Birdemic

Circular Motion Lab Apparatus User Manual

A Final Project for PHYS 351: Scientific Instrumentation Lab

Produced By

Christopher Egerstrom (Project Manager)
Seth Gnesin
Ali Hassan
Jake Silliman
Alex Toyryla
Max Weinhold

College of William & Mary
Physics Department

1 Our Story

1.1 Design Background

When we set out to build a new centripetal acceleration apparatus, our goals were simplicity and ease of use. We saw the experimental setup used in the introductory physics labs and noticed two fatal flaws which we wanted to address. First, people using the original experimental apparatus had to manually spin the axle to a specific speed such that the mass swung out at a specific radius. For measurement, they had to indirectly measure the angular acceleration and indirectly measure the rotation speed. Further, trying to get the radius of rotation to match the desired radius exactly, with no skew angle outward, while checking by eye and spinning the axle, is a recipe for poor measurements. We saw the error coming from the manual spinning and the eyeballed radii and our team mission was to remove the error by simplifying. Instead of adding complex sensors or devising a complex system for measuring the centripetal acceleration, we went for simplicity and created the **Birdemic**.

1.2 Build Process

Our team process began with a basic drawing and an idea. Instead of imprecise manual spinning, we decided to use a stepper motor so that we could control the rotation speed and ensure that sufficient torque was output to turn the apparatus regardless of the mass. From the start, we considered and then quickly discounted using any device that would have been placed on the articulating arm itself because it would have had a much higher risk of failure and been costlier to replace. Instead, our initial hand drawing, Figure 1, had an aluminum stock arm rotating on the stepper with a movable carriage on one side for the variable mass and radius and a counterweight on the opposite side. As visible in the first drawing, we were considering using a photodiode coupled with a LED for sensing the period of rotation and a laser rangefinder for measuring the radius, from which we could calculate the centripetal acceleration and force.

From a mechanical engineering perspective, the trajectory of the process was straight forward. We designed and redesigned carriages that would fit on our arm and hold the masses without providing them much room to shift. A mechanical drawing for the carriages, which were 3D printed with 0.5mm ABS Plastic filament on an Airwolf EVO, are included as part of the component list in Appendix B as Figure 5. The base itself is a simple plywood construction with 5/8" wooden dowel supports separating the 1/4" boards. Our initial designs, like the one in Figure 1, had a square base where the sensor column was connected, but to make a more portable apparatus, we slimmed down the base to a rectangle to get rid of extra wood.

Our team's electrical engineering process was very component-based. We got the photodiode, laser rangefinder, and stepper motor each working independently of the others and then worked to integrate them into a cohesive program. Initial tests for the photodiode, motor, and laser rangefinder are linked. Then, with the shaft assembly also complete, we ran a first test integrating all three electronic components together, albeit not setting them up properly, which is here. Coding was done in parallel to the electrical engineering work, so the set of functions for each component was written independently and then imported into

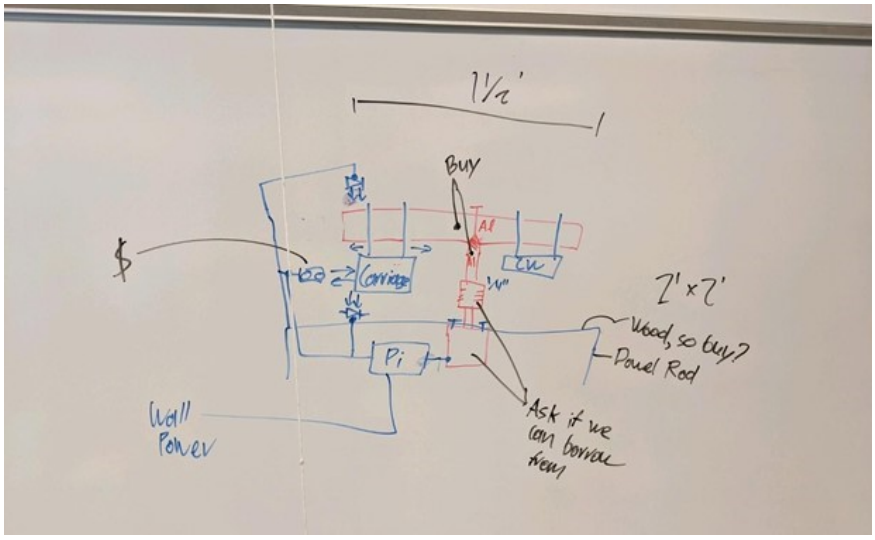

Figure 1: Drawing on a whiteboard of the initial design of the apparatus.

the final program. All the completed code is given in Appendix C with the program in C.1 and the source functions in C.2.

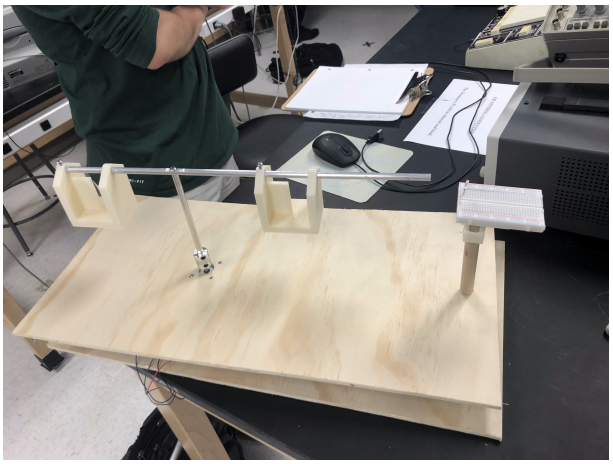

Figure 2: Photograph of the base with the motor and tower mounted but missing the electronics.

With everything coming together, we began to run testing and mount the motor and components on the wooden base as shown in Figure 2. The full circuit wiring diagram is shown in Figure 4. Video of the initial testing is linked here and here. In these tests, we were primarily looking to stabilize the system and debug the code, as well as determine any points of troubleshooting for inclusion for Section 5. To stabilize the system, we added gorilla tape to the flex coupling to make it more rigid. We also added a tape flag to the end of the rod in order to make the photodiode measurements easier to take. With all that done, we cleaned up the base to make the apparatus more aesthetically appealing and tightened up the code

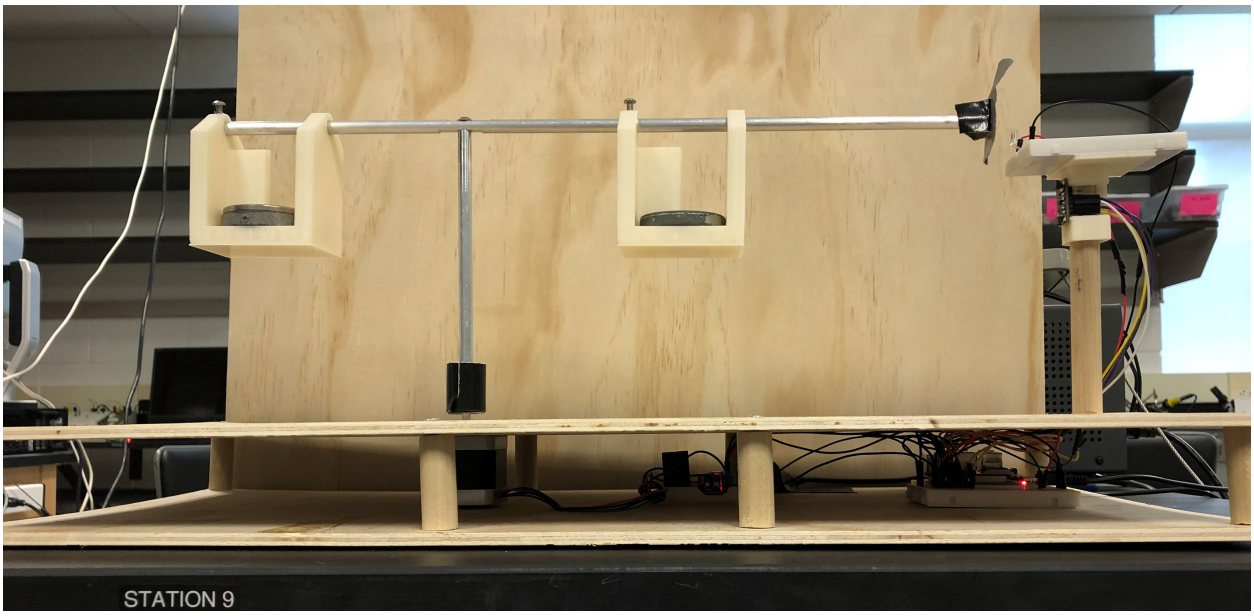

Figure 3: Photograph of the complete apparatus.

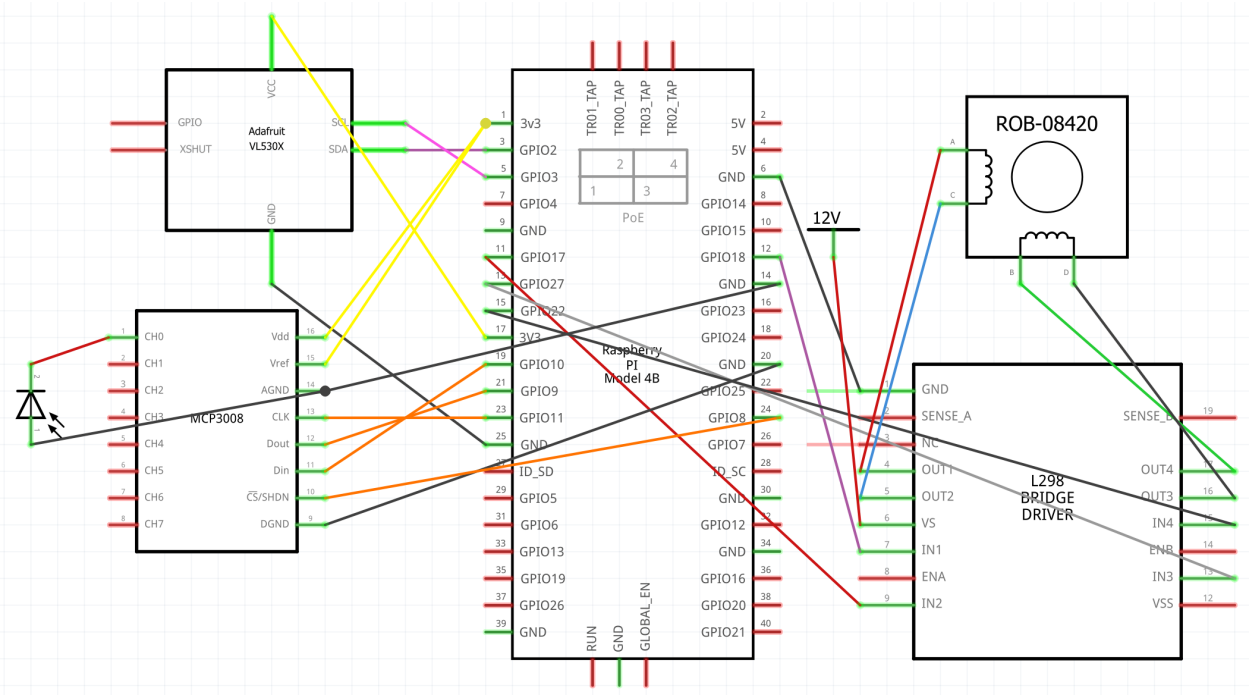

Figure 4: Wiring diagram for the completed electronics.

to result in the final product, pictured in Figure 3.

2 Device Specifications

Mechanical Specifications						
Parameter	**Symbol**	**Unit**	**Minimum**	**Typical**	**Maximum**	**Notes**
Radius	r	cm	5.2	-	25.4	Carriage CM
Mass	m	g	57	-	557	Carriage + Mass
Frequency	f	Hz	1.5	2	2.5	
Period	T	s	0.4	0.5	2	
Speed	ω	rad/s	3.14	12.57	15.71	
Step	θ	°	-	1.8	-	Motor Step
Count	n	-	-	17	-	Rotations
C-Radius	r_c	cm	-	11.64	-	Counter-radius
C-Mass	m_c	g	57	-	557	Counterweight

Electrical Specifications						
Parameter	**Symbol**	**Unit**	**Minimum**	**Typical**	**Maximum**	**Notes**
Pi Voltage In	V_{pi}	V	-	5.1	-	Wall Adapter
Aux Voltage In	V_{in}	V	5	12	18	For Stepper

Relevant Raspberry Pi Characteristics		
Quantity	**Value**	**Notes**
IPv4	10.45.137.29	
VNC Address	10.45.137.29	For VNC Viewer
MAC	dc:a6:32:af:18:c6	
Username	pi	Default
VNC Password	abc123	For VNC Viewer
SSH Password	pi	For SSH Connection

Component Data Sheets		
Name	**Use**	**Link**
Raspberry Pi 4 Model B	CPU	Documentation
Wantai 42BYGHW609 Stepper	Motor	Datasheet
L298N Dual H-Bridge	Motor Driver	Datasheet
PDB-C156 Photodiode	Homing and Sensing	Datasheet
MCP3008 Analog to Digital Converter	Photodiode Reading	Datasheet
VL53L0X Time of Flight Distance Sensor	Radius Measurement	Documentation

3 Physics Principles

The centripetal motion apparatus works by measuring the radius and period of a rotating mass and then calculating out the centripetal acceleration and force. This works because any object moving along a curve in space can have its change in position defined by a moved arc length

$$s = r\theta \tag{1}$$

For an object moving in a uniform circle with a constant radius r, this means that the object has a velocity v and an acceleration a such that

$$v = r\frac{\mathrm{d}\theta}{\mathrm{d}t} = r\omega \tag{2}$$

$$a = r\frac{\mathrm{d}\omega}{\mathrm{d}t} = r\alpha \tag{3}$$

Circular motion, however, is constrained to be periodic so the angular speed ω has relationships to rotation frequency f and period T

$$\omega = \frac{\mathrm{d}\theta}{\mathrm{d}t} = \frac{v}{r} = 2\pi f = \frac{2\pi}{T} \tag{4}$$

By measuring just one of ω, f, and T, the apparatus can return all three of them.

From Newton's second law for uniform circular motion (meaning that the radius does not change and the angular speed does not change), there are the following equivalencies

$$F = ma_c = \frac{mv^2}{r} = mr\omega^2 = mr(4\pi^2 f^2) = mr\frac{4\pi^2}{T^2} \tag{5}$$

This means that, by measuring some triplets of the following variables (F, m, a_c, r, v, T, f, ω), all others can be calculated. Our apparatus measures the radius r and period T or angular speed ω. These two parameters, along with the input mass, are sufficient for calculating all the rest, which the code does after the rotation.

4 User Instructions

4.1 Setting up the Raspberry Pi

There are two possible path for setting up the apparatus which are dependent on the setup of the lab work station. If there is a monitor, keyboard, and mouse available on the workstation, the Raspberry Pi can be connected through its micro-HDMI port to the monitor and its USB ports to the keyboard and mouse. If these accessories are not available, then the Raspberry Pi can be remotely controlled with the following steps:

1. Ensure your computer is connected to the WM-Welcome Wi-Fi network.
2. Download VNC Viewer to the computer which will control the Pi.
3. Run VNC Viewer

4. Make sure the Pi and external power supply are plugged in and powered on.

5. Enter the VNC Address from the **Relevant Raspberry Pi Characteristics** Table in Section 2.

6. When prompted, enter the username and password from **Relevant Raspberry Pi Characteristics** Table in Section 2.

7. The device should now be set up for remote use. See Section 5 for troubleshooting instructions.

Note: For the more technically adept user, the Raspberry Pi can be controlled from a shell on the host computer using the command *ssh Pi@10.45.137.29* and then inputting the password from the **Relevant Raspberry Pi Characteristics** Table in Section 2 when prompted. If doing so, simply go to Step 4.2.1(2) to continue.

4.2 Measurement Taking

Once the Raspberry Pi is set up and controllable either directly or remotely, the apparatus is ready to take measurements with the following process.

4.2.1 Initializing

1. Open the Terminal on the Pi (Black icon in the top left row of the interface).

2. Input the command *cd Desktop/Pumpkin_Centripetal/MainFiles*

3. Loosen the set screw on the carriage on the longer side, move it to the desired radius, then tighten it. Try to keep the carriage as vertical as possible.

4.2.2 Homing and Balancing

1. Ensure the rod is not initially aligned with the homing sensor.

2. Run the open code file by using the command *python3 Centripetal.py*.

3. The rod will rotate slowly as the motor undergoes its homing sequence. It will stop when the rod is aligned with the sensor tower. Sometimes it will rotate around several times before stopping on the sensor.

4. Once the rod is homed, the laser rangefinder will measure and return the radius of the center of mass of the carriage with the equation:

$$r = r_{max} - x + d \tag{6}$$

where $r_{max} =$ cm is the distance from the sensor to the motor shaft, x is the distance from the sensor to the close face of the carriage, and $d = 3.6$ cm is the distance from the carriage face to its center of mass.

5. Once the radius is measured, add mass to the movable carriage and input it in the program.

6. The program will return the required counterweight mass to be put on the other carriage to create a net vertical torque of zero with the equation:

$$m_c = \frac{m * r}{r_c} \tag{7}$$

where m_c is the counterweight mass, r is the radius, m is the input mass, and $r_c = 11.64$ cm is the anti-radius from the **Mechanical Specifications** Table in Section 2. If you cannot his the counterweight exactly, it is better to go over and round up to the nearest 50g.

4.2.3 Measuring

1. Once the counterweight is on, follow the text prompts on the screen to input the required quantities to make the measurement.

2. Step back while the arm articulates so as not to be injured.

3. Once the arm stops swinging, the program will return all the relevant statistics from the rotation, including the rotation speed, period, frequency, centripetal acceleration, and centripetal force.

4. With that, the program will terminate, but the terminal does not close. Simply run the command *python3 Centripetal.py* again to take another measurement after moving the carriage and going back to Section 4.2.2.

A link to a video tutorial of the entire process detailed above is here.

5 Troubleshooting

The following are a series of potential problems, frequently asked questions and possible solutions, as well as points of contact for solving the problems.

What do I do if the Raspberry Pi is not visible/won't show up with VNC Viewer?

- The best thing to do if the Raspberry Pi does not start up is to unplug its power cord and plug it back in. This will hard-restart the board and should resolve the issue.

- If this does not work or the Pi is still not viewable with VNC Viewer, try connecting a monitor to diagnose whether the problem is with VNC viewer or with the Pi itself. It is possible that the Pi is not connected to the internet, in which case contact Dr. Ran Yang for help.

- If the Pi itself is malfunctioning, see Appendix B.1(1) for replacement details.

- Contact Alex Toyryla or Jake Silliman for more details.

What do I do if my Pi will not run the program?

- The first step for fixing the Pi should be to hard-restart is by unplugging and replugging the power cord.
- If this does not work, the Pi might need to be updated with the following terminal commands *sudo apt-get update* and *sudo apt-get upgrade*. The Pi must be connected to the internet for this.
- Contact Alex Toyryla for more details.

What do I do if I lose the program files?

- Should the program be deleted, source code is available in both Appendix C and on Github.
- Contact Alex Toyryla for more help.

What do I do if the motor will not run?

- If the motor will not run, it is possible the battery pack is in the off position so switch it on.
- If the motor will still not run, it is possible the motor driver and/or motor are broken and could need to be replaced.

What do I do if my motor is vibrating loudly?

- If the motor is vibrating loudly, power is still going to it. Turn off the battery pack if you are done with the project.
- If not done, the code has likely hit a snag. Run the cleanup with the command *python3 cleanup.py* and then restart the measurement procedure.

What do I do if the code doesn't work right?

- For an automatic stop to the code, hit CTRL+C on the keyboard.
- If that does not work, unplug the device and hard-restart it.

What do I do if one of the carriages breaks?

- Models and instructions for 3d printing new carriages are included in Appendix B.2(4)
- Contact Seth Gnesin for more details.

What do I do if I need more masses than I have?

- More masses should be available in the prep room for the PHYS 101/102/107/108 Labs.
- See Appendix B.2(9) for more details.

What do I do if I can't fit enough masses on the carriage?

- The carriages each can hold up to the 500g mass or several of the 100g masses.
- If you cannot fit sufficient masses on the carriage, decrease the maximum mass being used in the project.
- If you cannot fit sufficient masses on the counterweight, decrease the mass on the other side and restart the measurement process.

What do I do if I cannot loosen the screw for the carriage?

- If the screw cannot be loosened, it might be at an off angle. Try to screw it in slightly and then unscrew it again.

What do I do if my carriage is sliding around on the shaft assembly?

- If the carriage is sliding, the screw is likely too lose. Tighten it further and try again.

What do I do if the shaft assembly is too pock-marked to be usable?

- If the shaft assembly becomes pock-marked from the screw, sand paper can be used to smooth out any sharp edges.
- If the assembly becomes too pockmarked to be fixed, details for creating a new shaft assembly are available in Appendix B.2(3).

Appendices

A Pumpkin Pie Team Members

Name	Class	Roles	Email
Christopher Egerstrom	Senior	Project Manager Mechanical Engineer	csegerstrom@email.wm.edu
Seth Gnesin	Junior	CAD Engineer Acquisition Manager Writer	sagnesin@email.wm.edu
Ali Hassan	Senior	Coder Electrical Engineer Mechanical Engineer	aahassan01@email.wm.edu
Jake Silliman	Junior	Coder Electrical Engineer	jssilliman@email.wm.edu
Alex Toyryla	Junior	Coder Electrical Engineer Media	amtoyryla@email.wm.edu
Max Weinhold	Senior	CAD Engineer Mechanical Engineer	msweinhold@email.wm.edu

B Components

B.1 Electronics

1. Raspberry Pi 4 Model B
    - The Raspberry Pi 4 Model B, running Raspberry Pi OS, is the CPU for the system.
    - A replacement can be purchased here for $35. Any standard keyboard, mouse, and monitor can be connected to the apparatus through the available ports. The SD card with the source code and operating system can also be transferred to the replacement board.
    - A replacement power supply cord for the Pi can be purchased here for $7.
2. Stepper Motor
    - The Wantai 42BYGHW609 Stepper Motor articulates the arm for the system.
    - If it stops working, a replacement can be found for $9.41.
3. L298N Dual H-Bridge Motor Driver
    - The L298N drives the stepper motor from the external power source.

- Replacement parts can be found here here for $2.30.

4. Photodiode

   - The PDB-C156 Photodiode is attached to the breadboard and is used for measuring the period of the stepper, as well as preforming the homing sequence.
   - A replacement for the photodiode can be bought from Digikey for $1.66.

5. MCP3008 Analog-to-Digital Converter

   - The MCP3008 Analog-to-Digital Converter is attached in the breadboard and is being used in conjunction with the photodiode for reading the relative light level to sense the incoming rod.
   - A replacement chip can be bought from Digikey for $2.19.

6. Adafruit VL53L0X Time of Flight Distance Sensor

   - The VL53L0X laser rangefinder is on the sensor mount on the tower and is used for finding the radius of rotation.
   - A replacement of the full breakout board with the sensor can be bought from Adafruit for $14.95

B.2 Mechanics

1. Wood Base

   - The wood base is made out of 1/4" plywood sheet ($17.99 at ACE Hardware for 2' x 4') and 5/8" dowel rod ($3.49 at ACE Hardware for 48"). It is all connected together with wood glue and screws.
   - The screws to mount the stepper to the base are M3 x 0.5 screws which can be bought from Home Depot or here for $0.62.
   - Contact Christopher Egerstrom for build details.

2. Flex Coupling

   - A 5mm to 8mm flex coupling is used to attach the shaft assembly to the stepper shaft.
   - A replacement part can be bought from Stepper Online for $1.27.

3. Shaft Assembly

   - The shaft assembly is build out of 8mm stock aluminum with the bottom shaft being 6" long and the crossbar being 1.5' long. The crossbar has a flattened edge 6" in from one side for the M5 x 0.8 screw between it and the bottom shaft.
   - Six feet of 8mm Aluminum 6601 stock can be bought from McMaster for $5.47.

- A replacement screw for connecting the bottom shaft to the crossbar can be found at Home Depot or here for$0.50.
- Contact Christopher Egerstrom for build details.

4. Carriages

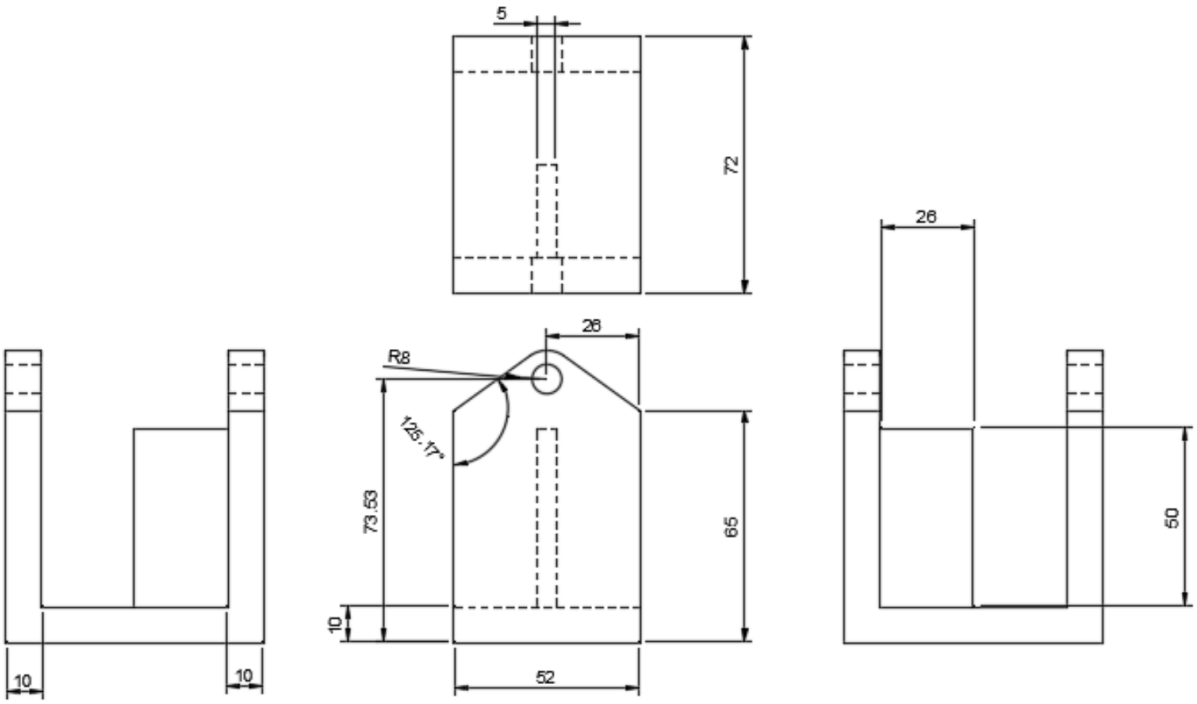

Figure 5: Mechanical Drawing of the Carriage done in Fusion360. All measurements done in milimeters.

- The carriages holding the masses were 3D printed with this STL File on an Airwolf EVO with 0.5mm ABS Plastic Filament. Figure 5 shows a mechanical drawing of the carriage for future modeling.
- The set screws in the carriages are M4 x 0.7 Socket-Cap Screws which can be bought from Home Depot or here for $0.98.
- Contact Seth Gnesin or Christopher Egerstrom for design and build details.

5. Sensor Housing

- The housing for the Laser Rangefinder, the Photodiode, and the ADC chip was 3D printed with this STL File on an Airwolf EVO with 0.5mm ABS Plastic Filament. Figure 6 shows a mechanical drawing of the sensor for future modeling.
- Contact Max Weinhold for design details.

6. Board Holders

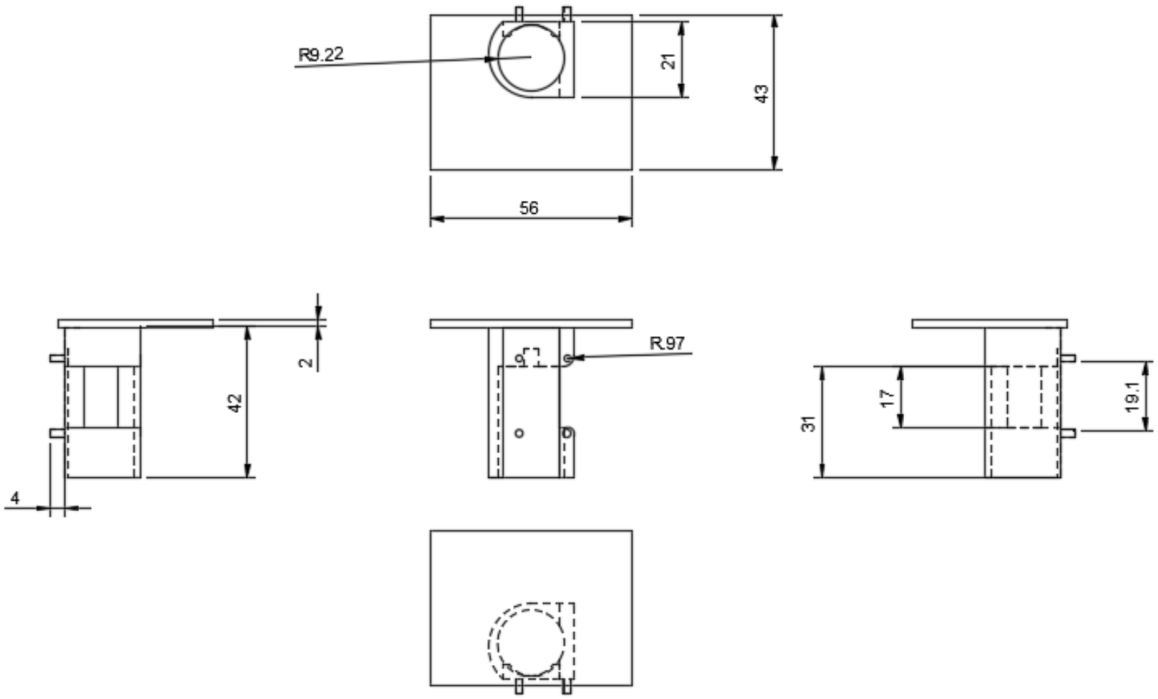

Figure 6: Mechanical Drawing of the Sensor Housing done in Fusion360. All measurements done in milimeters.

- The holder for the Raspberry Pi was 3D printed with this STL File on an Airwolf EVO with 0.5mm ABS Plastic Filament.
- The holder of the L298N H-Bridge was 3D printed with this STL File courtesy of PrintedHuman on Thingiverse on an Airwolf EVO with 0.5mm ABS Plastic Filament.
- Contact Seth Gnesin for more details.

7. Breadboard

   - The photodiode and ADC are held by two miniature breadboards, one inside and one outside.
   - Any mini breadboard could work but one easy replacement is from Sparkfun for $3.95.

8. Power Cord

   - The 12V power for the Stepper motor is taken from the wall and uses a terminal converter to connect to the L298N.
   - A replacement cord can be bought here for $13.95.
   - A replacement adapter can be bought from Sparkfun for $2.95.

9. Mass
    - This device uses the standard set of masses (20g to 500g) from the Physics Department as used for hanging mass applications.
    - For replacement, contact Dr. Ran Yang at rxyan2@wm.edu
10. Hex Wrench
    - To loosen the screws for the carriages, a 1.5 Hex Key is used.
    - A replacement can be found in the Makerspace or here for $..

C Source Code

All of the code below is also available in this Github Repository.

C.1 Primary Code

```
#!/usr/bin/env python
import threading, time, spidev, math
import RPi.GPIO as GPIO
import Functions2 as Functions
import VL53L0X

print("\n ~~~  Welcome to the Pumpkin Pals' Birdemic Centripetal Force Sensor
    ~~~\n\n")

out1 = 13
out2 = 11
out3 = 15
out4 = 12

GPIO.setmode(GPIO.BOARD)
GPIO.setup(out1,GPIO.OUT)
GPIO.setup(out2,GPIO.OUT)
GPIO.setup(out3,GPIO.OUT)
GPIO.setup(out4,GPIO.OUT)

spi = spidev.SpiDev() #open spi bus
spi.open(0,0) #open(bus, device)
spi.max_speed_hz=1000000
spi.mode = 0b00 #spi modes; 00,01,10,11

dummy = input("The arm will rotate automatically to the correct position for
    radial measurement. Make sure the two rod ends are approximately balanced
    , though exact counterweights will be calculated later. Press ENTER to
    continue.")
try:
    Functions.homing()
    print("\nCarriage lined up with distance sensor! Disregard the device
        info directly below.")
```

```
    tof = VL53L0X.VL53L0X()
    tof.start_ranging(VL53L0X.VL53L0X_GOOD_ACCURACY_MODE)
    distance = tof.get_distance()
    print("\nRadius of Center of Mass from Center Rod measured to be ",
        distance, " mm.")

    print("\nNow measure your desired mass, and after entering it the
        approximate counterweight mass will be given. Make sure the
        counterweight is close to the value given, though perfect accuracy isn
        't necessary.")
    mass = float(input("\nEnter your Mass in Grams: "))
    counterweight = ((mass+57)*distance)/120 - 57
    print("\nThe recommended counterweight is ", round(counterweight, 0), "
        grams. Add the counterweight now, rounding up to the nearest 50g.")

    print("Then, decide how many rotations per second to specify. The actual
        speed will be slower, so the device will measure it for you.")
    reps = 20
    speed = float(input("\nEnter Rotations per Second: "))
    timestep = 1/(speed*205)

    stepperthread = threading.Thread(target=Functions.stepper, args=(reps,
        timestep))
    photothread = threading.Thread(target=Functions.photodiode, args=(speed,
        mass,distance))

    stepperthread.start()
    photothread.start()

    stepperthread.join()
    photothread.join()

except KeyboardInterrupt:
    GPIO.cleanup()
except Exception as e:
    print('Unexpected Error: ', e)
    GPIO.cleanup()
```

C.2 Imported Functions

C.2.1 Photodiode and Stepper Motor

```
import RPi.GPIO as GPIO
import time, spidev, math

out1 = 13
out2 = 11
out3 = 15
out4 = 12

GPIO.setmode(GPIO.BOARD)
GPIO.setup(out1,GPIO.OUT)
```

```
GPIO.setup(out2,GPIO.OUT)
GPIO.setup(out3,GPIO.OUT)
GPIO.setup(out4,GPIO.OUT)

spi = spidev.SpiDev() #open spi bus
spi.open(0,0) #open(bus, device)
spi.max_speed_hz=1000000
spi.mode = 0b00 #spi modes; 00,01,10,11

def read_adc(channel):
    if not 0 <= channel <= 7:
        raise IndexError('Invalid. enter 0, 1, ..., 7')
    """datasheet page 19 about setting sgl/diff bit to high, hence we add 8 =
        0b1000
    left shift 4 bits to make space for the second byte of data[1]"""
    request = [0x1, (8+channel) << 4, 0x0] # [start bit, configuration, listen
        space]
    data = spi.xfer2(request) #data is recorded 3 bytes: data[0] - throw away,
        data[1] - keep last 2 bits, data[2] - keep all
    data10bit = ((data[1] & 3) << 8) + data[2] #shfit bits to get the 10 bit
        data
    return data10bit

def photodiode(speed, mass, distance):
    # Setup.  time_step is time between measurements, dark_volt is the
        fraction
    # of ambient light reached for the program to recognize the carriage has
    # passed over it, usually not much smaller than ambient voltage with our
        setup.
    time_step = 0.001
    volts = []
    times = []
    rotations = []
    print("\nReading AMBIENT LIGHT in 0.5 seconds")
    time.sleep(0.5)
    ambient = read_adc(0) * (3.3/1024)
    dark_volt = 0.93 * ambient
    nummeasure = int(round((20/speed)*1000*0.75, 0))

    # The main loop that runs while taking data on timestamps and voltage put
        out
    # by the photodiode.  Un-comment the print line below if a stream of real-
        time
    # measurements is desired.
    darkflag = False
    t = -time_step
    counts = 0
    print("\nTaking measurements!  For accurate measurements, do not hinder
        the device!")
    for i in range(nummeasure):
        t = t + time_step
        v_volt = read_adc(0) * (3.3/1024)
        #print("The input voltage is ", v_volt)
        if darkflag == False and v_volt < dark_volt:
```

```
            darkflag = True
            counts += 1
            print("Count: ", counts)
            rotations.append(time.time())
        if darkflag == True and v_volt > dark_volt:
            darkflag = False
        volts.append(v_volt)
        times.append(t)
        time.sleep(time_step)

    # The loop which creates a list of periods calculated with timestamps.
       Before
    # appending to the periods list, it removes timestamps unrealistically
       close to
    # the previous due to any error or "noise" in the signal.
    periods = []
    try:
        for i in range(0, len(rotations)-1):
            while rotations[i+1] - rotations[i] < 0.1:
                rotations.pop(i+1)
            periods.append(rotations[i+1] - rotations[i])
    except IndexError:
        pass
    try:
        finalperiod = sum(periods) / len(periods)
        centripetal = (4*(math.pi)**2 * (mass/1000) * (distance/1000)) / ((
           finalperiod)**2)
        print("\nUsing the formula Fc = (4pi^2 mr) / T^2, the centripetal
           force can be calculated:\n ")
        print("Centripetal Force: ", centripetal, " N")
        print("Mass: ", mass, " g")
        print("Radius: ", distance, " mm")
        print("Acceleration: ", centripetal/mass, " ms^-2")
        print("Period: ", finalperiod, " s")
        print("Frequency: ", 1/finalperiod, " Hz")
        print("Angular Frequency: ", 2*math.pi*(1/finalperiod), " rad/s")
        print("Counts while Measuring: ", counts)

        print("\n\nPowering Down in a few seconds... \nThank you for using the
            Birdemic Centripetal Force Sensor!\n")
    except:
        print("\nNo rotations were sensed!  Run the program again.\n")


def stepper(reps, timestep):
    out1 = 13
    out2 = 11
    out3 = 15
    out4 = 12

    GPIO.setmode(GPIO.BOARD)
    outs = [out1, out2, out3, out4]
    for out in outs:
        GPIO.setup(out, GPIO.OUT)
```

```
    #pattern = [[1,0,0,0], [1,0,1,0], [0,0,1,0], [0,1,1,0],
    #           [0,1,0,0], [0,1,0,1], [0,0,0,1], [1,0,0,1]]
    pattern = [[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]]

    x = int(reps*200)
    if x>0 and x<=100000:
        for i in range(x):
            throws = pattern[i%len(pattern)]
            for j in range(len(outs)):
                GPIO.output(outs[j], throws[j])
            time.sleep(timestep)
            #print(i)

    GPIO.cleanup()


def homing():
    i = 0
    print("Reading ambient light and homing in 3 seconds...")
    time.sleep(3)
    ambient = read_adc(0) * (3.3/1024)
    dark_volt = 0.94 * ambient
    v_volt = ambient
    timestep = 0.03 #Change this for speed!!! 0.003 is fast, 0.01 is slow
    out1 = 13
    out2 = 11
    out3 = 15
    out4 = 12

    GPIO.setmode(GPIO.BOARD)
    outs = [out1, out2, out3, out4]
    for out in outs:
        GPIO.setup(out, GPIO.OUT)

    #pattern = [[1,0,0,0], [1,0,1,0], [0,0,1,0], [0,1,1,0],
    #           [0,1,0,0], [0,1,0,1], [0,0,0,1], [1,0,0,1]]
    pattern = [[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]]

    i=0
    while v_volt > dark_volt:
        #print(v_volt)
        throws = pattern[i%len(pattern)]
        for j in range(len(outs)):
            GPIO.output(outs[j], throws[j])
        time.sleep(timestep)
        #print(i)
        i+=1
        v_volt = read_adc(0) * (3.3/1024)
    print("\nWaiting for vibrations to cease...")
    time.sleep(3) #freeze the stepper in place for 1s
    GPIO.cleanup()
```

C.2.2 Laser Rangefinder

```
import numpy as np
import time
from ctypes import *
import smbus
import statistics

VL53L0X_GOOD_ACCURACY_MODE      = 0   # Good Accuracy mode
VL53L0X_BETTER_ACCURACY_MODE    = 1   # Better Accuracy mode
VL53L0X_BEST_ACCURACY_MODE      = 2   # Best Accuracy mode
VL53L0X_LONG_RANGE_MODE         = 3   # Longe Range mode
VL53L0X_HIGH_SPEED_MODE         = 4   # High Speed mode

i2cbus = smbus.SMBus(1)

# i2c bus read callback
def i2c_read(address, reg, data_p, length):
    ret_val = 0;
    result = []

    try:
        result = i2cbus.read_i2c_block_data(address, reg, length)
    except IOError:
        ret_val = -1;

    if (ret_val == 0):
        for index in range(length):
            data_p[index] = result[index]

    return ret_val

# i2c bus write callback
def i2c_write(address, reg, data_p, length):
    ret_val = 0;
    data = []

    for index in range(length):
        data.append(data_p[index])
    try:
        i2cbus.write_i2c_block_data(address, reg, data)
    except IOError:
        ret_val = -1;

    return ret_val

# Load VL53L0X shared lib
tof_lib = CDLL("../bin/vl53l0x_python.so")

# Create read function pointer
READFUNC = CFUNCTYPE(c_int, c_ubyte, c_ubyte, POINTER(c_ubyte), c_ubyte)
read_func = READFUNC(i2c_read)

# Create write function pointer
```

```
WRITEFUNC = CFUNCTYPE(c_int, c_ubyte, c_ubyte, POINTER(c_ubyte), c_ubyte)
write_func = WRITEFUNC(i2c_write)

# pass i2c read and write function pointers to VL53L0X library
tof_lib.VL53L0X_set_i2c(read_func, write_func)

class VL53L0X(object):
    """VL53L0X ToF."""

    object_number = 0

    def __init__(self, address=0x29, TCA9548A_Num=255, TCA9548A_Addr=0, **
        kwargs):
        """Initialize the VL53L0X ToF Sensor from ST"""
        self.device_address = address
        self.TCA9548A_Device = TCA9548A_Num
        self.TCA9548A_Address = TCA9548A_Addr
        self.my_object_number = VL53L0X.object_number
        VL53L0X.object_number += 1
        self.ADDR_UNIT_ID_HIGH = 0x16
        self.ADDR_UNIT_ID_LOW = 0X17
        self.ADDR_I2C_ID_HIGH = 0X18
        self.ADDR_I2C_ID_LOW = 0X19
        self.ADDR_I2C_SEC_ADDR = 0X8a
    def start_ranging(self, mode = VL53L0X_GOOD_ACCURACY_MODE):
        """Start VL53L0X ToF Sensor Ranging"""
        tof_lib.startRanging(self.my_object_number, mode, self.device_address,
            self.TCA9548A_Device, self.TCA9548A_Address)

    def stop_ranging(self):
        """Stop VL53L0X ToF Sensor Ranging"""
        tof_lib.stopRanging(self.my_object_number)

    def get_distance(self):
        """Get distance from VL53L0X ToF Sensor"""
        max_d = 330
        run_time = time.time()
        elapsed = 0
        distances = []

        print ('\nFinding distance...')

        for i in range (0,4):
            while elapsed < 3:
                distance = tof_lib.getDistance(self.my_object_number)
                distances.append(distance)
                elapsed = time.time() - run_time
        result = round(max_d - statistics.mean(distances),0)

        print ('The distance at this point is', result, 'mm')
        pass
        return result

    # This function included to show how to access the ST library directly
```

```
    # from python instead of through the simplified interface
    def get_timing(self):
        Dev = POINTER(c_void_p)
        Dev = tof_lib.getDev(self.my_object_number)
        budget = c_uint(0)
        budget_p = pointer(budget)
        Status =  tof_lib.VL53L0X_GetMeasurementTimingBudgetMicroSeconds(Dev,
            budget_p)
        if (Status == 0):
            return (budget.value + 1000)
        else:
            return 0
```

C.3 Cleanup Code

```
import RPi.GPIO as GPIO
out1 = 13
out2 = 11
out3 = 15
out4 = 12

GPIO.setmode(GPIO.BOARD)
GPIO.setup(out1,GPIO.OUT)
GPIO.setup(out2,GPIO.OUT)
GPIO.setup(out3,GPIO.OUT)
GPIO.setup(out4,GPIO.OUT)

GPIO.cleanup()
```