



INTEGRATION

Engineering capstone project-EECE8040

JULY 24, 2019

VIBCHECK

Nivedita Rajendran, Lisler Thomson Pulikkottil, Shadaab Saiyed

Table of Contents

<i>Table of Figures</i>	2
<i>Table of Tables</i>	3
1.Introduction	4
2.Background Information	4
3. Testing	8
4. Summary	13
5. Reference	14
<i>Appendix A</i>	15
<i>Appendix B</i>	18
<i>Appendix C</i>	20

Table of Figures

Figure 1 Elevator 1 Module Prototype.....	7
Figure 2 Elevator 2 Module Prototype.....	7
Figure 4 Readings from Pump.....	8
Figure 5 Reading from Elevator	9
Figure 6 Graph obtained when sensor was kept inside the Elevator Cabin	10
Figure 7 Graph obtained when sensor was kept on the Hydraulic Pump	11
Figure 8 Screenshot of Text File saved.....	12

Table of Tables

Table 1 Arduino-ADXL345 Connections	4
Table 2 Arduino-RFM9x Connections	5
Table 3 Raspberrypi-RFM9x Connections	5
Table 4 Raspberrypi-MPU6050 Connections	6

1.Introduction

The portion of the project which was completed in the previous weeks were to research on the Lo-Ra peer to peer connection with the modules RFM9X (3), and successfully we had completed the coding part to analyze how the communication was taking place between the modules. We were also testing the range of Lo-Ra, how it behaved with obstacles and in different scenarios. We found that LoRa worked absolutely well in all our tests. Hence, we finally decided to go with Lo-Ra peer to peer instead of going Lo-Ra WAN which is much complicated than this one. So, our next step was to do the integration of Arduino-Uno with the RFM9X board hooked up with the 3 Axis accelerometer ADXL345 which acted as the vibration sensor in our project.

2.Background Information

We were researching on the ways to hook them up all together on the bread boards. We figure out that Lo-Ra module uses the SPI communication and accelerometer uses the I2C communication. For the accelerometer to Arduino-Uno connection, the pins were connected in the way given below on the table.

Table 1 Arduino-ADXL345 Connections

Arduino pins	ADXL345 pins
GND	GND
3V3	VCC
3V3	CS
GND	SDO
A4	SDA
A5	SCL

For the RFM9x Lo-Ra module to Arduino-Uno connection, the pins were connected in the way given below on the table.

Table 2 Arduino-RFM9x Connections

Arduino pins	RFM9X pins
5V	Vin
GND	GND
D3	G0
D13	SCK
D12	MISO
D11	MOSI
D4	CS
D2	RST

For the Raspberry pi 3 to Lo-Ra module RFM9X connections, pins were connected in the way given below on the table.

Table 3 Raspberrypi-RFM9x Connections

Raspberrypi 3 pins	RFM9X pins
3V3	Vin
GND	GND
#5	G0
SCLK	SCK
MISO	MISO
MOSI	MOSI
CE1	CS
#25	RST

For the Raspberry pi 3 to Accelerometer MPU6050 connections, pins were connected in the way given below on the table.

Table 4 Raspberry pi-MPU6050 Connections

MPU6050 pins	Raspberry pi 3 pins
Vin	3V3
GND	GND
SCL	SCL
SDA	SDA

Raspberry pi is used in this project to make the output values from vibration sensor displayed on the monitor as it was requested by the client that he needs the vibration values to be visually big to see ,easy for a tester to view that and note down the readings. The integration of the raspberry pi is still not yet completely done. We are working on how to get the data packets from Lo-Ra module to the raspberry pi without delays and with greater accuracy.

The images of the hooked up Lo-Ra module, accelerometer and Arduino-Uno is attached here below and it's the setup which is placed at the elevator walls or inside the elevator cab.

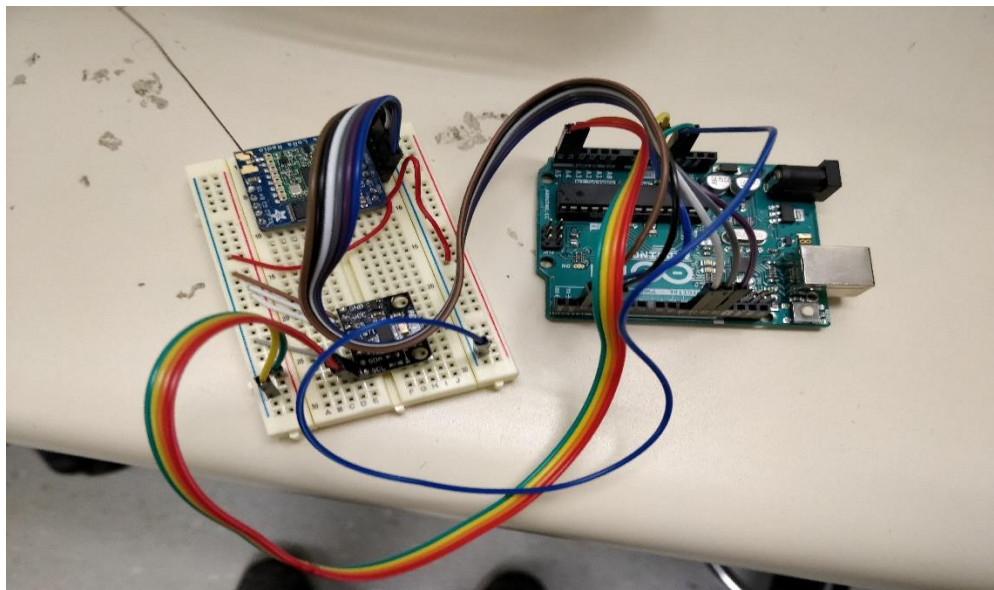


Figure 1 Elevator 1 Module Prototype

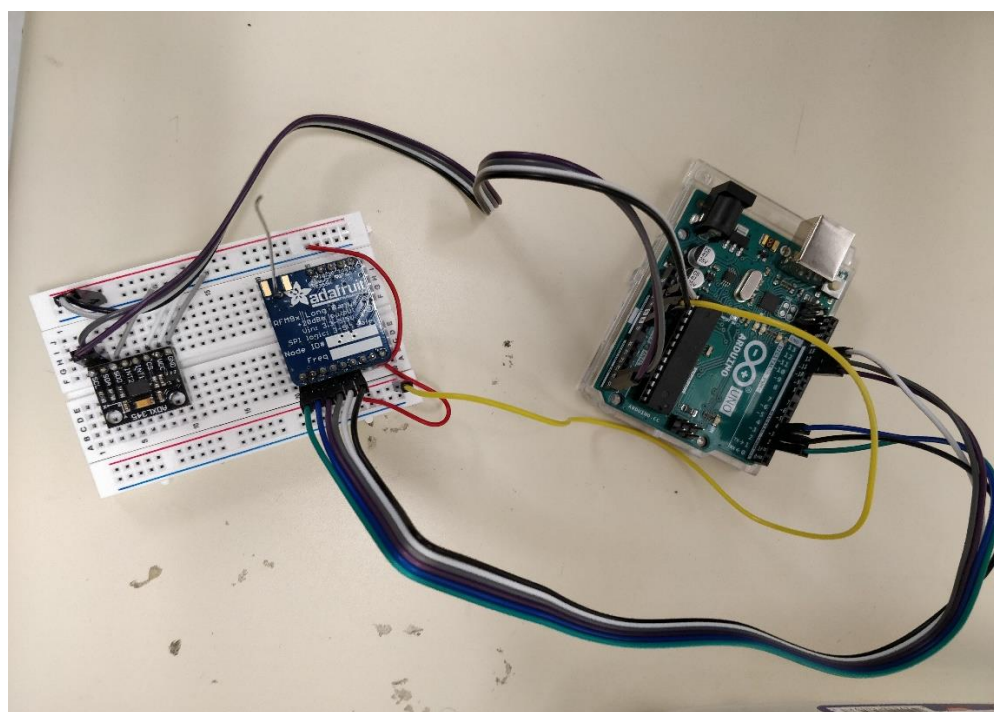


Figure 2 Elevator 2 Module Prototype

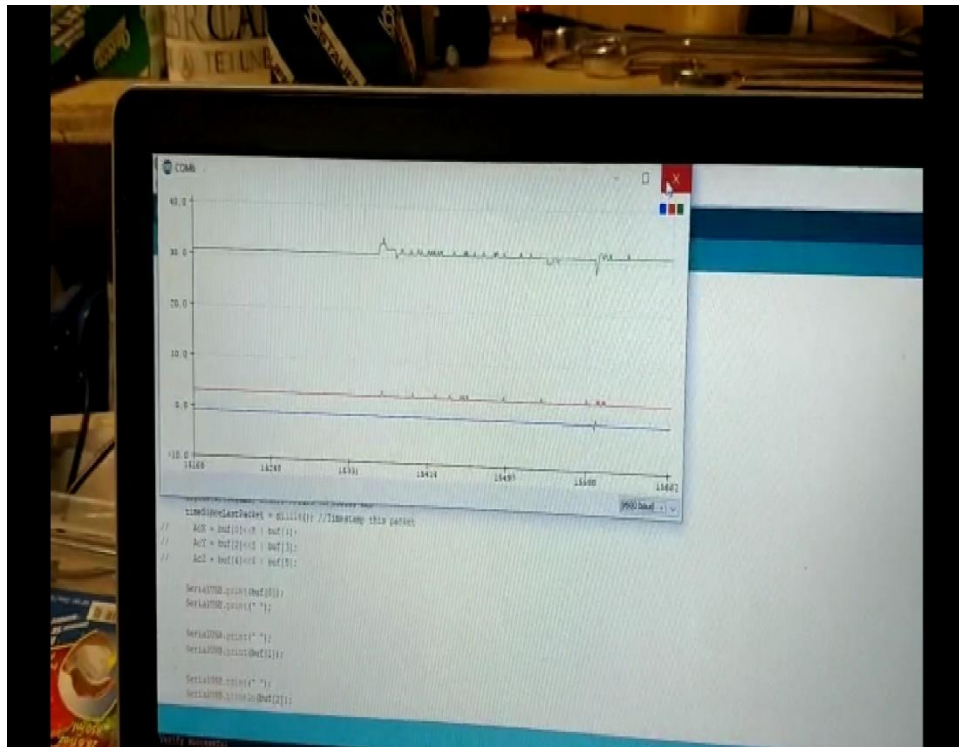


Figure 4 Reading from Elevator

I have also attached some videos of the working and testing of the whole setup on site. It was taken when the elevator was operated.



vibration_from_pump.mp4



vibration_from_elevatorcab.mp4

The following are the updated Tests after we adjusted the sensitivity of the sensor. The test was conducted in two stages - One when the sensor was kept inside the elevator cabin and another one when the sensor is kept on the elevator pump.

The following image shows the sensor reading from inside the cabin

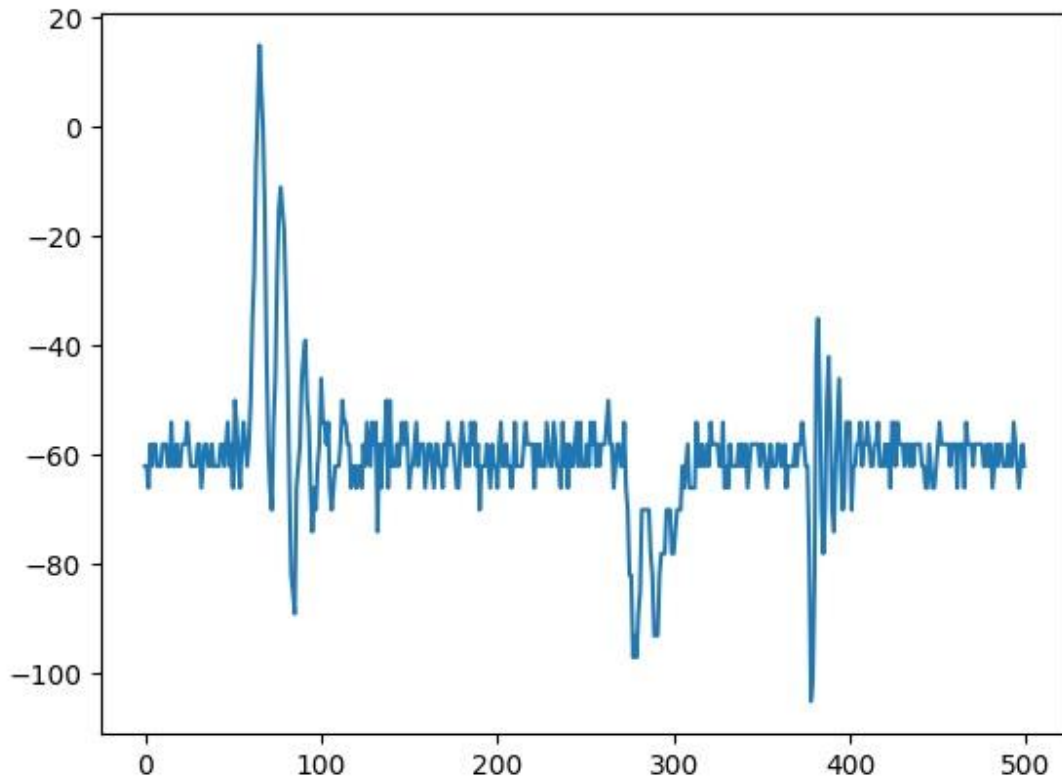


Figure 5 Graph obtained when sensor was kept inside the Elevator Cabin

The following image shows the sensor reading from the Hydraulic pump.

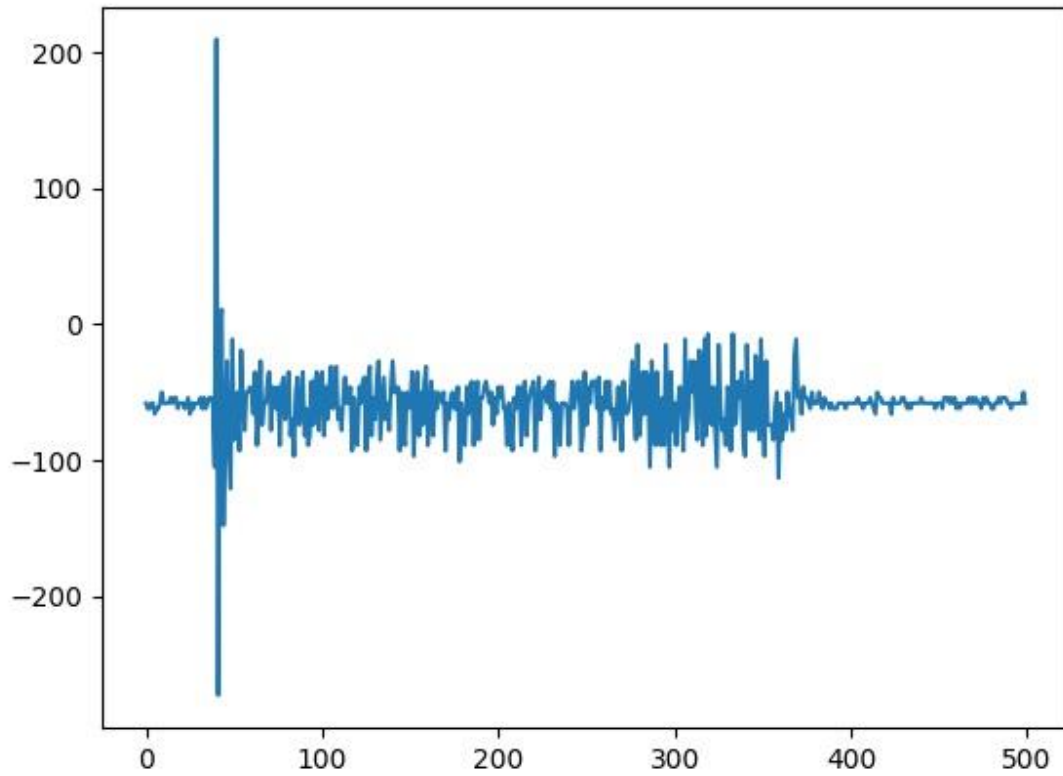
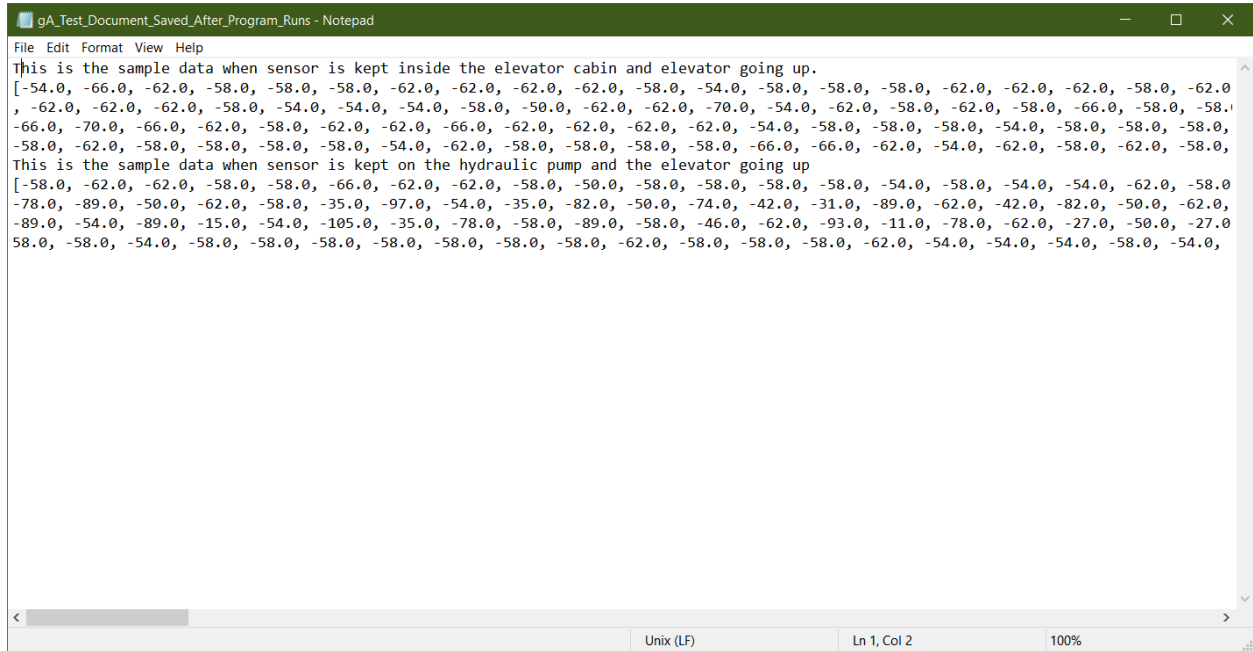


Figure 6 Graph obtained when sensor was kept on the Hydraulic Pump

We were also able to store the sensor reading to a text file so that the testers can access the readings on a later time.

The following image shows a screenshot of the above said text document.



The screenshot shows a Notepad window titled "gA_Test_Document_Saved_After_Program_Runs - Notepad". The text inside the window is as follows:

```
File Edit Format View Help
This is the sample data when sensor is kept inside the elevator cabin and elevator going up.
[-54.0, -66.0, -62.0, -58.0, -58.0, -62.0, -62.0, -62.0, -62.0, -58.0, -54.0, -58.0, -58.0, -62.0, -62.0, -62.0, -58.0, -62.0
, -62.0, -62.0, -62.0, -58.0, -54.0, -54.0, -58.0, -50.0, -62.0, -62.0, -70.0, -54.0, -62.0, -58.0, -62.0, -58.0, -66.0, -58.0, -58.0
-66.0, -70.0, -66.0, -62.0, -58.0, -62.0, -62.0, -66.0, -62.0, -62.0, -62.0, -62.0, -54.0, -58.0, -58.0, -58.0, -54.0, -58.0, -58.0, -58.0,
-58.0, -62.0, -58.0, -58.0, -58.0, -58.0, -54.0, -62.0, -58.0, -58.0, -58.0, -58.0, -66.0, -66.0, -62.0, -54.0, -62.0, -58.0, -62.0, -58.0,
This is the sample data when sensor is kept on the hydraulic pump and the elevator going up
[-58.0, -62.0, -62.0, -58.0, -58.0, -66.0, -62.0, -62.0, -58.0, -50.0, -58.0, -58.0, -58.0, -58.0, -54.0, -58.0, -54.0, -54.0, -62.0, -58.0
-78.0, -89.0, -50.0, -62.0, -58.0, -35.0, -97.0, -54.0, -35.0, -82.0, -50.0, -74.0, -42.0, -31.0, -89.0, -62.0, -42.0, -82.0, -50.0, -62.0,
-89.0, -54.0, -89.0, -15.0, -54.0, -105.0, -35.0, -78.0, -58.0, -89.0, -58.0, -46.0, -62.0, -93.0, -11.0, -78.0, -62.0, -27.0, -50.0, -27.0
58.0, -58.0, -54.0, -58.0, -58.0, -58.0, -58.0, -58.0, -58.0, -62.0, -58.0, -58.0, -58.0, -62.0, -54.0, -54.0, -54.0, -58.0, -54.0,
```

The status bar at the bottom indicates "Unix (LF)", "Ln 1, Col 2", and "100%".

Figure 7 Screenshot of Text File saved

4. Summary

The testing of the existing system has been made and it has been approved by the sponsor under his supervision at Cambridge elevating. Now onwards we will be focusing on making the system much more custom fit and tailored to the sponsor's needs. As per our sponsor, we are striving to make the display (monitor) much bigger and clearer using the raspberry pi and also trying to add on the option to save the real time sensed values of vibration of elevator on a hard drive storage. We also have planned to make into a product suitable for performing on site rather than the prototype.

5. Reference

https://conestoga.desire2learn.com/d2l/lms/dropbox/user/folder_submit_files.d2l?db=265395&grpId=279090&isprv=0&bp=0&ou=271571

Appendix A

Transmitter Code

```
/*Lo-Ra Transmitter code

  Authors: Lisler Thomson
           Nivedita Rajendran
           Shadab Saiyed

  Description: This Code implements the Transmitter end at the Elevator. It takes
the Vibration/accelerometer reading from the elevator and transmits over LoRa

  Revision: C

  Date: 23.07.2019
*/

#include <SPI.h>
//Radio Head Library:
#include <RH_RF95.h>
// SparkFun ADXL345 Accelerometer Library
#include <SparkFun_ADXL345.h>
ADXL345 adxl = ADXL345();           // USE FOR I2C COMMUNICATION
RH_RF95 rf95(4, 3); // RFM9x Chip Select and Interrupt Pins
int LED = 13; //Status LED is on pin 13
int packetCounter = 0; //Counts the number of packets sent
long timeSinceLastPacket = 0; //Tracks the time stamp of last packet received
float frequency = 921.2; //Broadcast frequency

void setup()
{
  pinMode(LED, OUTPUT);
  Serial.begin(9600);
  Serial.println("RFM Client!");
  //Initialize the Radio.
  if (rf95.init() == false) {
    Serial.println("Radio Init Failed - Freezing");
    while (1);
  }
}
```



```

else {
    //An LED indicator to let us know radio initialization has completed.
    Serial.println("Transmitter up!");
    digitalWrite(LED, HIGH);
    delay(500);
    digitalWrite(LED, LOW);
    delay(500);
}

// Set frequency
rf95.setFrequency(frequency);
rf95.setTxPower(14, false); // Sets the transmission frequency

// Power on the ADXL345
adxl.powerOn();
adxl.setRangeSetting(2);           // Give the range settings
adxl.setSpiBit(0);                 // Configure the device to be in 4 wire SPI
mode when set to '0' or 3 wire SPI mode when set to 1

adxl.setActivityXYZ(1, 0, 0);      // Set to activate movement detection in the
axes "adxl.setActivityXYZ(X, Y, Z);" (1 == ON, 0 == OFF)

adxl.setActivityThreshold(75);     // 62.5mg per increment // Set activity //
Inactivity thresholds (0-255)

adxl.setInactivityXYZ(1, 0, 0);    // Set to detect inactivity in all the axes
"adxl.setInactivityXYZ(X, Y, Z);" (1 == ON, 0 == OFF)

adxl.setInactivityThreshold(75);   // 62.5mg per increment // Set inactivity //
Inactivity thresholds (0-255)

adxl.setTimeInactivity(10);        // How many seconds of no activity is inactive?

adxl.setTapDetectionOnXYZ(0, 0, 1); // Detect taps in the directions turned ON
"adxl.setTapDetectionOnX(X, Y, Z);" (1 == ON, 0 == OFF)

// Set values for what is considered a TAP and what is a DOUBLE TAP (0-255)
adxl.setTapThreshold(50);          // 62.5 mg per increment
adxl.setTapDuration(15);           // 625  $\mu$ s per increment
adxl.setDoubleTapLatency(80);      // 1.25 ms per increment
adxl.setDoubleTapWindow(200);     // 1.25 ms per increment

// Set values for what is considered FREE FALL (0-255)
adxl.setFreeFallThreshold(7);      // (5 - 9) recommended - 62.5mg per increment
adxl.setFreeFallDuration(30);     // (20 - 70) recommended - 5ms per increment

```

```

// Turn on Interrupts for each mode (1 == ON, 0 == OFF)
adxl.InactivityINT(1);
adxl.ActivityINT(1);
adxl.FreeFallINT(1);
adxl.doubleTapINT(1);
adxl.singleTapINT(1);
}
void loop()
{
    int x, y, z;
    // Read the accelerometer values and store them in variables declared above x,y,z
    adxl.readAccel(&x, &y, &z);
    // Stores all three axis values to a single array to transmit
    uint8_t toSend[] = {x, y, z};
    // Sends Data through LoRa
    rf95.send(toSend, sizeof(toSend));
    rf95.waitPacketSent(); // Waits till the transmission is completed
}

```

Appendix B

Receiver Code:

```
/* LoRa Receiver Code

  Authors: Lisler Thomson
           Nivedita Rajendran
           Shadab Saiyed

  Description: This Code implements the Receiver end at the Elevator. It takes the
  Vibration/accelerometer reading from LoRa and Transmits to Raspberry Pi over Serial
  USB

  Revision: C
  Date: 23.07.2019
*/

#include <SPI.h>
//Radio Head Library:
#include <RH_RF95.h>
RH_RF95 rf95(12, 6);
int LED = 13; //Status LED on pin 13

int packetCounter = 0; //Counts the number of packets sent
long timeSinceLastPacket = 0; //Tracks the time stamp of last packet received
float frequency = 921.2;

void setup()
{
  pinMode(LED, OUTPUT);
  SerialUSB.begin(9600);
  while(!SerialUSB);
  //Initialize the Radio.
  if (rf95.init() == false){
    while (1);
  }
  else{
    // An LED indicator to let us know radio initialization has completed.
    digitalWrite(LED, HIGH);
  }
}
```

```

    delay(500);
    digitalWrite(LED, LOW);
    delay(500);
}
rf95.setFrequency(frequency);
}

void loop()
{
    if (rf95.available()){
        // Should be a message for us now
        int8_t buf[RH_RF95_MAX_MESSAGE_LEN];
        uint8_t len = sizeof(buf)-1;

        if (rf95.recv((uint8_t*)buf, &len)){
            digitalWrite(LED, HIGH); //Turn on status LED
            timeSinceLastPacket = millis(); //Timestamp this packet
            SerialUSB.println(buf[2])
        }
    }
    //Turn off status LED if we haven't received a packet after 1s
    if(millis() - timeSinceLastPacket > 1000){
        digitalWrite(LED, LOW); //Turn off status LED
        timeSinceLastPacket = millis(); //Don't write LED but every 1s
    }
}

```

Appendix C

Raspberry Pi Code for Graph and Data Storage

```
## VibCheck_Python.py
## Author: Lisler Thomson Pulikkotttil;
##         Nivedita Rajendran;
##         Shadaab Saiyed
## Description: This program takes the readings from sparkfun pro
##              RF through serial USB port and displays it on a
##              graph. It will also store the readings to a file

import time
import serial
import matplotlib.pyplot as plt

# Initialising the Gragph
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
xs = range(0,500)
ys = []

# Initialising Serial Port
ser = serial.Serial(

    port='/dev/ttyACM0',
    baudrate = 9600,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    timeout=1
)

# Storing Sensor data into an array
for t in range(0,500):
    data = ser.readline()
    print(float(data))
    ys.append(float(data))

# Plotting sensor readings to graph
ax.plot(xs,ys)

# Save plot
plt.savefig('datagraph.jpg')

# Show plot
plt.show()

# Open a file and store the sensor data
f = open("test1.txt","a+")
details = input("Enter Details:")
f.write(details)
```

```
f.write("\n")  
f.write(str(ys))  
f.write("\n")  
f.close()
```