ECE Makerspace Fan Control

This project works to control the 3D printer fans through the use of a raspberry pi and SEEED LIS3DHTR accelerometers. A significant portion of this project is code work in python, however there are some manufacturing processes for making one of these. This documentation will go through the process of making one and loading it up with the necessary code to function.

Materials:

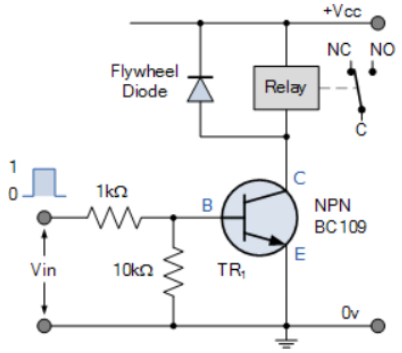
- Raspberry Pi 3B with ubuntu operating system
- 8 seeed LIS3DHTR 3-axis digital accelerometer
- I2C multiplexer for the pi
- 4 conductor wire
- 4 pin jst connector set
- AC relay with ~5 V dc input (the one used here has a range of 3-32V)
- NPN BJT
- Zener Diode
- 1k Ohm Resistor
- 10k Ohm Resistor

Process:

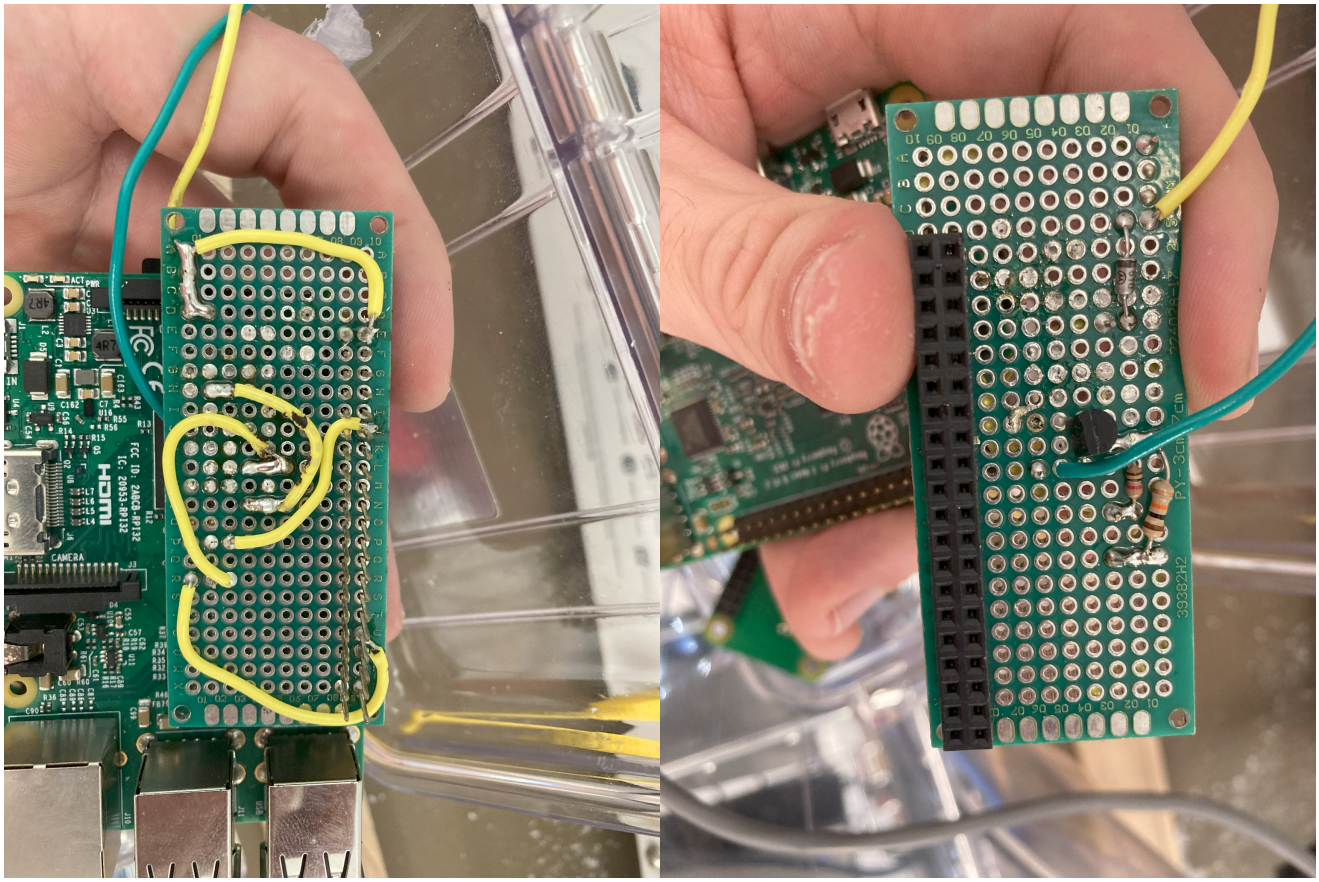
Building The Relay Circuit

The relay circuit is a very crucial part of this project as it allows for the delivery of power to the fan. It allows for a more stable voltage delivery than just using the GPIO output pin(which would still work however it is much closer to the 3 volt threshold than using the 5V). This part could be considered optional.

The basic circuit of the relay is this:



When converted to a circuit board it will look like this:



Assembly will require some soldering, however overall it should not be too difficult.

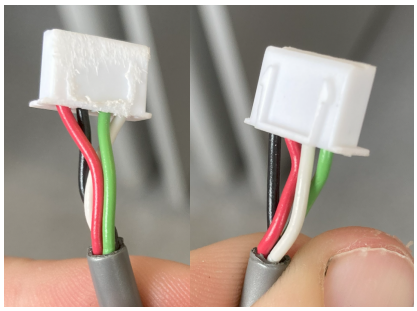
Wire Assembly

This is an important component of the fan control, without good wires the system will fall apart. Wire assembly has two parts, the LIS3DHTR wire and the connecting wire. These are made from two different wire types, the connecting wire has to be specially configured to interface between the PiHat and the LIS3DHTR.

Connecting Wire

The connecting wire is not that difficult to assemble, it has 2 major instructions, how much wire to take and connecting the JST connectors. The wire length you should aim for should be about 4 feet of 4 conductor telephone wire.

When setting up the connecting wire, you have to consider the 2 ends one goes to the pi hat and it needs to be sanded down. There is also a unique, as the connection for the pi hat is different from the i2c. Below there are photos of the correct wire orientation for the ends.



Pictured at the connections for the pi hat end and the LIS3DHTR end.

LIS3DHTR Wire

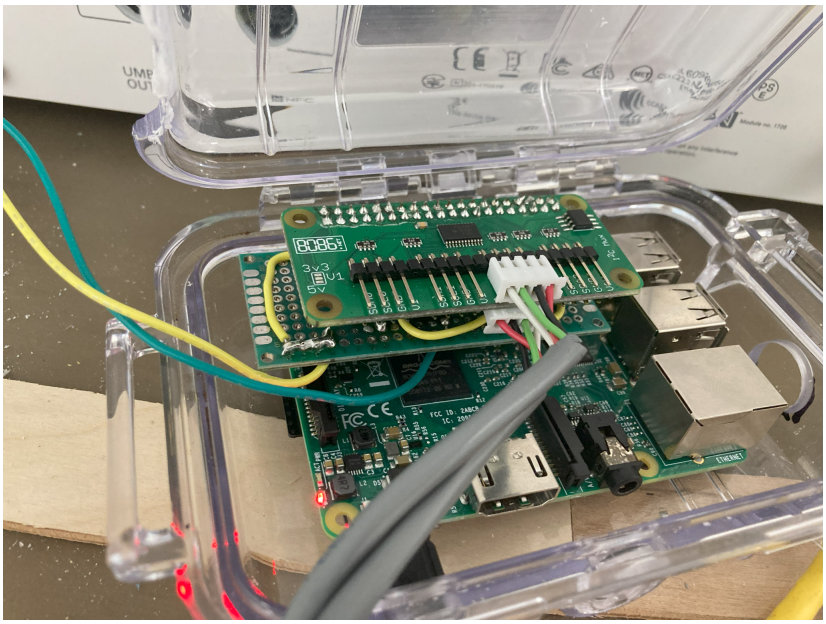
The LIS3DHTR wire is already almost complete out of the box, however the problem is that it only has female connectors at the end. As a result, we have to replace the female connector with a male connector, this is simple enough as it just involves trimming the wire, dabbing a little bit of solder onto the wire ends to fasten them to the connector and then shrink wrapping them. When applying the solder be careful not to hold heat for too long as it will melt through the plastic. Apply the shrink wraps as normal using the heat gun. The finished connector should look like this:

The Pi

The pi is the most important part of this system, it operates all the code that we put onto it. Going forward, we are very much underutilizing this pi with the code, we could add other functions onto it beyond just the fan code.

Physical Configuration

This is one of the easy parts of the configuration as all the parts should be assembled, you just need to make the box. The box should have a cut for wires, one large one of the control and sensor wires, micro usb power, and an ethernet cable. Beyond the box it is just placing the boards onto the raspberry pi gpio pins. When wiring the circuit, you will want the 5v, ground, and gpio 18 pin. 5v is Vcc, gpio 18 is Vin and ground is ground. The final assembled pi should look like this:



It has the gpio, circuit and then the pi hat, make sure all the pins are seated correctly.

Software Configuration

This is where most of the project is developed and where most errors will appear. The easiest way to work with this is to use the github repository and develop it off the system and pull it onto the pi using ssh. SSH allows for headless development and allows for more rapid testing of your changes.

SSH Configuration

SSH configurations is pretty simple on ubuntu, simply type the commands:
sudo apt update
sudo apt install openssh-server
These will set up ssh on the pi and allow for you to connect using an ssh software like PuTTY. Once you have ssh set up use
ip a
To find your address and put that into the ssh terminal. Once you are connected, you can start working on the system.

Pulling the Code From GitHub

Right now the code is hosted in this repository:
https://github.com/Ryan-Ficken/ECE-Makerspace-Accelerometer

Set up git on the device using:
sudo apt install git
git config --global user.name "your_name"
git config --global user.email "email@address.com"

This will let you use git commands on the device, you will also have to configure ssh for the device which can be done using these tutorials
https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent
https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account

Once you have these you can use
git pull “ssh portion of the repository”

To get the code onto the pi.

Enabling The Code

The pi hat requires some configuration before it can work, notably the overlay. To check that it is working input the command:
i2cdetect -l
You should receive an output similar to this:

```
i2c-1   i2c             bcm2835 (i2c@7e804000)                  I2C adapter
i2c-11  i2c             i2c-1-mux (chan_id 0)                   I2C adapter
i2c-12  i2c             i2c-1-mux (chan_id 1)                   I2C adapter
i2c-13  i2c             i2c-1-mux (chan_id 2)                   I2C adapter
i2c-14  i2c             i2c-1-mux (chan_id 3)                   I2C adapter
i2c-15  i2c             i2c-1-mux (chan_id 4)                   I2C adapter
i2c-16  i2c             i2c-1-mux (chan_id 5)                   I2C adapter
i2c-17  i2c             i2c-1-mux (chan_id 6)                   I2C adapter
i2c-18  i2c             i2c-1-mux (chan_id 7)                   I2C adapter
```

If you don’t receive an output similar to this, input this command:
sudo dtoverlay i2c-mux pca9548 addr=0x70

Then try
i2cdetect -l

If you don’t get this output or get this message:

```
* Failed to apply overlay '1_i2c-mux' (kernel)
Makerspace@Makerspace:~$
```

Input the command
sudo dtoverlay -r i2c-mux

Then disconnect the pi hat and reconnect it and input
sudo dtoverlay i2c-mux pca9548 addr=0x70

This should fix the error, if it doesn't try rebooting and doing this process again without removing the pi hat.

Operation

Now you can run the code, ensure your sensors are connected before running the code, if you add more you will likely have to restart the code. But once in operation it should be able to recover from sensor mistakes and should be able to run unless something physical happens with the sensors.

Setting up you will want to use the command

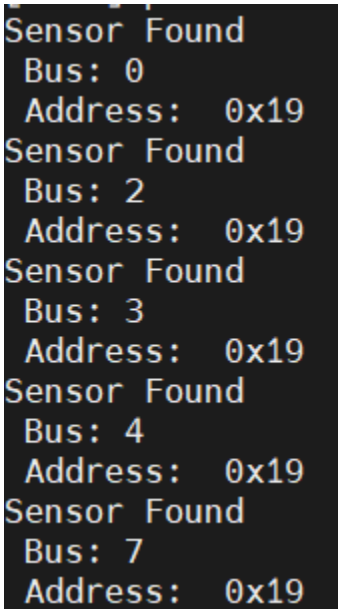
tmux

This will set up a separate session for the code allowing you to disconnect from the SSH.

To run the code go to the directory with LIS3DHTR.py and type

sudo python LIS3DHTR.py

And the code should start up and you should receive a message like this:



If not all the sensors are found you may have issues with your wire contacts try replugging them to make a better connection.

In operation the code should look like this:

```
Fan Turned Off
0.5
Data Read For:  4 Sensor:  0
{'x': [0.128], 'y': [-0.632], 'z': [0.756]} 4
4 Sensor:  0 Accels X:  0.0119999999999999997 0 0
[[0], [], [0], [0], [0], [], [], [0]]
[0]
Fan Turned Off
0.5
None 5
[[0], [], [0], [0], [0], [], [], [0]]
[]
Fan Turned Off
0.5
Checking 5
None 6
[[0], [], [0], [0], [0], [], [], [0]]
[]
Fan Turned Off
0.5
Checking 6
```

These outputs allow you to see which sensors have turned on the fan, when it is 0 the fan is not on because of this sensor. However when there is a 1 the fan will turn on. When a fan is on/turned on the code should output this:

```
Data Read For:  3 Sensor:  0
{'x': [0.296], 'y': [0.348], 'z': [0.728]} 3
3 Sensor:  0 Accels X:  0.0239999999999999966 0 1
[[1], [], [0], [1], [0], [], [], [0]]
[1]
TURN FAN ON!
Data Read For:  4 Sensor:  0
{'x': [0.12], 'y': [-0.644], 'z': [0.756]} 4
4 Sensor:  0 Accels X:  0.0279999999999999997 0 0
[[1], [], [0], [1], [0], [], [], [0]]
[0]
Fan Kept On
None 5
[[1], [], [0], [1], [0], [], [], [0]]
[]
Fan Kept On
Checking 5
None 6
[[1], [], [0], [1], [0], [], [], [0]]
[]
Fan Kept On
Checking 6
```

This will allow you to check sensor operation as the correlating sensors should be on if they are moving. You will have to track which sensors are plugged into the parts of the pi hat. To exit this press ctrl+b and then d. This will leave the session open, but you can close the ssh. To access the session, type:

tmux a -t 0

To stop the program press ctrl+z.

End

This should cover most of the theory of operation for this code and some common errors, other common errors can likely be troubleshot using the code outputs and looking at the sensor wire. Adding the sensors to the mess can be a bit difficult but you have to clip an entrance for the wire and feed it through to the back where you can connect the pi.

Code Backup

```
#!/usr/bin/env python

# Distributed with a free-will license.
# Use it any way you want, profit or free, provided it fits in the
licenses of its associated works.
# LIS3DHTR
# This code is designed to work with the LIS3DHTR_I2CS I2C Mini Module
available from ControlEverything.com.
#
https://www.controleverything.com/content/Accelorometer?sku=LIS3DHTR_I2CS#
tabs-0-product_tabset-2

import smbus
import time
import RPi.GPIO as GPIO

# Get I2C bus these parameters should be specified by the pihat best way
to find them is by using i2cdetect -l in terminal
buslist = [11,12,13,14,15,16,17,18]
bus =
[smbus.SMBus(buslist[0]),smbus.SMBus(buslist[1]),smbus.SMBus(buslist[2]),s
mbus.SMBus(buslist[3]),smbus.SMBus(buslist[4]),smbus.SMBus(buslist[5]),smb
us.SMBus(buslist[6]),smbus.SMBus(buslist[7])]

#This is basic config macros you can mostly ignore these or check the
datasheet if you want to operate in other modes

# I2C address of the device
LIS3DHTR_DEFAULT_ADDRESS            = 0x19
LIS3DHTR_SECOND_ADDRESS             = 0x18
LIS3DHTR_SECOND_ENABLE              = 0

# LIS3DHTR Register Map
LIS3DHTR_REG_WHOAMI                 = 0x0F # Who Am I Register
LIS3DHTR_REG_CTRL1                  = 0x20 # Control Register-1
LIS3DHTR_REG_CTRL2                  = 0x21 # Control Register-2
LIS3DHTR_REG_CTRL3                  = 0x22 # Control Register-3
LIS3DHTR_REG_CTRL4                  = 0x23 # Control Register-4
LIS3DHTR_REG_CTRL5                  = 0x24 # Control Register-5
```

```
LIS3DHTR_REG_CTRL6                  = 0x25 # Control Register-6
LIS3DHTR_REG_REFERENCE              = 0x26 # Reference
LIS3DHTR_REG_STATUS                 = 0x27 # Status Register
LIS3DHTR_REG_OUT_X_L                = 0x28 # X-Axis LSB
LIS3DHTR_REG_OUT_X_H                = 0x29 # X-Axis MSB
LIS3DHTR_REG_OUT_Y_L                = 0x2A # Y-Axis LSB
LIS3DHTR_REG_OUT_Y_H                = 0x2B # Y-Axis MSB
LIS3DHTR_REG_OUT_Z_L                = 0x2C # Z-Axis LSB
LIS3DHTR_REG_OUT_Z_H                = 0x2D # Z-Axis MSB

# Accl Datarate configuration
LIS3DHTR_ACCL_DR_PD                 = 0x00 # Power down mode
LIS3DHTR_ACCL_DR_1                  = 0x10 # ODR = 1 Hz
LIS3DHTR_ACCL_DR_10                 = 0x20 # ODR = 10 Hz
LIS3DHTR_ACCL_DR_25                 = 0x30 # ODR = 25 Hz
LIS3DHTR_ACCL_DR_50                 = 0x40 # ODR = 50 Hz
LIS3DHTR_ACCL_DR_100                = 0x50 # ODR = 100 Hz
LIS3DHTR_ACCL_DR_200                = 0x60 # ODR = 200 Hz
LIS3DHTR_ACCL_DR_400                = 0x70 # ODR = 400 Hz
LIS3DHTR_ACCL_DR_1620               = 0x80 # ODR = 1.620 KHz
LIS3DHTR_ACCL_DR_1344               = 0x90 # ODR = 1.344 KHz

# Accl Data update & Axis configuration
LIS3DHTR_ACCL_LPEN                  = 0x00 # Normal Mode, Axis disabled
LIS3DHTR_ACCL_XAXIS                 = 0x04 # X-Axis enabled
LIS3DHTR_ACCL_YAXIS                 = 0x02 # Y-Axis enabled
LIS3DHTR_ACCL_ZAXIS                 = 0x01 # Z-Axis enabled

# Acceleration Full-scale selection
LIS3DHTR_BDU_CONT                   = 0x00 # Continuous update, Normal
Mode, 4-Wire Interface
LIS3DHTR_BDU_NOT_CONT               = 0x80 # Output registers not updated
until MSB and LSB reading
LIS3DHTR_ACCL_BLE_MSB               = 0x40 # MSB first
LIS3DHTR_ACCL_RANGE_16G             = 0x30 # Full scale = +/-16g
LIS3DHTR_ACCL_RANGE_8G              = 0x20 # Full scale = +/-8g
LIS3DHTR_ACCL_RANGE_4G              = 0x10 # Full scale = +/-4g
LIS3DHTR_ACCL_RANGE_2G              = 0x00 # Full scale = +/-2g, LSB first
LIS3DHTR_HR_DS                      = 0x00 # High-Resolution Disabled
LIS3DHTR_HR_EN                      = 0x08 # High-Resolution Enabled
```

```
LIS3DHTR_ST_0                       = 0x02 # Self Test 0
LIS3DHTR_ST_1                       = 0x04 # Self Test 1
LIS3DHTR_SIM_3                      = 0x01 # 3-Wire Interface

#Self Defined Regs
LIS3DHTR_INT1_SRC           = 0x31 # Interrupt 1 source register
LIS3DHTR_INT1_CFG           = 0x30 # Interrupt 1 configuration register
LIS3DHTR_INT1_THS                   = 0x32 # Interrupt 1 Threshold
register
LIS3DHTR_INT1_DURATION              = 0x33 # Interrupt 1 Duration register
LIS3DHTR_INT1_MOTION_DETECT         = 0x0A # 6-Direction Movement
Recognition



#LIS3DHTR Object
#most of the params are default, however some modification had to be made
to allow for concurrent running with the i2cmux
#most of this was the addition of the input params like busnum,
addressList and numSensors
#these allow for you to reference parts of the objects when reading the
data
#there is also a try and except block on the datarate and config this
prevents the error where a sensor disconnects
#it also logs these instances, however the system should be robust enough
to keep running.
#potentially could cause an issue when log file fills up, simple enough to
remove however

class LIS3DHTR():
    def __init__ (self,busnum,addressList,numSensors):
        self.objaddressList = addressList
        self.numSensors = numSensors
        self.busnum = busnum
        self.select_datarate()
        self.select_data_config()

    def select_datarate(self):
        """Select the data rate of the accelerometer from the given
provided values"""
```

```
        DATARATE_CONFIG = (LIS3DHTR_ACCL_DR_1 | LIS3DHTR_ACCL_XAXIS |
LIS3DHTR_ACCL_YAXIS | LIS3DHTR_ACCL_ZAXIS)
        try:
            for i in range(self.numSensors):
                if self.objaddressList:

bus[self.busnum].write_byte_data(self.objaddressList[i],
LIS3DHTR_REG_CTRL1, DATARATE_CONFIG)
        except:
            print("Initalization Failed")
            log.write("{0},{1}\n".format(time.strftime("%Y-%m-%d
%H:%M:%S"),"Sensor Failed to Initalize Despite Finding One")) #remove this
line if log file fills up
            self.numSensors = 0
            self.objaddressList = []

    def select_data_config(self):
        """Select the data configuration of the accelerometer from the
given provided values"""
        DATA_CONFIG = (LIS3DHTR_ACCL_RANGE_2G | LIS3DHTR_BDU_CONT |
LIS3DHTR_HR_DS)
        try:
            for i in range(self.numSensors):
                if self.objaddressList:

bus[self.busnum].write_byte_data(self.objaddressList[i],
LIS3DHTR_REG_CTRL4, DATA_CONFIG)
        except:
            print("Initalization Failed")
            log.write("{0},{1}\n".format(time.strftime("%Y-%m-%d
%H:%M:%S"),"Sensor Failed to Initalize Despite Finding One")) #remove this
line if log file fills up
            self.numSensors = 0
            self.objaddressList = []

    def read_accl(self):
        """Read data back from LIS3DHTR_REG_OUT_X_L(0x28), 2 bytes
        X-Axis Accl LSB, X-Axis Accl MSB"""
        xAccl = [0]*self.numSensors
        yAccl = [0]*self.numSensors
```

```
        zAccl = [0]*self.numSensors
        for i in range(self.numSensors):
            if self.objaddressList:
                try:
                    print("Data Read For: ",self.busnum, "Sensor: ",i)
                    data0 =
bus[self.busnum].read_byte_data(self.objaddressList[i],
LIS3DHTR_REG_OUT_X_L)
                    data1 =
bus[self.busnum].read_byte_data(self.objaddressList[i],
LIS3DHTR_REG_OUT_X_H)

                    xAccl[i] = data1 * 256 + data0
                    if xAccl[i] > 32767 :
                        xAccl[i] -= 65536
                    xAccl[i] /= 16000
                    """Read data back from LIS3DHTR_REG_OUT_Y_L(0x2A), 2
bytes
                    Y-Axis Accl LSB, Y-Axis Accl MSB"""
                    data0 =
bus[self.busnum].read_byte_data(self.objaddressList[i],
LIS3DHTR_REG_OUT_Y_L)
                    data1 =
bus[self.busnum].read_byte_data(self.objaddressList[i],
LIS3DHTR_REG_OUT_Y_H)

                    yAccl[i] = data1 * 256 + data0
                    if yAccl[i] > 32767 :
                        yAccl[i] -= 65536
                    yAccl[i] /= 16000
                    """Read data back from LIS3DHTR_REG_OUT_Z_L(0x2C), 2
bytes
                    Z-Axis Accl LSB, Z-Axis Accl MSB"""
                    data0 =
bus[self.busnum].read_byte_data(self.objaddressList[i],
LIS3DHTR_REG_OUT_Z_L)
                    data1 =
bus[self.busnum].read_byte_data(self.objaddressList[i],
LIS3DHTR_REG_OUT_Z_H)
```

```
                    zAccl[i] = data1 * 256 + data0
                    if zAccl[i] > 32767 :
                        zAccl[i] -= 65536
                    zAccl[i] /= 16000
                except:
                    xAccl[i] = None
                    yAccl[i] = None
                    zAccl[i] = None
            return {'x' : xAccl, 'y' : yAccl, 'z' : zAccl}

from LIS3DHTR import LIS3DHTR
#reinit function
#zeros the sensor and checks the i2c if it finds a sensor, it keeps it
def SensorReinitalization(i):
        c = 0
        addressListtemp = []
        for j in range(2,120):
                try:
                    bus[i].read_byte_data(j,LIS3DHTR_REG_OUT_X_L)
                    addressListtemp.append(j)
                    c = c + 1
                    print("Sensor Found \n Bus:",i,"\n Address: ",hex(j))
                except:
                    pass
                numAddresses[i] = c
                addressList[i] = addressListtemp
        if i == 0:
            print(numAddresses[i],addressList[i])
        return LIS3DHTR(i,addressList[i],numAddresses[i])

with open("/home/Makerspace/ECE-Makerspace-Accelerometer/error_log.csv",
"a") as log: #remove this line if log file fills up
    #initalization of data storage structures for enabling the LIS3DHTR
    c = 0
    addressList = []
    addressListtemp = []
    numAddresses = []
    lis3dhtr = []
    reinit_count = [0]*8
```

```
    #first intitalization loop basically runs through all the i2c ports on
the pi and checks if it got a read for one if so it found a sensor
    for i in range(0,len(bus)):
        for j in range(2,120):
            try:
                bus[i].read_byte_data(j,LIS3DHTR_REG_OUT_X_L)
                addressListtemp.append(j)
                c = c + 1
                print("Sensor Found \n Bus:",i,"\n Address: ",hex(j))
            except:
                pass
        numAddresses.append(c)
        c = 0
        addressList.append(addressListtemp)
        lis3dhtr.append(LIS3DHTR(i,addressList[i],numAddresses[i]))
        addressListtemp = []

    time.sleep(1)
    #setting up tracking variables
    accl_old = []
    count =
[[0]*numAddresses[0],[0]*numAddresses[1],[0]*numAddresses[2],[0]*numAddres
ses[3],[0]*numAddresses[4],[0]*numAddresses[5],[0]*numAddresses[6],[0]*num
Addresses[7]]
    lowcount =
[[0]*numAddresses[0],[0]*numAddresses[1],[0]*numAddresses[2],[0]*numAddres
ses[3],[0]*numAddresses[4],[0]*numAddresses[5],[0]*numAddresses[6],[0]*num
Addresses[7]]
    fanOn =
[[0]*numAddresses[0],[0]*numAddresses[1],[0]*numAddresses[2],[0]*numAddres
ses[3],[0]*numAddresses[4],[0]*numAddresses[5],[0]*numAddresses[6],[0]*num
Addresses[7]]
    sensorOn =
[[0]*numAddresses[0],[0]*numAddresses[1],[0]*numAddresses[2],[0]*numAddres
ses[3],[0]*numAddresses[4],[0]*numAddresses[5],[0]*numAddresses[6],[0]*num
Addresses[7]]
    time_before_next_loop = .5
    #getting old values, so we have something to compare against on the
first run
    for i in range(0,len(bus)):
```

```
        accl_old.append(lis3dhtr[i].read_accl())
    print(accl_old)
    #configuring the output pin
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(18,GPIO.OUT)
    GPIO.output(18,GPIO.LOW)
    #the operation loop
    #try except block allows for robustness as if a sensor disconnects it
won't throw an error and stop code
    #instead it will deinit the sensor
    #basic operation is that the sensor will read a new value, compare it
to the old and if the difference is greater than the threshold
    #it will add a count to the fan, if the fan gets more counts than its
threshold it will turn on
    #while it is on the loop will be less frequent until the fan turns off
    while True:
        for i in range(0,len(bus)):
            try:
                accl = lis3dhtr[i].read_accl()
                print(accl,i)
                for j in range(numAddresses[i]):
                    if abs(accl_old[i]['x'][j] - accl['x'][j]) > .15 or
abs(accl_old[i]['y'][j] - accl['y'][j]) > .15 or abs(accl_old[i]['z'][j] -
accl['z'][j]) > .15:
                        count[i][j] += 1
                        lowcount[i][j] = 0
                        if count[i][j] > 2:
                            sensorOn[i][j] = 1
                            count[i][j] = 0
                    else:
                        count[i][j] = 0
                        lowcount[i][j] += 1
                        if lowcount[i][j] > 4:
                            sensorOn[i][j] = 0
                            lowcount[i][j] = 0
                    print(i,"Sensor: ",j,"Accels X:
",abs(accl_old[i]['x'][j] - accl['x'][j]), count[i][j],sensorOn[i][j])
                    accl_old[i] = accl
            except:
```

```
                log.write("{0},{1}\n".format(time.strftime("%Y-%m-%d
%H:%M:%S"),"Sensor"+str(i)+"Failed"))
                print("Exception Happened")
                print(accl)
                lis3dhtr[i] = SensorReinitalization(i)
                time.sleep(1)
                sensorOn[i] = [0]*lis3dhtr[i].numSensors
            print(sensorOn)
            print(sensorOn[i])
            if 1 in  sensorOn[i]:
                print("TURN FAN ON!")
                GPIO.output(18,GPIO.HIGH)
                time_before_next_loop = 5
            else:
                #poor wording on this if but basically checks every single
part of the sensorOn list for a 1
                #if theres is no 1 it will turn off
                if 1 in (item for sublist in sensorOn for item in
sublist):
                    print("Fan Kept On")
                else:
                    print("Fan Turned Off")
                    GPIO.output(18,GPIO.LOW)
                    print(time_before_next_loop)
                    time_before_next_loop = .5
            #This check is here to see if a new sensor was
connected/reconnected
            if accl == None:
                print("Checking", i)
                if accl_old[i]:
                    time.sleep(1)
                lis3dhtr[i] = SensorReinitalization(i)
                if lis3dhtr[i].numSensors >= 1:
                    time.sleep(1)
                    reinit_count[i] += 1
                    print("Sensor Reinitalized",reinit_count)
            time.sleep(1)
        print(time_before_next_loop)
        time.sleep(time_before_next_loop)
```