Overview

This project teaches you how to build up a system connecting sensors with pi and accessible from other client through internet. Also we try to visualize the data which we get from the sensor for easier analyzation. There will be four sections below including: mraa, MQTT, mongoDB and Bokeh. Mraa is an API used for linkit 7688 GPIO control. MQTT is a Machine-to-machine protocols used in the project for the communication between sensor and raspberry pi. mongoDB is used for storing the data into database and Bokeh is used in data visualization.

The system is made up with sensor connecting linkit 7688 and linkit 7688 is then connected to raspberry pi. In this situation, linkit 7688 is used for linking sensors to the raspberry pi, and raspberry pi is meant to be running as a web server to provide data access and visualization.

mraa

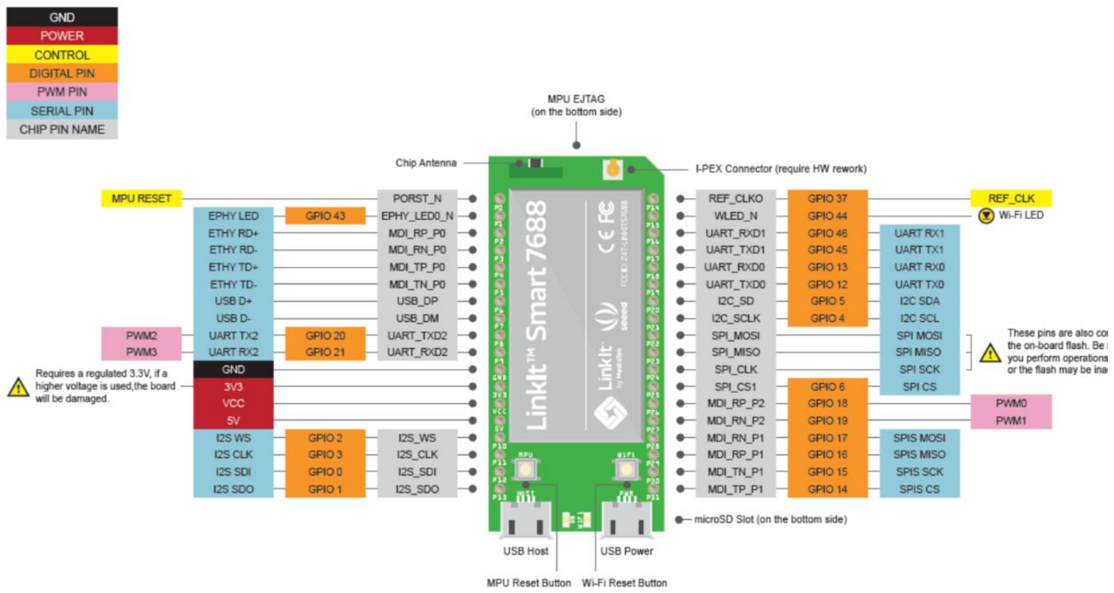


Figure above are the pins for linkit 7688(take note that linkit 7688 duo has different pin)

Additional information for linkit 7688 duo:

Linkit 7688 duo has both MPU and MCU. The sensors are all connected to MCU(ATmega), we need torun standard fermata code in MPU so that it can connect to MCU through UART, then we can use pymata in MCU to manipulate sensor data.

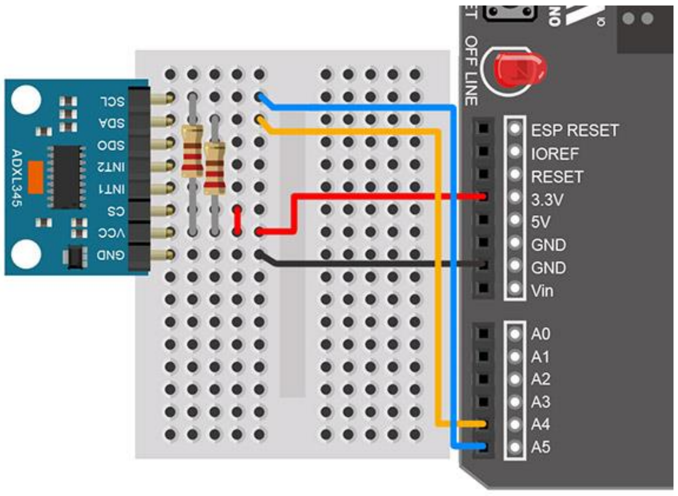
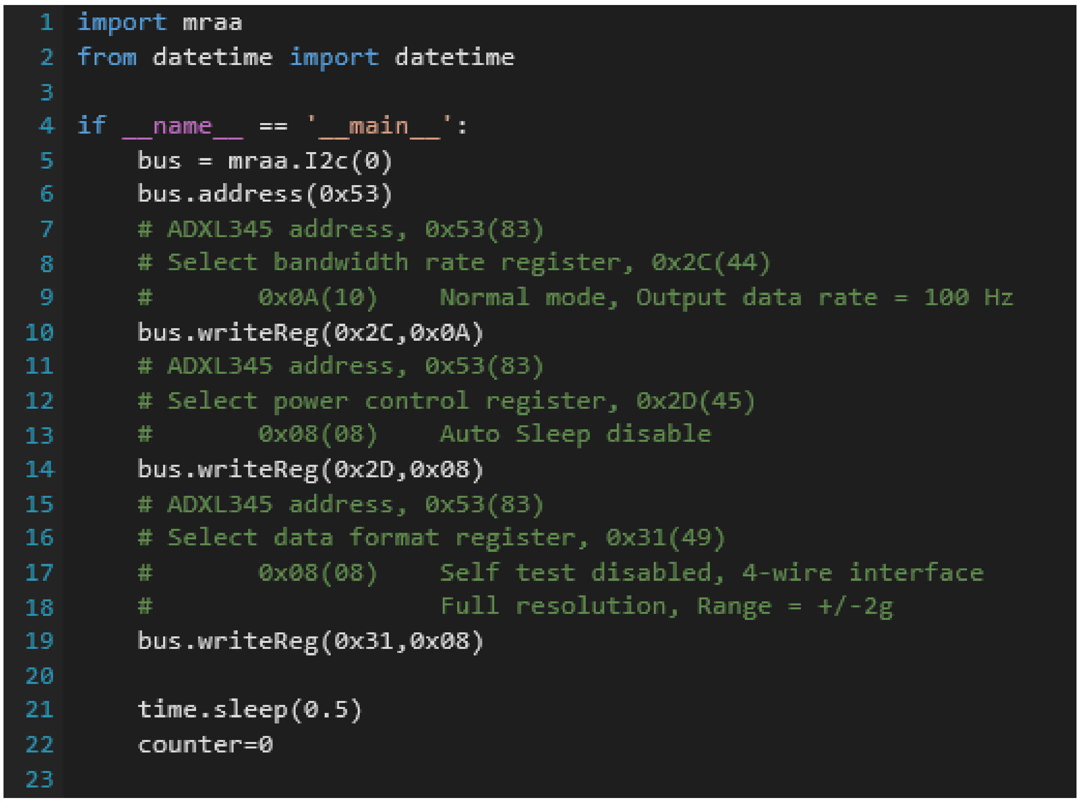


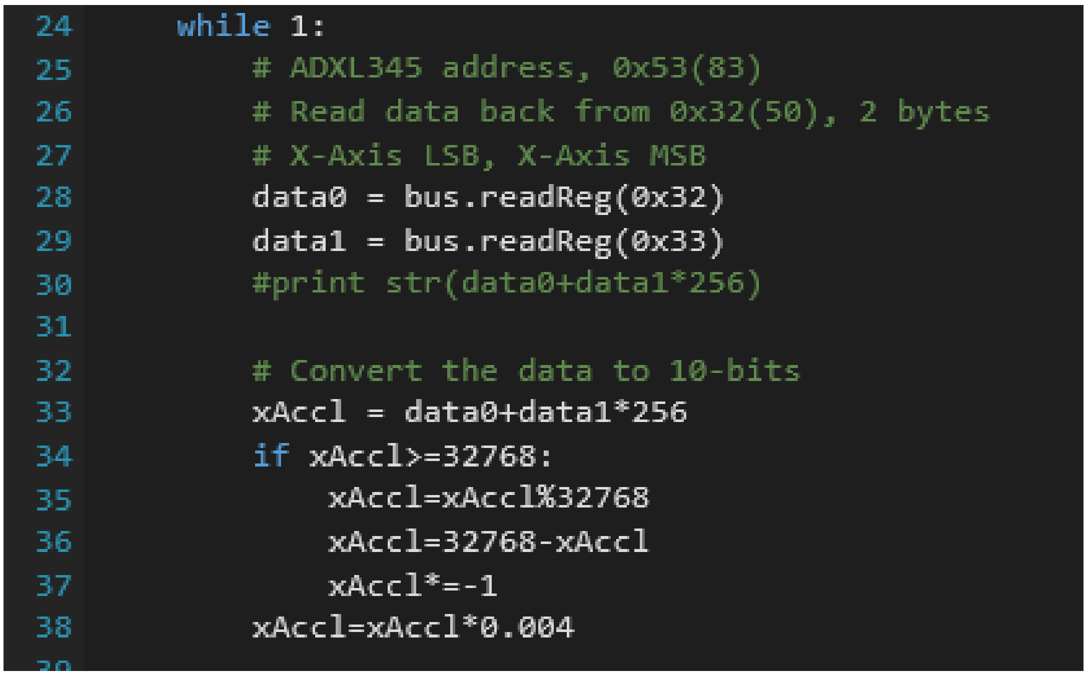
Figure above is how we connect linkit 7688 with sensor adxl345, note that the A4 and A5 pins here are P20 and P21 in linkit 7688

Mraa is an API in python to control GPIO of linkit 7688. It is installed in linkit 7688 by default so there is no installation issues. The code is very simple too. Below is an easy code of how to fetch data from adxl345:



```python
import mraa
from datetime import datetime

if __name__ == '__main__':
    bus = mraa.I2c(0)
    bus.address(0x53)
    # ADXL345 address, 0x53(83)
    # Select bandwidth rate register, 0x2C(44)
    #       0x0A(10)    Normal mode, Output data rate = 100 Hz
    bus.writeReg(0x2C,0x0A)
    # ADXL345 address, 0x53(83)
    # Select power control register, 0x2D(45)
    #       0x08(08)    Auto Sleep disable
    bus.writeReg(0x2D,0x08)
    # ADXL345 address, 0x53(83)
    # Select data format register, 0x31(49)
    #       0x08(08)    Self test disabled, 4-wire interface
    #                   Full resolution, Range = +/-2g
    bus.writeReg(0x31,0x08)

    time.sleep(0.5)
    counter=0

```

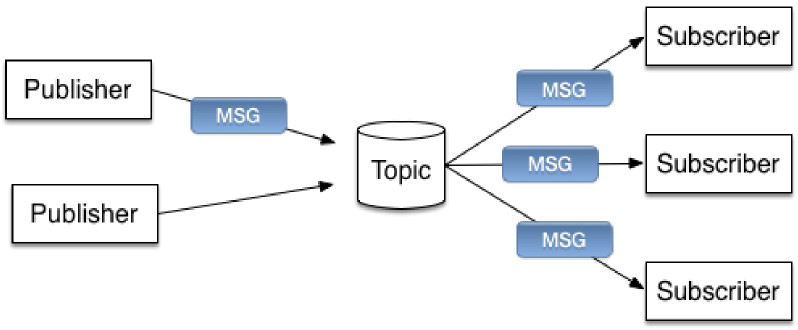
This code is for making configuration and initialization, detailed are written in the green colored annotation



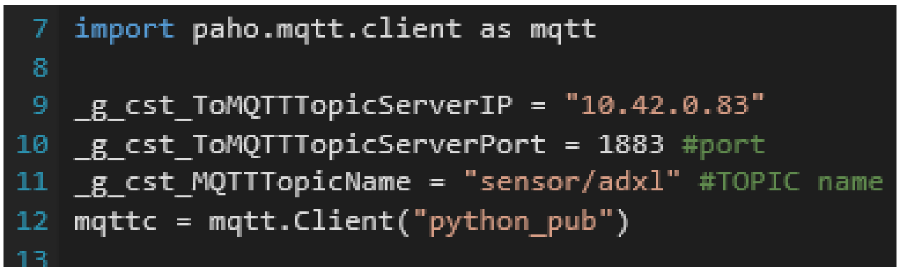
```
    while 1:
        # ADXL345 address, 0x53(83)
        # Read data back from 0x32(50), 2 bytes
        # X-Axis LSB, X-Axis MSB
        data0 = bus.readReg(0x32)
        data1 = bus.readReg(0x33)
        #print str(data0+data1*256)

        # Convert the data to 10-bits
        xAccl = data0+data1*256
        if xAccl>=32768:
            xAccl=xAccl%32768
            xAccl=32768-xAccl
            xAccl*=-1
        xAccl=xAccl*0.004
```

Code above is a simple example on how to get x acceleration of adxl345, do the same things on register 0x34, 0x35 for y acceleration and 0x36, 0x37 for z acceleration.

MQTT



MQTT uses the concept of publisher and subscriber for data transfer. In our situation publisher are linkit 7688 and sensor, while subscribers are raspberry pi(at same time raspberry pi is acting as a broker too). The publisher publish a data to a topic and then people who subscribed the topic will be able to receive the data. This connection is established based on TCP/IP.

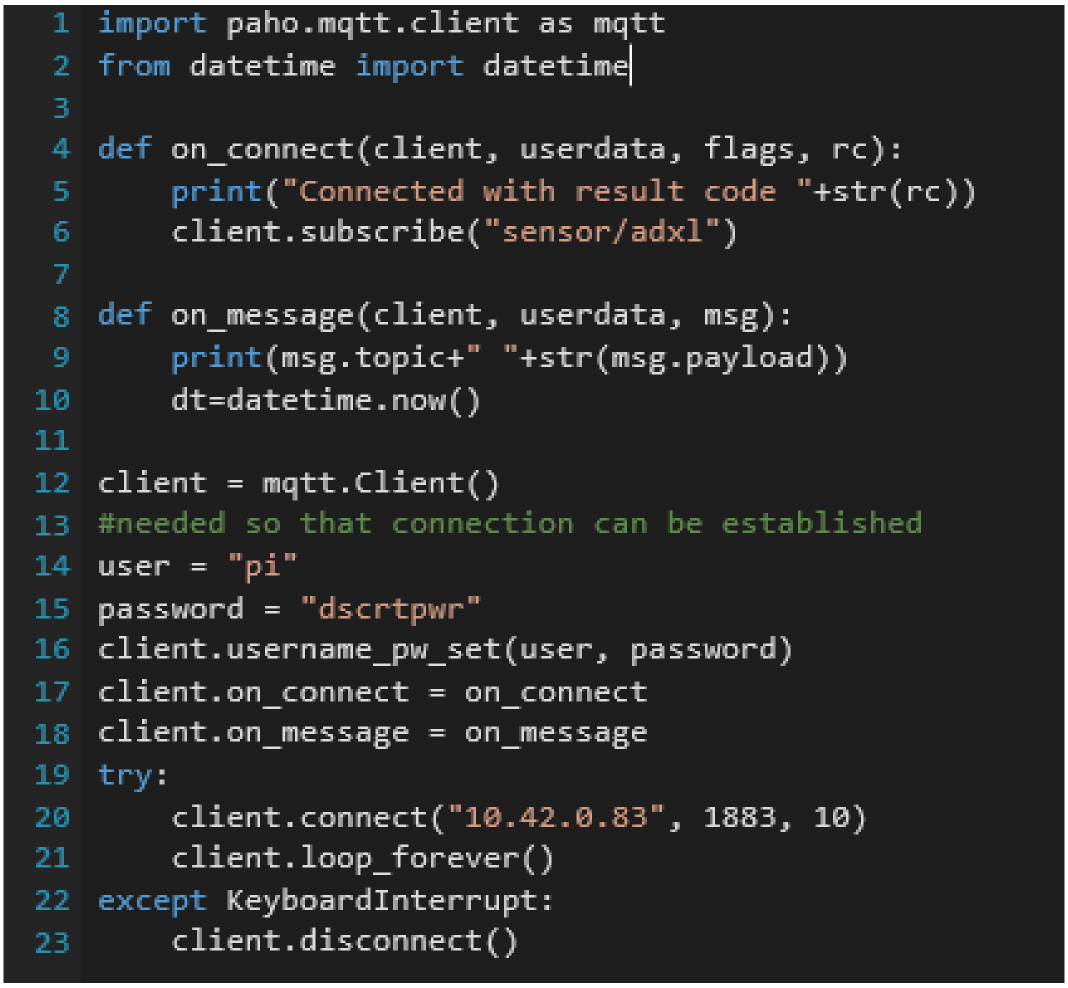


```
import paho.mqtt.client as mqtt

_g_cst_ToMQTTTopicServerIP = "10.42.0.83"
_g_cst_ToMQTTTopicServerPort = 1883 #port
_g_cst_MQTTTopicName = "sensor/adxl" #TOPIC name
mqttc = mqtt.Client("python_pub")
```

Code above shows how to initialize paho-mqtt(mqtt python API) on publisher.

mqttc.connect(_g_cst_ToMQTTTopicServerIP,_g_cst_ToMQTTTopicServerPort) to connect to mosquito mqt broker
mqttc.publish(_g_cst_MQTTTopicName, “data”) to publish data

As for Subscriber we can use the code below:



```python
import paho.mqtt.client as mqtt
from datetime import datetime

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe("sensor/adxl")

def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))
    dt=datetime.now()

client = mqtt.Client()
#needed so that connection can be established
user = "pi"
password = "dscrtpwr"
client.username_pw_set(user, password)
client.on_connect = on_connect
client.on_message = on_message
try:
    client.connect("10.42.0.83", 1883, 10)
    client.loop_forever()
except KeyboardInterrupt:
    client.disconnect()
```

We assign the function on_connect to notify us that mqtt is connected and on_message to define how we manipulate the received data. Note that use and password here are the user and password for the broker (in this case it is our pi username and password).

Socket(Replace MQTT)

Socket is a faster transmission method compare to MQTT. Since we wants a achieve high frequency of 200Hz sampling rate, we decided to replace MQTT with socket.

mongoDB

We have tried both mysql and mongoDB they work at similar rate, but we found capped collection in mongoDB is very suitable for this project so we decided to use mongoDB.

SQL	MongoDB
database	database
table	collection
row	document
column	field

Table above show the difference between the words used in mysql and mongoDB.

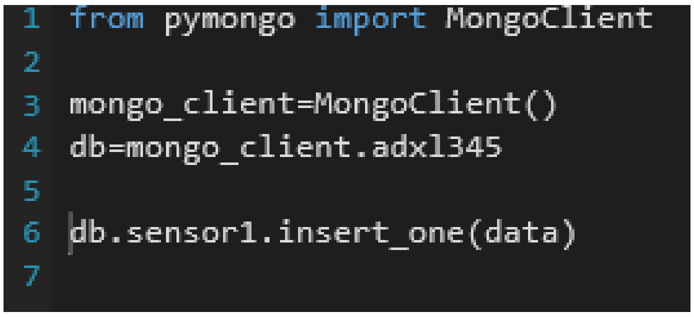
The advantage of using capped collection in mongoDB provide us a convenient vode writing. We don't need to control the size of the data in collection, instead mongoDB handle it well for us. The capped collection function like a queue structure, it push in the data from the end of collection and pop out the first data in collection to maintain the size of collection.

To use this concept we need to open the mongo command line and created capped collection by command:

```
>use “your_database”
>db.createCollection("your_cappedLogCollection",{capped:true,size:10000,max:
1000})
```

Where size is the size of collection in bytes and max as the maximum number of documents in collection. Then we uses the pymongo to manipulate our mongo database through python.

Below is and easy example of how to insert data in python:



```python
from pymongo import MongoClient

mongo_client=MongoClient()
db=mongo_client.adxl345

db.sensor1.insert_one(data)

```

Adxl345 as the database and sensor1 as the collection.

Bokeh

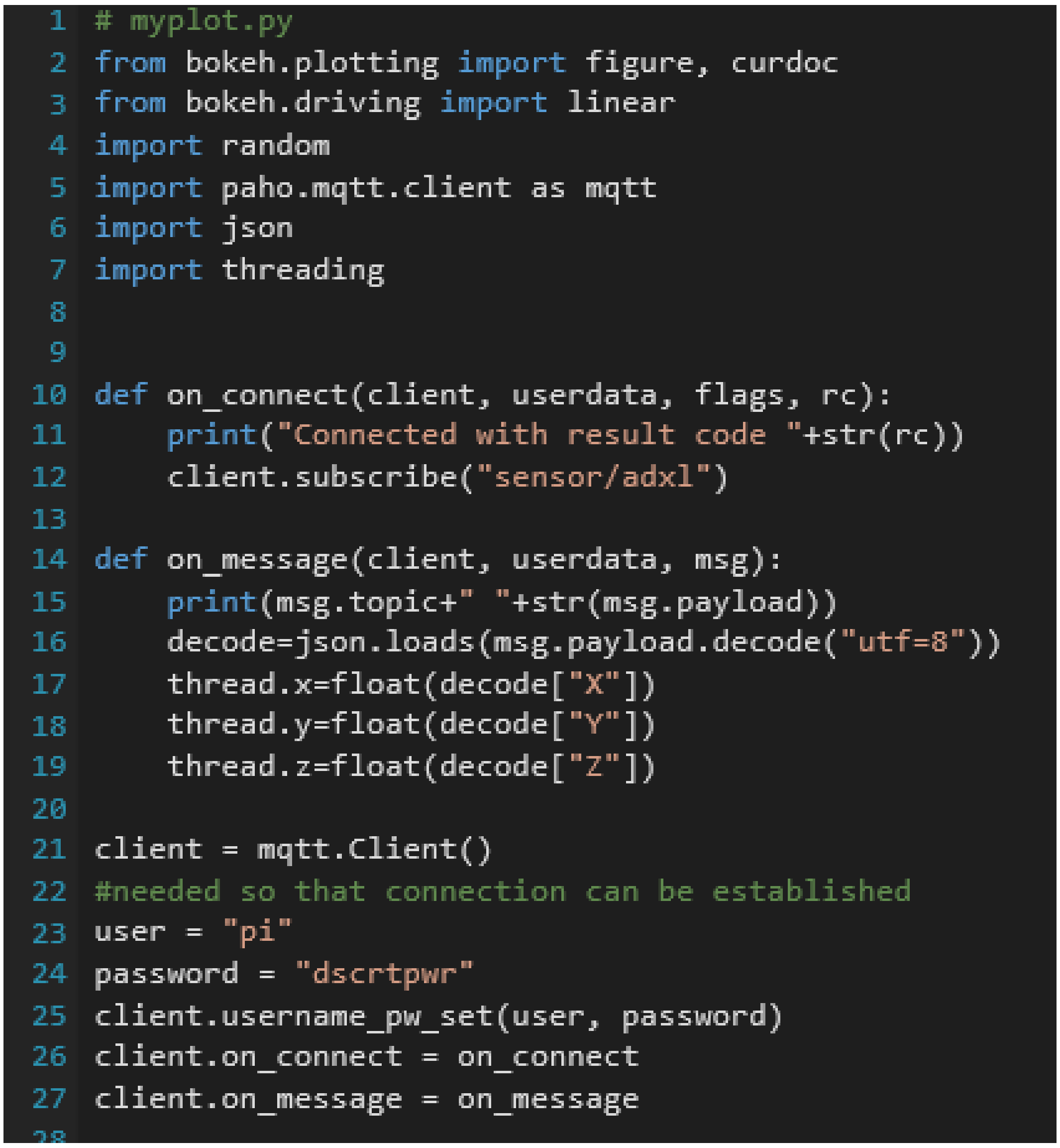
Bokeh is a python API that visualize data and is able to display on other client side through websocket. A bokeh server visualize data by creating bokeh application and the client-side bowser is able to see data visualization through the bokehJS. The application are written in python and we uses the command below to see the result locally:

```
>bokeh serve -show myapplication.py
```

If you want it to work over internet please add these prefixes:

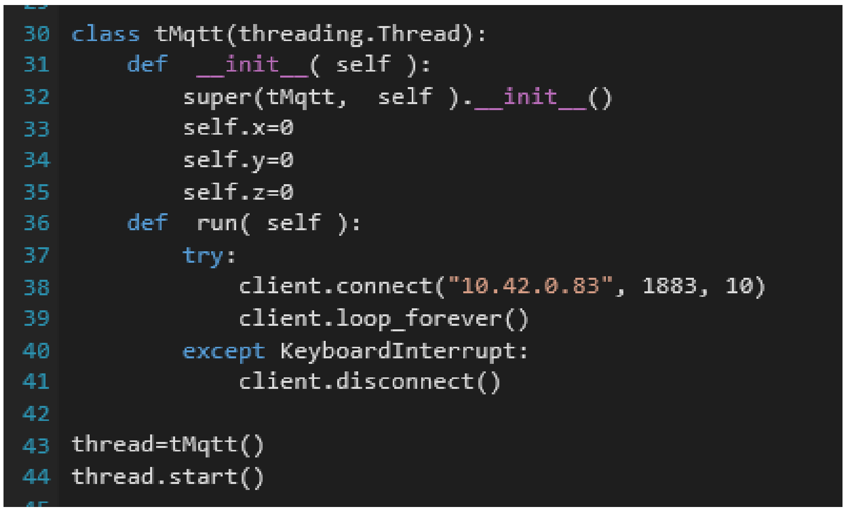
```
>bokeh serve --show --address [server ip] --allow-websocket-origin [server I
P]:[bokeh port] --allow-websocket-origin [allowed IP] myapplication.py
```

Below is some example on how to write a real-time streaming data plot application with mqtt:



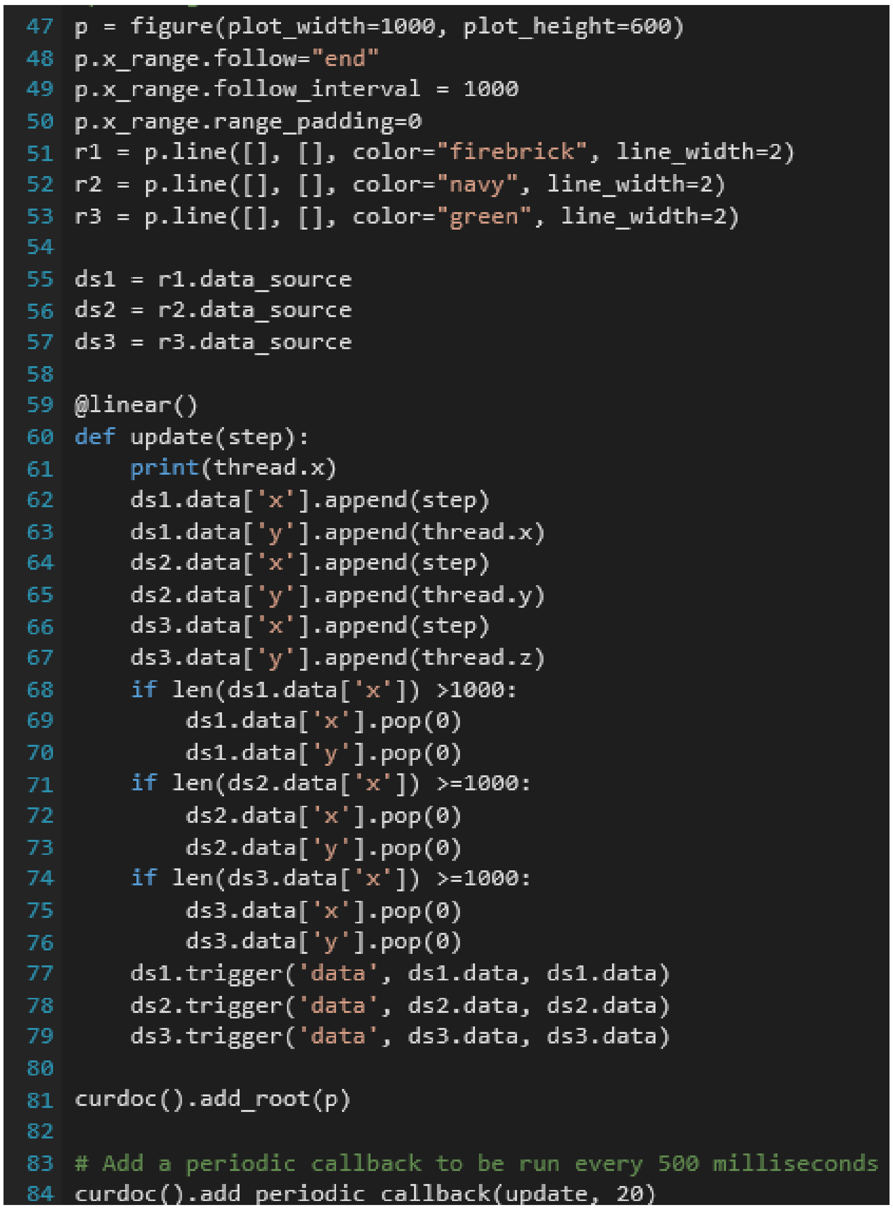
```python
# myplot.py
from bokeh.plotting import figure, curdoc
from bokeh.driving import linear
import random
import paho.mqtt.client as mqtt
import json
import threading


def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe("sensor/adxl")

def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))
    decode=json.loads(msg.payload.decode("utf=8"))
    thread.x=float(decode["X"])
    thread.y=float(decode["Y"])
    thread.z=float(decode["Z"])

client = mqtt.Client()
#needed so that connection can be established
user = "pi"
password = "dscrtpwr"
client.username_pw_set(user, password)
client.on_connect = on_connect
client.on_message = on_message
```

Code above defined what to do when we received mqtt message, which is to update the x , y and z acceleration.



```python
class tMqtt(threading.Thread):
    def  __init__( self ):
        super(tMqtt,  self ).__init__()
        self.x=0
        self.y=0
        self.z=0
    def  run( self ):
        try:
            client.connect("10.42.0.83", 1883, 10)
            client.loop_forever()
        except KeyboardInterrupt:
            client.disconnect()

thread=tMqtt()
thread.start()
```

Next we uses thread so that we can update x,y,z acceleration and do real-time plotting simultaneously.



```python
p = figure(plot_width=1000, plot_height=600)
p.x_range.follow="end"
p.x_range.follow_interval = 1000
p.x_range.range_padding=0
r1 = p.line([], [], color="firebrick", line_width=2)
r2 = p.line([], [], color="navy", line_width=2)
r3 = p.line([], [], color="green", line_width=2)

ds1 = r1.data_source
ds2 = r2.data_source
ds3 = r3.data_source

@linear()
def update(step):
    print(thread.x)
    ds1.data['x'].append(step)
    ds1.data['y'].append(thread.x)
    ds2.data['x'].append(step)
    ds2.data['y'].append(thread.y)
    ds3.data['x'].append(step)
    ds3.data['y'].append(thread.z)
    if len(ds1.data['x']) >1000:
        ds1.data['x'].pop(0)
        ds1.data['y'].pop(0)
    if len(ds2.data['x']) >=1000:
        ds2.data['x'].pop(0)
        ds2.data['y'].pop(0)
    if len(ds3.data['x']) >=1000:
        ds3.data['x'].pop(0)
        ds3.data['y'].pop(0)
    ds1.trigger('data', ds1.data, ds1.data)
    ds2.trigger('data', ds2.data, ds2.data)
    ds3.trigger('data', ds3.data, ds3.data)

curdoc().add_root(p)

# Add a periodic callback to be run every 500 milliseconds
curdoc().add_periodic_callback(update, 20)
```

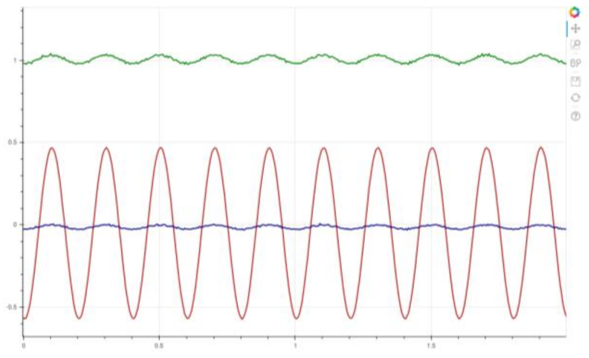
The rest of the code are for plotting, as you can do some configuration on the display screen and let it follow along the data while streaming. For convenient we limit the size of data to 1000.

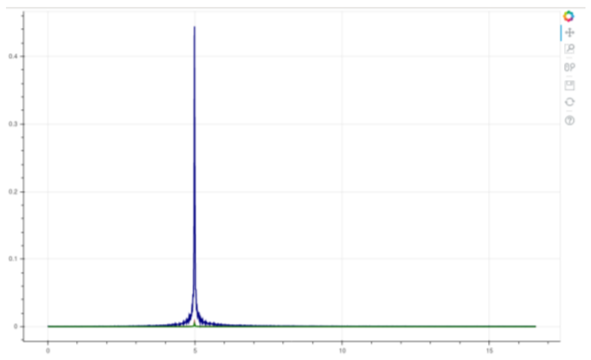
Clustering(kmeans)

The clustering is meant to analyze the collected data. Throuh clustering methods we can simply classify signal into **k** groups. The kmeans is initialized according to kmeans++ and then iterate until the error converges. We randomly choose one signal as initial centers, then we choose the next initial center, such that the sum of distance to closest center is smallest. With these initial centers we follow the pseudo code below to iterate:

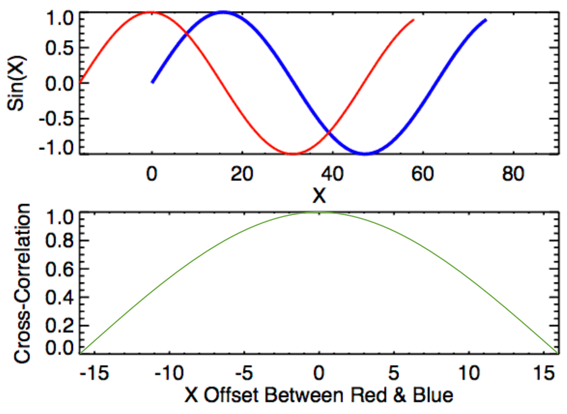
```
for each cluster center k in image:
        find the best center for each point
        compute new center(average)
        error=total distance between new and old center
stop until error converge
```

To apply kmeans to the signal data, we need to first define the distance (dissimilarity) between two signals. This is define using cross correlation, first we do fast Fourier transform of the signal along x, y and z axis.





Then we add zero padding to one of the signal(padding of size 10 before and after the signal), do cross relation on two signal according to their x, y and z Fourier transform.



The max value of cross relation will be the distance between two signal along an axis. We can either sum up the max value along x, y and z axis or using Euclidean function:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2}.$$

Python Code

adxl355_linkit.py
This the python code run on linkit7688 to transmit acceleration information to server through socket.

auto-generate.py
auto generate python code is for simulation purpose, it can create data of target frequency and amplitude with some noises to make the data more realistic.

bokeh_socket.py
bokeh server for receiving data from linkit7688 and plot the streaming data on web html. (For more information please refer to Bokeh category above)

bokeh_socket_fft.py
bokeh server for receiving data from linkit7688 and convert the streaming data to fast fourier transform and plot on web html . (For more information please refer to Bokeh category above, the window fo fft is 100 datas, in other word 5 seconds)

fft_csv.py
The data saved or simulated into csv will be converted to fast fourier transform and visualize using this code.

kmeans.py
provide a directory path with csv inside, using kmeans algorithm to classify the signal into k groups.

plot_csv.py
The data saved or simulated into csv can be visualize using this code.

socket_csv.py
Code receiving data from linkit7688 and saved into csv file.