LAB2: MOTION SENSING SYSTEM IMPLEMENTATION USING RASPBERRY PI

By: Pranati R Trivedi
ECE 442

Lab Date: 03-08-2019
Due Date: 03-15-2019

Acknowledgment: I acknowledge all of the work (including figures and codes) belongs to me and/or persons who are referenced.

Signature : ________________________

Purpose

In this lab, students are required to build a motion sensing system using a Raspberry Pi single-board computer, where the sensing data are transmitted to a server through a client-server communication. By using UDP connection between a Raspberry Pi and a server, encrypted data from the Raspberry Pi are sent to the server, where the data are decrypted and displayed in a graphical format.
The purpose of this experiment is to introduce the following items:
• Raspberry Pi single-board computer
• Adafruit ADXL345 triple-axis accelerometer
• Linux development environment
• I2C (Inter-Integrated Circuit) bus
• SPI (Serial Peripheral Interface) bus
• Client-server internet communication architecture

The purpose of this exercise is to introduce students with different methods of communication between a microcomputer and I/O devices using Raspberry Pi, a simple data encryption-decryption method using a symmetric key, and a client-server UDP connection.
2.0 Component Requirements
• 1 x Raspberry Pi 3 single-board computer
• 1 x Adafruit ADXL345 triple-axis accelerometer
• 1 x Desktop (as a data storage server)

a. Theory:

This experiment is divided into two tasks-

1) Implementation of real-time motion sensing device using I2C protocol

I2C is used to transfer data between master and slave devices. It uses two pins SDA and SCL. SCL is the clock line that is used to synchronize all data transfers over the bus. SDA is the data line that is used to specify the addresses and transfer data. Master has the task to drive the SCL clock line. Slaves always respond to the request of the master device. In this experiment, the accelerometer acts as a slave device while the Pi acts as a master device. Pi tells the slave to send the readings for the axes as an input data. This input data for all the three axes is received by master Pi and displayed and stored.

2) Implementation of real-time motion sensing device using SPI protocol and establishing server-client communication

The concept of SPI also lies in transferring data between master and slave devices. SPI uses 3-4 wires for connection. SCK, MOSI, MISO and SS are used. SCK is for the clock line which synchronizes all data transfers on the bus. MOSI is Master Out Slave In and is used for transferring data from master to slave devices. MISO is Master In Slave Out and is used for slave to master. SS is slave select signal generated by master device, slave device with logic 0. Master initiates the communication by driving SS of slave to low. First byte sent by master will have the MSB for writing. In second byte, slave will respond to master by sending data from register on to the master. The Pi will initiate the communication and act as master while the slave accelerometer will send the data to the master.

This data is sent to server using serve-client communication. The Pi acts as the client and the PC acts as the server. The data is encrypted using AES and transmitted over the internet. The server decrypts the data using the same key as encryption and then uses it. The initialization vector for this encryption-decryption are already specified for this project.

b. **Preliminary work**

1)

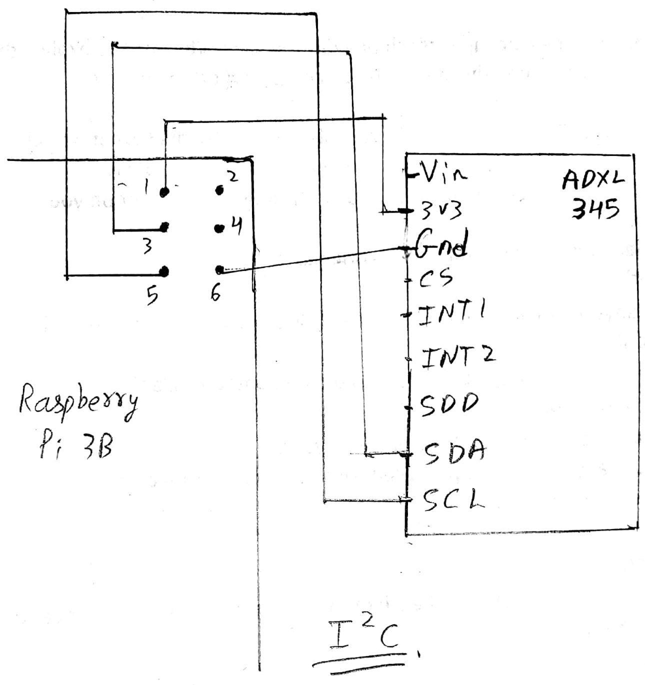


Fig.1 schematic for I2C connection of Pi and accelerometer

2)

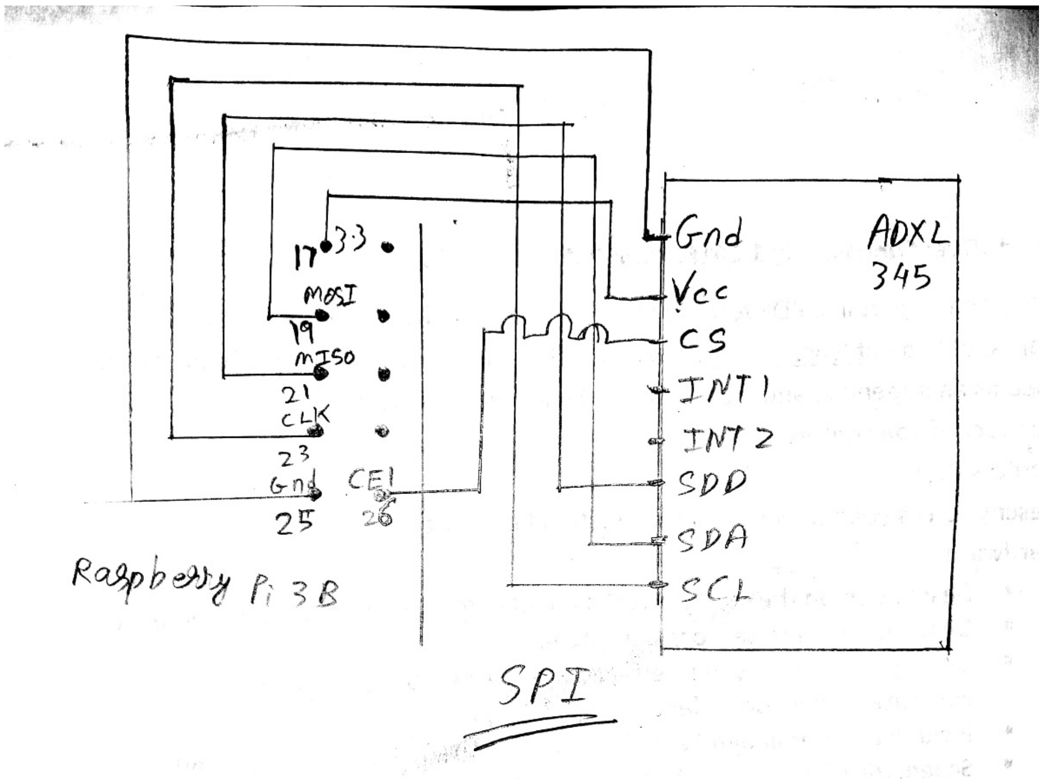


Fig.2 schematic for SPI connection of Pi and accelerometer

Experimental procedure:

a. Schematic:

Please refer to the schematics above.

b. Procedure:

The connection of accelerometer with raspberry Pi is as follows:

I2C connection:

Raspberry pi	**Accelerometer**
3.3V	3.3V
Gnd	Gnd
SDA	SDA
SCL	SCL

Connect both the devices using I2C connection. Program the Pi according to I2C connection. Now, execute the code to obtain the values of the three axes from accelerometer. The code is developed to store the values to an output text file.

SPI connection:

Raspberry Pi	Accelerometer
3.3V	3.3V
Gnd	Gnd
SPI_MOSI (pin 19)	SDA
SPI_MISO (pin 21)	SDO
SPIO_CLK (pin 23)	SCL
SPI_CE1_N (pin 26)	CS

Connect the devices using SPI connection. Program the Pi so as to receive the values for this SPI connection. The code developed receives the value for axes of accelerometer and the values are stored in an output text file.

c. All apparatus
1) Raspberry Pi 3B
2) Accelerometer ADXL345
3) Jumper wires
4) PC for server

d. Results:

1) I2C connection results:

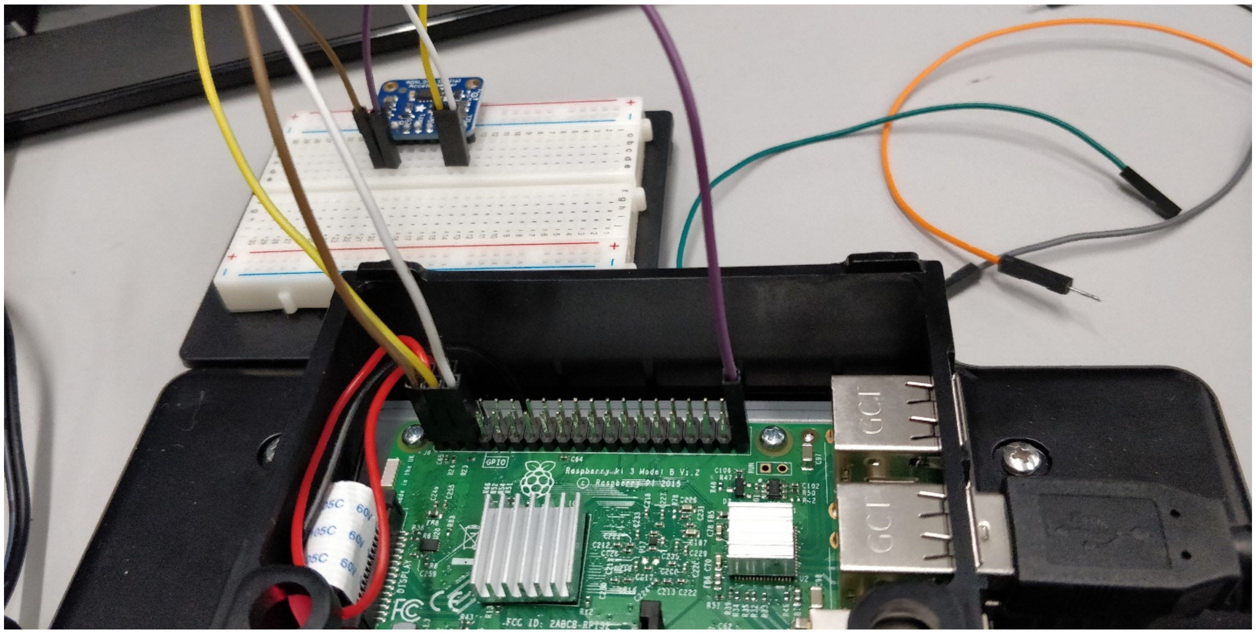


Fig.3 I2C connection with Pi

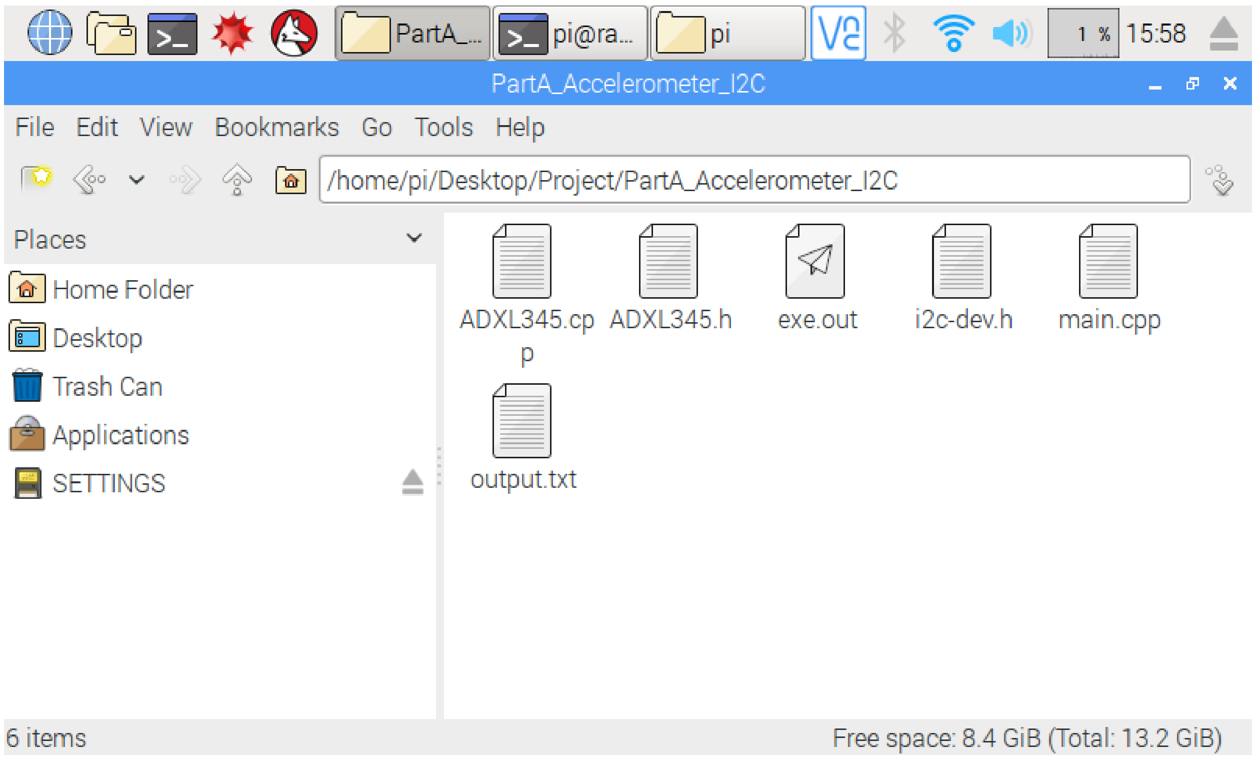
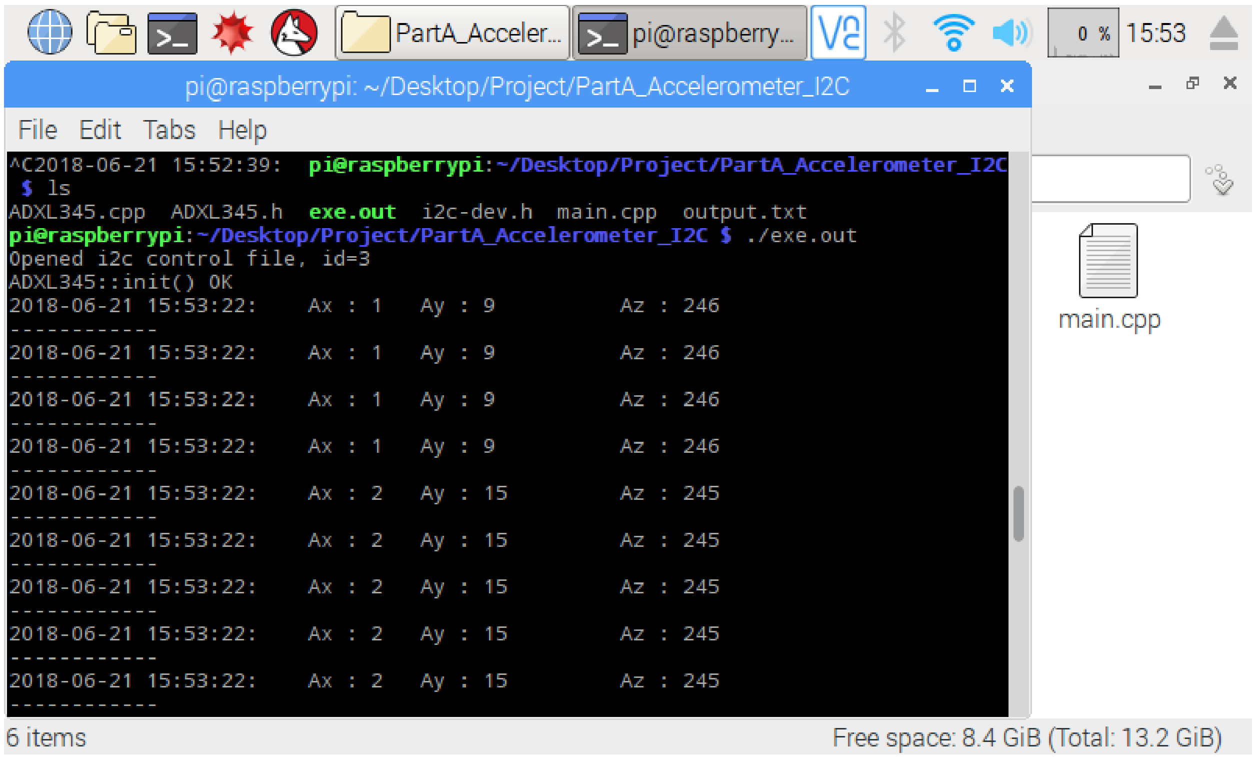
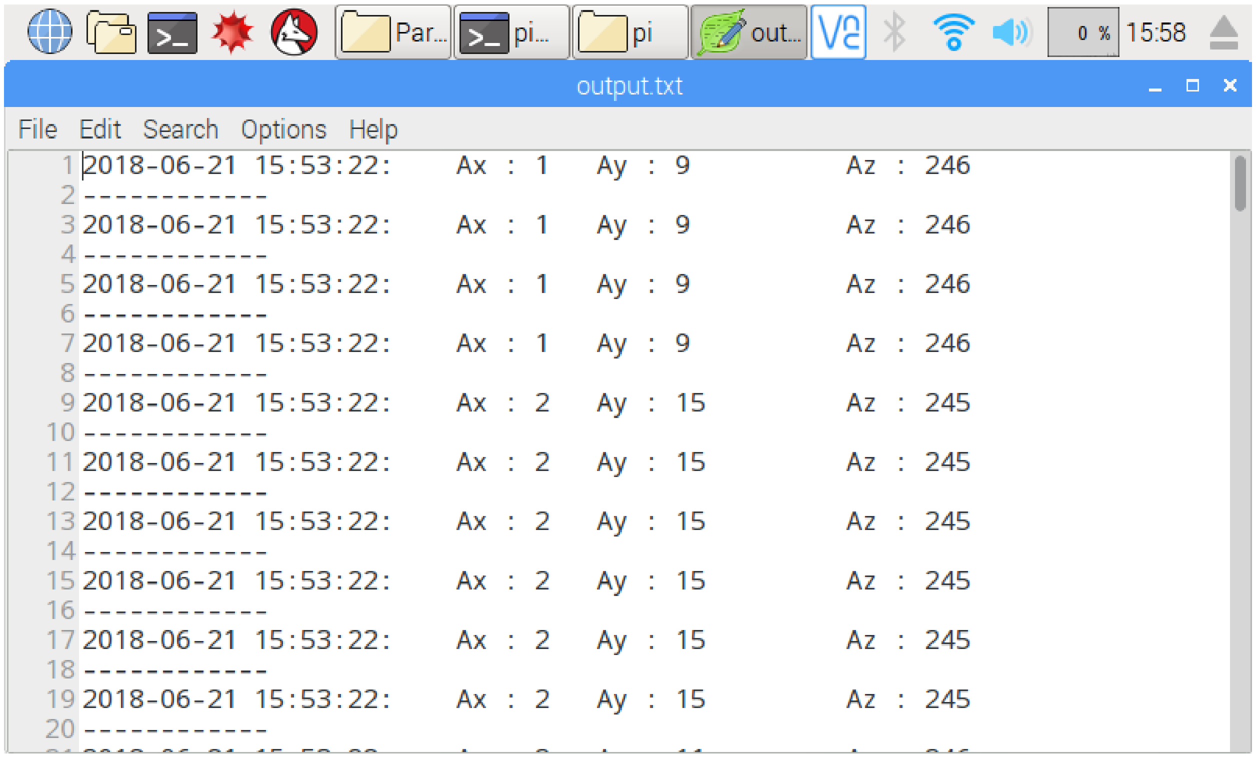


Fig.4 generated exe.out file


PartA_Acceler...
pi@raspberry...
0 %
15:53
pi@raspberrypi: ~/Desktop/Project/PartA_Accelerometer_I2C
File Edit Tabs Help
main.cpp
6 items
Free space: 8.4 GiB (Total: 13.2 GiB)

```
^C2018-06-21 15:52:39:  pi@raspberrypi:~/Desktop/Project/PartA_Accelerometer_I2C
 $ ls
ADXL345.cpp  ADXL345.h  exe.out  i2c-dev.h  main.cpp  output.txt
pi@raspberrypi:~/Desktop/Project/PartA_Accelerometer_I2C $ ./exe.out
Opened i2c control file, id=3
ADXL345::init() OK
2018-06-21 15:53:22:    Ax : 1   Ay : 9          Az : 246
------------
2018-06-21 15:53:22:    Ax : 1   Ay : 9          Az : 246
------------
2018-06-21 15:53:22:    Ax : 1   Ay : 9          Az : 246
------------
2018-06-21 15:53:22:    Ax : 1   Ay : 9          Az : 246
------------
2018-06-21 15:53:22:    Ax : 2   Ay : 15         Az : 245
------------
2018-06-21 15:53:22:    Ax : 2   Ay : 15         Az : 245
------------
2018-06-21 15:53:22:    Ax : 2   Ay : 15         Az : 245
------------
2018-06-21 15:53:22:    Ax : 2   Ay : 15         Az : 245
------------
2018-06-21 15:53:22:    Ax : 2   Ay : 15         Az : 245
------------
```

Fig.5 output of accelerometer using I2C connection



```
2018-06-21 15:53:22:     Ax : 1    Ay : 9           Az : 246
------------
2018-06-21 15:53:22:     Ax : 1    Ay : 9           Az : 246
------------
2018-06-21 15:53:22:     Ax : 1    Ay : 9           Az : 246
------------
2018-06-21 15:53:22:     Ax : 1    Ay : 9           Az : 246
------------
2018-06-21 15:53:22:     Ax : 2    Ay : 15          Az : 245
------------
2018-06-21 15:53:22:     Ax : 2    Ay : 15          Az : 245
------------
2018-06-21 15:53:22:     Ax : 2    Ay : 15          Az : 245
------------
2018-06-21 15:53:22:     Ax : 2    Ay : 15          Az : 245
------------
2018-06-21 15:53:22:     Ax : 2    Ay : 15          Az : 245
------------
2018-06-21 15:53:22:     Ax : 2    Ay : 15          Az : 245
------------
```

Fig.6 accelerometer data values stored in output.txt (I2C)

2) <u>SPI connection results:</u>

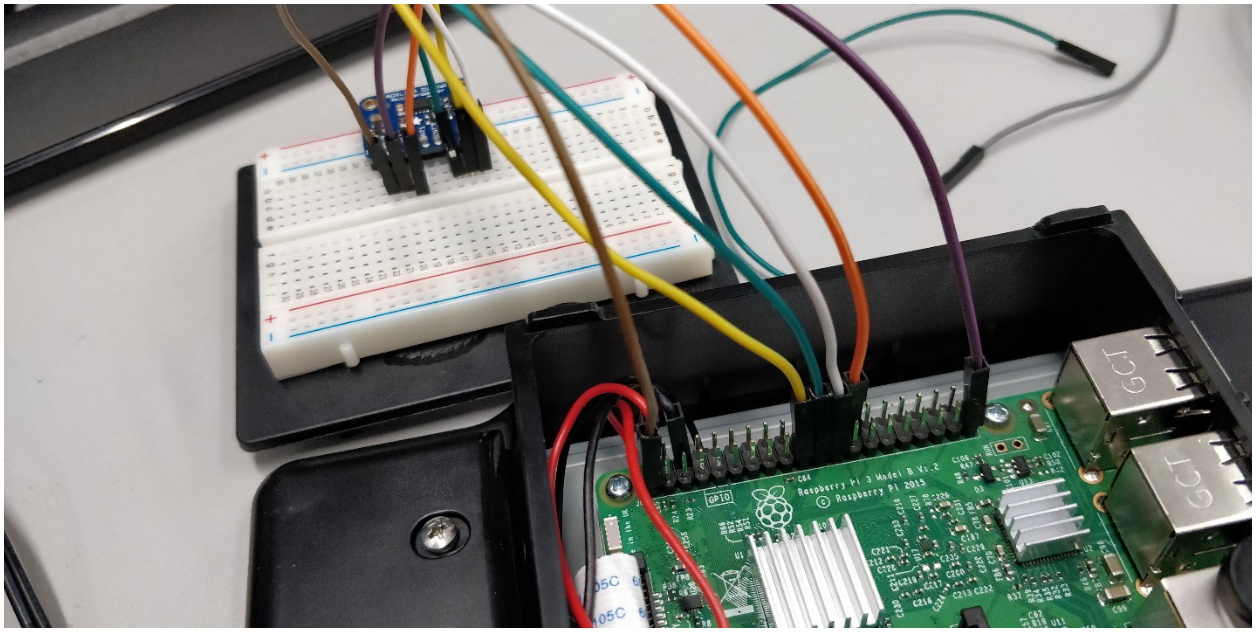
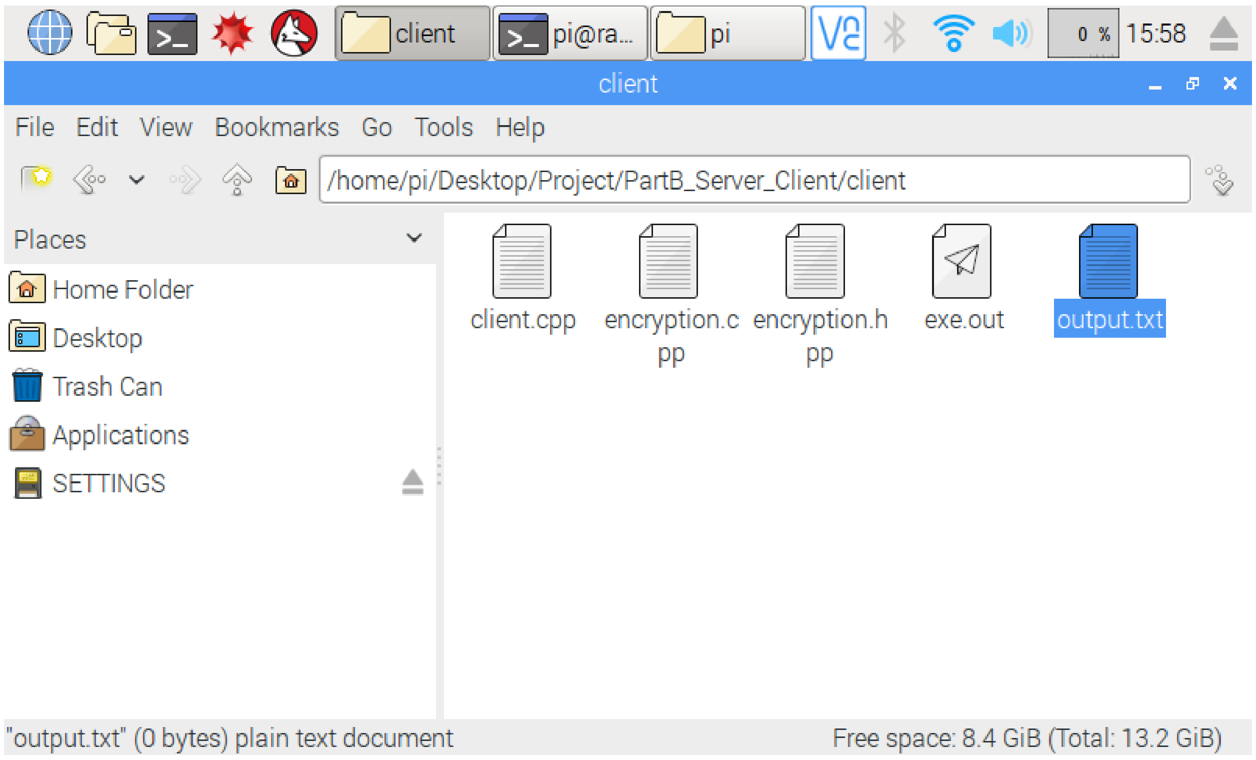


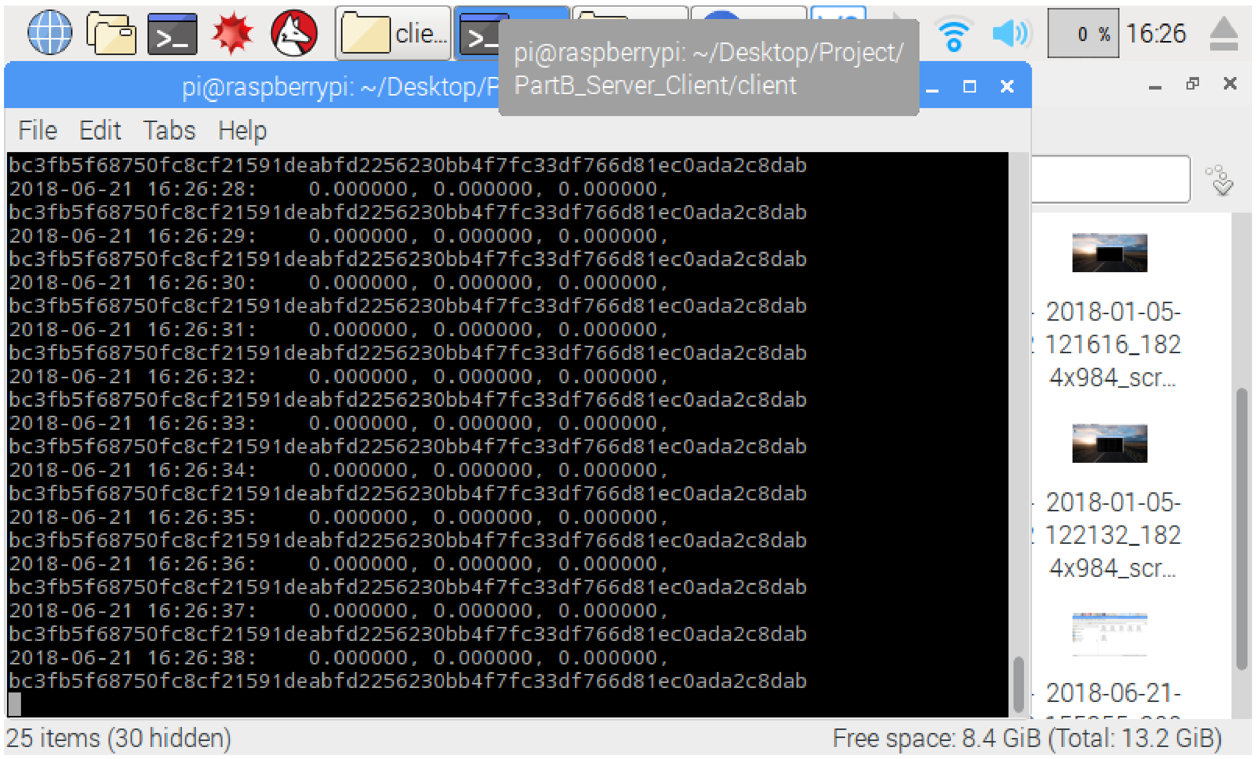
Fig.7 SPI connection with Pi



/home/pi/Desktop/Project/PartB_Server_Client/client

"output.txt" (0 bytes) plain text document   Free space: 8.4 GiB (Total: 13.2 GiB)

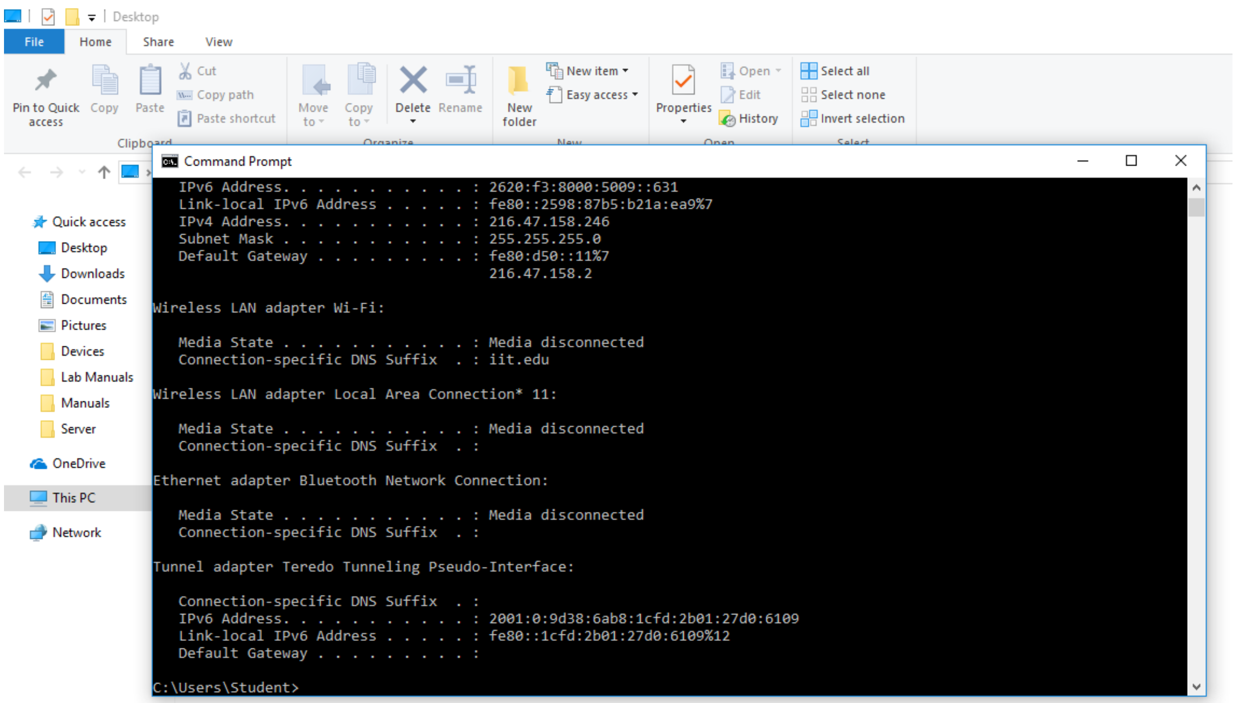
Fig.8 generated exe.out file


clie...
16:26
pi@raspberrypi: ~/Desktop/Project/PartB_Server_Client/client
File Edit Tabs Help
2018-01-05-121616_1824x984_scr...
2018-01-05-122132_1824x984_scr...
2018-06-21-

```
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
2018-06-21 16:26:28:    0.000000, 0.000000, 0.000000,
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
2018-06-21 16:26:29:    0.000000, 0.000000, 0.000000,
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
2018-06-21 16:26:30:    0.000000, 0.000000, 0.000000,
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
2018-06-21 16:26:31:    0.000000, 0.000000, 0.000000,
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
2018-06-21 16:26:32:    0.000000, 0.000000, 0.000000,
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
2018-06-21 16:26:33:    0.000000, 0.000000, 0.000000,
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
2018-06-21 16:26:34:    0.000000, 0.000000, 0.000000,
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
2018-06-21 16:26:35:    0.000000, 0.000000, 0.000000,
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
2018-06-21 16:26:36:    0.000000, 0.000000, 0.000000,
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
2018-06-21 16:26:37:    0.000000, 0.000000, 0.000000,
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
2018-06-21 16:26:38:    0.000000, 0.000000, 0.000000,
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
```

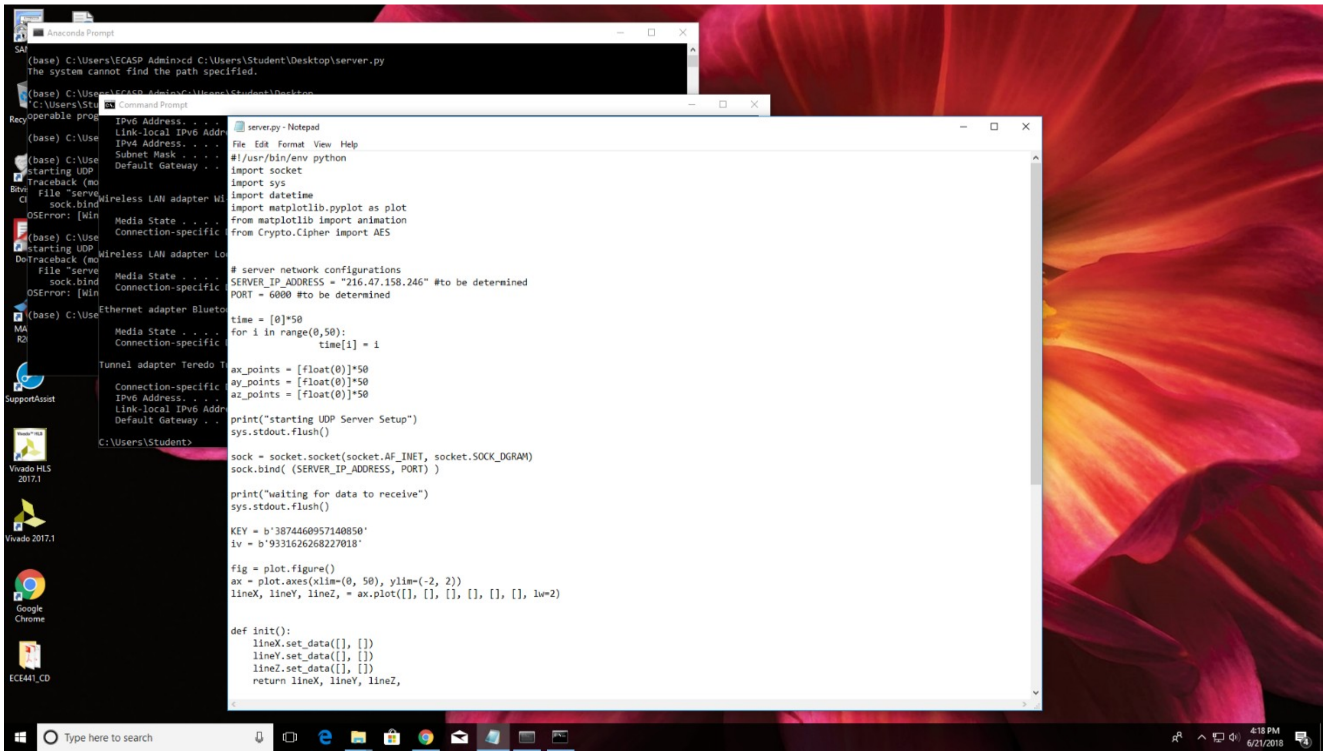
25 items (30 hidden)   Free space: 8.4 GiB (Total: 13.2 GiB)

Fig.9 output for accelerometer using SPI connection



```
   IPv6 Address. . . . . . . . . . . : 2620:f3:8000:5009::631
   Link-local IPv6 Address . . . . . : fe80::2598:87b5:b21a:ea9%7
   IPv4 Address. . . . . . . . . . . : 216.47.158.246
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80:d50::11%7
                                       216.47.158.2

Wireless LAN adapter Wi-Fi:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . : iit.edu

Wireless LAN adapter Local Area Connection* 11:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Tunnel adapter Teredo Tunneling Pseudo-Interface:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2001:0:9d38:6ab8:1cfd:2b01:27d0:6109
   Link-local IPv6 Address . . . . . : fe80::1cfd:2b01:27d0:6109%12
   Default Gateway . . . . . . . . . :

C:\Users\Student>
```

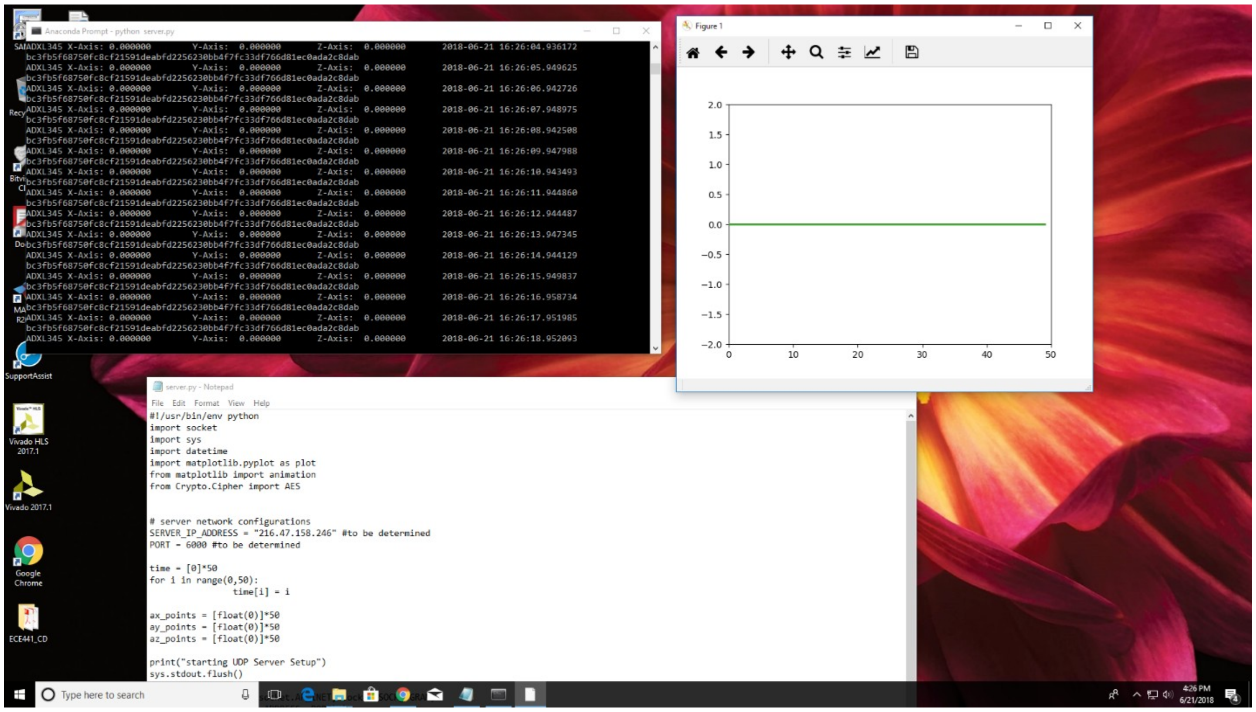
Fig.10 IP address for server



```
(base) C:\Users\ECASP Admin>cd C:\Users\Student\Desktop\server.py
The system cannot find the path specified.
```

```
#!/usr/bin/env python
import socket
import sys
import datetime
import matplotlib.pyplot as plot
from matplotlib import animation
from Crypto.Cipher import AES


# server network configurations
SERVER_IP_ADDRESS = "216.47.158.246" #to be determined
PORT = 6000 #to be determined

time = [0]*50
for i in range(0,50):
                time[i] = i

ax_points = [float(0)]*50
ay_points = [float(0)]*50
az_points = [float(0)]*50

print("starting UDP Server Setup")
sys.stdout.flush()

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind( (SERVER_IP_ADDRESS, PORT) )

print("waiting for data to receive")
sys.stdout.flush()

KEY = b'3874460957140850'
iv = b'9331626268227018'

fig = plot.figure()
ax = plot.axes(xlim=(0, 50), ylim=(-2, 2))
lineX, lineY, lineZ, = ax.plot([], [], [], [], [], [], lw=2)


def init():
    lineX.set_data([], [])
    lineY.set_data([], [])
    lineZ.set_data([], [])
    return lineX, lineY, lineZ,
```

Fig.11 assigning IP address of server to the code


Anaconda Prompt - python server.py

```
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:04.936172
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:05.949625
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:06.942726
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:07.948975
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:08.942508
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:09.947988
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:10.943493
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:11.944860
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:12.944487
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:13.947345
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:14.944129
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:15.949837
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:16.958734
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:17.951985
bc3fb5f68750fc8cf21591deabfd2256230bb4f7fc33df766d81ec0ada2c8dab
ADXL345 X-Axis: 0.000000        Y-Axis:  0.000000        Z-Axis:  0.000000        2018-06-21 16:26:18.952093
```

server.py - Notepad

```
#!/usr/bin/env python
import socket
import sys
import datetime
import matplotlib.pyplot as plot
from matplotlib import animation
from Crypto.Cipher import AES


# server network configurations
SERVER_IP_ADDRESS = "216.47.158.246" #to be determined
PORT = 6000 #to be determined

time = [0]*50
for i in range(0,50):
                time[i] = i

ax_points = [float(0)]*50
ay_points = [float(0)]*50
az_points = [float(0)]*50

print("starting UDP Server Setup")
sys.stdout.flush()
```

Fig.12 accelerometer data displayed on server PC

e. <u>Discussion</u>

1. <u>Refer appendix A1 for the code</u>
2.

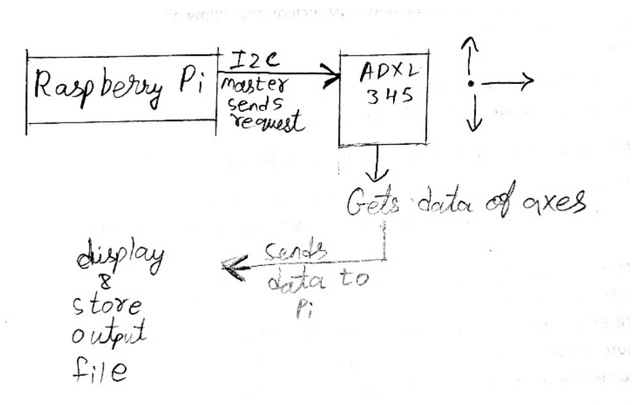


Fig.13 flow chart for I2C operation

Raspberry Pi acts as a master and using I2C connection sends the request to slave accelerometer to send the data. The accelerometer SCL clock line takes care of the synchronization between Pi and accelerometer. The SDA data line is responsible for sending data to the Pi. On the request of master, it sends data of the coordinates of the axes to the Pi. The Pi displays it on the screen and also stores the values in an output file.

3. For an I2C type of connection. The SDA pin of the slave device is generally used which takes care of the data transmission. But if more than one slave devices are connected than SDO pin of the other accelerometer acts as the alternate pin for the I2C protocol. Pi determines slave devices using which of these pins send the data to it.

   For a SPI type of connection, all other connections can be mutual between two or more slaves with the Pi. But one connection of CS (chip select) is different for different slave sensors. On the Pi also, the CS pin of each slave accelerometer is connected to different pin of Pi. So, Pi can easily differentiate between two slave devices using which CS pin is connected.

   To differentiate between an I2c slave and SPI slave, Pi can determine it because the pins used for I2C connection and SPI connection are different. SPI uses Pi's MISO, MOSI, pins also. While I2C just uses SCL and SDA pins for the Pi. So the connections being different the protocol being different it is easy for the Pi to identify the slaves whether are connected to I2C or SPI.

4. The register 0x31 is for data format control, to read or write. The 0x0A activates the D3 bit that is Full resolution bit to set. In this mode, output resolution increases the g range set by the range bits to maintain a 4 mg/LSB scale factor. It also activates the D1 bit for range. It signifies that it can take a range value from -8g to +8g.

5. Sample rate of PC: 44.1 khz, 44100 samples per second.

   And there are different sampling rate according to different peripheral for RPI.
   Here for I2C bus bandwidth sampling rate in theory would be 400000/27 = 14814 samples per second. Because to transfer 2 bytes through bus needs atleast 27 bits (address+2 data bytes).
   For the maximum clock rate is 3.6 Mhz when running at 5V. The device requires 18 clocks per data samples (8 for addr. and setup and 10 for data) .
   So 3.6 Mhz/ 18 = 200k samples per second.
   Sampling Rate for SPI = 3.6 Mhz/ 24= 150ksps

Interpretation:
It is evident from the results that, the output received to the server are as expected. There are no errors found in performing this experiment. The accelerometer is successfully able to send the axes data to the Pi and then to the server. The server also received the values correctly in real time and displayed the values on the screen and on the graph as expected.

Conclusion:
It proves that this method is good for sending sensor data and reliable too. It can be accessed from the server and then the data can be used for manipulation and controlling other sensors and devices. This serves the purpose of internet of things.

Appendix

Appendix A1:
Code for I2C connection:
Main.cpp

```
/**********************
Use example and information in Appendix A in Lab 10 to finish this file
**********************/
#include<stdio.h>
#include<signal.h>
#include<sys/time.h>

#include "i2c-dev.h"
#include "ADXL345.h"

#define I2C_FILE_NAME "dev/i2c-1" // for Rpi B+
void INThandler(int sig);

int main(int argc, char **argv)
{
	//Open a connection to the I2C userspace control file.
	int i2c_fd = open(I2C_FILE_NAME,O_RDWR);
	if(i2c_fd <0)
	{
		print("Unable to open i2c control file, err=%d\n", i2c_fd);
		exit(1);
	}

	printf("Opened i2c control file, id=%d\n", i2c_fd);
	ADXL345 myAcc(i2c_fd);
	int ret = myAcc.init();
	if (ret)
	{
		printf("fialed init ADXL345, ret=%d\n", ret);
		exit(1);
	}
	usleep(100 * 1000);

	signal(SIGNT, INThandler);
	short ax, ay, az;
	//create file IO
	FILE *fp;
	fp = fopen("./output.txt","w+");

	char TimeString[128];
	timeval curTime;
	while(1)
	{
		// get the current time
		gettimeoftheday(&curTime, NULL);
		strftime(TimeString, B0, "%Y-%m-%d %H:%M:%S",
localtime(&curTime.tv_sec);
		printf(TimeString);
		printf(": ");
		// now, fetch data from sensor
		myAcc.readXYZ(ax, ay, az);
		//print to screen
		printf("Ax : %hi \t Ay : %hi \t Az : %hi \n", ax,ay,az);
		printf("----------------------\n");
		//print to file
		fprintf(fp,TimeString);
		fprintf(fp, ": ");
		fprintf(fp, "Ax : %hi \t Ay : %hi \t Az : %hi \n", ax,ay,az);
		fprintf(fp, "----------------------\n");
		if (getchar() == 'q') break;
	}

	fclose(fp);

	return 0;
}

void INThandler(int sig)
{
	signal(sig, SIG_IGN);
	exit(0);
}
```

ADXL345.cpp

```
/*
Basic readout of ADXL345 accelerometer via I2C

Oryginal code taken from the very bottom of this page:
http://www.raspberrypi.org/forums/viewtopic.php?t=55834

Updated by Jan Balewski, August 2014
*/

#include <assert.h>
#include "ADXL345.h"

//==========================================
//==========================================
bool ADXL345::selectDevice(){
   if (ioctl(fd, I2C_SLAVE, myAddr) < 0) {
      fprintf(stderr, "device ADXL345 not present\n");
      return false;
   }
   return true;
}

//==========================================
//==========================================
bool ADXL345::writeToDevice(char * buf, int len){
   if (write(fd, buf, len) != len)   {
     fprintf(stderr, "Can't write to device ADXL345 buf=%s len=%d\n",fd,buf,len);
     return false;
   }
  return true;
}


//===============
bool  ADXL345::readXYZ( short &x , short &y, short &z) {
  assert(fd>0); // crash if port was not opened earlier
  if(!selectDevice())   return false;
  //   printf("selectDevice(fd,ADXL345...)  passed\n");
  char buf[7];
  buf[0] = 0x32;     // This is the register we wish to read from
  if(!writeToDevice(buf,2))     return false;

  if (read(fd, buf, 6) != 6) {  // Read back data into buf[]
    printf("Unable to read from slave for ADXL345\n");
    return false;
  }  else {
    x = (buf[1]<<8) |  buf[0];
    y = (buf[3]<<8) |  buf[2];
    z = (buf[5]<<8) |  buf[4];
  }
  return true;
}


//=========================================
//=========================================
//=========================================
int ADXL345::init()  {
  assert(fd>0); // crash if port was not opened earlier
  char buf[6];        // Buffer for data being read/ written on the i2c bus

  if(!selectDevice()) return -1;

  buf[0] = 0x2d;                    // Commands for performing a ranging
  buf[1] = 0x18;

  if(!writeToDevice(buf,2))  return -2;

   buf[0] = 0x31;               // Commands for performing a ranging
   buf[1] = 0x0A; //09 4g , A 8g

   if(!writeToDevice(buf,2))  return -3;
   printf("ADXL345::init() OK\n");
   return 0;
}
```

ADXL345.h

```
/*
Basic readout of ADXL345 accelerometer via I2C bus

Oryginal code taken from the very bottom of this page:
http://www.raspberrypi.org/forums/viewtopic.php?t=55834

Updated by Jan Balewski, August 2014
*/

#ifndef ADXL345_HH
#define ADXL345_HH
#include <stdio.h>
#include <stdlib.h>
#include "i2c-dev.h"
#include <fcntl.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <iostream>


class ADXL345 {
 public:
  ADXL345 (int fdx, unsigned char addx=0x53) { fd=fdx; myAddr=addx;}
  int init();
  bool readXYZ(short &ax , short &ay, short &az);

 private:
  bool selectDevice();
  bool writeToDevice(char * buf, int len);

  unsigned char myAddr;
  int fd;// File descriptor
};
#endif
```

I2c-dev.h

```
/*
   i2c-dev.h - i2c-bus driver, char device interface

*/

#ifndef LIB_I2CDEV_H
#define LIB_I2CDEV_H

#include <linux/types.h>
#include <sys/ioctl.h>
//#include <linux/i2c-dev.h>

/* -- i2c.h -- */



/*
 * I2C Message - used for pure i2c transaction, also from /dev interface
 */
struct i2c_msg {
	__u16 addr;	/* slave address			*/
	unsigned short flags;
#define I2C_M_TEN	0x10	/* we have a ten bit chip address	*/
#define I2C_M_RD	0x01
#define I2C_M_NOSTART	0x4000
#define I2C_M_REV_DIR_ADDR	0x2000
#define I2C_M_IGNORE_NAK	0x1000
#define I2C_M_NO_RD_ACK		0x0800
	short len;		/* msg length				*/
	char *buf;		/* pointer to msg data			*/
};

/* To determine what functionality is present */

#define I2C_FUNC_I2C			0x00000001
#define I2C_FUNC_10BIT_ADDR		0x00000002
#define I2C_FUNC_PROTOCOL_MANGLING	0x00000004 /*
I2C_M_{REV_DIR_ADDR,NOSTART,..} */
#define I2C_FUNC_SMBUS_PEC		0x00000008
#define I2C_FUNC_SMBUS_BLOCK_PROC_CALL	0x00008000 /* SMBus 2.0 */
#define I2C_FUNC_SMBUS_QUICK		0x00010000
#define I2C_FUNC_SMBUS_READ_BYTE	0x00020000
#define I2C_FUNC_SMBUS_WRITE_BYTE	0x00040000
#define I2C_FUNC_SMBUS_READ_BYTE_DATA	0x00080000
#define I2C_FUNC_SMBUS_WRITE_BYTE_DATA	0x00100000
#define I2C_FUNC_SMBUS_READ_WORD_DATA	0x00200000
```

```
#define I2C_FUNC_SMBUS_WRITE_WORD_DATA       0x00400000
#define I2C_FUNC_SMBUS_PROC_CALL          0x00800000
#define I2C_FUNC_SMBUS_READ_BLOCK_DATA       0x01000000
#define I2C_FUNC_SMBUS_WRITE_BLOCK_DATA 0x02000000
#define I2C_FUNC_SMBUS_READ_I2C_BLOCK 0x04000000 /* I2C-like block xfer  */
#define I2C_FUNC_SMBUS_WRITE_I2C_BLOCK         0x08000000 /* w/ 1-byte reg. addr.
*/

#define I2C_FUNC_SMBUS_BYTE (I2C_FUNC_SMBUS_READ_BYTE | \
                         I2C_FUNC_SMBUS_WRITE_BYTE)
#define I2C_FUNC_SMBUS_BYTE_DATA (I2C_FUNC_SMBUS_READ_BYTE_DATA | \
                             I2C_FUNC_SMBUS_WRITE_BYTE_DATA)
#define I2C_FUNC_SMBUS_WORD_DATA (I2C_FUNC_SMBUS_READ_WORD_DATA | \
                             I2C_FUNC_SMBUS_WRITE_WORD_DATA)
#define I2C_FUNC_SMBUS_BLOCK_DATA (I2C_FUNC_SMBUS_READ_BLOCK_DATA | \
                              I2C_FUNC_SMBUS_WRITE_BLOCK_DATA)
#define I2C_FUNC_SMBUS_I2C_BLOCK (I2C_FUNC_SMBUS_READ_I2C_BLOCK | \
                             I2C_FUNC_SMBUS_WRITE_I2C_BLOCK)

/* Old name, for compatibility */
#define I2C_FUNC_SMBUS_HWPEC_CALC        I2C_FUNC_SMBUS_PEC

/*
 * Data for SMBus Messages
 */
#define I2C_SMBUS_BLOCK_MAX         32        /* As specified in SMBus standard */
#define I2C_SMBUS_I2C_BLOCK_MAX   32        /* Not specified but we use same structure */
union i2c_smbus_data {
          __u8 byte;
          __u16 word;
          __u8 block[I2C_SMBUS_BLOCK_MAX + 2]; /* block[0] is used for length */
                                         /* and one more for PEC */
};

/* smbus_access read or write markers */
#define I2C_SMBUS_READ 1
#define I2C_SMBUS_WRITE          0

/* SMBus transaction types (size parameter in the above functions)
   Note: these no longer correspond to the (arbitrary) PIIX4 internal codes! */
#define I2C_SMBUS_QUICK                       0
#define I2C_SMBUS_BYTE                1
#define I2C_SMBUS_BYTE_DATA     2
#define I2C_SMBUS_WORD_DATA              3
#define I2C_SMBUS_PROC_CALL     4
#define I2C_SMBUS_BLOCK_DATA              5
```

```
#define I2C_SMBUS_I2C_BLOCK_BROKEN  6
#define I2C_SMBUS_BLOCK_PROC_CALL   7               /* SMBus 2.0 */
#define I2C_SMBUS_I2C_BLOCK_DATA    8


/* ----- commands for the ioctl like i2c_command call:
 * note that additional calls are defined in the algorithm and hw
 *        dependent layers - these can be listed here, or see the
 *        corresponding header files.
 */
                                    /* -> bit-adapter specific ioctls          */
#define I2C_RETRIES          0x0701           /* number of times a device address      */
                                    /* should be polled when not           */
                        /* acknowledging                              */
#define I2C_TIMEOUT          0x0702           /* set timeout - call with int               */


/* this is for i2c-dev.c */
#define I2C_SLAVE   0x0703           /* Change slave address                              */
                                    /* Attn.: Slave address is 7 or 10 bits */
#define I2C_SLAVE_FORCE           0x0706           /* Change slave address
          */
                                    /* Attn.: Slave address is 7 or 10 bits */
                                    /* This changes the address, even if it */
                                    /* is already taken!                          */
#define I2C_TENBIT 0x0704           /* 0 for 7 bit addrs, != 0 for 10 bit      */

#define I2C_FUNCS  0x0705           /* Get the adapter functionality */
#define I2C_RDWR   0x0707           /* Combined R/W transfer (one stop only)*/
#define I2C_PEC              0x0708           /* != 0 for SMBus PEC                   */

#define I2C_SMBUS  0x0720           /* SMBus-level access */

/* -- i2c.h -- */


/* Note: 10-bit addresses are NOT supported! */

/* This is the structure as used in the I2C_SMBUS ioctl call */
struct i2c_smbus_ioctl_data {
          char read_write;
          __u8 command;
          int size;
          union i2c_smbus_data *data;
};
```

```
/* This is the structure as used in the I2C_RDWR ioctl call */
struct i2c_rdwr_ioctl_data {
        struct i2c_msg *msgs;/* pointers to i2c_msgs */
        int nmsgs;              /* number of i2c_msgs */
};


static inline __s32 i2c_smbus_access(int file, char read_write, __u8 command,
                        int size, union i2c_smbus_data *data)
{
        struct i2c_smbus_ioctl_data args;

        args.read_write = read_write;
        args.command = command;
        args.size = size;
        args.data = data;
        return ioctl(file,I2C_SMBUS,&args);
}


static inline __s32 i2c_smbus_write_quick(int file, __u8 value)
{
        return i2c_smbus_access(file,value,0,I2C_SMBUS_QUICK,NULL);
}

static inline __s32 i2c_smbus_read_byte(int file)
{
        union i2c_smbus_data data;
        if (i2c_smbus_access(file,I2C_SMBUS_READ,0,I2C_SMBUS_BYTE,&data))
                return -1;
        else
                return 0x0FF & data.byte;

}

static inline __s32 i2c_smbus_write_byte(int file, __u8 value)
{
return i2c_smbus_access(file,I2C_SMBUS_WRITE,value,
I2C_SMBUS_BYTE,NULL);
}

static inline __s32 i2c_smbus_read_byte_data(int file, __u8 command)
{
union i2c_smbus_data data;
if (i2c_smbus_access(file,I2C_SMBUS_READ,command,
I2C_SMBUS_BYTE_DATA,&data))
return -1;
else
return 0x0FF & data.byte;
}

static inline __s32 i2c_smbus_write_byte_data(int file, __u8 command,
                                              __u8 value)
{
union i2c_smbus_data data;
data.byte = value;
return i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
I2C_SMBUS_BYTE_DATA, &data);
}

static inline __s32 i2c_smbus_read_word_data(int file, __u8 command)
{
union i2c_smbus_data data;
if (i2c_smbus_access(file,I2C_SMBUS_READ,command,
I2C_SMBUS_WORD_DATA,&data))
return -1;
else
return 0x0FFFF & data.word;
}

static inline __s32 i2c_smbus_write_word_data(int file, __u8 command,
                                              __u16 value)
{
union i2c_smbus_data data;
data.word = value;
return i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
I2C_SMBUS_WORD_DATA, &data);
}

static inline __s32 i2c_smbus_process_call(int file, __u8 command, __u16 value)
{
union i2c_smbus_data data;
data.word = value;
if (i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
I2C_SMBUS_PROC_CALL,&data))
return -1;
else
return 0x0FFFF & data.word;
}


/* Returns the number of read bytes */
```

```
static inline __s32 i2c_smbus_read_block_data(int file, __u8 command,
                                              __u8 *values)
{
union i2c_smbus_data data;
int i;
if (i2c_smbus_access(file,I2C_SMBUS_READ,command,
I2C_SMBUS_BLOCK_DATA,&data))
return -1;
else {
for (i = 1; i <= data.block[0]; i++)
values[i-1] = data.block[i];
return data.block[0];
}
}

static inline __s32 i2c_smbus_write_block_data(int file, __u8 command,
                                               __u8 length, const __u8 *values)
{
union i2c_smbus_data data;
int i;
if (length > 32)
length = 32;
for (i = 1; i <= length; i++)
data.block[i] = values[i-1];
data.block[0] = length;
return i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
I2C_SMBUS_BLOCK_DATA, &data);
}

/* Returns the number of read bytes */
/* Until kernel 2.6.22, the length is hardcoded to 32 bytes. If you
ask for less than 32 bytes, your code will only work with kernels
2.6.23 and later. */
static inline __s32 i2c_smbus_read_i2c_block_data(int file, __u8 command,
                                                  __u8 length, __u8 *values)
{
union i2c_smbus_data data;
int i;

if (length > 32)
length = 32;
data.block[0] = length;
if (i2c_smbus_access(file,I2C_SMBUS_READ,command,
length == 32 ? I2C_SMBUS_I2C_BLOCK_BROKEN :
I2C_SMBUS_I2C_BLOCK_DATA,&data))
return -1;
else {
for (i = 1; i <= data.block[0]; i++)
values[i-1] = data.block[i];
return data.block[0];
}
}

static inline __s32 i2c_smbus_write_i2c_block_data(int file, __u8 command,
                                                   __u8 length,
                                                   const __u8 *values)
{
union i2c_smbus_data data;
int i;
if (length > 32)
length = 32;
for (i = 1; i <= length; i++)
data.block[i] = values[i-1];
data.block[0] = length;
return i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
I2C_SMBUS_I2C_BLOCK_BROKEN, &data);
}

/* Returns the number of read bytes */
static inline __s32 i2c_smbus_block_process_call(int file, __u8 command,
                                                 __u8 length, __u8 *values)
{
union i2c_smbus_data data;
int i;
if (length > 32)
length = 32;
for (i = 1; i <= length; i++)
data.block[i] = values[i-1];
data.block[0] = length;
if (i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
I2C_SMBUS_BLOCK_PROC_CALL,&data))
return -1;
else {
for (i = 1; i <= data.block[0]; i++)
values[i-1] = data.block[i];
return data.block[0];
}
}


#endif /* LIB_I2CDEV_H */
```
Appendix A2:

Encryption.cpp

```
#include "encryption.hpp"

int encrypt(const char * key, const char * iv, char * msg, char * msgCiphered)
{
  int key_length, iv_length, msg_length;
  key_length = strlen(key);
  iv_length = strlen(iv);
  msg_length = strlen(msg);

  const EVP_CIPHER *cipher;
  int cipher_key_length, cipher_iv_length;
  cipher = EVP_aes_128_cbc();
  cipher_key_length = EVP_CIPHER_key_length(cipher);
  cipher_iv_length = EVP_CIPHER_iv_length(cipher);

  if (key_length != cipher_key_length) {
    fprintf(stderr, "Error: key length must be %d, %d found\n", cipher_key_length, key_length);
    exit(EXIT_FAILURE);
  }
  if (iv_length != cipher_iv_length) {
    fprintf(stderr, "Error: iv length must be %d, %d found\n", cipher_iv_length, iv_length);
    exit(EXIT_FAILURE);
  }

  EVP_CIPHER_CTX ctx;
  int i, cipher_length, final_length;

  EVP_CIPHER_CTX_init(&ctx);
  EVP_EncryptInit_ex(&ctx, cipher, NULL, (unsigned char *)key, (unsigned char *)iv);
  cipher_length = msg_length + EVP_MAX_BLOCK_LENGTH;

  EVP_EncryptUpdate(&ctx, (unsigned char *)msgCiphered, &cipher_length, (unsigned char
*)msg, msg_length);
  EVP_EncryptFinal_ex(&ctx, (unsigned char *)msgCiphered + cipher_length, &final_length);

  return cipher_length + final_length;
}
```

Encrytption.hpp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/evp.h>

int encrypt(const char * key, const char * iv, char * msg, char * msgCiphered);
```

client.cpp

```
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <termios.h>
#include <fcntl.h>

#include <wiringPiSPT.h>

#include <netdb.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>

#include "encryption.hpp"

//specify the size of buffer and port for transmit

#define BUFFERSIZE 64
#define PORTNUMBER 50123

//specify key and IV
const char KEY[]= "3874460957140850";
const char IV[]= "9331626268227018";

//specify the address for server
const char hostname[] = "216.47.158.2";

//specify the channel used for SPI
const int spichannel(0);

void initialSPI()
{
	writingPiSPISetupMode(spichannel, 1000000, 3);
	usleep(10);
	unsigned char buf[2];
	//buf[0] = 0x80;
	//writingPiSPIDataRW(spichannel, (unsigned char *)&buf, 2);
	//std::cout << std::hex << (short)buf[1] << std::end1;
	//std::cout << "finished testing" << std::end1;

	//configure power
	buf[0] = 0x2D;
	buf[1] = 0x18;
	writingPiSPIDataRW(spichannel, (unsigned char *)&buf, 2);
	//std::cout << "finished setting up powerctl" << std::end1;

	//configure data format (Full_res, left justify, +-2g)
	buf[0]= 0x31;
	buf[1]= 0x00;
	writingPiSPIDataRW(spichannel, (unsigned char *)&buf, 2);
	//std::cout << "finished setting up dataformat" << std::end1;
	return;
}

void readRawXYZ(short &X, short &Y, short &Z)
{
	unsigned char txRxData[2];
	unsigned char buf[6];
	//read data
	for (unsigned short i(0); i<&; i++)
	{
		txRxData[0] = (unsigned char) ((unsigned short)0xB2 +i);
		writingPiSPIDataRW(spichannel, (unsigned char *)&txRxData, 2);
		buf[i] = txRxData[i];
	}

	X = (buf[1] << 8) | buf[0];
	Y = (buf[3] << 8) | buf[2];
	Z = (buf[5] << 8) | buf[4];

return;
}

void readXYZ(float &X, float &Y,float &Z, const short &scale = 2)
{
	short x_raw, y_raw, _raw;
	readRawXYZ(x_raw, y_raw, z_raw);
	X = (float x_raw / 1024 * scale);
	Y = (float y_raw / 1024 * scale);
	Z = (float z_raw / 1024 * scale);

	return;
```

```
}

//return true when keyboard been pressed, false other wise
bool kbhit(void)
{
        struct termios oldt, newt;
        int ch;
        int oldf;

        tcgetattr(STDIN_FILENO, &oldt);
        newt = oldt;
        newt.c_lflag &= ~(ICANON | ECHO);
        tcsetattr(STDIN_FILENO, TCSANOW, &newt);
        oldf = fcn1(STDIN_FILENO, f_GETFL, 0);
        fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);

        ch = getchar();

        tcsetattr(STDIN_FILENO, tcsanow, &oldt);
        fcntl(STDIN_FILENO, F_SETFL, oldf);

        if(ch != EOF)
        {
                ungetc(ch, stdin);
                return true;
        }

        return false;
}

int main()
{
        intitialSPI();
        float x, y, z;

        //verify the socket;
        int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
        if(sockfd <0)
        {
                printf("ERROR opening socket\n");
                exit(1);
        }

        //verify host name/address
        struct  hostent *server = gethostbyname(hostname);
        if(server == NULL)
        {
                printf("ERROR, no such host as %s\n",hostname);
                exit(1);
        }

        //Build the server internet address
        struct sockaddr_in serveraddr;
        bzero((char *) &serveraddr, sizeof(serveraddr));
        serveraddr.sin_family = AF_INET;

bcopy((char *}server->h_addr,(char *)&serveraddr.sin_addr.s_addr, server->h_length);
serveraddr.sin_port = htons(PORTNUMBER);

char bufRaw[BUFFERSIZE], bufCiphered[BUFFERSIZE *2];
while(!kbhit())
{
        readXYZ(x,y,z);
        //format the buffers with output
        //float with 2 and 6 digits before and after decimal point
        snprint(bufRaw, BUFFERSIZE, "%2.6f, %2.6f, %2.6f,\n", x,y,z);
        printf(bufRaw);


        int length = encrypt((const char *)&KEY, (const char *)&IV, &bufRaw[0], (char
*)&bufCiphered);
        for(uint i = 0; i< length; i++)
        printf("%02x", bufCiphered[i]);
        printf("\n");

        int sendStatus = sendto(sockfd, bufCiphered, length, 0, (struct sockaddr *)&serveraddr,
sizeof(serveraddr));
        if(sendStatus<0)
        {
                printf("sent failed with status %d\n", sendStatus);
                exit();
        }

        usleep(1000000);

        }
        return 0;

}
```

server.py

```
#!/usr/bin/env python
import socket
import sys
import datetime
import matplotlib.pyplot as plot
from matplotlib import animation
from Crypto.Cipher import AES


# server network configurations
SERVER_IP_ADDRESS = "216.47.158.2" #to be determined
PORT = 6000 #to be determined

time = [0]*50
for i in range(0,50):
            time[i] = i

ax_points = [float(0)]*50
ay_points = [float(0)]*50
az_points = [float(0)]*50

print("starting UDP Server Setup")
sys.stdout.flush()

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind( (SERVER_IP_ADDRESS, PORT) )

print("waiting for data to receive")
sys.stdout.flush()

KEY = b'3874460957140850'
iv = b'9331626268227018'

fig = plot.figure()
ax = plot.axes(xlim=(0, 50), ylim=(-2, 2))
lineX, lineY, lineZ, = ax.plot([], [], [], [], [], [], lw=2)


def init():
   lineX.set_data([], [])
   lineY.set_data([], [])
   lineZ.set_data([], [])
   return lineX, lineY, lineZ,

def updateData(i):
      decryption_suite = AES.new(KEY, AES.MODE_CBC, IV=iv)
```

```
        data, addr = sock.recvfrom(64)
        print (''.join('{:02x}'.format(x) for x in data))
        plain_text = decryption_suite.decrypt(data)

        data = plain_text.decode('utf-8')
        ax,ay,az,dump = data.split(",")
        print("ADXL345 X-Axis: " + ax + "\tY-Axis: " + ay + "\tZ-Axis: " + az + "\t" +
datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f"))
        sys.stdout.flush()

        del ax_points[0]
        del ay_points[0]
        del az_points[0]
        ax_points.append(float(ax))
        ay_points.append(float(ay))
        az_points.append(float(az))
        lineX.set_data(time,ax_points)
        lineY.set_data(time,ay_points)
        lineZ.set_data(time,az_points)
        return lineX, lineY, lineZ,

anim = animation.FuncAnimation(fig, updateData, init_func=init,
                    frames=200, interval=20, blit=True)
plot.show()
```