calculations

```

def query_fromdb():
    # Connect to the DB
    db_connection = sqlite3.connect('user_location_tracking.db')
    # Create a Cursor
    db_cursor = db_connection.cursor()

    db_cursor.execute("SELECT oid FROM tag_location ORDER BY oid DESC ")
    print(db_cursor.fetchone())
    current_oid = db_cursor.fetchone()[0]
    print(current_oid)
    nth_oid = current_oid - n
    print(nth_oid)

    db_cursor.execute("""
        SELECT *,oid
        FROM tag_location
        WHERE oid == ?
        ORDER BY oid DESC
        """,
        (nth_oid,))
    print(db_cursor.fetchall())

    db_cursor.execute("""
        SELECT x_coord
        FROM tag_location
        WHERE oid == ?
        ORDER BY oid DESC
        """,
        (nth_oid,))
    xn_coord = int(db_cursor.fetchone()[0])

    db_cursor.execute("""
        SELECT y_coord
        FROM tag_location
        WHERE oid == ?
        ORDER BY oid DESC
        """,
        (nth_oid,))
    yn_coord = int(db_cursor.fetchone()[0])

    db_cursor.execute("""
        SELECT time_stamp
        FROM tag_location
        WHERE oid == ?
        ORDER BY oid DESC
        """,
        (nth_oid,))
    timen = db_cursor.fetchone()[0]
    print("x5 = ", xn_coord, " y5 = ", yn_coord, " timen = ", timen)

    db_connection.commit()
    db_connection.close()
    return xn_coord, yn_coord, timen

```

## Decision Algorithm

Use of the fuzzy logic python library requires the following to be present

Matplotlib >= 3.1

NumPy >= 1.6

SciPy >= 0.9

NetworkX >= 1.9

Use pip install -U <package to install> to install any missing packages

Then run

```
$ pip install -U scikit-fuzzy
```

My environment required : pip install --upgrade pip

The code used was adapted from the following source :

[https://pythonhosted.org/scikit-fuzzy/auto\\_examples/plot\\_tipping\\_problem\\_newapi.html](https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_tipping_problem_newapi.html)

Further analysis of this is contained within the final report as well as supporting demonstrations of code in video and test data on the git hub repository here:

[https://github.com/ashification/ee580\\_meng](https://github.com/ashification/ee580_meng)

**Actuator node***GPIO control*

Integrate GPIO control with broker

Adapted from: <https://www.circuitbasics.com/how-to-control-led-using-raspberry-pi-and-python/>

Implementing subscription to multiple topics to control individual LEDs for simulation of a locking mechanism being triggered

```
def on_message(client, userdata, msg):
    print(msg.topic + " " + str(msg.qos) + " " + str(msg.payload))
    raw_payload = str(msg.payload)
    print(raw_payload)
    if ("actuator_topic/" in msg.topic):
        trigger = int(re.sub('[^0-9]', '', raw_payload)) # remove non
alphanuemic characters
        print(trigger)
    if (msg.topic == "device_topic/device"):
        message = raw_payload # [1:-1]
        print(message)
    if (msg.topic == "actuator_topic/act1"):
        if (trigger == 1):
            print("Turn ON Orange LED")
            # set GPIO pin to HIGH
            GPIO.output(13, GPIO.HIGH) # Orange
        if (trigger == 0):
            print("Turn Off Orange LED")
            # set GPIO pin to Low
            GPIO.output(13, GPIO.LOW) # Orange
    if (msg.topic == "actuator_topic/act2"):
        if (trigger == 1):
            print("Turn ON Yellow LED")
            # set GPIO pins to HIGH
            GPIO.output(19, GPIO.HIGH) # Yellow
        if (trigger == 0):
            print("Turn Off Yellow LED")
            # set GPIO pins to Low
            GPIO.output(19, GPIO.LOW) # Yellow
    if (msg.topic == "actuator_topic/act3"):
        if (trigger == 1):
            print("Turn ON Blue LED")
            # set GPIO pins to HIGH
            GPIO.output(26, GPIO.HIGH) # Yellow
        if (trigger == 0):
            print("Turn Off Blue LED")
            # set GPIO pins to Low
            GPIO.output(26, GPIO.LOW) # Yellow
    if (msg.topic == "device_topic/device"):
        if ("Offline" in message):
            print("Turn OFF Red LED")
            # set GPIO pin to Low
            GPIO.output(6, GPIO.LOW) # Red
            GPIO.output(13, GPIO.LOW) # Orange
            GPIO.output(19, GPIO.LOW) # Yellow
            GPIO.output(26, GPIO.LOW) # Blue
        if (message == "Simulator Online"):
            print("Turn OFF Red LED")
            # set GPIO pin to HIGH
            GPIO.output(6, GPIO.HIGH) # Red
```