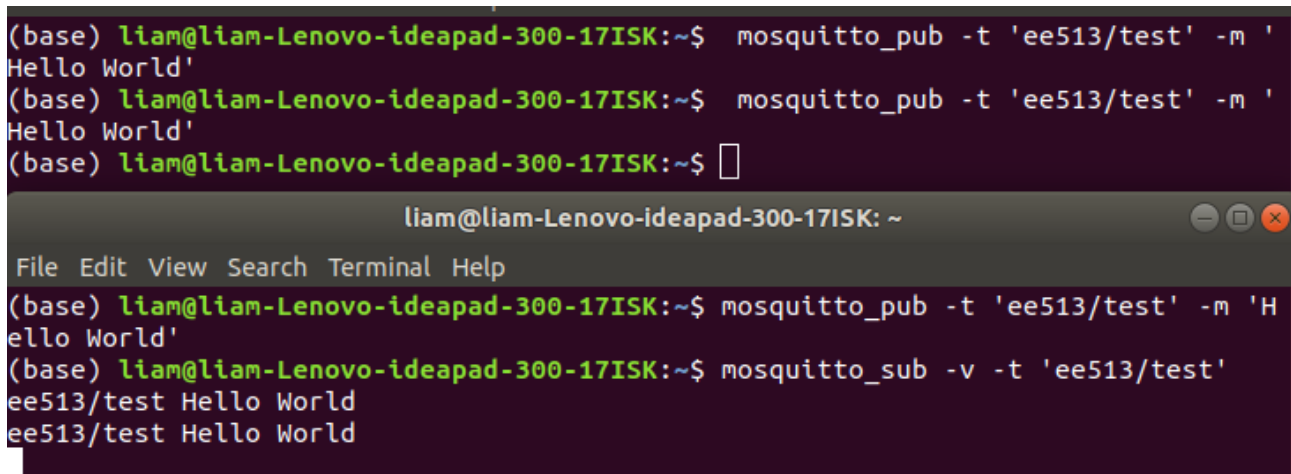


Section 1 – Establishing an MQTT Framework

Initial testing involved installing the mosquitto client/server on a dedicated Linux machine and using provided scripts to send messages between two terminal windows.



```
(base) liam@liam-Lenovo-ideapad-300-17ISK:~$ mosquitto_pub -t 'ee513/test' -m 'Hello World'
(base) liam@liam-Lenovo-ideapad-300-17ISK:~$ mosquitto_pub -t 'ee513/test' -m 'Hello World'
(base) liam@liam-Lenovo-ideapad-300-17ISK:~$ █

liam@liam-Lenovo-ideapad-300-17ISK: ~
File Edit View Search Terminal Help
(base) liam@liam-Lenovo-ideapad-300-17ISK:~$ mosquitto_pub -t 'ee513/test' -m 'Hello World'
(base) liam@liam-Lenovo-ideapad-300-17ISK:~$ mosquitto_sub -v -t 'ee513/test'
ee513/test Hello World
ee513/test Hello World
█
```

Next, the service was adapted to require a password in order to subscribe to the topic and the ‘-d’ optional flag added to the mosquitto client request to show more detailed output. The communication between client & server involves sending of ‘CONNECT’ and ‘SUBSCRIBE’ requests from the client and receipt of ‘CONNACK’ and ‘SUBACK’ messages to acknowledge successful connection & subscription respectively. In addition; the periodic PING requests & responses can also be seen as the client/server pair check the connection is still open.

To setup password protection a new folder was created in /etc /mosquitto named ‘/passwd’, where usernames and associated passwords are stored. Next the mosquitto configuration file ‘default.conf’ was altered using the nano editor to block unauthenticated users (allow anonymous false) and indicate where usernames and passwords are stored (‘/passwd’ folder created previously)

Contents of default.conf after editing:
allow_anonymous false
password_file /etc/mosquitto/passwd

After setting this authentication requirement; calls to subscribe or publish which were not accompanied by the credentials were refused as expected. This feature was then turned off for ease & speed of debugging.

<https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-the-mosquitto-mqtt-messaging-broker-on-ubuntu-16-04>

Command to specify the qos from client side
mosquitto_sub -d -u liambanus -P pw -q 2 -t "ee513/1CPUtem"

```
(base) liam@liam-Lenovo-ideapad-300-17ISK:~$ mosquitto_pub -t 'ee513/test' -m 'Hello World'
Connection Refused: not authorised.
Error: The connection was refused.
(base) liam@liam-Lenovo-ideapad-300-17ISK:~$ mosquitto_pub -t 'ee513/test' -u 'liambanus' -P 'pw' -m 'Hello World'
(base) liam@liam-Lenovo-ideapad-300-17ISK:~$

liam@liam-Lenovo-ideapad-300-17ISK: ~
File Edit View Search Terminal Help
Use 'mosquitto_sub --help' to see usage.
(base) liam@liam-Lenovo-ideapad-300-17ISK:~$ mosquitto_sub -v -u 'liambanus' -P 'pw' -d -t 'ee513/test'
Client mosqsub|12163-liam-Leno sending CONNECT
Client mosqsub|12163-liam-Leno received CONNACK
Client mosqsub|12163-liam-Leno sending SUBSCRIBE (Mid: 1, Topic: ee513/test, QoS: 0)
Client mosqsub|12163-liam-Leno received SUBACK
Subscribed (mid: 1): 0
Client mosqsub|12163-liam-Leno received PUBLISH (d0, q0, r0, m0, 'ee513/test', ... (11 bytes))
ee513/test Hello World
Client mosqsub|12163-liam-Leno sending PINGREQ
Client mosqsub|12163-liam-Leno received PINGRESP
```

Section 2 – Publisher or publish.cpp

The first step was to create a simple helper function within the ‘publish’ script to record the current time and the CPU temperature as an easily accessible data value for testing. The function `getCPUtemperature()` read in the current temperature in degrees Celsius from the appropriate “sys/class” folder and the value was returned as a float. The string buffer function `sprintf()` was used to prepare the char array “*str_payload*” for transmitting the temperature reading as a JSON object as shown below (S2.i). The choice of ‘d’ as the name field was used for downstream cross compatibility with IBM Watson. The message was published to the topic “ee513/1CPUTem” and this format of using an integer prefix for each topic was implemented as a means of identifying & storing received messages downstream in the Subscriber application (see Section 3)/

```
sprintf(str_payload, "{\"d\":\"1CPUTem\": %f }\"", getCPUtemperature());
```

Once this procedure was verified using MQTT-Spy tool (and the Qt Application); a second topic was developed based around the ADX’s pitch reading. An object of the ADXL class was created in the publish program’s `main()` function and initialised (switched to read mode). As with the temperature value above; the ‘pitch’ value was packaged into JSON format and published to the topic “ee513/2Ptch”. While the values were not calibrated; manual inspection and observation of readings as the sensor was moved confirmed the pitch values were approximately accurate. To demonstrate the Last Will functionality; an additional object from the MQTTClient library was defined: *MQTTClient-willOptions*; whose parameters were adjusted to define the topic for the last will message, the message itself and the QOS. A last will message was defined for the topic “/1CPUTem” and the contrasting behaviour examined by using MQTT-Spy to subscribe to both 1CPUTem and /2Ptch simultaneously before simulating a disconnection.

New All ee513/1CPUTem ee513/2Ptch

Message 1 / 38 ☒ Show latest Search Tools

Topic ee513/1CPUTem Retained ☐ QoS 0 Time 2020/04/15 13:08:36:945

Data (30B) {"d":{"1CPUTem": 33.627998 }}

▼ Received messages summary [search topics:] (1 topic, 38 messages, load: 0.8/0.9/0.1)

Topic	Content	Browse	Messages	Last received
ee513/1CPUTem	{"d":{"1CPUTem": ...	<input checked="" type="checkbox"/>	38	2020/04/15 13:08:36:945

```

Waiting for up to 10 seconds for publication of {"d":{"2Ptch": 1.175760 }}
on topic ee513/2Ptch for ClientID: rp1l
Message with token 2 delivered.
Current local time and date: Wed Apr 15 13:08:36 2020
Current local time and date: Wed Apr 15 13:08:36 2020
Waiting for up to 10 seconds for publication of {"d":{"1CPUTem": 33.627998 }}
on topic ee513/1CPUTem for ClientID: rp1l
Message with token 1 delivered.
Waiting for up to 10 seconds for publication of {"d":{"2Ptch": 0.783370 }}
on topic ee513/2Ptch for ClientID: rp1l
Message with token 2 delivered.

```

New All ee513/1CPUTem ee513/2Ptch

Message 1 / 107 ☒ Show latest Search Tools

Topic ee513/1CPUTem Retained ☐ QoS 0 Time 2020/04/15 13:09:56:385

Data (21B) I told you I was sick

▼ Received messages summary [search topics:] (1 topic, 107 messages, load: 0.0/0.7/0.4)

Topic	Content	Browse	Messages	Last received
ee513/1CPUTem	I told you I was sick	<input checked="" type="checkbox"/>	107	2020/04/15 13:09:56:385

```

on topic ee513/1CPUTem for ClientID: rp1l
Message with token 1 delivered.
Waiting for up to 10 seconds for publication of {"d":{"2Ptch": 0.783370 }}
on topic ee513/2Ptch for ClientID: rp1l
Message with token 2 delivered.
Current local time and date: Wed Apr 15 13:09:55 2020
Current local time and date: Wed Apr 15 13:09:55 2020
Waiting for up to 10 seconds for publication of {"d":{"1CPUTem": 33.627998 }}
on topic ee513/1CPUTem for ClientID: rp1l
Message with token 1 delivered.
^C
pi@raspberrypi:~/assign2 $

```

New All ee513/1CPUTem ee513/2Ptch

Message 1 / 50 ☒ Show latest Search Tools

Topic ee513/2Ptch Retained ☐ QoS 0 Time 2020/04/15 13:09:54:445

Data (27B) {"d":{"2Ptch": 0.783370 }}

▼ Received messages summary [search topics:] (1 topic, 50 messages, load: 0.0/0.0/0.2)

Topic	Content	Browse	Messages	Last received
ee513/2Ptch	{"d":{"2Ptch": 0.78...	<input checked="" type="checkbox"/>	50	2020/04/15 13:09:54:445

```

on topic ee513/1CPUTem for ClientID: rp1l
Message with token 1 delivered.
Waiting for up to 10 seconds for publication of {"d":{"2Ptch": 0.783370 }}
on topic ee513/2Ptch for ClientID: rp1l
Message with token 2 delivered.
Current local time and date: Wed Apr 15 13:09:55 2020
Current local time and date: Wed Apr 15 13:09:55 2020
Waiting for up to 10 seconds for publication of {"d":{"1CPUTem": 33.627998 }}
on topic ee513/1CPUTem for ClientID: rp1l
Message with token 1 delivered.
^C

```

The message itself was a homage to Spike Milligan and in Figure it can be seen that on unexpected disconnection caused by manually quitting the publisher application (or disabling the ethernet connection between the desktop machine subscriber (MQTT-Spy) and the publisher on the Rpi); the Last Will message is sent to the subscriber to indicate the publisher has gone offline. In contrast, by calling the manual client disconnect function and subsequently quitting the program the Last Will message is *not* sent; nor is it sent when the 'Disconnect' button is pushed from the Qt application (See Section 4). The code for including a Last Will message involved a separate initialiser with similar parameters to that of the connectOptions:

```
MQTTClient willOptions will_opts = MQTTClient_willOptions_initializer;  
will_opts.topicName = TOPIC tem;  
will_opts.message = "I told you I was ill";  
will_opts.qos = QOS_1CPUtem;  
opts.will = &will_opts;
```

Under the MQTT protocol; three distinct levels of 'Quality of Service' (QoS) are available with the minimum QoS level 0 having no guarantee of delivery above that provided by the underlying TCP protocol. There is no acknowledgement when a message is received and is neither stored or resent by the transmitter. Also known as At Most Once or Fire & Forget

QoS Level 1 guarantees delivery at least once; the defining feature is that the sender retains the message until a publication acknowledgement (PUBACK) packet is received from the receiver. Hence it is possible for a message to be sent/delivered multiple times. If a message is re-sent and duplicate (DUP) flag bit is set to 1.

QoS Level 2 ensures a message is received *exactly* once. It is the most reliable but slowest method and involves a 4 step conversation between client & broker for each message. To avoid processing a message multiple times; the receiver must respond to a published message with a PUBREC packet. If it fails to do so the sender will repeat transmission with the DUP bit set to 1 until it receives a PUBREC acknowledgement. Only then can the sender discard the initial packet to be published and respond with a PUBREL packet. On receipt of the PUBREL packet the receiver can discard references to the original packet ID and send a PUBCOMP packet to the publisher. The packet id is then freed for re-use.

To demonstrate the impact of the different QoS levels; the three different levels 0,1 and 2 were used to define the MQTT session for the three topics 1CPUtem, 2Ptch and 3Roll respectively. Again, the verbose debug output flag in Mosquitto was used to monitor the flow of messages. An additional feature of MQTT is the use of persistent vs clean sessions. In a clean session; each subscription is treated as a stand-alone event with the consequence that if a subscriber briefly goes offline or is interrupted and then reconnects it will miss the interim messages. In contrast; persistent sessions require the broker to associate a client ID with it's message history and store unreceived messages until the client is back online; provided QoS level 1 or 2 were used. The clean session flag was set to 0 for all topics and the behaviour following a disconnect/reconnect observed

```

Client sub1 received PUBLISH (d0, q0, r0, m0, 'ee513/1CPUTem', ... (30 bytes))
{"d":{"1CPUTem": 33.627998 }}
^C
(base) liam@liam-Lenovo-ideapad-300-17ISK:/etc/mosquitto$ mosquitto_sub -d -u li
iambanus -P pw -q 0 -i sub1 -c -t "ee513/1CPUTem"
Client sub1 sending CONNECT
Client sub1 received CONNACK
Client sub1 sending SUBSCRIBE (Mid: 1, Topic: ee513/1CPUTem, QoS: 0)
Client sub1 received SUBACK
Subscribed (mid: 1): 0
Client sub1 received PUBLISH (d0, q0, r0, m0, 'ee513/1CPUTem', ... (30 bytes))
{"d":{"1CPUTem": 33.627998 }}
^CClient sub1 received PUBLISH (d0, q0, r0, m0, 'ee513/1CPUTem', ... (30 bytes))
)
{"d":{"1CPUTem": 33.627998 }}

```

QoS Level 0:

Observe that the message identification field m(N) is 0 for each message in two sessions despite clean sessions being disabled. The QoS level is set using the -q flag in the mosquitto call and in the publish.cpp script.

```

Client sub2 received PUBLISH (d0, q1, r1, m141, 'ee513/2Ptch', ... (27 bytes))
Client sub2 sending PUBACK (Mid: 141)
{"d":{"2Ptch": 0.193477 }}
^C
(base) liam@liam-Lenovo-ideapad-300-17ISK:/etc/mosquitto$ mosquitto_sub -d -u li
"ee513/2Ptch"
Client sub2 sending CONNECT
Client sub2 received CONNACK
Client sub2 sending SUBSCRIBE (Mid: 1, Topic: ee513/2Ptch, QoS: 1)
Client sub2 received PUBLISH (d0, q1, r0, m142, 'ee513/2Ptch', ... (27 bytes))
Client sub2 sending PUBACK (Mid: 142)
{"d":{"2Ptch": 0.000000 }}
Client sub2 received PUBLISH (d0, q1, r0, m143, 'ee513/2Ptch', ... (27 bytes))
Client sub2 sending PUBACK (Mid: 143)
{"d":{"2Ptch": 0.000000 }}

```

QoS Level 1:

Observe that in addition to the initial CONNACK and SUBACK packets; each new message is accompanied by a PUBACK packet which is sent to the publisher to confirm receipt of each message. This PUBACK packet references the message id. The message id can also be seen to resume where it left off following disconnection of the subscriber; i.e. we can see that missed messages were stored and transmitted to the subscriber on reconnect.

```
(base) liam@liam-Lenovo-ideapad-300-17ISK:/etc/mosquitto$ mosquitto_sub -d -u li
amabus -P pw -q 2 -i sub3 -c -t "ee513/3Roll"
Client sub3 sending CONNECT
Client sub3 received CONNACK
Client sub3 sending SUBSCRIBE (Mid: 1, Topic: ee513/3Roll, QoS: 2)
Client sub3 received PUBLISH (d0, q2, r0, m17, 'ee513/3Roll', ... (28 bytes))
Client sub3 sending PUBREC (Mid: 17)
Client sub3 received SUBACK
Subscribed (mid: 1): 2
Client sub3 received PUBREL (Mid: 17)
{"d":{"3Roll": -0.787479  }}
Client sub3 sending PUBCOMP (Mid: 17)
```

QoS Level 2:

Observe that in addition to the PUBREC packet the two additional features of QoS 2 are present; namely the PUBREL and PUBCOMP packets.

Attempts to simulate a client disconnection and repeat transmission of messages (in QoS 1 & 2) were unsuccessful. It was hoped to display messages with the duplicate flag indicating they were re-sent by the publisher on failure to deliver the original. Lastly; while the current time before each reading

Section 3 – Subscriber & Actuator

In developing the subscriber application the greatest challenge was correctly parsing the messages received from the publisher. A means of decoding the messages which were in JSON format using a more elegant function library was not completed and instead the message was stored as an array of characters and a 'topic identifier' integer number (N) placed in the same position for each topic e.g. "ee513/Ntopic" or "ee513/3Roll":

```
char topic_identifier[1] = {topicName[6]}; //topics coded 1-9
cout << "top name char[6] :" << topic_identifier[0] << endl;
```

Similarly, the value of interest for each topic was also extracted from the payload using the atof() function with the 'payloadptr' variable offset by an empirically determined number to retrieve the signed float values for Pi CPU temperature, ADXL pitch & roll. As a simple proof of concept a warning message and lighting of a red LED in response to a CPU temperature exceeding 34.5 degrees constituted the first application. The provided C++ template class for controlling the Rpi GPIO's was used throughout as a simple means of controlling the LEDs. As this is the same publisher script as used in Section 2; the 'last will' message is also displayed following an unexpected disconnect from the publisher.

```
if (topic_identifier[0] == '1'){

float temp16 = atof(payloadptr+16);
cout << "temp 16 is " << temp16 << endl;
if (temp16 > 36.5) {
cout << "Flashing LED for danger" << endl;
dled21.turnOn();
dled21.turnOff();
}
```


As a slightly more intricate example; a total of 4 LED's were connected to different GPIO pins and a common ground (using a breadboard power-rail) and activated in response to movement of the ADXL. The aim was to simulate the indicator and reverse lights of a small vehicle with yellow LEDs lighting in response to left & right turns (subscribed to ee513/3Roll) of the ADXL and green/red LEDs lighting in response to forward and backward pitch tilts respectively (subscribed to ee513/2Ptch). The thresholds were determined empirically and despite a short lag time the breadboard lights were quite responsive to changes in the accelerometer. A short video demonstration is available on request. Enlarged images with the ADXL and breadboard in frame are included in the appendix; the spatial orientation of the ADXL and correct response of the actuator application are visible in each case.

A persistent session was ensured by disabling the clean session flag in the connect options and ensuring a unique client id for the subscriber. A means of outputting the message id's similar to the verbose mosquitto server output above could not be determined but on quitting the subscribe application and re-subscribing; the cache of missed messages did appear in the terminal followed by rapid cycling of the LED's corresponding to the missed movement of the accelerometer. In ensuring a persistent session an important requirement was discovered; that a unique client ID should be supplied as a command line argument to avoid disconnecting other instances of the subscribe.cpp application. The command line was also used as a simple means of selecting between an instance of the temperature monitoring and accelerometer monitoring applications. While these scripts could run concurrently; due to a shortage of resistors and breadboard real estate the same LED's were used; causing contention when both applications were ran together.

```
pi@raspberrypi:~/assign2 $ ./subscribe adxl rpi8
Message arrived
  topic: ee513/3Roll
  message: Subbing to adxl
Subscribing to topic ee513/2Ptch
for client rpi8 using QoS1

Press Q<Enter> to quit

top name char[6] :3
roll id'd
roll 14 is -2.96078
Right Turn
{"d":{"3Roll": -2.960778 }}
Message arrived
  topic: ee513/2Ptch
  message: top name char[6] :2
pitch 14 is 0.197373
{"d":{"2Ptch": 0.197373 }}
Message arrived
  topic: ee513/3Roll
^C
```

```
pi@raspberrypi:~/assign2 $ ./subscribe temp rpi9
Subbing to temp
Subscribing to topic ee513/1CPUTem
for client rpi9 using QoS1

Press Q<Enter> to quit

Message arrived
  topic: ee513/1CPUTem
  message: top name char[6] :1
temp 16 is 33.628
Temperature Safe
{"d":{"1CPUTem": 33.627998 }}
Message arrived
  topic: ee513/1CPUTem
  message: top name char[6] :1
```

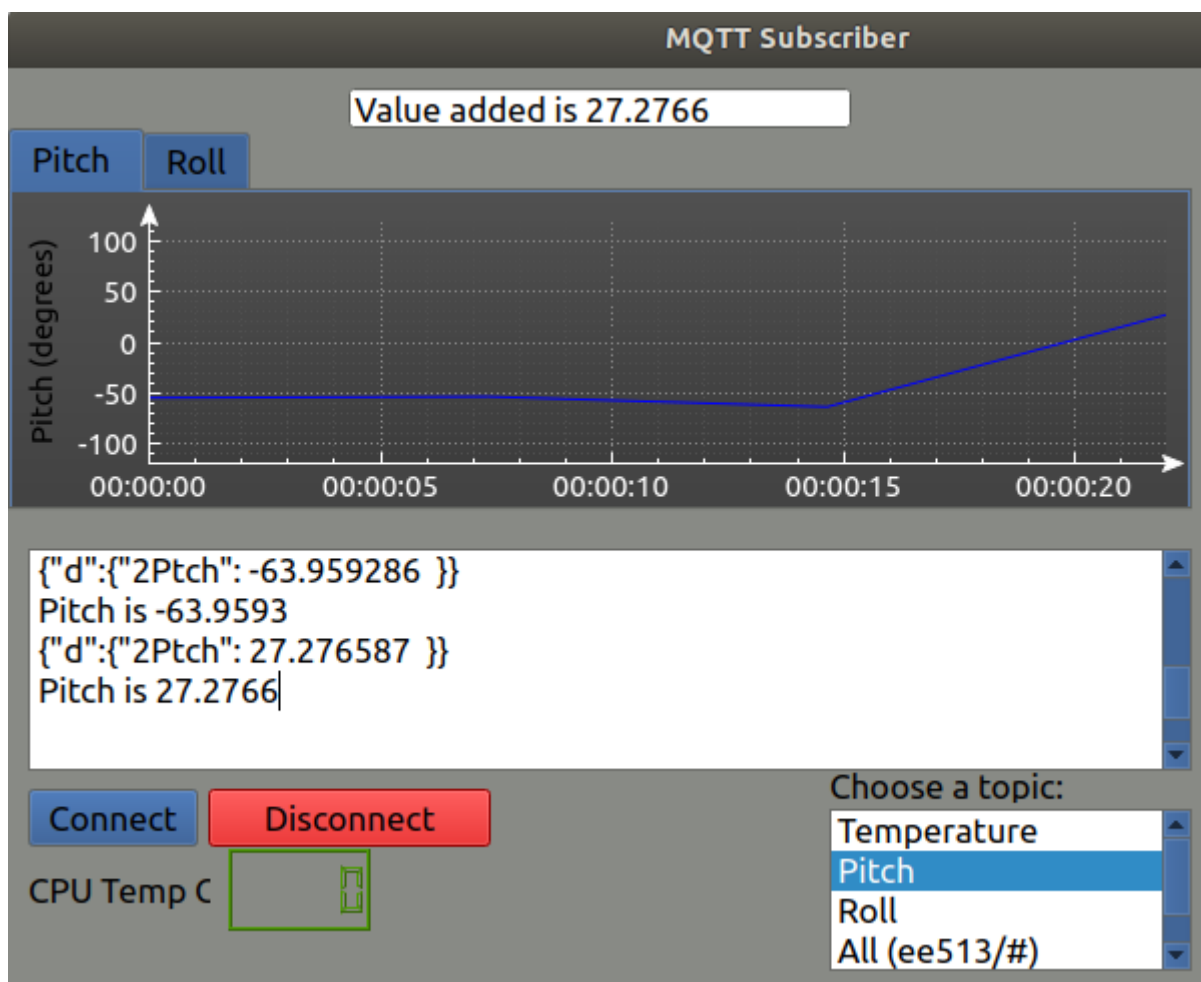
Left: The 'adxl' actuator program which subscribes to pitch & roll and lights a series of LED's in response to the measurements. Some of the additional debug output is displayed to show that the topic identifier field has been identified as 3 for roll and 2 for pitch (the top name char [6] value). The actual 'float' value corresponding to the roll reading in the payload message is at position payload ptr+14. The value is -2.96078 which is below the threshold set at -1.5; triggering a 'Right Turn' message and lighting of the right yellow LED/.

Right:

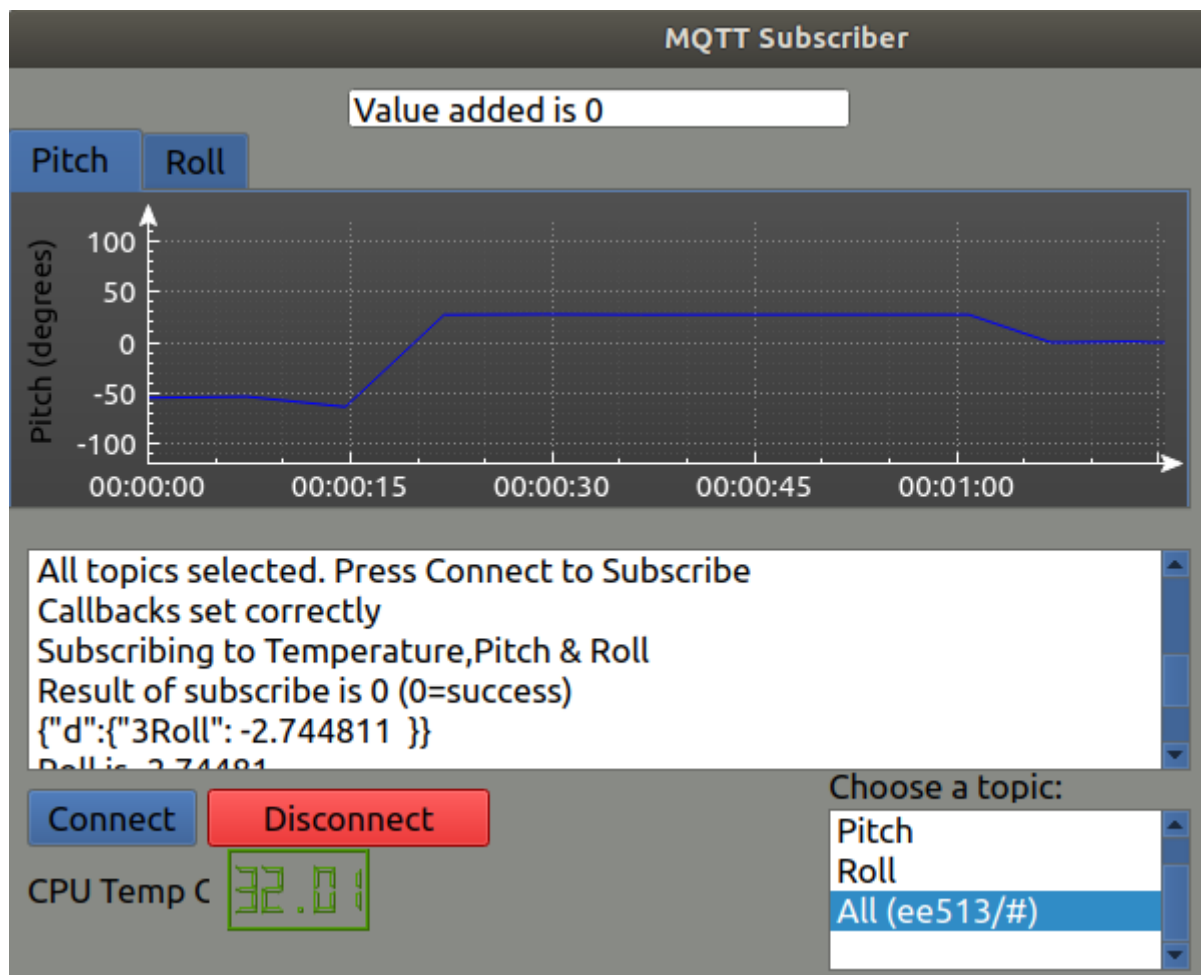
The simpler temperature actuator which subscribes to ee513/temp only. This program lights the same green LED provided the temperature is below the threshold (and displays a temperature safe message) and flashes a red LED if the temperature exceeds 34.5 degrees.

Section 4 – Qt Visualisation

The final element was a subscriber application with a GUI developed using Qt which included near real-time plotting of sensor data published from the Pi & ADXL. On start-up; the application screen provided the users with text instructions and provided a list of subscribable topics as well as the option to use the wildcard ‘/#’ to subscribe to all topics published to ee513/. This default topic folder was hardwired into the application but could be easily adapted to accept text input if/when new topics are added. On selecting a topic, a prompt appeared to indicate that pressing the blue connect button was required to begin subscription. Confirmation of the topic name and notice of successful subscription were displayed in the text ticker. Received messages in raw JSON format were also output and depending on the topic; the data was displayed on one of the graph tabs (for pitch & roll) or in a small LCD display for CPU temperature. Once subscribed; the text field above the graph which initially displays instructions changes to display the most recent pitch or roll value to have been added. Complex behaviour such as handling of changing from being subscribed solely to pitch or roll and then selecting ‘All’ was accomplished by modifying the sequence of updates etc. On submission the only significant remaining bug was the display of a large negative value for temperature in between messages on the ‘1CPUTem’ topic.



App homescreen following subscription to 'Pitch' topic.



Switching to ‘All’ topics in the same session. The pitch graph resumes plotting and current CPU Temperature is shown in the LCD display. An additional image from the same session shows the roll data in the appendix.

Long periods of inactivity on one topic did not affect the graphical display and while adding a feature to reset the time axis at each subscription (e.g. going from just pitch to All) was considered; it was felt that the automatic rescaling of the axes in the template provided was adequate. The varying parts of the application are discussed in more detail function by function with reference or inserts of source code for clarity. The only script which required substantial modification was `mainwindow.cpp`.

i) The constructor required only minor changes apart from the verbose but uncomplicated settings for ‘prettifying’ the graph backgrounds in the two custom plot instances. See source code.

ii) `parseJSONData()`

```
int MainWindow::parseJSONData(QString str)
    QString cur = ui->listWidget->currentItem()->text();
```

The current value selected in the `listWidget` was pulled and used to inform the processing of the different JSON formatted payloads. The easiest method of avoiding excessive repetition of code was a series of if statements which would execute if the current selection was matched **or** the ‘all’ field was selected. The extra if statement is designed as an (imperfect) solution to a recurrent problem whereby a message would be processed even if the application was not subscribed to that topic. This led to the unused variables (pitch, roll or temp) being set equal to 0 in the intervening period between two messages of the selected topic arriving when messages from other topics would arrive. This was partially overcome by initialising pitch & roll to 1 and inserting a catching

statement like the second last line below to prevent the value being over-written incorrectly. The effectiveness of this method is implicit from the static pictures of the graph as the value of roll does not return to 0 in between readings (as occurred before this bug was handled).

```
if (cur == "Pitch" || cur == "All (ee513/#)") {
    QJsonDocument doc = QJsonDocument::fromJson(str.toUtf8());
    QJsonObject obj = doc.object();
    QJsonObject sample = obj["d"].toObject();
    this->pitch = (float) sample["2Ptch"].toDouble();
    if (pitch != 0){
        ui->outputText->appendPlainText(QString("Pitch is %1").arg(this->pitch));
    }
}
```

iii) update()

The only edit worth mentioning in the update() function was the workaround which after basic testing appeared to have solved the erroneous temperature display issue. When the topic is changed the last known value persists indefinitely; some kind of flag to indicate this was a historic reading would be a beneficial extra feature.

```
void MainWindow::update()
if (this->temperature > 1){
    ui->lcdNumber->display(this->temperature);}
```

iv) on_connectButton_clicked()

While a global state holding the current selection of the list widget would allow it to be accessed from within any function; attempts to develop this feature were unsuccessful and the selection was converted to a QString as in the parseJSONData() function:

```
QString cur_sel = ui->listWidget->currentItem()->text();
```

This cur_sel variable was used to dictate which topics to subscribe to. Again, over-writing a global variable named TOPIC was the intended approach but this could not be coerced into working. A new const char array named TOPICN[] was used as an argument for the MQTTClient_subscribe function. QOS was hardwired to be 1 for all topics/subscriptions as this feature was more thoroughly examined in previous sections.

```
else if (cur_sel == "All (ee513/#)" ){
    ui->outputText->appendPlainText(QString("Subscribing to
Temperature,Pitch & Roll"));
    const char TOPIC4[] = "ee513/#";
    int x = MQTTClient_subscribe(client, TOPIC4, QOS);
    ui->outputText->appendPlainText(QString("Result of subscribe is %1
(0=success)").arg(x));
}
```

v) on_MQTTmessage()

This function was chosen as the best location to call the parseJSONData function and subsequently update the graphs etc. However; a more elegant approach may have been to carry out some basic filtering of messages here and only parse messages to which the application was subscribed. With this approach messages were passed to the function regardless of whether they were displayed or not (leading to bugs such as 0 values in between readings).

```
parseJSONData(payload); //This is slightly wasteful; parsing the whole payload
even when not subscribed but a more elegant solution was not developed.
this->update();
```

Getting the time to display would be nice

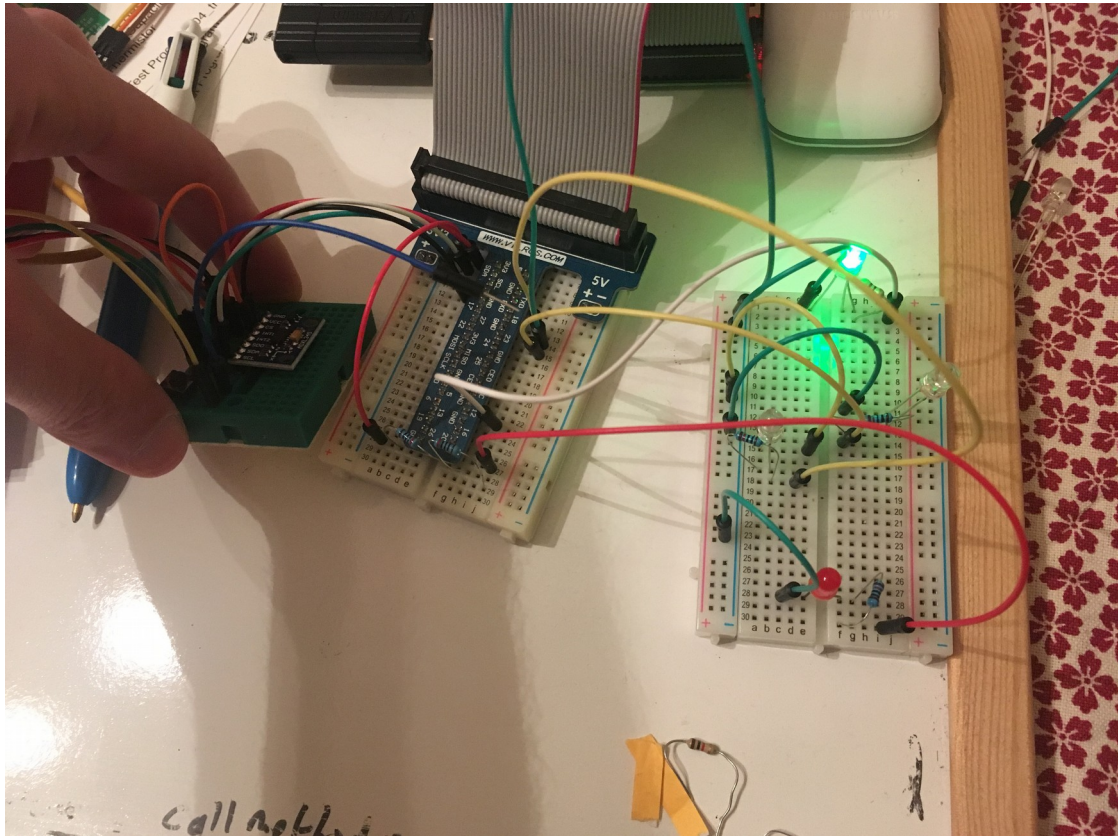
vi) on_listWidget_currentItemChanged()

Initially the application immediately changed subscription based on selecting an item from the list using this function. While quicker and more responsive; it was deemed likely that this behaviour was un-intuitive and a prompt to click the connect button after selecting a topic was included

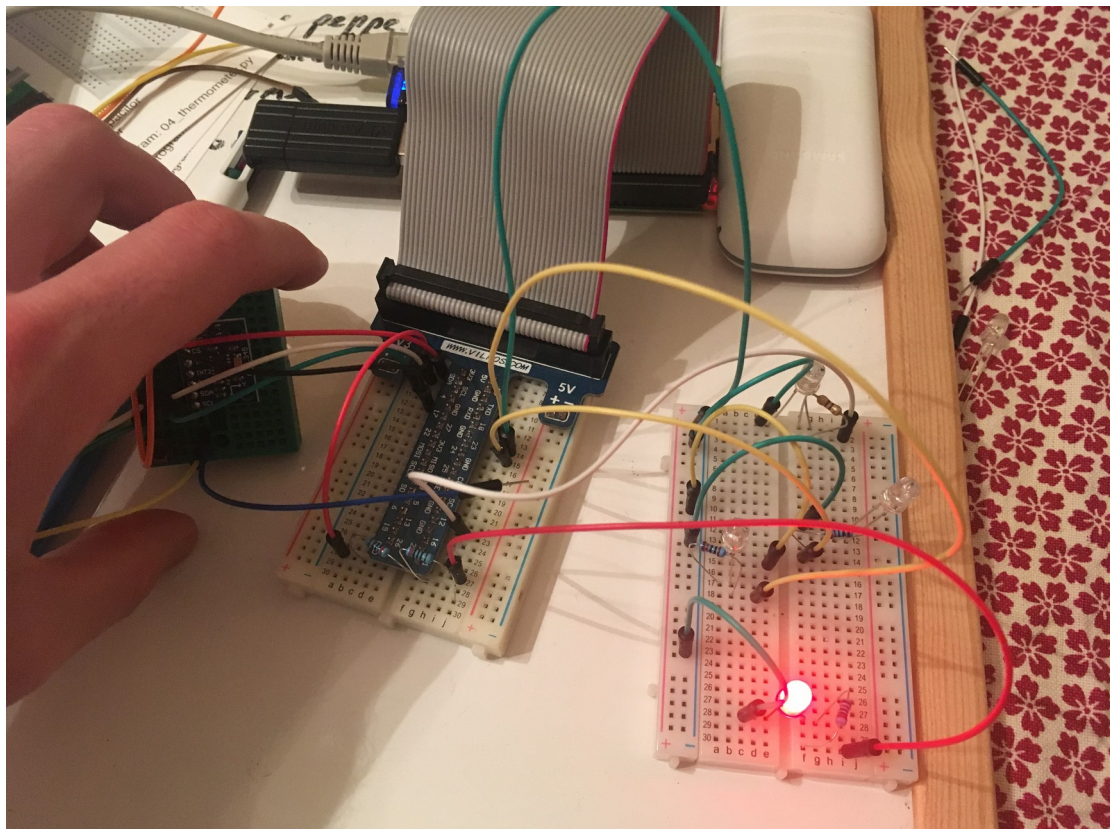
```
QString cur = ui->listWidget->currentItem()->text();
if (cur == "Temperature"){
    ui->outputText->appendPlainText("Temperature selected. Press Connect to
Subscribe");}
```

Actuator Demonstration

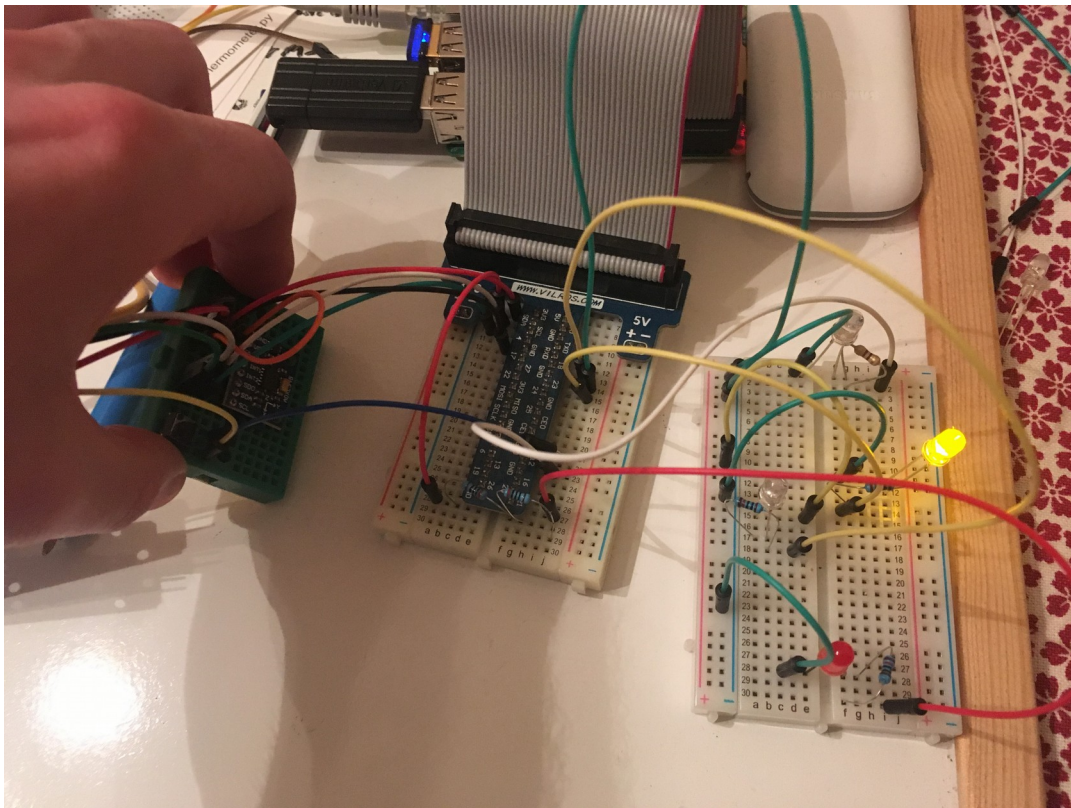
Forward Tilt: Pitch exceeding +10 and lighting of green LED



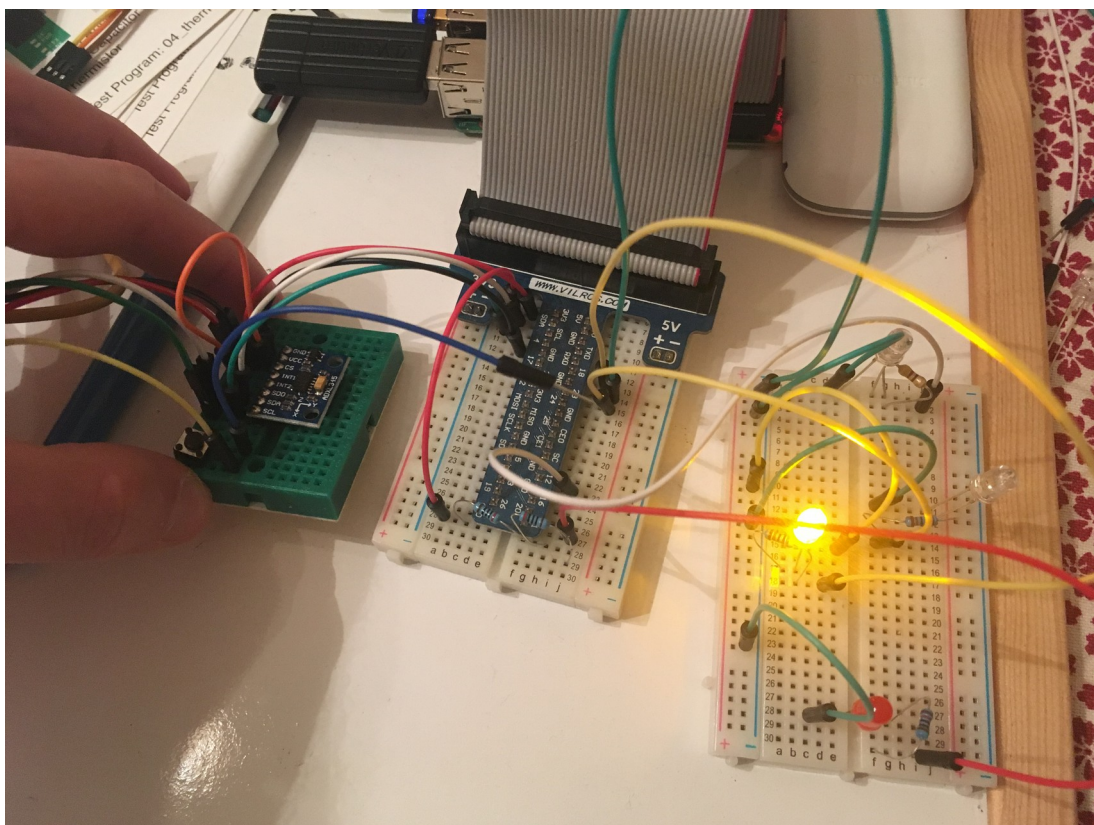
Backward Tilt: Pitch exceeding -10 and lighting red LED

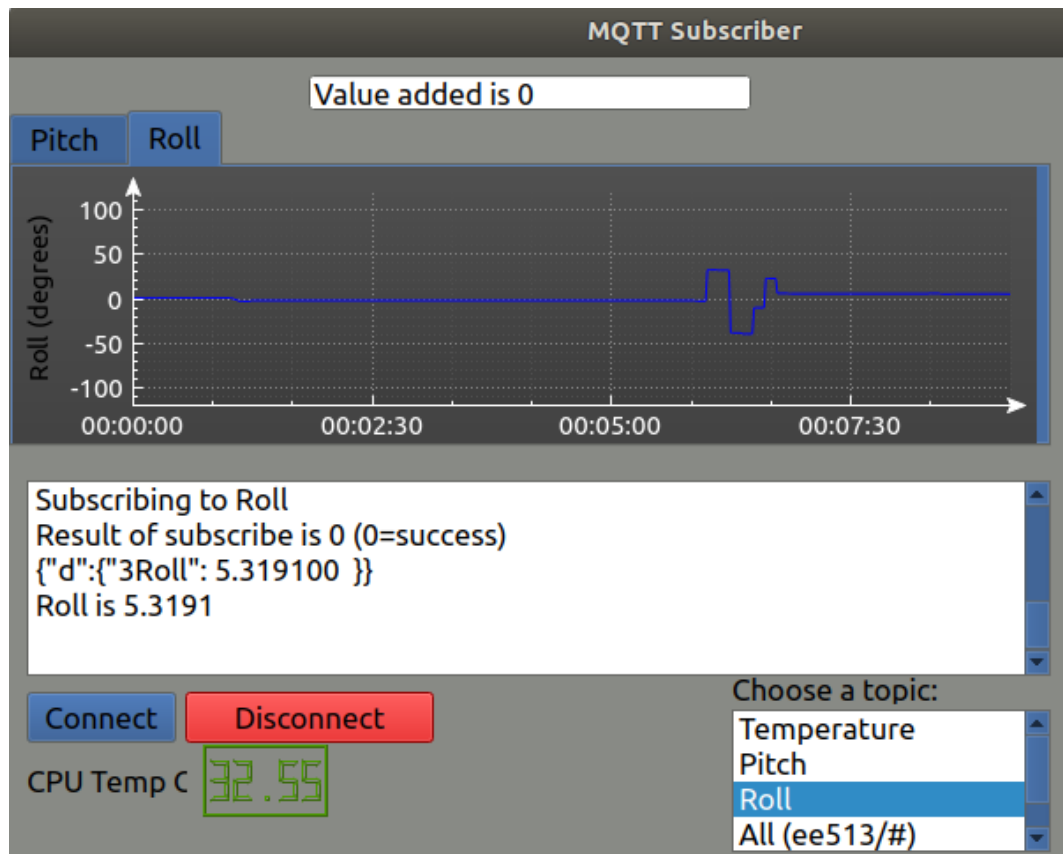


Right Turn: Roll exceeding 1.5

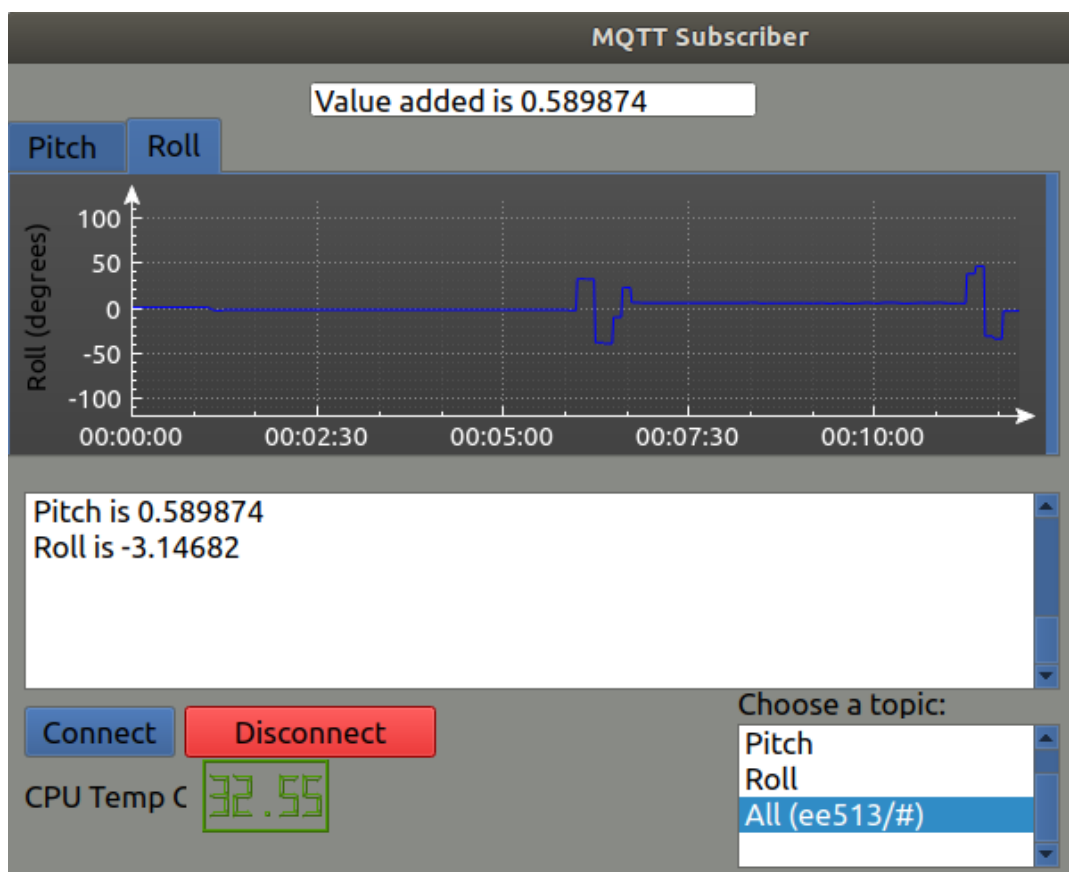


Left Turn: Roll exceeding -1.5





Plotting of Roll data – Disconnected from pitch/temperature but session maintained.



Changing back to 'All' in the same session. Note value added refers to the pitch.

