Lab3 Report

Lab section 402 Wednesday

Xinyun Guo (xg284) Zhuoheng Li (zl764)

2019-10-23

Table of Content

Introduction

In lab 3, we completed an autonomous robot that can be controlled by piTFT screen and display real-time status and control information. The servos are powered by battery packs, and the robot is self-contained and not connect to any external devices.

Design and Testing

1. Part 1

We first added output capability to the R-Pi through the exploration of a number of attached components.

1.1 Using PWM in RPi.GPIO

First, we learned about how to use rpi.GPIO library to control the servo with GPIO output. The function we need to use are:

Functionality	rpi.GPIO library
create a PWM instance	*p = GPIO.PWM(GPIO_pin, frequency)*
start PWM	*p.start(dc) # where dc is the duty cycle (0.0 <= dc <= 100.0)*
change the frequency	*p.ChangeFrequency(freq)*
change the duty cycle	*p.ChangeDutyCycle(dc) # where 0.0 <= dc <= 100.0*
stop PWM	*p.stop()*

Table 1 the rpi.GPIO library implemented

Using these rpi.GPIO library functions, we designed a python code, blink.py (Appendix Figure 1) to implement an LED circuit on the GPIO pin 5 and experiment with blink rates 1 per second (blink the LED on and off over a second) using PWM settings. The circuit we use shown in the figure below.

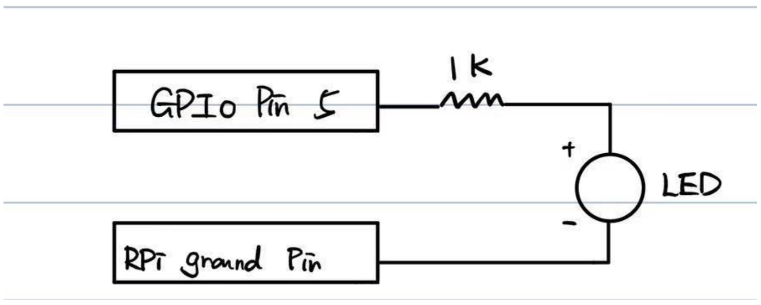
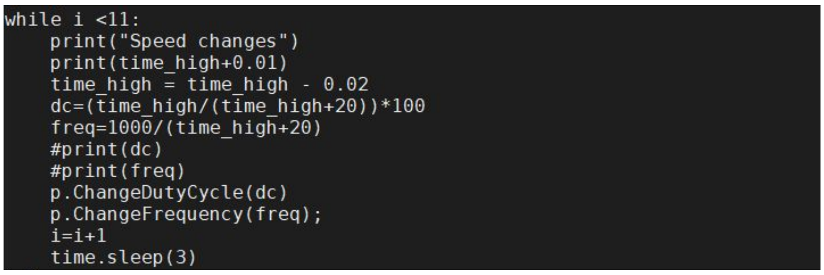


Figure 1 circuit for LED connection

Next, we developed a python application 'pwm_calibrate.py' 'servo_control.py' (Appendix Figure 2) to calibrate the servo. When the pulse is 1.5ms, the servo would hold still. We calculated the duty cycle and frequency of this case, the duty cycle is approximately 6.98%, the frequency is 46.51HZ. We wrote the python code using this data and calibrated the servo to still state with a screw-driver while running the code.

After calibration, we further developed a python application ‘servo_control.py’ (Appendix Figure 3) to start the servo from stop state, increase the speed of the servo through ten-speed steps in the clockwise, decrease the speed of the servo through ten-speed steps in the counter-clockwise direction, and stop at last. We used the following formula to calculate each increment and use a *while* loop to implement ten-speed steps increments.
t*dc=(time_high/(time_high+20))*100*
freq=1000/(time_high+20)
Also, we use *sleep(3)* to control servo running at each speed for 3 seconds.



```
while i <11:
    print("Speed changes")
    print(time_high+0.01)
    time_high = time_high - 0.02
    dc=(time_high/(time_high+20))*100
    freq=1000/(time_high+20)
    #print(dc)
    #print(freq)
    p.ChangeDutyCycle(dc)
    p.ChangeFrequency(freq);
    i=i+1
    time.sleep(3)
```

Figure 2 a snapshot of python code servo_control.py

1.2 Left and Right Servos

In this part, we used PWM to control two servos. In specific, we followed the lab instructions to generate the PWM signal to drive both the servo to rotate clockwise and counter-clockwise at full speed. We also set up six buttons to control how the servos rotate. Each servo has three buttons that control it to rotate clockwise, counter-clockwise and stop.

In the python code, we wrote, we use changeFrequency(0) to control let the servo stop. In addition, to drive the servo rotate clockwise at full speed, we set the changeFrequency() to implement two_wheel.py. The circuit we use shown in the figure below.

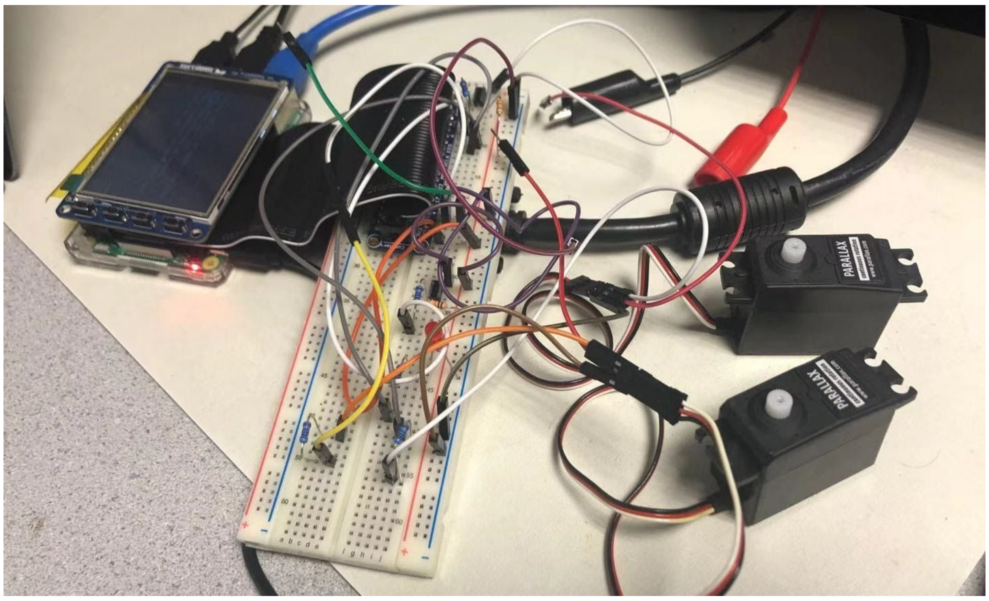


Figure 2 the circuit for python code two_wheel.py

Discussion: consider the difference between a calibration signal and a stopped state for the servo. What input signal will cause the servo to completely stop?

The calibration signal is a 1.5ms pulse width signal which should let the servo hold still. And based on this knowledge, we calibrate the servo to let it hold still. However, to let the servo completely stop, we need to send a constant signal. Thus, we use changeFrequency(0) to control let the servo stop.

1.3 Rolling Control

In this step, we wrote the python code 'rolling_control.py' to implement functions as required. As we did in Lab2, we created *rect* for three touch buttons: a red 'panic stop' button, a green 'resume' button, and a 'quit' button. When the quit button is pressed, the system will exit (same as what we did in Lab2). For the 'panic stop' button, we used a flag (*isPause* in Appendix Figure 3) to record whether the 'panic stop' button has been pressed. When the 'panic stop' button is pressed, the flag is *True*, and PiTFT will display the green 'resume' button. Similarly, when the 'resume' button is pressed, the flag is *False*, PiTFT will display the red 'panic stop' button. Further, we need to create the *rect* to display the three history states for the left and right servo. After setting this, the piTFT will initially display as shown in the figure below.

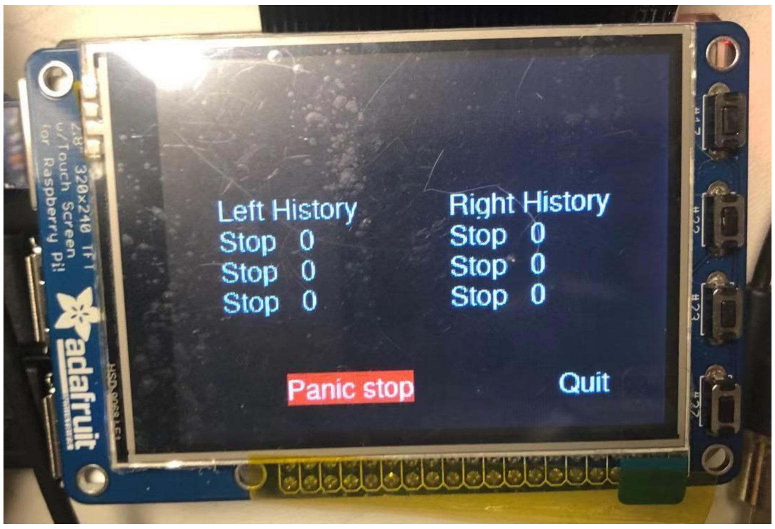
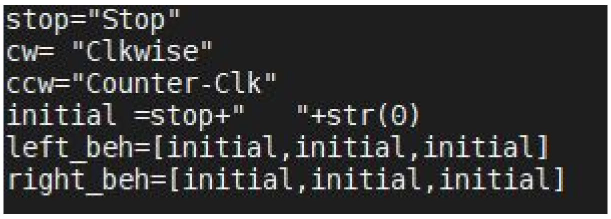
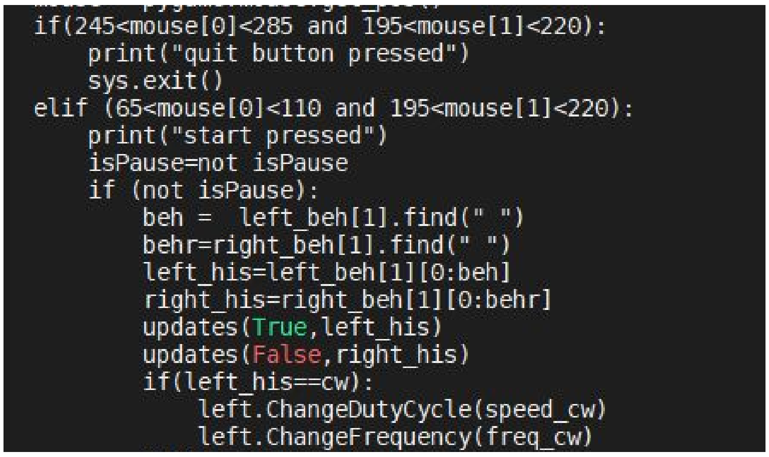


Figure 3 the initial display of piTFT screen

The challenge is to record start-time/direction pairs for each motor and display a scrolling history of the most recent motion. To achieve this, we first created some parameters and two arrays to store the 3 past entries for each motor separately.


```
stop="Stop"
cw= "Clkwise"
ccw="Counter-Clk"
initial =stop+"   "+str(0)
left_beh=[initial,initial,initial]
right_beh=[initial,initial,initial]
```
Figure 4 the initial display of piTFT screen

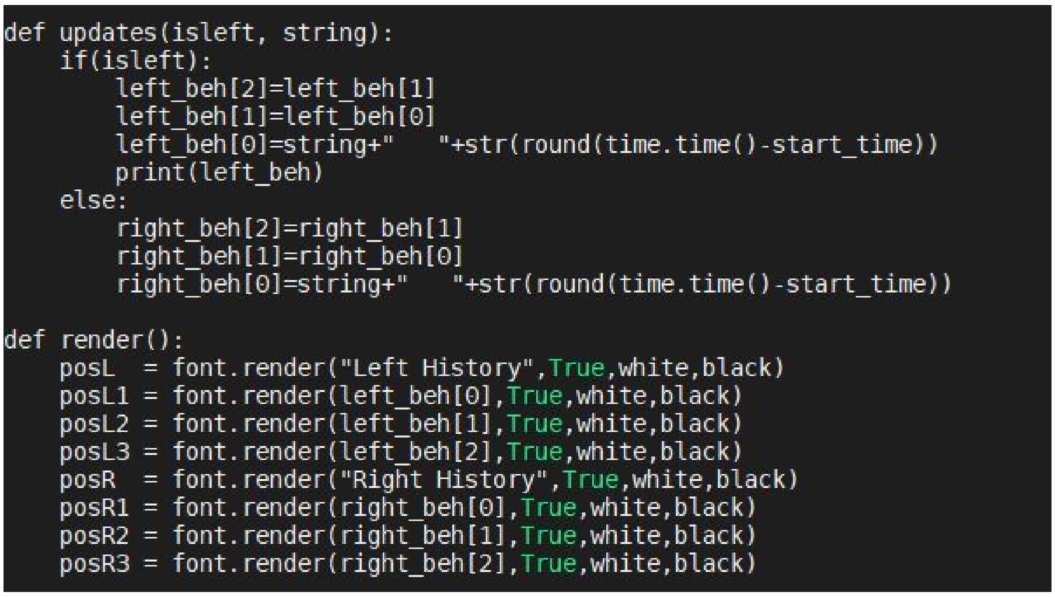
First, we need to implement the functionality of ‘panic stop’ button. To do this, we set two servos ‘*stop’* when the ‘panic stop’ is pressed; when the ‘resume’ button is pressed, we set two servos with their previous status, which record in the index 1 of *left_beh* and *right_beh*. The figure below shows one scenario, the full code can be found in the appendix.



```
if(245<mouse[0]<285 and 195<mouse[1]<220):
    print("quit button pressed")
    sys.exit()
elif (65<mouse[0]<110 and 195<mouse[1]<220):
    print("start pressed")
    isPause=not isPause
    if (not isPause):
        beh =  left_beh[1].find(" ")
        behr=right_beh[1].find(" ")
        left_his=left_beh[1][0:beh]
        right_his=right_beh[1][0:behr]
        updates(True,left_his)
        updates(False,right_his)
        if(left_his==cw):
            left.ChangeDutyCycle(speed_cw)
            left.ChangeFrequency(freq_cw)
```

Figure 5 when the ‘resume’ button is pressed, two servos back to its previous state

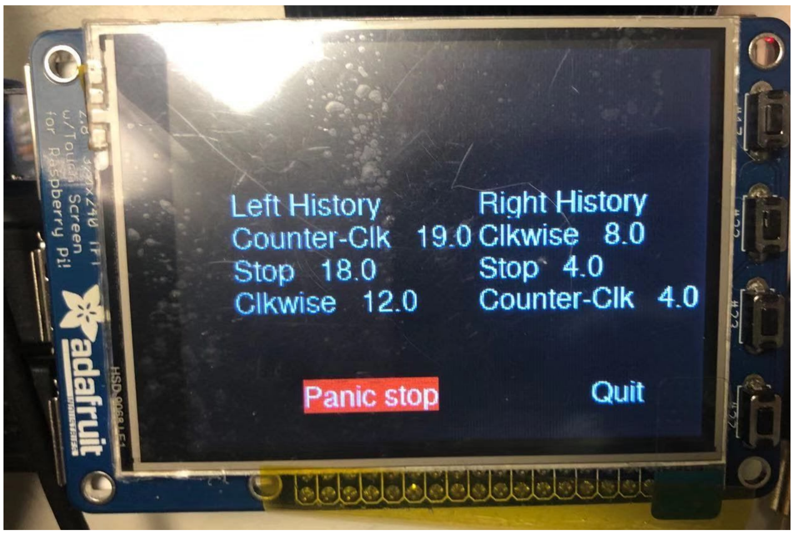
Then, we wrote two functions: ‘updates’ to updates any changes of either servo; ‘render’ to render the new history entries of left and right servos. The function ‘updates’ read two parameters *‘isleft’*, which indicates which servo updated and a *‘string’* indicated the change of the update. We use *time.time()* to calculate each update time.



```
def updates(isleft, string):
    if(isleft):
        left_beh[2]=left_beh[1]
        left_beh[1]=left_beh[0]
        left_beh[0]=string+"   "+str(round(time.time()-start_time))
        print(left_beh)
    else:
        right_beh[2]=right_beh[1]
        right_beh[1]=right_beh[0]
        right_beh[0]=string+"   "+str(round(time.time()-start_time))

def render():
    posL  = font.render("Left History",True,white,black)
    posL1 = font.render(left_beh[0],True,white,black)
    posL2 = font.render(left_beh[1],True,white,black)
    posL3 = font.render(left_beh[2],True,white,black)
    posR  = font.render("Right History",True,white,black)
    posR1 = font.render(right_beh[0],True,white,black)
    posR2 = font.render(right_beh[1],True,white,black)
    posR3 = font.render(right_beh[2],True,white,black)
```

Figure 6 two functions we wrote in rolling_control.py

Then, we integrated this with the servo_control.py. When a physical button is pressed, the corresponding servo will change its duty cycle and frequency to the corresponding status (clockwise, counter-clockwise or stop), meanwhile it will call functions ‘updates’ and ‘render’ to updates the display of the piTFT screen. One of the scenarios is shown in the figure below.



```
Left History          Right History
Counter-Clk   19.0    Clkwise   8.0
Stop   18.0           Stop   4.0
Clkwise   12.0        Counter-Clk   4.0
```

Figure 7 a scenario to show the update of piTFT display

2. Part 2

2.1 Robot assembly

In this step, we built a robot platform for the RPi. We first attached servo brackets, the upper frame piece, and the second level to the base plate. Then, we connected two motors to the bracket. After this, we installed bearing to the rear dowel and attached the dowel to the center of the frame using the middle hole along the outer rim. This is necessary because the bearing helps the whole robot car frame to keep balance.

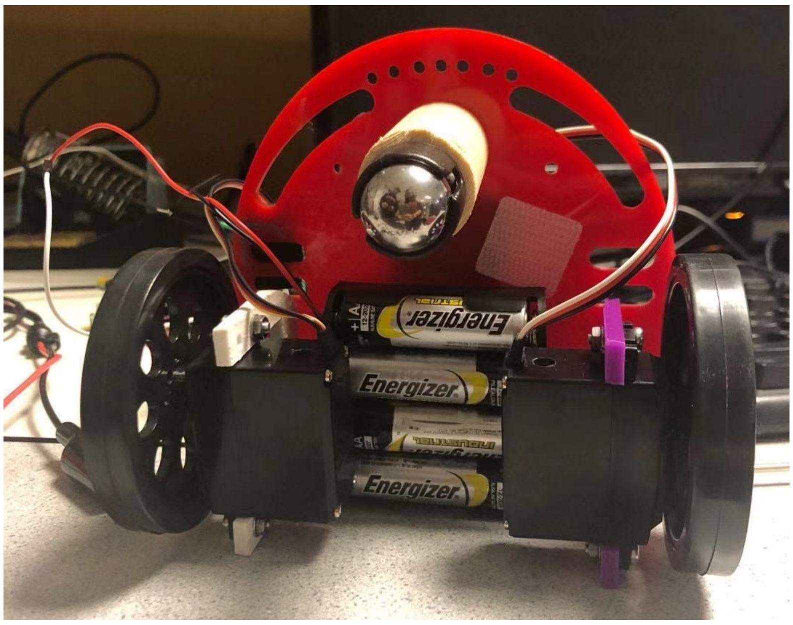


Figure 8 the bottom view of the platform, with bearing and battery holders attached

Next, we attached the battery holders to the underside of the main body using Velcro. Then, we installed two wheels to two motors.

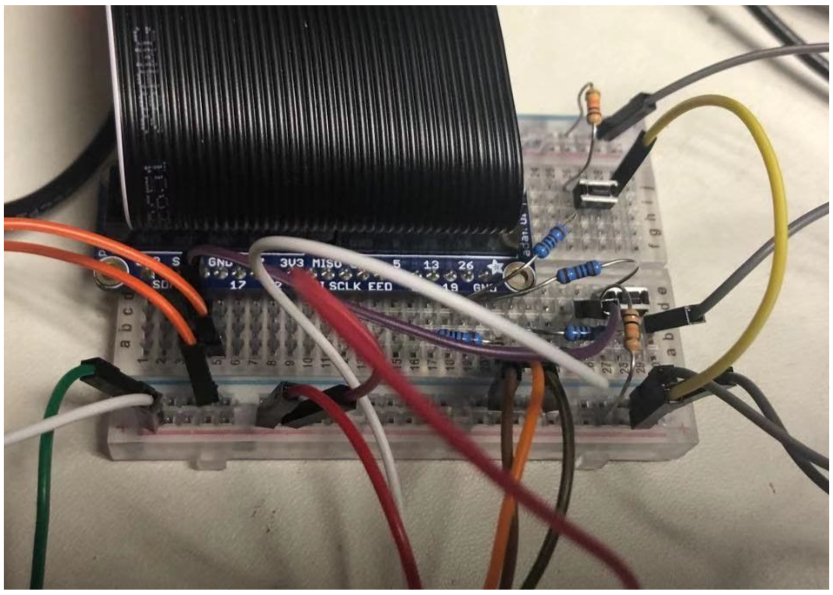


Figure 9 Wire RPi into motor controls using the small proto-board

At last, we attached the Rpi and wired proto-board to the top-level of the platform. The final robot looks like what is shown in figure 10 below.

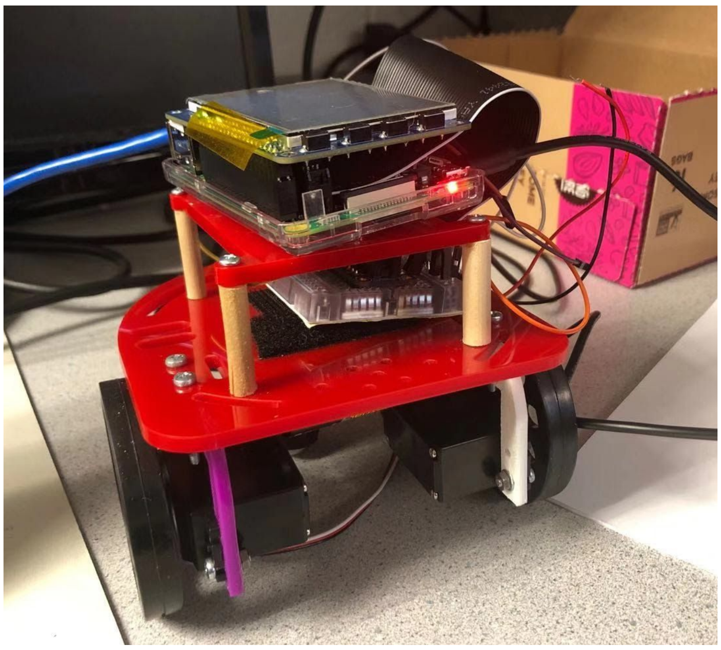


Figure 10 the final platform

2.2 Test

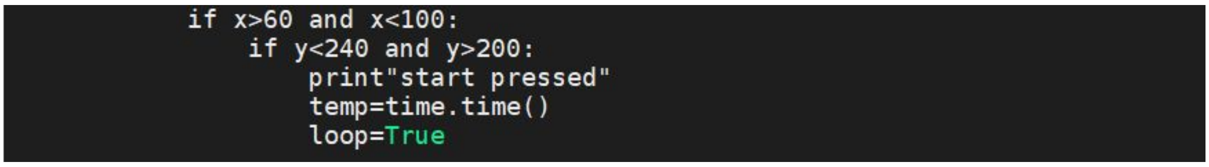
After assembling the robot frame, we connected two motors with the external 5V power supply and kept using the 5V RPi power supply for physical buttons. Then we tested the functionality of rolling_control.py and two_wheel.py. Both python codes worked exactly the same as it performs in part 1.

2.3 Building Battery Into the Circuit

In this step, we connected a switch to the 4-AA battery pack to allow the motors to be disconnected from the battery pack. This is important because if we turn off the switch, both motors will not be powered up. This will protect the circuit and motors from any unexpected movement which may break the motors. Then, we wired the battery power supply into the + and – rails of the small protoboard and rewired motor to use the 6V battery power.

2.4 Initial Battery Test

In this step, we design a python code ‘run_test.py’ to achieve the requirements on the lab instruction. First, we need to create a ‘start’ button, when we pressed the button, the control of the servo will start. We use a flag *‘loop’* to indicate whether the button is pressed or not. If the ‘start’ button hasn’t been pressed, the flag ‘*loop’* is false, and the robot will not move; If *‘loop’* is *True*, the servo controlling program will be started.



```
        if x>60 and x<100:
            if y<240 and y>200:
                print"start pressed"
                temp=time.time()
                loop=True
```

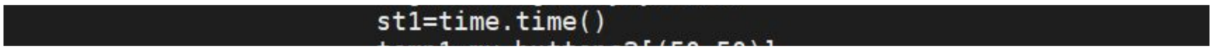
Figure 11 the code for ‘start’ button

Then, we need to build a state machine with seven states for servo control. After we test the time that the robot takes at half-speed of the motors, we know it will take around 3 seconds to move the robot forward or backward about 1 foot, and 11 seconds to go over one round, from state 1 to state 7. Therefore, the full state machine loop will be indicated as the table below.

State	Movement	Time range
State 1	Move the robot forward about 1 foot	From time 0 to time 3
State 2	Stop	From time 3 to time 4
State 3	Move the robot backward about 1 foot	From time 4 to time 7
State 4	Pivot left	From time 7 to time 8
State 5	Stop	From time 8 to time 9
State 6	Pivot right	From time 9 to time 10
State 7	Stop	From time 10 to time 11

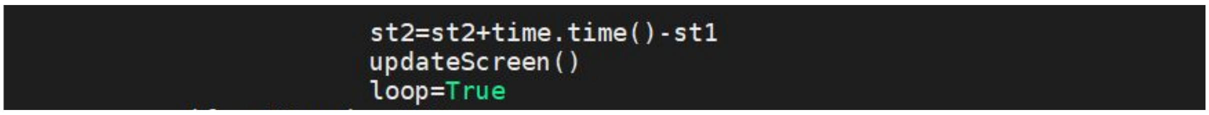
Table 2 seven states of performance

Since the robot may be paused during this loop, we need to know the real running time of this robot. To do that, we use a variable *temp* to store the start time of this loop. Also, we need to know much time the robot remains in the stopped state. In order to keep track of that, we use ***st1*** to record the time when we press the “panic stop” button and the robot stops. Then, when we press the “resume” button, we calculate the time the robot spends in the stop state which is ***time.time() - st1.*** After calculating the duration of the current stopped state, we add this time to variable ***st2***, we are able to get the total time after the robot starts the state machine without any pause.



```
st1=time.time()
```

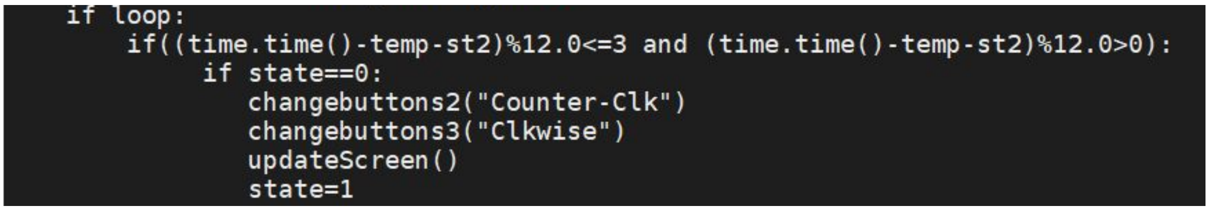
Figure 12 record the stop time



```
st2=st2+time.time()-st1
updateScreen()
loop=True
```

Figure 13 recalculates the time after the state machine starts

After getting these data, we are able to calculate which state the robot should be in. Because one round of the movement loop takes 11 seconds, and we also know the time the robot in this loop is ***time.time() - temp - st2***. And we let ***time.time() - temp - st2*** modulo ***12***, we can get the corresponding time of the state. For instance, If the ***(time.time()-temp-st2) %12 <=3 and (time.time()-temp-st2) %12 >0***, which means the time right now is within 3 seconds of the total 11 seconds loop of the state machine, so the robot should keep moving forward until reach 3 seconds. The full code can be found in the appendix.



```
    if loop:
        if((time.time()-temp-st2)%12.0<=3 and (time.time()-temp-st2)%12.0>0):
            if state==0:
               changebuttons2("Counter-Clk")
               changebuttons3("Clkwise")
               updateScreen()
               state=1
```

Figure 14 an example of the implement of the state machine of servo control

With the help of the state machine, we are able to achieve the functionality of ‘panic stop’ button. When the ‘panic stop’ button is pressed, both servos will stop; when the ‘resume’ button’ is pressed, the servo will go back to its previous state, similar to how we implement in part 1 1.3 rolling_control.py.

2.5 Enable WIFI

We did not use the WIFI for RPi because TAs told us WIFI in the lab is not very strong and stable.

2.6 Configure the RPi to launch your application at boot

To make the RPi run the python code when it reboots, we used method 1 shown on the instruction. We added a call to run_test.py in /home/pi/.bashrc. Then, we used *sudo raspi-config*, configure the RPi to start without login.

>Select ‘Boot Options’

> Select ‘Desktop/CLI’

> Select ‘Console Autologin’

After configuration, we reboot the RPi, and it will automatically run the python code run_test.py when it powers up.

Result：

Following the lab procedure, we implemented and tested using PWM to control the servo. At first, we learned the importance of calibration before using servos. Then, we learned how to change the duty cycle and the frequency to modulate the pulse width to drive the servo to rotate in different directions and different speeds. In addition, we set up six GPIO buttons to control the running direction and running speed of the two servos. We also implemented the log display function to display the events of two servos . The test showed that all the functions worked well.

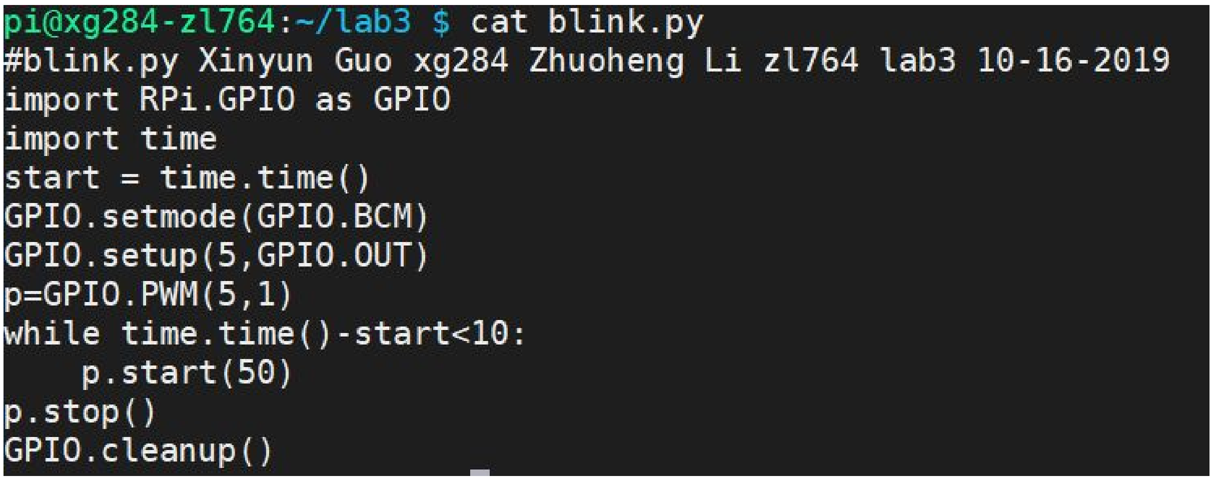
In week 2, we assembled the robot and tested the run_test.py we wrote. In this python program, we are supposed to pause and resume the robot. To make this, we used a state machine to record the real-time state of the robot. Thus, when we press the resume button, the robot can continue its previous movement. In the end, we followed the instruction and add the ***python run_test.py*** command to *.bashrc* so that the program will be launched automatically when the Raspberry pi boot.

Conclusion

In lab 3, we completed most steps smoothly, and most tasks performed well as we expected. During the lab, we encountered a problem that the run_test.py we wrote at first was wrong that we couldn't let the robot stop in the middle of a series of movements. Then, we turned to seek help from TA and we learned that we were supposed to use a state machine to keep track of the state of the robot. Then, we modified our codes and used the method mentioned above to achieve the goal, and it worked as expected. In addition, we also learned that a good coding style can help us a lot when writing a long program. In this lab, we separated several functions into individual methods that keep the code cleaner and easier to read. An organized code also helped us with debugging.

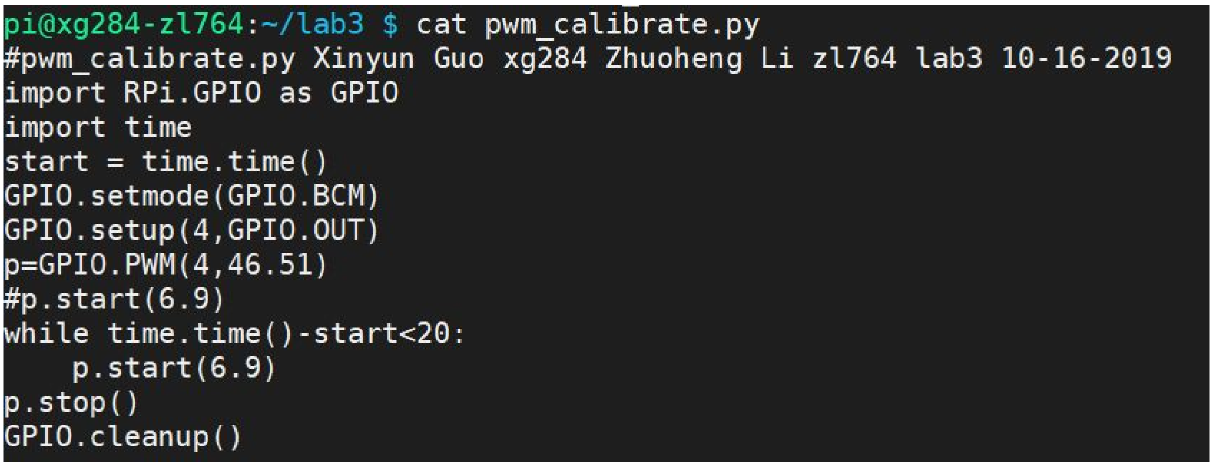
We think the lab description is clear and organized, and we are able to follow each step during the lab. However, we did not check out the lab within the lab section because the last task takes us a long time coding and testing.

Appendix



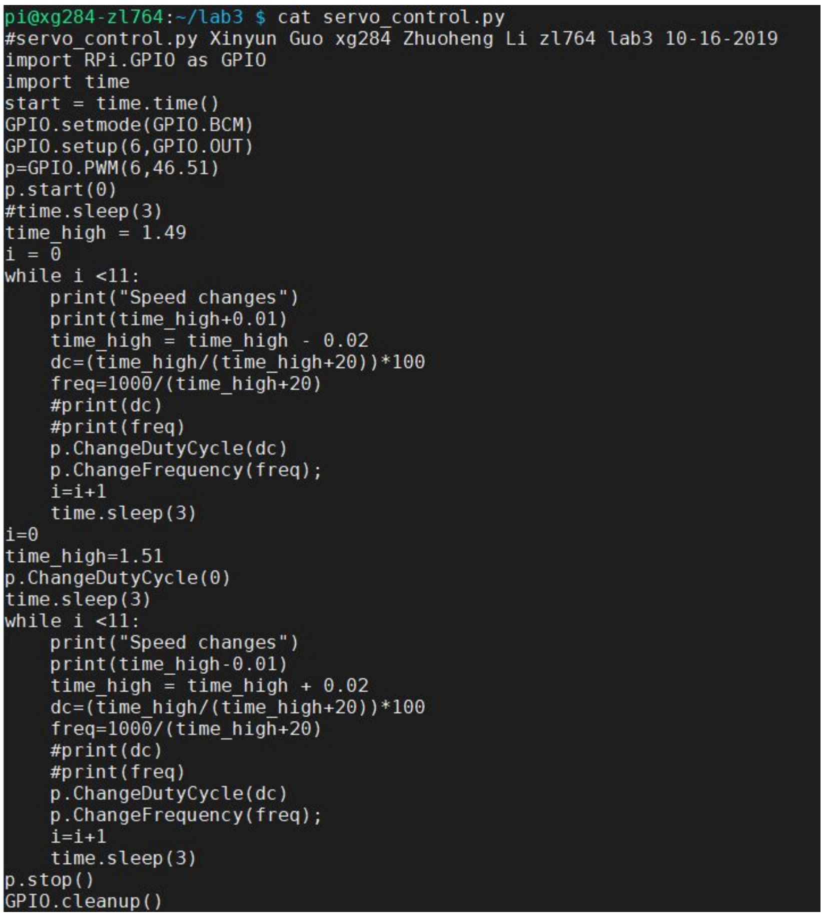
```
pi@xg284-zl764:~/lab3 $ cat blink.py
#blink.py Xinyun Guo xg284 Zhuoheng Li zl764 lab3 10-16-2019
import RPi.GPIO as GPIO
import time
start = time.time()
GPIO.setmode(GPIO.BCM)
GPIO.setup(5,GPIO.OUT)
p=GPIO.PWM(5,1)
while time.time()-start<10:
    p.start(50)
p.stop()
GPIO.cleanup()
```

Figure 1 Python code for blink.py



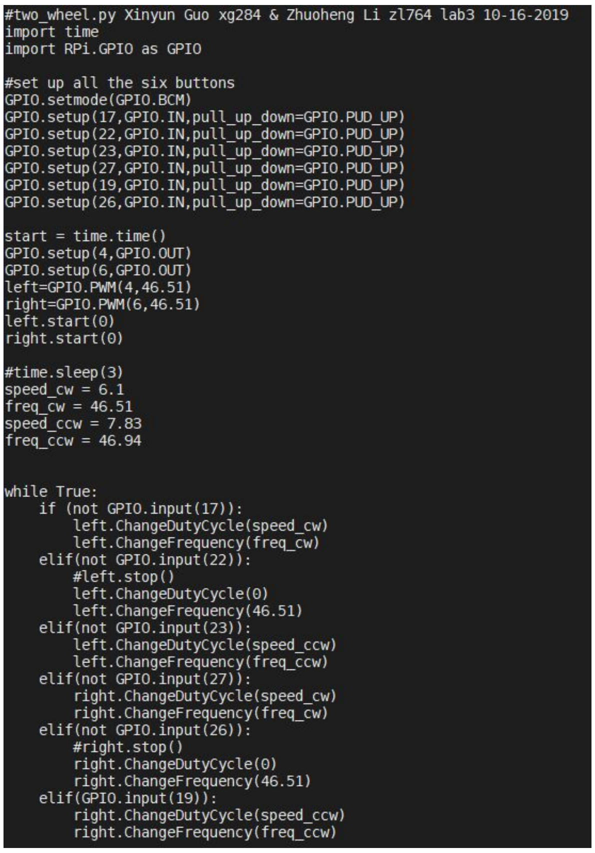
```
pi@xg284-zl764:~/lab3 $ cat pwm_calibrate.py
#pwm_calibrate.py Xinyun Guo xg284 Zhuoheng Li zl764 lab3 10-16-2019
import RPi.GPIO as GPIO
import time
start = time.time()
GPIO.setmode(GPIO.BCM)
GPIO.setup(4,GPIO.OUT)
p=GPIO.PWM(4,46.51)
#p.start(6.9)
while time.time()-start<20:
    p.start(6.9)
p.stop()
GPIO.cleanup()
```

Figure 2 Python code for pwm_calibrate.py



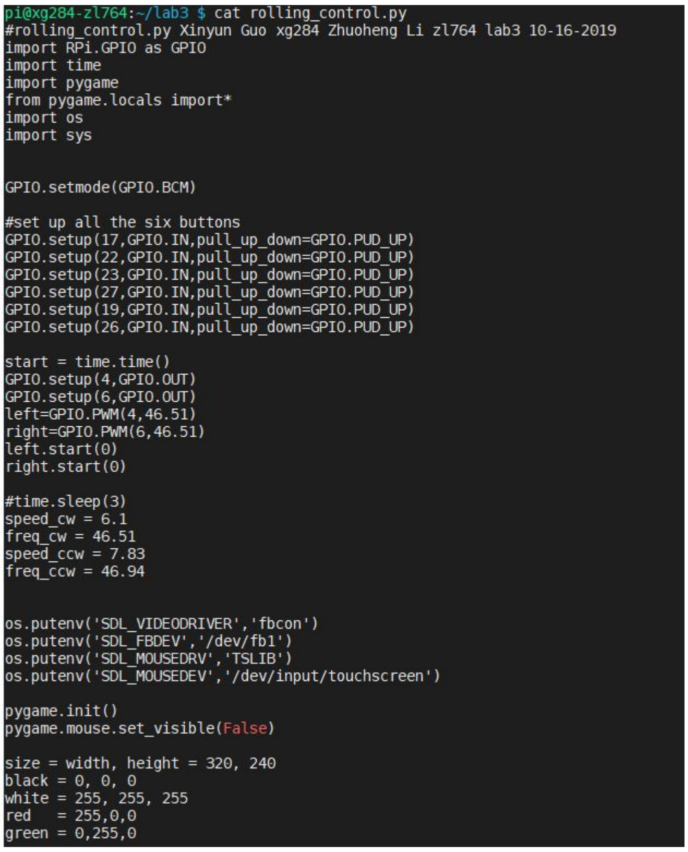
```
pi@xg284-zl764:~/lab3 $ cat servo_control.py
#servo_control.py Xinyun Guo xg284 Zhuoheng Li zl764 lab3 10-16-2019
import RPi.GPIO as GPIO
import time
start = time.time()
GPIO.setmode(GPIO.BCM)
GPIO.setup(6,GPIO.OUT)
p=GPIO.PWM(6,46.51)
p.start(0)
#time.sleep(3)
time_high = 1.49
i = 0
while i <11:
    print("Speed changes")
    print(time_high+0.01)
    time_high = time_high - 0.02
    dc=(time_high/(time_high+20))*100
    freq=1000/(time_high+20)
    #print(dc)
    #print(freq)
    p.ChangeDutyCycle(dc)
    p.ChangeFrequency(freq);
    i=i+1
    time.sleep(3)
i=0
time_high=1.51
p.ChangeDutyCycle(0)
time.sleep(3)
while i <11:
    print("Speed changes")
    print(time_high-0.01)
    time_high = time_high + 0.02
    dc=(time_high/(time_high+20))*100
    freq=1000/(time_high+20)
    #print(dc)
    #print(freq)
    p.ChangeDutyCycle(dc)
    p.ChangeFrequency(freq);
    i=i+1
    time.sleep(3)
p.stop()
GPIO.cleanup()
```

Figure 3 Python code for servo_control.py

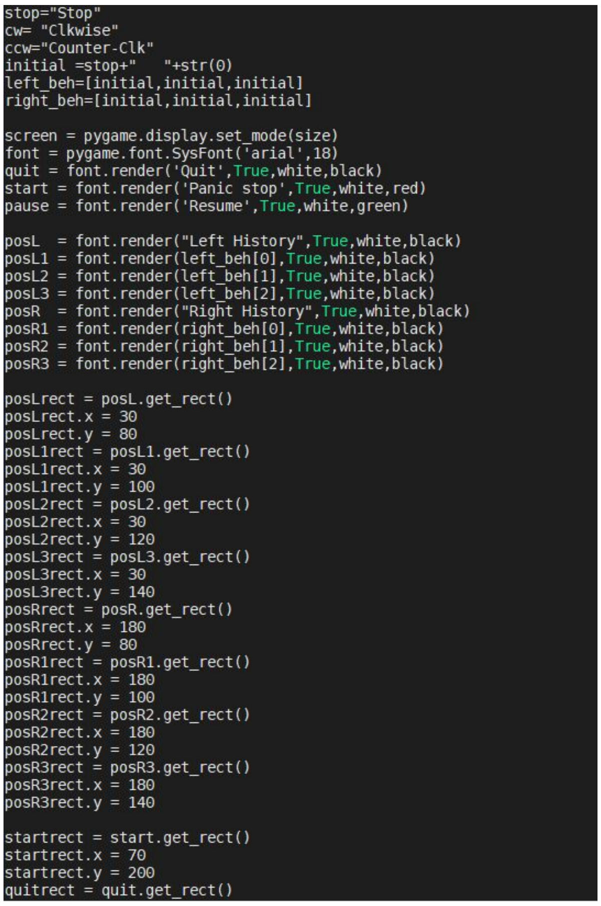


```
#two_wheel.py Xinyun Guo xg284 & Zhuoheng Li zl764 lab3 10-16-2019
import time
import RPi.GPIO as GPIO

#set up all the six buttons
GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(22,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(23,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(27,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(19,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(26,GPIO.IN,pull_up_down=GPIO.PUD_UP)

start = time.time()
GPIO.setup(4,GPIO.OUT)
GPIO.setup(6,GPIO.OUT)
left=GPIO.PWM(4,46.51)
right=GPIO.PWM(6,46.51)
left.start(0)
right.start(0)

#time.sleep(3)
speed_cw = 6.1
freq_cw = 46.51
speed_ccw = 7.83
freq_ccw = 46.94


while True:
    if (not GPIO.input(17)):
        left.ChangeDutyCycle(speed_cw)
        left.ChangeFrequency(freq_cw)
    elif(not GPIO.input(22)):
        #left.stop()
        left.ChangeDutyCycle(0)
        left.ChangeFrequency(46.51)
    elif(not GPIO.input(23)):
        left.ChangeDutyCycle(speed_ccw)
        left.ChangeFrequency(freq_ccw)
    elif(not GPIO.input(27)):
        right.ChangeDutyCycle(speed_cw)
        right.ChangeFrequency(freq_cw)
    elif(not GPIO.input(26)):
        #right.stop()
        right.ChangeDutyCycle(0)
        right.ChangeFrequency(46.51)
    elif(GPIO.input(19)):
        right.ChangeDutyCycle(speed_ccw)
        right.ChangeFrequency(freq_ccw)
```

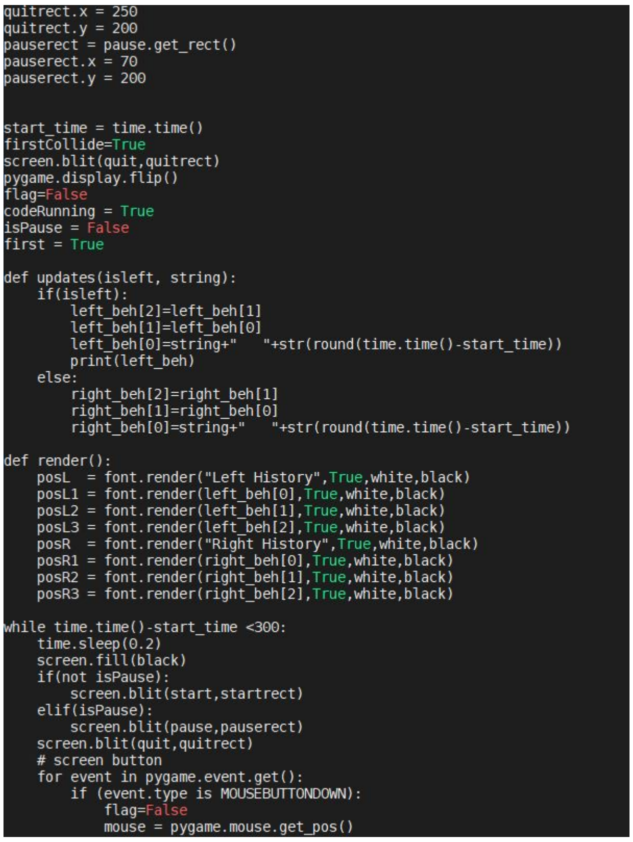
Figure 4 Python code for two_wheel.py



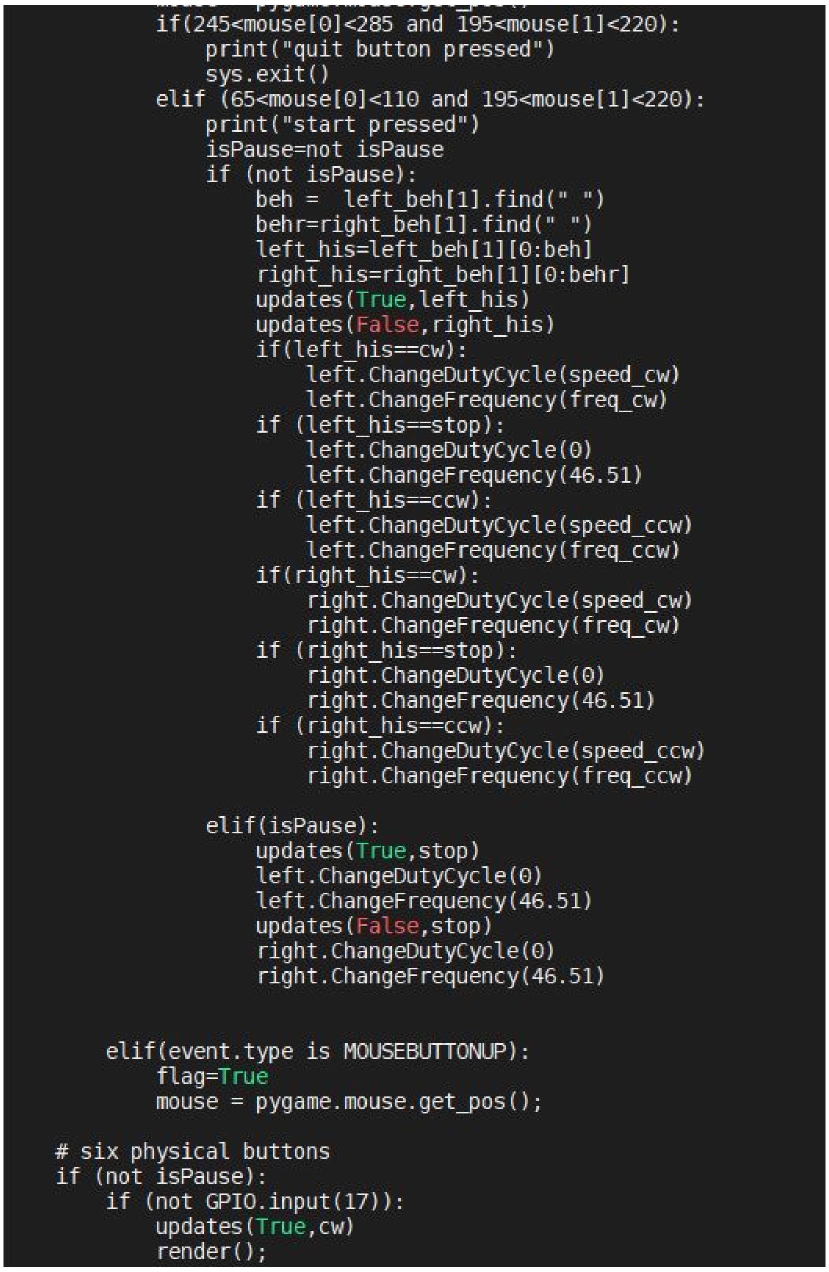
```
pi@xg284-zl764:~/lab3 $ cat rolling_control.py
#rolling_control.py Xinyun Guo xg284 Zhuoheng Li zl764 lab3 10-16-2019
import RPi.GPIO as GPIO
import time
import pygame
from pygame.locals import*
import os
import sys


GPIO.setmode(GPIO.BCM)

#set up all the six buttons
GPIO.setup(17,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(22,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(23,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(27,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(19,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(26,GPIO.IN,pull_up_down=GPIO.PUD_UP)

start = time.time()
GPIO.setup(4,GPIO.OUT)
GPIO.setup(6,GPIO.OUT)
left=GPIO.PWM(4,46.51)
right=GPIO.PWM(6,46.51)
left.start(0)
right.start(0)

#time.sleep(3)
speed_cw = 6.1
freq_cw = 46.51
speed_ccw = 7.83
freq_ccw = 46.94


os.putenv('SDL_VIDEODRIVER','fbcon')
os.putenv('SDL_FBDEV','/dev/fb1')
os.putenv('SDL_MOUSEDRV','TSLIB')
os.putenv('SDL_MOUSEDEV','/dev/input/touchscreen')

pygame.init()
pygame.mouse.set_visible(False)

size = width, height = 320, 240
black = 0, 0, 0
white = 255, 255, 255
red   = 255,0,0
green = 0,255,0
```



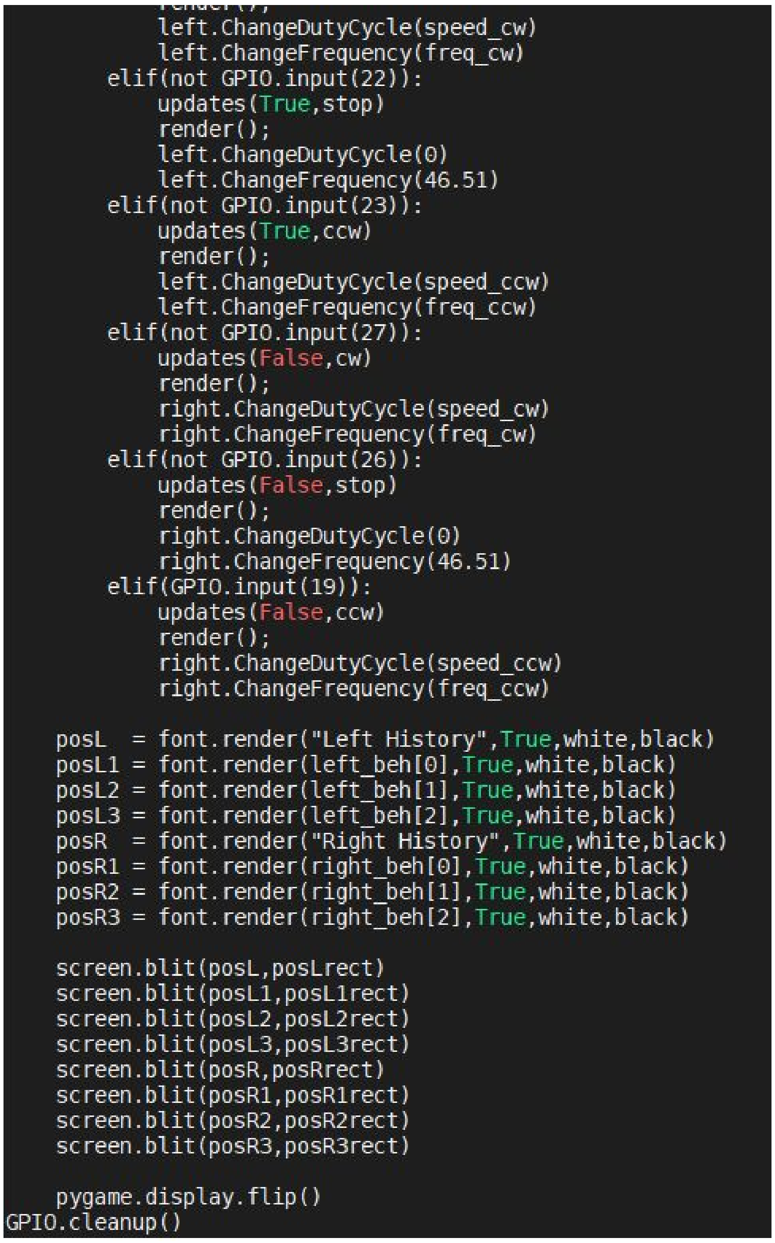
```
stop="Stop"
cw= "Clkwise"
ccw="Counter-Clk"
initial =stop+"    "+str(0)
left_beh=[initial,initial,initial]
right_beh=[initial,initial,initial]

screen = pygame.display.set_mode(size)
font = pygame.font.SysFont('arial',18)
quit = font.render('Quit',True,white,black)
start = font.render('Panic stop',True,white,red)
pause = font.render('Resume',True,white,green)

posL  = font.render("Left History",True,white,black)
posL1 = font.render(left_beh[0],True,white,black)
posL2 = font.render(left_beh[1],True,white,black)
posL3 = font.render(left_beh[2],True,white,black)
posR  = font.render("Right History",True,white,black)
posR1 = font.render(right_beh[0],True,white,black)
posR2 = font.render(right_beh[1],True,white,black)
posR3 = font.render(right_beh[2],True,white,black)

posLrect = posL.get_rect()
posLrect.x = 30
posLrect.y = 80
posL1rect = posL1.get_rect()
posL1rect.x = 30
posL1rect.y = 100
posL2rect = posL2.get_rect()
posL2rect.x = 30
posL2rect.y = 120
posL3rect = posL3.get_rect()
posL3rect.x = 30
posL3rect.y = 140
posRrect = posR.get_rect()
posRrect.x = 180
posRrect.y = 80
posR1rect = posR1.get_rect()
posR1rect.x = 180
posR1rect.y = 100
posR2rect = posR2.get_rect()
posR2rect.x = 180
posR2rect.y = 120
posR3rect = posR3.get_rect()
posR3rect.x = 180
posR3rect.y = 140

startrect = start.get_rect()
startrect.x = 70
startrect.y = 200
quitrect = quit.get_rect()
```



```
quitrect.x = 250
quitrect.y = 200
pauserect = pause.get_rect()
pauserect.x = 70
pauserect.y = 200


start_time = time.time()
firstCollide=True
screen.blit(quit,quitrect)
pygame.display.flip()
flag=False
codeRunning = True
isPause = False
first = True

def updates(isleft, string):
    if(isleft):
        left_beh[2]=left_beh[1]
        left_beh[1]=left_beh[0]
        left_beh[0]=string+"    "+str(round(time.time()-start_time))
        print(left_beh)
    else:
        right_beh[2]=right_beh[1]
        right_beh[1]=right_beh[0]
        right_beh[0]=string+"    "+str(round(time.time()-start_time))

def render():
    posL  = font.render("Left History",True,white,black)
    posL1 = font.render(left_beh[0],True,white,black)
    posL2 = font.render(left_beh[1],True,white,black)
    posL3 = font.render(left_beh[2],True,white,black)
    posR  = font.render("Right History",True,white,black)
    posR1 = font.render(right_beh[0],True,white,black)
    posR2 = font.render(right_beh[1],True,white,black)
    posR3 = font.render(right_beh[2],True,white,black)

while time.time()-start_time <300:
    time.sleep(0.2)
    screen.fill(black)
    if(not isPause):
        screen.blit(start,startrect)
    elif(isPause):
        screen.blit(pause,pauserect)
    screen.blit(quit,quitrect)
    # screen button
    for event in pygame.event.get():
        if (event.type is MOUSEBUTTONDOWN):
            flag=False
            mouse = pygame.mouse.get_pos()
```

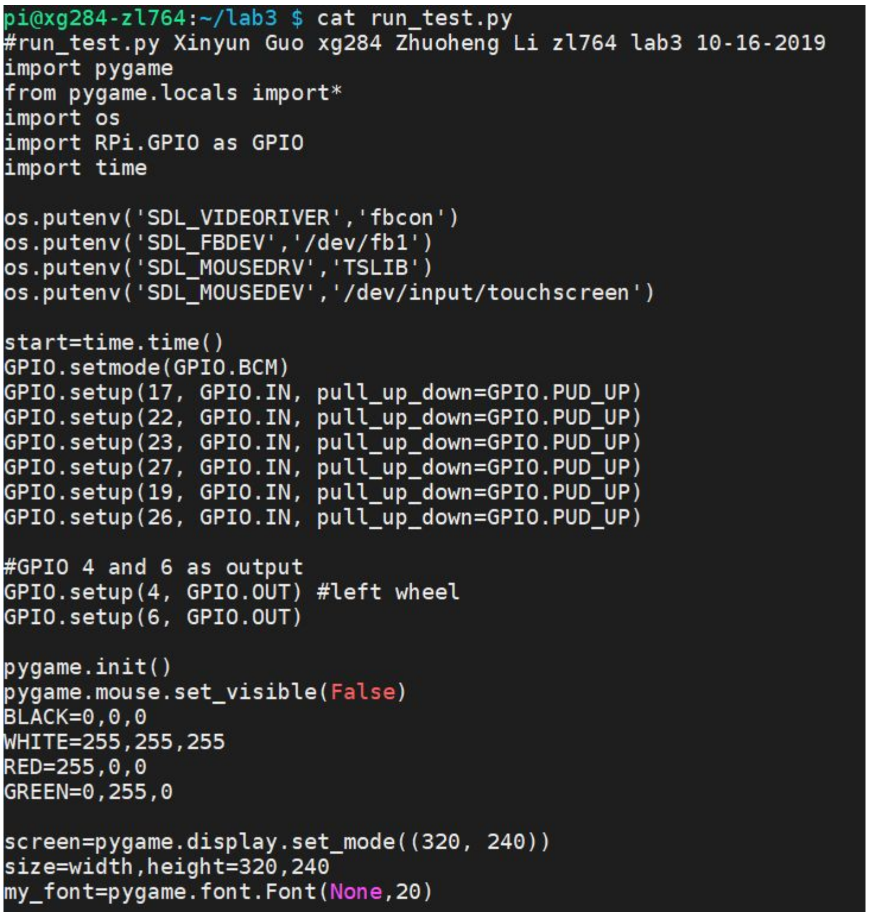


```
        if(245<mouse[0]<285 and 195<mouse[1]<220):
            print("quit button pressed")
            sys.exit()
        elif (65<mouse[0]<110 and 195<mouse[1]<220):
            print("start pressed")
            isPause=not isPause
            if (not isPause):
                beh =  left_beh[1].find(" ")
                behr=right_beh[1].find(" ")
                left_his=left_beh[1][0:beh]
                right_his=right_beh[1][0:behr]
                updates(True,left_his)
                updates(False,right_his)
                if(left_his==cw):
                    left.ChangeDutyCycle(speed_cw)
                    left.ChangeFrequency(freq_cw)
                if (left_his==stop):
                    left.ChangeDutyCycle(0)
                    left.ChangeFrequency(46.51)
                if (left_his==ccw):
                    left.ChangeDutyCycle(speed_ccw)
                    left.ChangeFrequency(freq_ccw)
                if(right_his==cw):
                    right.ChangeDutyCycle(speed_cw)
                    right.ChangeFrequency(freq_cw)
                if (right_his==stop):
                    right.ChangeDutyCycle(0)
                    right.ChangeFrequency(46.51)
                if (right_his==ccw):
                    right.ChangeDutyCycle(speed_ccw)
                    right.ChangeFrequency(freq_ccw)

            elif(isPause):
                updates(True,stop)
                left.ChangeDutyCycle(0)
                left.ChangeFrequency(46.51)
                updates(False,stop)
                right.ChangeDutyCycle(0)
                right.ChangeFrequency(46.51)


    elif(event.type is MOUSEBUTTONUP):
        flag=True
        mouse = pygame.mouse.get_pos();

# six physical buttons
if (not isPause):
    if (not GPIO.input(17)):
        updates(True,cw)
        render();
```

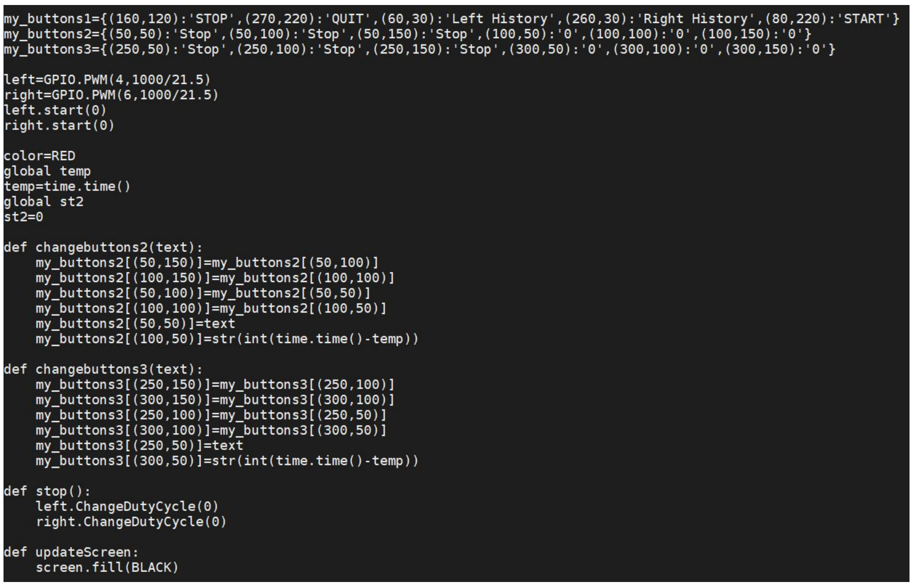


```
            left.ChangeDutyCycle(speed_cw)
            left.ChangeFrequency(freq_cw)
        elif(not GPIO.input(22)):
            updates(True,stop)
            render();
            left.ChangeDutyCycle(0)
            left.ChangeFrequency(46.51)
        elif(not GPIO.input(23)):
            updates(True,ccw)
            render();
            left.ChangeDutyCycle(speed_ccw)
            left.ChangeFrequency(freq_ccw)
        elif(not GPIO.input(27)):
            updates(False,cw)
            render();
            right.ChangeDutyCycle(speed_cw)
            right.ChangeFrequency(freq_cw)
        elif(not GPIO.input(26)):
            updates(False,stop)
            render();
            right.ChangeDutyCycle(0)
            right.ChangeFrequency(46.51)
        elif(GPIO.input(19)):
            updates(False,ccw)
            render();
            right.ChangeDutyCycle(speed_ccw)
            right.ChangeFrequency(freq_ccw)

    posL  = font.render("Left History",True,white,black)
    posL1 = font.render(left_beh[0],True,white,black)
    posL2 = font.render(left_beh[1],True,white,black)
    posL3 = font.render(left_beh[2],True,white,black)
    posR  = font.render("Right History",True,white,black)
    posR1 = font.render(right_beh[0],True,white,black)
    posR2 = font.render(right_beh[1],True,white,black)
    posR3 = font.render(right_beh[2],True,white,black)

    screen.blit(posL,posLrect)
    screen.blit(posL1,posL1rect)
    screen.blit(posL2,posL2rect)
    screen.blit(posL3,posL3rect)
    screen.blit(posR,posRrect)
    screen.blit(posR1,posR1rect)
    screen.blit(posR2,posR2rect)
    screen.blit(posR3,posR3rect)

    pygame.display.flip()
GPIO.cleanup()
```

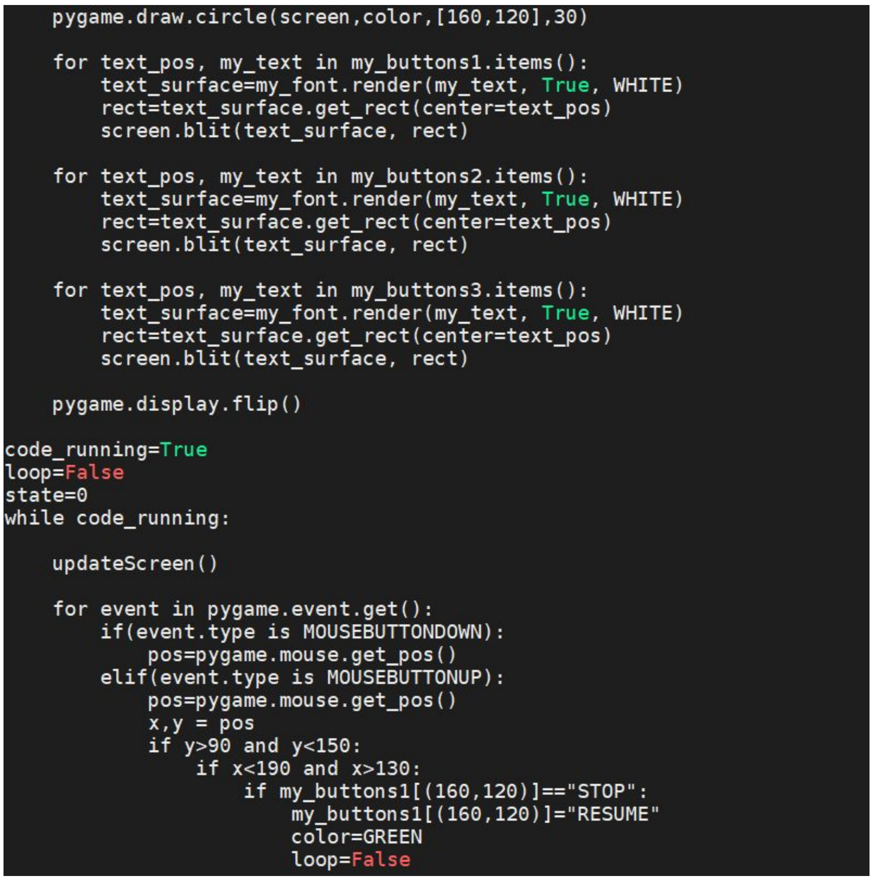
Figure 5 code for rolling_control.py



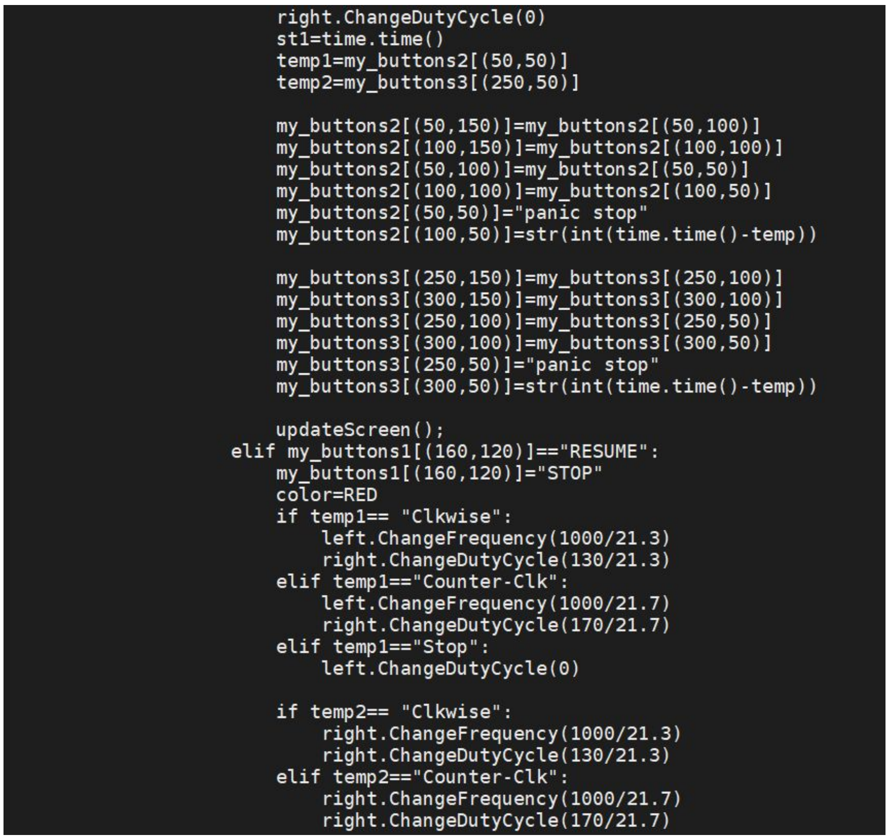
```
pi@xg284-zl764:~/lab3 $ cat run_test.py
#run_test.py Xinyun Guo xg284 Zhuoheng Li zl764 lab3 10-16-2019
import pygame
from pygame.locals import*
import os
import RPi.GPIO as GPIO
import time

os.putenv('SDL_VIDEORIVER','fbcon')
os.putenv('SDL_FBDEV','/dev/fb1')
os.putenv('SDL_MOUSEDRV','TSLIB')
os.putenv('SDL_MOUSEDEV','/dev/input/touchscreen')

start=time.time()
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(22, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(27, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(19, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(26, GPIO.IN, pull_up_down=GPIO.PUD_UP)

#GPIO 4 and 6 as output
GPIO.setup(4, GPIO.OUT) #left wheel
GPIO.setup(6, GPIO.OUT)

pygame.init()
pygame.mouse.set_visible(False)
BLACK=0,0,0
WHITE=255,255,255
RED=255,0,0
GREEN=0,255,0

screen=pygame.display.set_mode((320, 240))
size=width,height=320,240
my_font=pygame.font.Font(None,20)
```



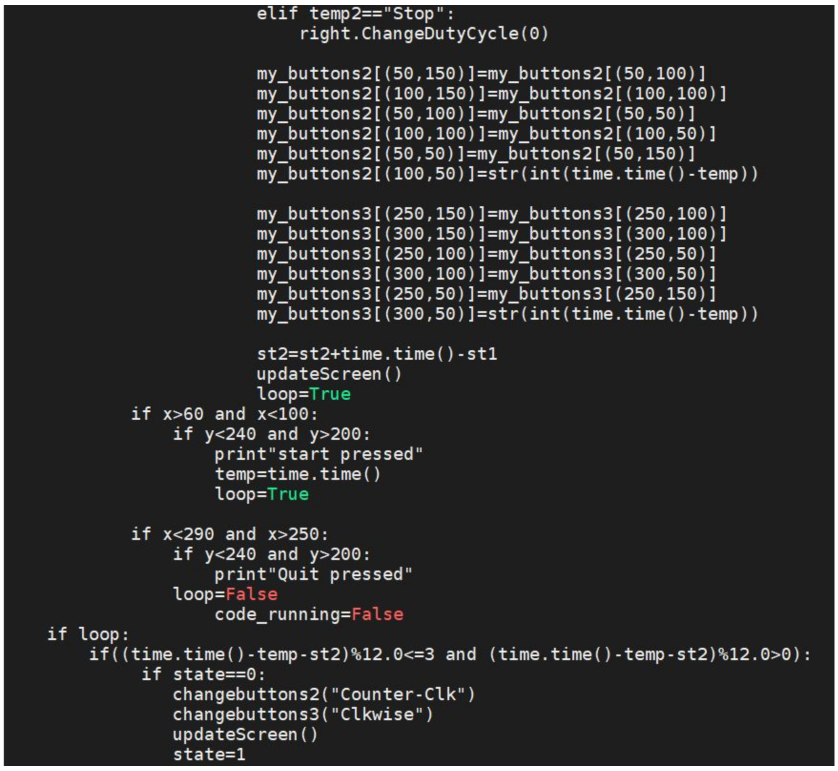
```
my_buttons1={(160,120):'STOP',(270,220):'QUIT',(60,30):'Left History',(260,30):'Right History',(80,220):'START'}
my_buttons2={(50,50):'Stop',(50,100):'Stop',(50,150):'Stop',(100,50):'0',(100,100):'0',(100,150):'0'}
my_buttons3={(250,50):'Stop',(250,100):'Stop',(250,150):'Stop',(300,50):'0',(300,100):'0',(300,150):'0'}

left=GPIO.PWM(4,1000/21.5)
right=GPIO.PWM(6,1000/21.5)
left.start(0)
right.start(0)

color=RED
global temp
temp=time.time()
global st2
st2=0

def changebuttons2(text):
    my_buttons2[(50,150)]=my_buttons2[(50,100)]
    my_buttons2[(100,150)]=my_buttons2[(100,100)]
    my_buttons2[(50,100)]=my_buttons2[(50,50)]
    my_buttons2[(100,100)]=my_buttons2[(100,50)]
    my_buttons2[(50,50)]=text
    my_buttons2[(100,50)]=str(int(time.time()-temp))

def changebuttons3(text):
    my_buttons3[(250,150)]=my_buttons3[(250,100)]
    my_buttons3[(300,150)]=my_buttons3[(300,100)]
    my_buttons3[(250,100)]=my_buttons3[(250,50)]
    my_buttons3[(300,100)]=my_buttons3[(300,50)]
    my_buttons3[(250,50)]=text
    my_buttons3[(300,50)]=str(int(time.time()-temp))

def stop():
    left.ChangeDutyCycle(0)
    right.ChangeDutyCycle(0)

def updateScreen:
    screen.fill(BLACK)
```



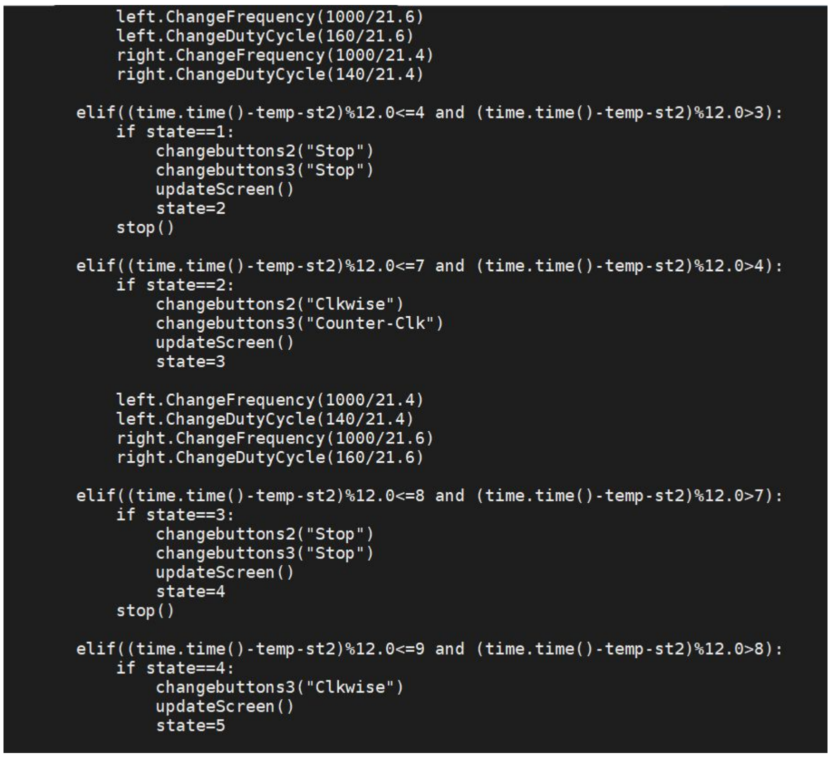
```
    pygame.draw.circle(screen,color,[160,120],30)

    for text_pos, my_text in my_buttons1.items():
        text_surface=my_font.render(my_text, True, WHITE)
        rect=text_surface.get_rect(center=text_pos)
        screen.blit(text_surface, rect)

    for text_pos, my_text in my_buttons2.items():
        text_surface=my_font.render(my_text, True, WHITE)
        rect=text_surface.get_rect(center=text_pos)
        screen.blit(text_surface, rect)

    for text_pos, my_text in my_buttons3.items():
        text_surface=my_font.render(my_text, True, WHITE)
        rect=text_surface.get_rect(center=text_pos)
        screen.blit(text_surface, rect)

    pygame.display.flip()

code_running=True
loop=False
state=0
while code_running:

    updateScreen()

    for event in pygame.event.get():
        if(event.type is MOUSEBUTTONDOWN):
            pos=pygame.mouse.get_pos()
        elif(event.type is MOUSEBUTTONUP):
            pos=pygame.mouse.get_pos()
            x,y = pos
            if y>90 and y<150:
                if x<190 and x>130:
                    if my_buttons1[(160,120)]=="STOP":
                        my_buttons1[(160,120)]="RESUME"
                        color=GREEN
                        loop=False
```



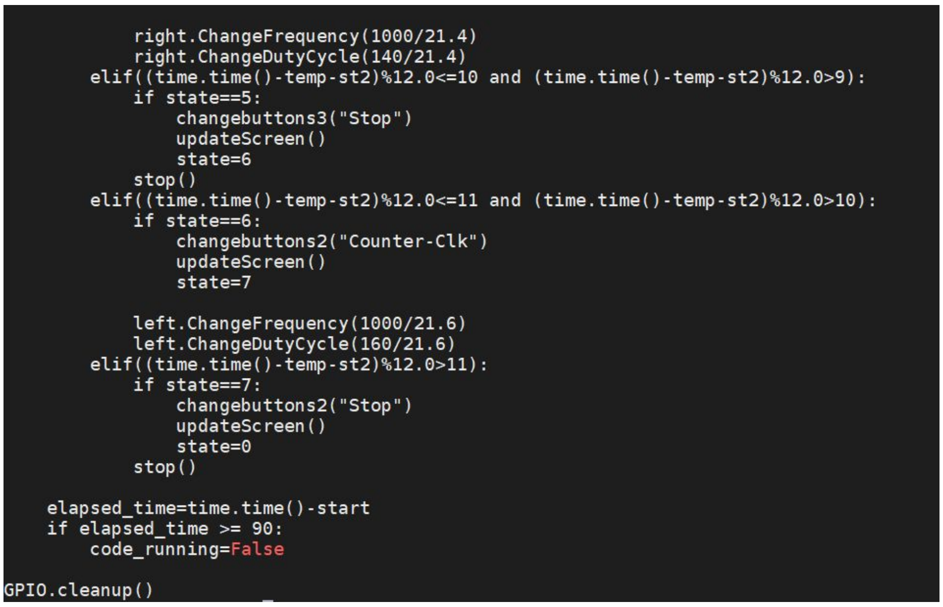
```
                      right.ChangeDutyCycle(0)
                      st1=time.time()
                      temp1=my_buttons2[(50,50)]
                      temp2=my_buttons3[(250,50)]

                      my_buttons2[(50,150)]=my_buttons2[(50,100)]
                      my_buttons2[(100,150)]=my_buttons2[(100,100)]
                      my_buttons2[(50,100)]=my_buttons2[(50,50)]
                      my_buttons2[(100,100)]=my_buttons2[(100,50)]
                      my_buttons2[(50,50)]="panic stop"
                      my_buttons2[(100,50)]=str(int(time.time()-temp))

                      my_buttons3[(250,150)]=my_buttons3[(250,100)]
                      my_buttons3[(300,150)]=my_buttons3[(300,100)]
                      my_buttons3[(250,100)]=my_buttons3[(250,50)]
                      my_buttons3[(300,100)]=my_buttons3[(300,50)]
                      my_buttons3[(250,50)]="panic stop"
                      my_buttons3[(300,50)]=str(int(time.time()-temp))

                      updateScreen();
                  elif my_buttons1[(160,120)]=="RESUME":
                      my_buttons1[(160,120)]="STOP"
                      color=RED
                      if temp1== "Clkwise":
                          left.ChangeFrequency(1000/21.3)
                          right.ChangeDutyCycle(130/21.3)
                      elif temp1=="Counter-Clk":
                          left.ChangeFrequency(1000/21.7)
                          right.ChangeDutyCycle(170/21.7)
                      elif temp1=="Stop":
                          left.ChangeDutyCycle(0)

                      if temp2== "Clkwise":
                          right.ChangeFrequency(1000/21.3)
                          right.ChangeDutyCycle(130/21.3)
                      elif temp2=="Counter-Clk":
                          right.ChangeFrequency(1000/21.7)
                          right.ChangeDutyCycle(170/21.7)
```



```
                    elif temp2=="Stop":
                        right.ChangeDutyCycle(0)

                    my_buttons2[(50,150)]=my_buttons2[(50,100)]
                    my_buttons2[(100,150)]=my_buttons2[(100,100)]
                    my_buttons2[(50,100)]=my_buttons2[(50,50)]
                    my_buttons2[(100,100)]=my_buttons2[(100,50)]
                    my_buttons2[(50,50)]=my_buttons2[(50,150)]
                    my_buttons2[(100,50)]=str(int(time.time()-temp))

                    my_buttons3[(250,150)]=my_buttons3[(250,100)]
                    my_buttons3[(300,150)]=my_buttons3[(300,100)]
                    my_buttons3[(250,100)]=my_buttons3[(250,50)]
                    my_buttons3[(300,100)]=my_buttons3[(300,50)]
                    my_buttons3[(250,50)]=my_buttons3[(250,150)]
                    my_buttons3[(300,50)]=str(int(time.time()-temp))

                    st2=st2+time.time()-st1
                    updateScreen()
                    loop=True
        if x>60 and x<100:
            if y<240 and y>200:
                print"start pressed"
                temp=time.time()
                loop=True

        if x<290 and x>250:
            if y<240 and y>200:
                print"Quit pressed"
            loop=False
                code_running=False
if loop:
    if((time.time()-temp-st2)%12.0<=3 and (time.time()-temp-st2)%12.0>0):
         if state==0:
            changebuttons2("Counter-Clk")
            changebuttons3("Clkwise")
            updateScreen()
            state=1
```



```
        left.ChangeFrequency(1000/21.6)
        left.ChangeDutyCycle(160/21.6)
        right.ChangeFrequency(1000/21.4)
        right.ChangeDutyCycle(140/21.4)

    elif((time.time()-temp-st2)%12.0<=4 and (time.time()-temp-st2)%12.0>3):
        if state==1:
            changebuttons2("Stop")
            changebuttons3("Stop")
            updateScreen()
            state=2
        stop()

    elif((time.time()-temp-st2)%12.0<=7 and (time.time()-temp-st2)%12.0>4):
        if state==2:
            changebuttons2("Clkwise")
            changebuttons3("Counter-Clk")
            updateScreen()
            state=3

        left.ChangeFrequency(1000/21.4)
        left.ChangeDutyCycle(140/21.4)
        right.ChangeFrequency(1000/21.6)
        right.ChangeDutyCycle(160/21.6)

    elif((time.time()-temp-st2)%12.0<=8 and (time.time()-temp-st2)%12.0>7):
        if state==3:
            changebuttons2("Stop")
            changebuttons3("Stop")
            updateScreen()
            state=4
        stop()

    elif((time.time()-temp-st2)%12.0<=9 and (time.time()-temp-st2)%12.0>8):
        if state==4:
            changebuttons3("Clkwise")
            updateScreen()
            state=5
```



```
            right.ChangeFrequency(1000/21.4)
            right.ChangeDutyCycle(140/21.4)
        elif((time.time()-temp-st2)%12.0<=10 and (time.time()-temp-st2)%12.0>9):
            if state==5:
                changebuttons3("Stop")
                updateScreen()
                state=6
            stop()
        elif((time.time()-temp-st2)%12.0<=11 and (time.time()-temp-st2)%12.0>10):
            if state==6:
                changebuttons2("Counter-Clk")
                updateScreen()
                state=7

            left.ChangeFrequency(1000/21.6)
            left.ChangeDutyCycle(160/21.6)
        elif((time.time()-temp-st2)%12.0>11):
            if state==7:
                changebuttons2("Stop")
                updateScreen()
                state=0
            stop()

    elapsed_time=time.time()-start
    if elapsed_time >= 90:
        code_running=False

GPIO.cleanup()
```

Figure 6 run_test.py