TWST: An Automated Measurement System for Centripetal Force

Steve Bao, Anthea Empson, Carson Stillman, Cutter Fugett, Percival Skalski, Reehan Siraj

November 2020

Contents

1 Story

We were motivated to create an automated centripetal force measuring device that would be both creative and practical. With these goals in mind, TWST was born. Our sleek and simple design doesn't lack sophistication or robustness and is easily implemented in any lab. Building upon the common instructional experiment in first year physics, we sought to streamline this process by marrying a simple hardware design to an aesthetically pleasing and functional GUI in order to give the user centripetal force readings at the click of a button. Additionally, our loud system is one of the safest on the market ensuring everyone in the room knows when to get out of the way. Customers purchasing TWST will be pleasantly surprised by how easy it is to setup and take measurements.

Click here to view our design story.

1.1 Engineering plan

Mounted on an ABS platform, we have a bearing that allows a rod to rotate. This rod is fixed in a T-shape using various ABS fixtures with the horizontal rod supporting a variable hanging mass at an adjustable radius. Secured to this hanging mass is a three axis LIS3DH accelerometer. Our design affixes the accelerometer at a variable radius using a 3D printed effector which connects to the rotating rod. Since the mass is not allowed to swing, it is only necessary to measure one axis of the accelerometer as it rotates to obtain the centripetal acceleration, from which centripetal force can be calculated using the equation:

$$F_c = ma_c$$

where m is the user inputted mass. Thus it is crucial for the acclerometer to be placed in the holder in the correct orientation. The y-axis (the axis that goes through the mass hanger) must line up parallel with the radial rod (for more information, refer to the Operation Section). In order to ensure the wiring isn't tangled as the mass is rotated, a slip ring was affixed under the base to keep the wiring to the Pi stationary.

In order to power the rotation, we used a gearing system with a Nema-17 stepper motor. To connect the Pi to the stepper, it was necessary to utilize a motor driver. The motor is powered by an AC to DC adapter to barrel jack connector. The Raspberry Pi is powered using the standard Raspberry Pi USB-C power supply.

Click here to view a product commercial.

1.2 CAD Models

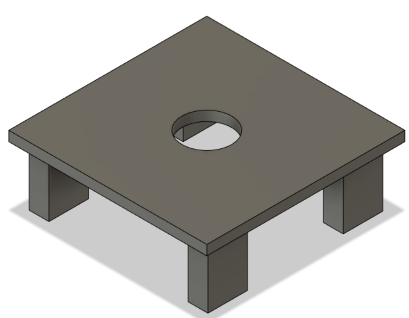


Figure 1: Bottom of base structure

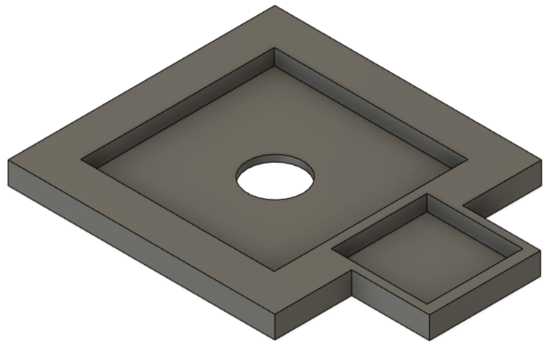


Figure 2: Top of base structure

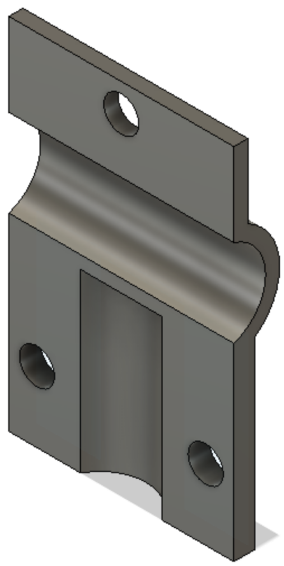


Figure 3: End effecter allowing us to connect the two rods in a T-shape

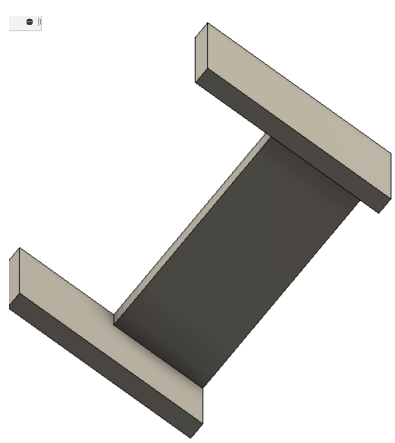


Figure 4: Clamping mechanism to hold the stepper more securely in the base

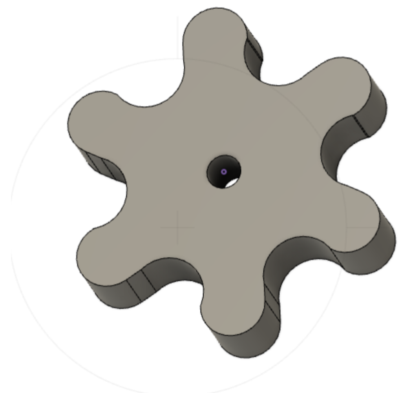


Figure 5: Gear for the motor

Figure 6: Gear for the rod

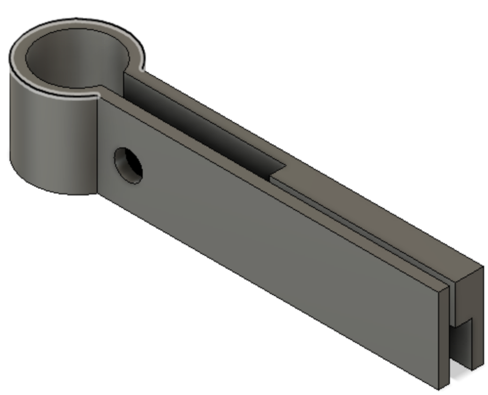


Figure 7: Ring to hold the mass in place

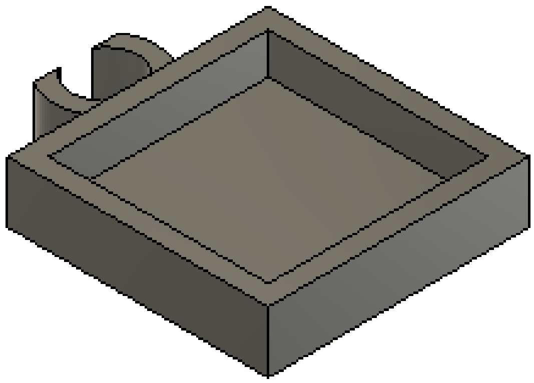


Figure 8: Case for Accelerometer

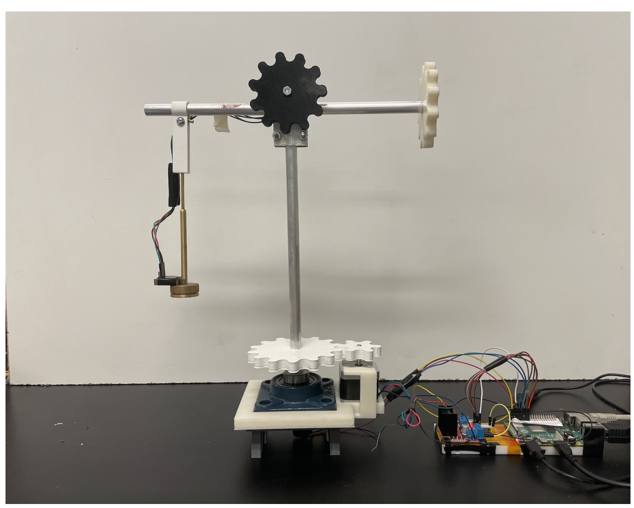


Figure 9: Fully Assembled System

1.3 Full System Blueprint

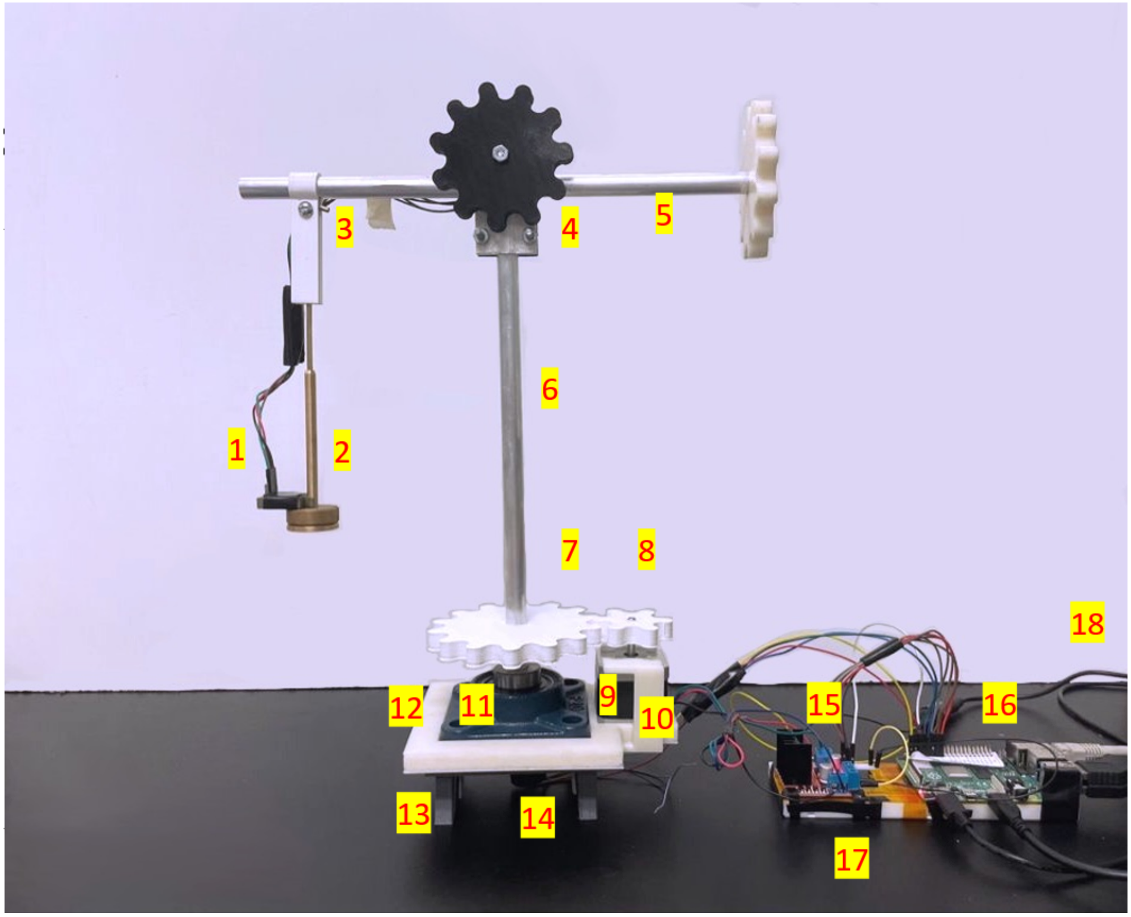


Figure 10: Full System Blueprint

1. Accelerometer in 3D printed carriage
2. Mass holder
3. 3D printed ring to restrict the mass holder's movement
4. Two end effectors holding the horizontal and vertical rods in T formation
5. Horizontal Rod
6. Vertical rod, through which the wires to the accelerometer are fed
7. 3D printed gear, larger
8. 3D printed gear, smaller, affixed to stepper motor
9. Stepper Motor
10. 3D printed clamp to hold stepper motor in place
11. Flange bearing
12. 3D printed base, top piece

13. 3D printed base, bottom piece, affixed to (12) with screws
14. Slipring to allow wire movement
15. Motor Driver for stepper motor
16. Raspberry Pi
17. Portable breadboard, insulated from other components with Kapton Tape
18. Power supply, 12V 2Amp power conversion from wall socket

2 Specs

2.1 Features

- High performance, accuracy
- 12V DC Supply Voltage
- Minimum added mass: 50g
- Maximum added mass: 300g
- Complete Graphical User Interface
- Portability
- Easy Adjustment of Rotation Speed and Radius
- Replaceable assemblies
- Note the period displayed is approximate and should not be used for calculation purposes

2.2 Major Design Decisions

Sensor Selection Choices		
Requirements	**Acceleometer**	Photodiode Ultrasonic Sensor System
Less Source of Error	3	2
Ease of Assembly	3	2
Accuracy	3	2
Portability	3	3
Ease of Wiring	2	3
Total Score	14	12

*Bolder text are better choices chosen in the design process

Motor Selection Choices		
Requirements	**Stepper Motor**	DC Motor
High Torque	3	1
Ease of Assembly	2	3
Control Accuracy	3	2
Price	3	3
Ease of Wiring	2	3
Ease of Motor Holder Design	3	1
Total Score	16	13

Mass Hanger Ring Design Choices		
Requirements	**Constraining Mass Movement**	Enabling Mass Swing Out Freely
Less Source of Error	3	2
Ease of Assembly	3	3
Accuracy	3	2
Portability	3	3
Ease of Wiring	3	3
Ease of Calculation	3	2
System Stability	3	2
Total Score	21	17

Motor and Central Rod Connection Design Choices		
Requirements	**Gear Assemblies**	Directly Connect Central Rod to the Motor
Less Source of Error	3	3
Ease of Assembly	2	3
Accuracy	3	2
Portability	3	3
Ease of Wiring	2	3
Ease of Calculation	3	3
System Stability	3	1
Total Score	19	18

User Interface Design Choices		
Requirements	**Graphical User Interface with PyQt**	Command Line Interface
User Friendly	3	1
Aesthetic	3	1
Coding Difficulty	1	3
Functionality	3	2
Total Score	10	7

3 Tutorial

3.1 System Construction and Electronic Connection

The whole system is preassembled, all the wiring and mechanical parts are connected together and ready to use. However, in case of bad connection or damaged parts. The instruction below provides a guidance to reassemble the system:

1. First, put together the system. Attach the two rods together in a T-shape using the provided screws and the end effecters seen in Figure 3. Next place the bottom end of the vertical rod in the bearing, and attach the bearing in the correct space in the top of the base. Attach the mass hanger to the provided ring, and slip on one end of the horizontal rod. **Ensure the connection ring is tightened securely before operating the system.**

2. Next, it is time to set up the wiring. This is the most complicated part of the set up process. For more information concerning the operation of the accelerometer of the motor driver please refer to the attached documentation. Feed the wires from the slip ring up the vertical rod and out the side of the end effecters so they can reach the accelerometer attached to the mass. Next, connect the accelerometer as follows:

   (a) Vdd to GPIO1 3.3V
   (b) Gnd to any GPIO gnd
   (c) SCL to GPIO3
   (d) SDA to GPIO2

   Make sure that the accelerometer is in the correct orientation.

3. Now, connect the motor. Start by connecting the motor to the motor driver. As follows:

   (a) Black → OUT1
   (b) Blue → OUT2
   (c) Green → OUT3
   (d) Red → OUT4

4. Now, connect the motor driver to the Raspberry Pi as follows:

   (a) ENA pin → pin 18
   (b) IN1 pin → pin 4
   (c) IN2 pin → pin 17
   (d) IN3 → pin 27
   (e) IN4 → pin 22

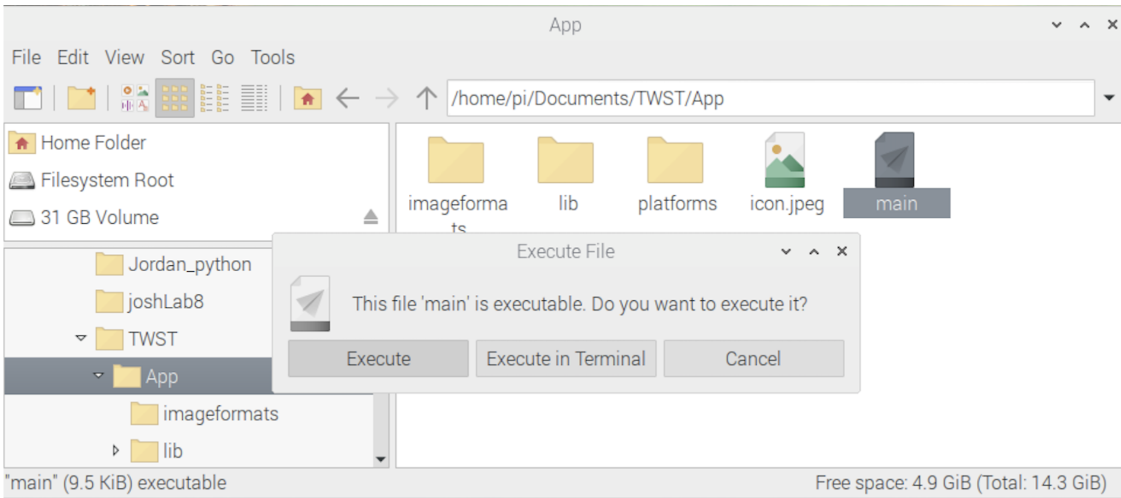
5. Now, time to connect the power source:

   (a) Plug the yellow wire (12v) of the power converter into any red slots on the mini bread board.
   (b) Plug the black wire (ground) of the power converter into any blue slots on the mini bread board.
   (c) Connect the 12v input of the motor driver to the positive 12v converter output on the breadboard.
   (d) Connect the ground of the motor driver to the converter ground on the breadboard.
   (e) Connect the Raspberry Pi ground to the converter ground on the breadboard.
   (f) Plug the converter into wall and the motor driver will light up.

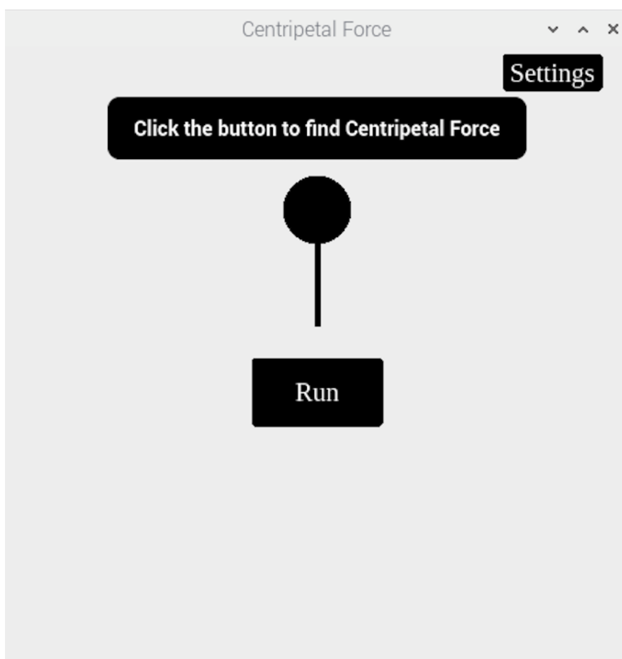
6. Add mass to the system not exceeding 300 g.

3.2 Graphical User Interface Tutorial

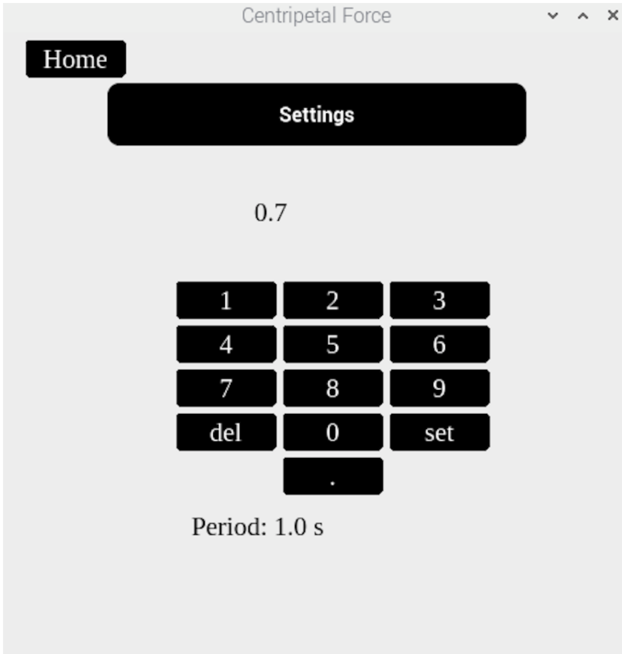
1. To run the program find the executable file 'main' in the enclosing folder. Click on it to open the application and click 'Execute' when prompted. **It is critical all files remain in their appropriate directories. Please do not move any files to different folders.**



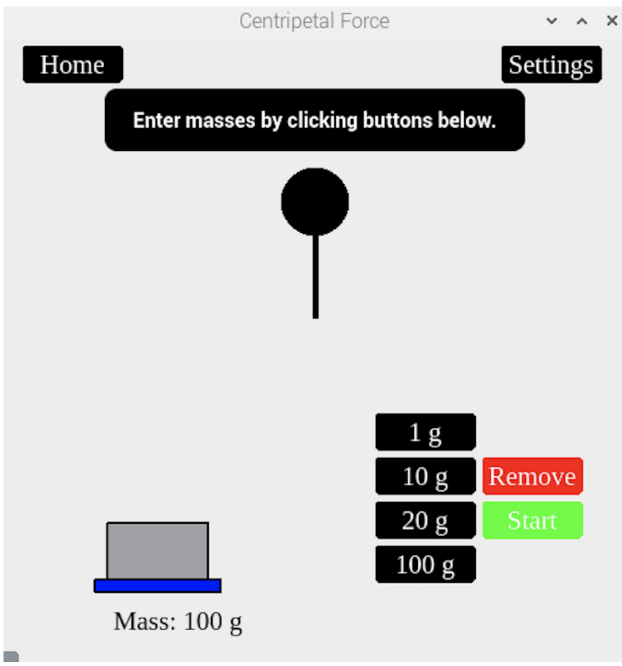
2. Click run to go to the mass screen. In the upper left hand corner the home button will appear. Use this at any time to stop data collections and return back to the main screen.



3. To change the rotation period of the motor, click the 'Settings' button in the upper right corner of the screen. Enter the desired period and click 'Set' to set that as the period. Note this period is approximate and should not be used for calculation purposes.



4. Enter the correct mass you have hung, by clicking the appropriate buttons until the Mass variable reads properly. To remove masses you have erroneously added click 'Remove'.



5. Click 'Run' to initiate data collection. **Take Care to step back after clicking run. The system will start rotating in 3 seconds!**

3.3 Warnings

1. Be careful with the power supply!!!
2. Motor driver heats up, use caution upon handling.
3. Use caution when operating the system, the mass can rotate at high speeds which can be dangerous to those in your immediate surroundings.
4. Ensure not to place more than 300 g of mass on the system
5. Do not enter a period that is too long, this would cause vibration of the stepper motor and affect accuracy.

3.4 FAQ

How do I add mass to the system?
Slip on a labeled mass at the top of the hanging mass shaft and move all the way to the bottom where the shaft widens.

The system appears to be operating fine, yet I get weird readings/errors for the Centripetal Force?
First, check the accelerometer is in the correct orientation, with the y-axis pointing away from the center rod. Second, check to make sure the mass you entered matches the mass on the system.

The motor is spinning but the gears aren't turning, what do I do?
The gears may be slipping due to erosion of the secure fit around the motor shaft. A new gear may be required. If the motor is vibrating rather than spinning, the orientation of the wires may be incorrect.

How do I connect the motor to the motor driver?
Take the four wires of the motor–black, blue, green, and red–and connect them to OUT1, OUT2, OUT3, and OUT4 of the motor driver, respectively.

How do I connect the motor driver to the raspberry pi?
The driver's ENA should be connected to pin 18 of the raspberry pi, IN1 to pin 4, IN2 to pin 17, IN3 to 27, and IN4 to pin 22. The 12VDC source should be connected to the driver's power pin farthest from the inputs (the 5V pin is not used), while the ground is connected to the middle power pin.

Where can I find replacement parts?
Please see the appendix for a list of components and prices.

I keep getting a check your wires error/ it says my accelerometer is set up incorrectly?
This means the system is having trouble establishing connection with the acclerometer. Please ensure all your wires are functioning and that they are connected to the Pi as prescribed in the operation section.

4 Appendix

4.1 Team

Steve Bao – Team Leader and Materials

Anthea Empson – CAD

Carson Stillman – Media, Coder

Cutter Fugett – CAD

Percival Skalski – Electrical Engineering, Coder

Reehan Siraj – Electrical Engineering, Coder

4.2 Additional Documentation

LIS3DH Data Sheet – Click here
Raspberry Pi 4 Data Sheet – Click here
Nema-17 Stepper Motor Data Sheet – Click here
L298N Motor Driver Data Sheet – Click here

4.3 Components

1. 1/2 inch outer diameter aluminum rod $3.21 – Click here
2. 1/2 inch Bore Diameter Bearing $22 – Click here
3. Slip ring $14.95– Click here
4. Nema-17 Stepper Motor $14.00 – Click here
5. L298N Motor Driver $6.69 – Click here
6. Bottom Base – Email cmstillman@email.wm.edu for custom part.
7. Top Base – Email cmstillman@email.wm.edu for custom part.
8. End Effecter – Email cmstillman@email.wm.edu for custom part.
9. Motor Clamp – Email cmstillman@email.wm.edu for custom part.
10. Motor Gear – Email cmstillman@email.wm.edu for custom part.
11. Rod Gear – Email cmstillman@email.wm.edu for custom part.
12. Mass Ring – Email cmstillman@email.wm.edu for custom part.

4.4 Source Code

A GitHub repository with all of the source code and the app can be found and downloaded here: https://github.com/resiraj/TWST. If you are interested in accessing the source code, download the whole repository. If not, only download the folder 'App'.

4.4.1 Main Screen Driver

```
import sys, signal
from PyQt5.QtGui import QPainter, QColor, QFont, QPen, QBrush, QImage
from PyQt5.QtCore import Qt, QRect, QLine
from PyQt5.QtWidgets import QWidget, QApplication, QPushButton, QLabel,QLineEdit
from Button import Button
import math
import time
import threading
from AccelerometerMath import *
from callstepper import *
from statistics import mean
import LIS3DHMOD as acc
import RPi.GPIO as GPIO
import argparse

W_WIDTH = 500
W_HEIGHT = 500


x_list=[]
y_list=[]
z_list=[]

#This code creates all the buttons used in the program
runButton = Button(100,50, W_WIDTH//2-50, W_HEIGHT-250, 'Run')
homeButton = Button(75,25,20,8,'Home')
homeButton.turnOff()
button1 = Button(75,25,W_WIDTH//2+50,W_HEIGHT-200,'1 g')
button1.turnOff()
button10 = Button(75,25,W_WIDTH//2+50,W_HEIGHT-165,'10 g')
button10.turnOff()
button20 = Button(75,25,W_WIDTH//2+50,W_HEIGHT-130,'20 g')
button20.turnOff()
button100 = Button(75,25,W_WIDTH//2+50,W_HEIGHT-95,'100 g')
button100.turnOff()
removeMassButton = Button(75,25,W_WIDTH//2+135,W_HEIGHT-165,'Remove',Qt.red)
removeMassButton.turnOff()
startButton = Button(75,25,W_WIDTH//2+135,W_HEIGHT-130,'Start',Qt.green)
startButton.turnOff()
settingButton = Button(75,25,400,8,'Settings')
settingButton.turnOn()
oneButton = Button(75,25,W_WIDTH//2-110,W_HEIGHT//2- 50,'1')
oneButton.turnOff()
twoButton = Button(75,25,W_WIDTH//2-25,W_HEIGHT//2- 50,'2')
twoButton.turnOff()
```

```
threeButton = Button(75,25,W_WIDTH//2+60,W_HEIGHT//2- 50,'3')
threeButton.turnOff()
fourButton = Button(75,25,W_WIDTH//2-110,W_HEIGHT//2- 15,'4')
fourButton.turnOff()
fiveButton = Button(75,25,W_WIDTH//2-25,W_HEIGHT//2- 15,'5')
fiveButton.turnOff()
sixButton = Button(75,25,W_WIDTH//2+60,W_HEIGHT//2- 15,'6')
sixButton.turnOff()
sevenButton = Button(75,25,W_WIDTH//2-110,W_HEIGHT//2+20,'7')
sevenButton.turnOff()
eightButton = Button(75,25,W_WIDTH//2-25,W_HEIGHT//2+20,'8')
eightButton.turnOff()
nineButton = Button(75,25,W_WIDTH//2+60,W_HEIGHT//2+20,'9')
nineButton.turnOff()
delButton = Button(75,25,W_WIDTH//2-110,W_HEIGHT//2+55,'del')
delButton.turnOff()
zeroButton = Button(75,25,W_WIDTH//2-25,W_HEIGHT//2+55,'0')
zeroButton.turnOff()
setButton = Button(75,25,W_WIDTH//2+60,W_HEIGHT//2+55,'set')
setButton.turnOff()
pointButton = Button(75,25,W_WIDTH//2-25,W_HEIGHT//2+90,'.')
pointButton.turnOff()
#To draw all the buttons they needed to be added to a button list
buttonList = [pointButton,runButton,button1,button10,button20,button100,
    homeButton,removeMassButton,startButton,settingButton,oneButton,
        twoButton,threeButton,fourButton,fiveButton,sixButton,sevenButton,
            eightButton,nineButton,delButton,zeroButton,setButton]

class main_screen(QWidget):
    def __init__(self):
        #initializes all the variables necessary to run the program
        super().__init__()
        self.setWindowTitle('Centripetal Force')
        self.setGeometry(610, 300, W_WIDTH,W_HEIGHT)
        self.circle_Xcoord = W_WIDTH//2-25
        self.circle_Ycoord = W_HEIGHT//2-145
        self.rod_Xcoord = W_WIDTH//2
        self.rod_Ycoord = W_HEIGHT//2-30
        self.angle = 270
        self.reset = False
        self.displayResults = False
        self.errorScreen = False
        self.settingScreen = False
        self.period = .7
        self.calc = None
        self.force = 0
```

```
        self.xinit = 0
        self.yinit = 0
        self.zinit = 0

        self.text = 'Click the button to find Centripetal Force'
        self.topLabel = QLabel(self.text,self)
        self.topLabel.move(W_WIDTH//2-167,30)
        self.topLabel.setAlignment(Qt.AlignCenter)
        self.topLabel.setStyleSheet("background-color: black; border: none;
         color: white; font: bold 16px; border-style: outset;border-radius: 10px;")
        self.topLabel.setGeometry(W_WIDTH//2-167,40,334,50)

        try:
            self.setUpAccel()   #Tries to set up accelerometer
        except:
            self.text = 'You set up the accelerometer wrong!'
            #if it cant prints an error to the program
            runButton.turnOff()
            homeButton.turnOn()

        self.massScreen = False #keeps track of where the program is in its execution
        self.massesEntered = False
        self.massArray = []
        self.mass = 0
        self.show()

    def paintEvent(self, event):
        self.topLabel.setText(self.text)
        qp = QPainter()
        qp.begin(self)

        qp.setPen(QPen(QBrush(Qt.black),5))
        qp.setFont(QFont('Times', 16))

        if self.settingScreen == False:
            #draws the circle and bar in the middle of the screen,
            # not drawn when in the setting screen
            qp.drawLine(QLine(self.rod_Xcoord,self.rod_Ycoord,self.circle_Xcoord+25,
                self.circle_Ycoord+25))
            qp.setBrush(QBrush(Qt.black, Qt.SolidPattern))
            qp.drawEllipse(self.circle_Xcoord,self.circle_Ycoord,50,50)

        for item in buttonList:
            #draws all the buttons
            item.draw(qp)
```

```
if self.settingScreen == False:
    if self.massScreen == True:
        # when the mass screen needs to be drawn
        qp.setPen(QPen(QBrush(Qt.blue), 2))
        qp.setBrush(QBrush(Qt.blue))
        qp.drawRect(75,W_HEIGHT-70,100,10)
        qp.setPen(QPen(QBrush(Qt.black),2))
        qp.drawRect(75,W_HEIGHT-70,100,10)
        drawNextMassY = W_HEIGHT-70
        #draws the masses in the bottom left corner
        for item in self.massArray:
            qp.setBrush(QBrush(Qt.gray))
            if item == 1:
                height = 6
            elif item == 10:
                height = 15
            elif item == 20:
                height = 30
            else:
                height = 45
            qp.drawRect(85,drawNextMassY-(height),80,height)
            drawNextMassY = drawNextMassY-(height)
            qp.drawText(90,W_HEIGHT-30,'Mass: '+str(self.mass)+' g')
else:
    # this happens when youre in the settings screen
    qp.setPen(QPen(QBrush(Qt.black), 3))
    qp.drawText(150,400,'Period: '+str(self.period) +' s')
    if self.calc != None:
        qp.drawText(200,150,str(self.calc))

if self.displayResults == True:
    #this happens when results are displayed
    qp.setPen(QPen(QBrush(Qt.black), 2))
    if self.force == 0:
        #gives the user some input to let them know things are operating as
        # expected
        qp.drawText(125,250,'Calculating...')
    else:
        if isinstance(self.force,str)==False:
            qp.drawText(125,250,'Centripetal Force: '+str(round(self.force,2))+'
            qp.drawText(125,300,'Mass: '+str(self.mass)+' g')
            qp.drawText(125,350,'Period: '+str(self.period)+' s')
        else:

            qp.drawText(125,250,'Centripetal Force: '+self.force)
            qp.drawText(125,300,'Mass: '+str(self.mass)+' g')
```

```python
def mousePressEvent(self, event):
    xCord = event.x()
    yCord = event.y()

    if settingButton.buttonClicked(xCord,yCord):
        self.settingScreen = True
        self.text = 'Settings'
        homeButton.turnOn()
        runButton.turnOff()
        settingButton.turnOff()
        button1.turnOff()
        button10.turnOff()
        button20.turnOff()
        button100.turnOff()
        removeMassButton.turnOff()
        startButton.turnOff()
        oneButton.turnOn()
        twoButton.turnOn()
        threeButton.turnOn()
        fourButton.turnOn()
        fiveButton.turnOn()
        sixButton.turnOn()
        sevenButton.turnOn()
        eightButton.turnOn()
        nineButton.turnOn()
        delButton.turnOn()
        zeroButton.turnOn()
        setButton.turnOn()
        pointButton.turnOn()
        self.update()

    if homeButton.buttonClicked(xCord,yCord):
        runButton.turnOn()
        settingButton.turnOn()
        homeButton.turnOff()
        button1.turnOff()
        button10.turnOff()
        button20.turnOff()
        button100.turnOff()
        removeMassButton.turnOff()
        oneButton.turnOff()
        twoButton.turnOff()
        threeButton.turnOff()
        fourButton.turnOff()
        fiveButton.turnOff()
```

```
        sixButton.turnOff()
        sevenButton.turnOff()
        eightButton.turnOff()
        nineButton.turnOff()
        delButton.turnOff()
        zeroButton.turnOff()
        setButton.turnOff()
        pointButton.turnOff()
        x_list=[]
        y_list=[]
        z_list=[]
        self.settingScreen = False
        startButton.turnOff()
        self.text = 'Click the button to find Centripetal Force'
        self.mass = 0
        self.calc = None
        self.force = 0
        try:
            self.setUpAccel()
        except:
            self.text = 'You set up the accelerometer wrong!'
            runButton.turnOff()
            homeButton.turnOn()
        self.massScreen = False
        self.displayResults = False
        self.errorScreen = False
        self.reset = True
        self.massesEntered = False
        self.circle_Xcoord = W_WIDTH//2-25
        self.circle_Ycoord = W_HEIGHT//2-145
        self.rod_Xcoord = W_WIDTH//2
        self.rod_Ycoord = W_HEIGHT//2-30
        self.massArray = []
        self.update()

    if runButton.buttonClicked(xCord,yCord):
        runButton.turnOff()
        homeButton.turnOn()
        self.text = 'Enter masses by clicking buttons below.'
        button1.turnOn()
        button10.turnOn()
        button20.turnOn()
        button100.turnOn()
        removeMassButton.turnOn()
        startButton.turnOn()
        self.massScreen = True
```

```
        self.update()

    if button1.buttonClicked(xCord,yCord):
        self.massArray.append(1)
        self.mass += 1
        self.update()
    if button10.buttonClicked(xCord,yCord):
        self.massArray.append(10)
        self.mass += 10
        self.update()
    if button20.buttonClicked(xCord,yCord):
        self.massArray.append(20)
        self.mass += 20
        self.update()
    if button100.buttonClicked(xCord,yCord):
        self.massArray.append(100)
        self.mass += 100
        self.update()
    if removeMassButton.buttonClicked(xCord,yCord):
        if len(self.massArray)>0:
            massToRemove = self.massArray.pop()
            self.mass = self.mass - massToRemove
        self.update()
    if startButton.buttonClicked(xCord,yCord):
        self.massesEntered = True
        button1.turnOff()
        button10.turnOff()
        button20.turnOff()
        button100.turnOff()
        removeMassButton.turnOff()
        settingButton.turnOff()
        startButton.turnOff()
        self.countdown_thread =
            threading.Thread(target=self.countdown, args=())
        self.countdown_thread.start()
        self.animation_thread =
            threading.Thread(target = self.moveCircle,
                    args=(self.angle,(self.circle_Xcoord,self.circle_Ycoord+90)))
        self.animation_thread.start()

    if oneButton.buttonClicked(xCord,yCord):
        if self.calc != None:
            self.calc += '1'
        else:
            self.calc = '1'
        self.update()
```

```
if twoButton.buttonClicked(xCord,yCord):
    if self.calc != None:
        self.calc += '2'
    else:
        self.calc = '2'
    self.update()

if threeButton.buttonClicked(xCord,yCord):
    if self.calc != None:
        self.calc += '3'
    else:
        self.calc = '3'
    self.update()

if fourButton.buttonClicked(xCord,yCord):
    if self.calc != None:
        self.calc += '4'
    else:
        self.calc = '4'
    self.update()

if fiveButton.buttonClicked(xCord,yCord):
    if self.calc != None:
        self.calc += '5'
    else:
        self.calc = '5'
    self.update()

if sixButton.buttonClicked(xCord,yCord):
    if self.calc != None:
        self.calc += '6'
    else:
        self.calc = '6'
    self.update()

if sevenButton.buttonClicked(xCord,yCord):
    if self.calc != None:
        self.calc += '7'
    else:
        self.calc = '7'
    self.update()

if eightButton.buttonClicked(xCord,yCord):
    if self.calc != None:
        self.calc += '8'
```

```
        else:
            self.calc = '8'
        self.update()

    if nineButton.buttonClicked(xCord,yCord):
        if self.calc != None:
            self.calc += '9'
        else:
            self.calc = '9'
        self.update()

    if zeroButton.buttonClicked(xCord,yCord):
        if self.calc != None:
            self.calc += '0'
        else:
            self.calc = '0'
        self.update()

    if pointButton.buttonClicked(xCord,yCord):
        if self.calc != None:
            self.calc += '.'
        else:
            self.calc = '.'
        self.update()

    if delButton.buttonClicked(xCord,yCord):
        if self.calc != None:
            self.calc = self.calc[:len(self.calc)-1]
        self.update()

    if setButton.buttonClicked(xCord,yCord):
        if self.calc != None and self.calc != '0':
            try:
                self.period = float(self.calc)
                self.calc = None
                self.text = 'Settings'
            except:
                self.text = 'You need to enter a number you dolt!'
                self.calc = None
        if self.calc == '0':
            self.text = 'Please enter a non-zero value!'
            self.calc = None


        self.update()
```

```
def runMotor(self):
    # 4-wire bipolar stepper motor - NEMA-17 42BYGHW609
    GPIO.setmode(GPIO.BCM)

    # Enable pins for IN1-4
    control_pin = [4,17,27,22]
    delay = self.period/1390.0 #change for speed

    # IN1-4 pin setup
    for pin in control_pin:
        GPIO.setup(pin, GPIO.OUT)
        GPIO.output(pin, 0) #counter-clockwise rotation

    halfstep_seq = [[1,0,0,0],
                    [1,1,0,0],
                    [0,1,0,0],
                    [0,1,1,0],
                    [0,0,1,0],
                    [0,0,1,0],
                    [0,0,0,1],
                    [1,0,0,1]]

    #setting stepper
    timevar = 10.0//delay
    counter = 0
    while counter<timevar:
        for i in range(512):
            if self.reset== True:
                GPIO.cleanup()
                break
            for step in range(8):
                for pin in range(4):
                    GPIO.output(control_pin[pin],halfstep_seq[step][pin])
                time.sleep(delay)
                counter += 1
    GPIO.cleanup()

def moveCircle(self,angle,coordsCirc):
    self.reset = False
    while self.countdown_thread.is_alive():
        pass

    stepperThread = threading.Thread(target=self.runMotor, args=())
    stepperThread.start()
    timeInitial = time.time()
    while 1:
```

```
try:
    if time.time()-timeInitial > 2 and time.time()-timeInitial < 8:
        x = self.config.getX()
        y = self.config.getY()
        z = self.config.getZ()

        x_list.append(x)
        y_list.append(y)
        z_list.append(z)
except:
    self.errorScreen = True
    self.massScreen = False
    self.text = "Check your wires. Something's amiss!"

angle += 4
theta = math.radians(angle)
self.circle_Xcoord = coordsCirc[0] + (90 * math.cos(theta))
self.circle_Ycoord = coordsCirc[1] + (90 * math.sin(theta))
self.update()
time.sleep(.01)

if stepperThread.is_alive() == False:
    xmean = mean(x_list)
    ymean = mean(y_list)
    zmean = mean(z_list)

    accelArray = [xmean,ymean,zmean]

    accelArray0 = [self.xinit, self.yinit, self.zinit]

    self.circle_Xcoord = W_WIDTH//2-25
    self.circle_Ycoord = W_HEIGHT//2-145
    self.rod_Xcoord = W_WIDTH//2
    self.rod_Ycoord = W_HEIGHT//2-30

    self.displayResults = True
    self.massScreen = False
    self.text = 'Centripetal Force'
    self.update()

    #print(accelArray)
    try:
        self.force = findforce(accelArray,accelArray0,self.mass)
    except:
        self.force = '''There was an unexpected error'''
```

```
                self.update()
                break

            if self.reset == True:
                self.circle_Xcoord = W_WIDTH//2-25
                self.circle_Ycoord = W_HEIGHT//2-145
                self.rod_Xcoord = W_WIDTH//2
                self.rod_Ycoord = W_HEIGHT//2-30
                break

    def countdown(self):
        while self.massesEntered == False:
            pass
        for i in range(3,0,-1):
            self.text = 'Please Stand Back! Starting in '+str(i)
            self.update()
            time.sleep(1)
            if self.reset == True:
                break
        if self.reset == False:
            self.text = 'Please Stand Back!'

    def setUpAccel(self):
        self.config = acc.LIS3DH()
        self.config.setRange(0b00)
        self.config.setDataRate(0b0101)

        self.xinit = self.config.getX()
        self.yinit = self.config.getY()
        self.zinit = self.config.getZ()


def main():
  app = QApplication(sys.argv)
  ex = main_screen()
  sys.exit(app.exec_())
```

4.4.2 Custom Button Class

```
from PyQt5.QtGui import QPainter, QColor, QFont, QPen, QBrush
from PyQt5.QtCore import Qt, QRect
from PyQt5.QtWidgets import QWidget, QApplication, QPushButton
from AccelerometerMath import *

class Button():
    def __init__(self, buttonWidth, buttonHeight, xPos, yPos, text,
         buttonColor = Qt.black, textColor = Qt.white):
```

```
        self.buttonWidth = buttonWidth
        self.buttonHeight =  buttonHeight
        self.xPos = xPos
        self.yPos = yPos
        self.text = text
        self.buttonColor = buttonColor
        self.textColor = textColor
        self.clickable = True

    def isClickable(self):
        return self.clickable

    def turnOn(self):
        self.clickable = True

    def turnOff(self):
        self.clickable = False

    def draw(self, qp):
        if self.clickable==True:
            textColorPen = QPen(QBrush(self.textColor), 5)
            buttonColorPen = QPen(QBrush(self.buttonColor), 5)
            qp.setPen(buttonColorPen)
            qp.setBrush(self.buttonColor)
            qp.drawRect(self.xPos, self.yPos, self.buttonWidth,
                self.buttonHeight)
            qp.setPen(textColorPen)
            qp.drawText(QRect(self.xPos, self.yPos, self.buttonWidth,
                 self.buttonHeight), Qt.AlignCenter, self.text)

    def buttonClicked(self, x, y):
        if self.clickable == False:
            return False
        return(self.xPos<x<self.xPos+self.buttonWidth and
            self.yPos<y<self.yPos+self.buttonHeight)

    def getX(self):
        return self.xPos

    def getY(self):
        return self.yPos

    def getButtonWidth(self):
        return self.buttonWidth

    def getButtonHeight(self):
```

```
        return self.getButtonHeight
```

4.4.3 Accelerometer Math

```
from sympy import Symbol, solve, cos, sin, atan

'''
This code will do the math to turn the accelerometer reading into a
measurement of the centrifugal force
'''

#e1 points down
#e2 points in the radial direction
#e3 points tangent to the radial direction

def findforce(accelArray,accelArray0, mass):

    e1 = accelArray[2]  #z
    e2 = accelArray[1]  #y
    e3 = accelArray[0]  #x

    e10 = accelArray0[2]  #z0
    e20 = accelArray0[1]  #y0
    e30 = accelArray0[0]  #x0

    theta = atan(e10/e20)

    a_c = abs(e1*cos(theta)-e2*sin(theta))

    f_c = 9.8 * a_c * mass /1000

    return f_c
```

4.4.4 Accelerometer Code

```
#!/usr/bin/env python3
"""LIS3DH triple-axis accelerometer
LIS3DH Python module for Raspberry Pi

   ~ Created by Matt Dyson (mattdyson.org)
   ~ Bill Cooke adapted this to remove the Adafruit I2C
   library in favor of standard smbus2
   ~ Ran Yang modified this module  9/25/2020

££ Python 2 reached EOL on 1/1/2020
££ Enforce this script to run by default in python3, currently python3.8.5
```

```
££ chip address:
   ~£ sudo 12cdetect -y 1
        0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- 18 -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
   ~ the channel showing is 1
   ~ bus = smbus2.SMBus(1)    # 0 = /dev/i2c-0 (port I2C0),
   1 = /dev/i2c-1 (port I2C1)
   ~ the address showing is 0x18 (Hex 19 = decimal 52)

££ Pin Configuration
Raspberry Pi 4 GPIO2 - SDA and GPIO3 - SCL are I2C serial pins.
To connect LIS3DH to a Raspberry Pi using I2C:
   ~ Vdd to GPIO1 3.3V
   ~ Gnd to any GPIO gnd
   ~ SCL to GPIO3
   ~ SDA to GPIO2
   ~ Leave 3Vo float
   ~ Leave the rest float
To connect LIS3DH to a Raspberry Pi using SPI:
   ~ Vdd to GPIO1 3.3V
   ~ Gnd to any GPIO gnd
   ~ SCL to GPIO11 SCLK
   ~ SDA to GPIO10 MOSI
   ~ SDO to GPIO09 MISO
   ~ CS  to GPIO anypin, active low, so drop it low to start SPI
   data communication
   ~ Leave 3Vo float, this is a output pin from the chip, max 100mA
   ~ Leave the INI float
   ~ Connect multiple chips to the same master, the chips can share
   the data and clock with different chip select pins

"""

import smbus2, time
import RPi.GPIO as GPIO        #needed for Hardware interrupt

class LIS3DH:

   # Ranges
```

```
RANGE_2G          = 0b00
RANGE_4G          = 0b01
RANGE_8G          = 0b10
RANGE_16G   = 0b11

# Refresh rates
DATARATE_400HZ          = 0b0111 # 400Hz
DATARATE_200HZ          = 0b0110 # 200Hz
DATARATE_100HZ          = 0b0101 # 100Hz
DATARATE_50HZ           = 0b0100 # 50Hz
DATARATE_25HZ           = 0b0011 # 25Hz
DATARATE_10HZ           = 0b0010 # 10Hz
DATARATE_1HZ            = 0b0001 # 1Hz
DATARATE_POWERDOWN      = 0      # Power down
DATARATE_LOWPOWER_1K6HZ = 0b1000 # Low power mode (1.6KHz)
DATARATE_LOWPOWER_5KHZ  = 0b1001 # Low power mode (5KHz) /
#Normal power mode (1.25KHz)

# Registers
REG_STATUS1       = 0x07
REG_OUTADC1_L     = 0x08
REG_OUTADC1_H     = 0x09
REG_OUTADC2_L     = 0x0A
REG_OUTADC2_H     = 0x0B
REG_OUTADC3_L     = 0x0C
REG_OUTADC3_H     = 0x0D
REG_INTCOUNT      = 0x0E
REG_WHOAMI        = 0x0F # Device identification register
REG_TEMPCFG       = 0x1F
REG_CTRL1         = 0x20 # Used for data rate selection,
#and enabling/disabling individual axis
REG_CTRL2         = 0x21
REG_CTRL3         = 0x22
REG_CTRL4         = 0x23 # Used for BDU, scale selection,
#resolution selection and self-testing
REG_CTRL5         = 0x24
REG_CTRL6         = 0x25
REG_REFERENCE     = 0x26
REG_STATUS2       = 0x27
REG_OUT_X_L       = 0x28
REG_OUT_X_H       = 0x29
REG_OUT_Y_L       = 0x2A
REG_OUT_Y_H       = 0x2B
REG_OUT_Z_L       = 0x2C
REG_OUT_Z_H       = 0x2D
REG_FIFOCTRL      = 0x2E
```

```
REG_FIFOSRC        = 0x2F
REG_INT1CFG        = 0x30
REG_INT1SRC        = 0x31
REG_INT1THS        = 0x32
REG_INT1DUR        = 0x33
REG_CLICKCFG       = 0x38
REG_CLICKSRC       = 0x39
REG_CLICKTHS       = 0x3A
REG_TIMELIMIT      = 0x3B
REG_TIMELATENCY    = 0x3C
REG_TIMEWINDOW     = 0x3D

# Values
DEVICE_ID      = 0x33
INT_IO                  = 0x04        # GPIO pin for interrupt
CLK_NONE       = 0x00
CLK_SINGLE     = 0x01
CLK_DOUBLE     = 0x02


AXIS_X         = 0x00
AXIS_Y         = 0x01
AXIS_Z         = 0x02

def __init__(self, address=0x18, bus=-1, debug=False):
   self.isDebug = debug
   self.debug("Initialising LIS3DH")
   self.bus = smbus2.SMBus(1)
   self.address = address

   try:
      val = self.bus.read_byte_data(self.address,self.REG_WHOAMI)
      #val = self.i2c.readU8(self.REG_WHOAMI)
      if val!=self.DEVICE_ID:
         raise Exception("Device ID incorrect - expected 0x%X,
            got 0x%X at address 0x%X" % (self.DEVICE_ID, val, self.address))
      self.debug(
         "Successfully connected to LIS3DH at address 0x%X" % (self.address))
   except Exception as e:
      print("Error establishing connection with LIS3DH")
      print(e)

   # Enable all axis
   self.setAxisStatus(self.AXIS_X, True)
   self.setAxisStatus(self.AXIS_Y, True)
   self.setAxisStatus(self.AXIS_Z, True)
```

```
    # Set 400Hz refresh rate
    self.setDataRate(self.DATARATE_400HZ)

    self.setHighResolution()
    self.setBDU()

    self.setRange(self.RANGE_2G)

# Get reading from X axis
def getX(self):
    return self.getAxis(self.AXIS_X)

# Get reading from Y axis
def getY(self):
    return self.getAxis(self.AXIS_Y)

# Get reading from Z axis
def getZ(self):
    return self.getAxis(self.AXIS_Z)

# Get a reading from the desired axis
def getAxis(self, axis):
    base = self.REG_OUT_X_L + (2 * axis) # Determine which register we need
        #to read from (2 per axis)

    low = self.bus.read_byte_data(self.address,base)
    # Read the first register (lower bits)
    high = self.bus.read_byte_data(self.address,base+1)
    # Read the next register (higher bits)
    res = low | (high << 8) # Combine the two components
    res = self.twosComp(res) # Calculate the twos compliment of the result

    # Fetch the range we're set to, so we can accurately calculate the result
    range = self.getRange()
    divisor = 1
    if range==self.RANGE_2G:    divisor = 16380
    elif range==self.RANGE_4G:  divisor = 8190
    elif range==self.RANGE_8G:  divisor = 4096
    elif range==self.RANGE_16G: divisor = 1365.33

    return float(res) / divisor

# Get the range that the sensor is currently set to
def getRange(self):
    val = self.bus.read_byte_data(self.address,self.REG_CTRL4)
```

```
    # Get value from register
    val = (val >> 4) # Remove lowest 4 bits
    val &= 0b0011 # Mask off two highest bits

    if val==self.RANGE_2G: return self.RANGE_2G
    elif val==self.RANGE_4G: return self.RANGE_4G
    elif val==self.RANGE_8G: return self.RANGE_8G
    else: return self.RANGE_16G

# Set the range of the sensor (2G, 4G, 8G, 16G)
def setRange(self, range):
    if range<0 or range>3:
        raise Exception("Tried to set invalid range")

    val = self.bus.read_byte_data(self.address,self.REG_CTRL4)
    # Get value from register
    val &= ~(0b110000) # Mask off lowest 4 bits
    val |= (range << 4) # Write in our new range
    self.writeRegister(self.REG_CTRL4, val)
    # Write back to register

# Enable or disable an individual axis
# Read status from CTRL_REG1, then write
# back with appropriate status bit changed
def setAxisStatus(self, axis, enable):
     if axis<0 or axis>2:
         raise Exception("Tried to modify invalid axis")

     current = self.bus.read_byte_data(self.address,self.REG_CTRL1)
     status = 1 if enable else 0
     final = self.setBit(current, axis, status)
     self.writeRegister(self.REG_CTRL1, final)

def setInterrupt(self,mycallback):
      GPIO.setmode(GPIO.BCM)
      GPIO.setup(self.INT_IO, GPIO.IN)
      GPIO.add_event_detect(self.INT_IO, GPIO.RISING, callback=mycallback)

def setClick(self,clickmode,clickthresh=80,timelimit=10,
    timelatency=20,timewindow=100,mycallback=None):
      if (clickmode==self.CLK_NONE):
          val = self.bus.read_byte_data(self.address,REG_CTRL3)
          # Get value from register
          val &= ~(0x80) # unset bit 8 to disable interrupt
          self.writeRegister(self.REG_CTRL3, val) # Write back to register
          self.writeRegister(self.REG_CLICKCFG, 0) # disable all interrupts
```

```
            return
        self.writeRegister(self.REG_CTRL3, 0x80)  # turn on int1 click
        self.writeRegister(self.REG_CTRL5, 0x08)  # latch interrupt on int1

        if (clickmode == self.CLK_SINGLE):
            self.writeRegister(self.REG_CLICKCFG, 0x15) # turn on all axes & singletap
        if (clickmode == self.CLK_DOUBLE):
            self.writeRegister(self.REG_CLICKCFG, 0x2A) # turn on all axes & doubletap

# set timing parameters
        self.writeRegister(self.REG_CLICKTHS, clickthresh)
        self.writeRegister(self.REG_TIMELIMIT, timelimit)
        self.writeRegister(self.REG_TIMELATENCY, timelatency)
        self.writeRegister(self.REG_TIMEWINDOW, timewindow)

        if mycallback != None:
            self.setInterrupt(mycallback)


   def getClick(self):
        reg = self.bus.read_byte_data(self.address,self.REG_CLICKSRC)
        # read click register
        self.bus.read_byte_data(self.address,self.REG_INT1SRC)
         # reset  interrupt flag
        return reg

   # Set the rate (cycles per second) at which data is gathered

   def setDataRate(self, dataRate):
      val = self.bus.read_byte_data(self.address,self.REG_CTRL1) # Get current value
      val &= 0b1111 # Mask off lowest 4 bits
      val |= (dataRate << 4) # Write in our new data rate to highest 4 bits
      self.writeRegister(self.REG_CTRL1, val) # Write back to register

   # Set whether we want to use high resolution or not
   def setHighResolution(self, highRes=True):
      val = self.bus.read_byte_data(self.address,self.REG_CTRL4) # Get current value
      status = 1 if highRes else 0
      final = self.setBit(val, 3, status) # High resolution is bit 4 of REG_CTRL4
      self.writeRegister(self.REG_CTRL4, final)

   # Set whether we want to use block data update or not
   # False = output registers not updated until MSB and LSB reading
   def setBDU(self, bdu=True):
      val = self.bus.read_byte_data(self.address,self.REG_CTRL4) # Get current value
      status = 1 if bdu else 0
```

```python
        final = self.setBit(val, 7, status) # Block data update is bit 8 of REG_CTRL4
        self.writeRegister(self.REG_CTRL4, final)

    # Write the given value to the given register
    def writeRegister(self, register, value):
        self.debug("WRT %s to register 0x%X" % (bin(value), register))
        self.bus.write_byte_data(self.address, register, value)

    # Print a register
    def printRegister(self, register):
        print(f'LIS3DH I2C address is {self.bus.read_byte_data(self.address,register)}')

    # Set the bit at index 'bit' to 'value' on 'input' and return
    def setBit(self, input, bit, value):
        mask = 1 << bit
        input &= ~mask
        if value:
            input |= mask
        return input

    # Return a 16-bit signed number (two's compliment)
    def twosComp(self,x) :
        if (0x8000 & x):
            x = - (0x010000 - x)
        return x

    # Print an output of all registers
    def dumpRegisters(self):
        for x in range(0x0, 0x3D):
            read = self.bus.read_byte_data(self.address,x)
            print( "%X: %s" % (x, bin(read)))

    def debug(self, message):
        if not self.isDebug: return
        print(message)
```