Lab3 Week1 & Week2 Report

Names: Zixiao Wang, Yue Zhang

NetID: zw579, yz2455

Date: Nov 3, 2019

Introduction

We did lab3 through following steps:

1. Exploring a number of attached components with adding output capability of R-Pi.
2. Build robot platform and frame and designed robot application with a user interface.
3. Modularize code blocks that can be reused for future labs.

Design and Testing

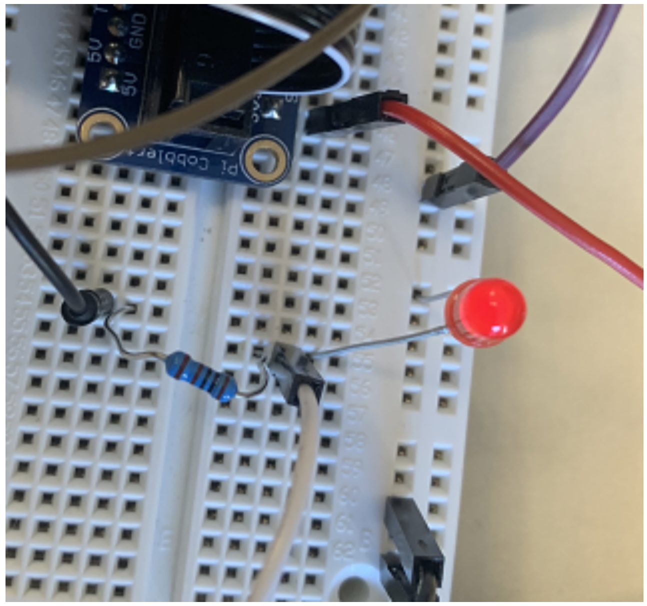
Here we listed the steps we took and issues we came into.

1. Implement an LED circuit on the selected output pin and change the blink rates using PWM settings with RPi.GPIO PWM in python code. Verify the PWM signal using oscilloscope.

- Testing:
  We wrote a python code *blink.py* to blink the LED on and off over a second, and change the blink frequency every 1s.
  Here's our blink.py:

```python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(16, GPIO.OUT)

p = GPIO.PWM(16, 1)
p.start(50)
print(type(p))
try:
    while True:
        for f in range(1, 11):
            p.ChangeFrequency(f)
            time.sleep(1)
        for f in range(11, 0, -1):
            p.ChangeFrequency(f)
            time.sleep(1)
except KeyboardInterrupt:
    pass

p.stop()
GPIO.cleanup()
```

  Here's how blinking LED looking like:



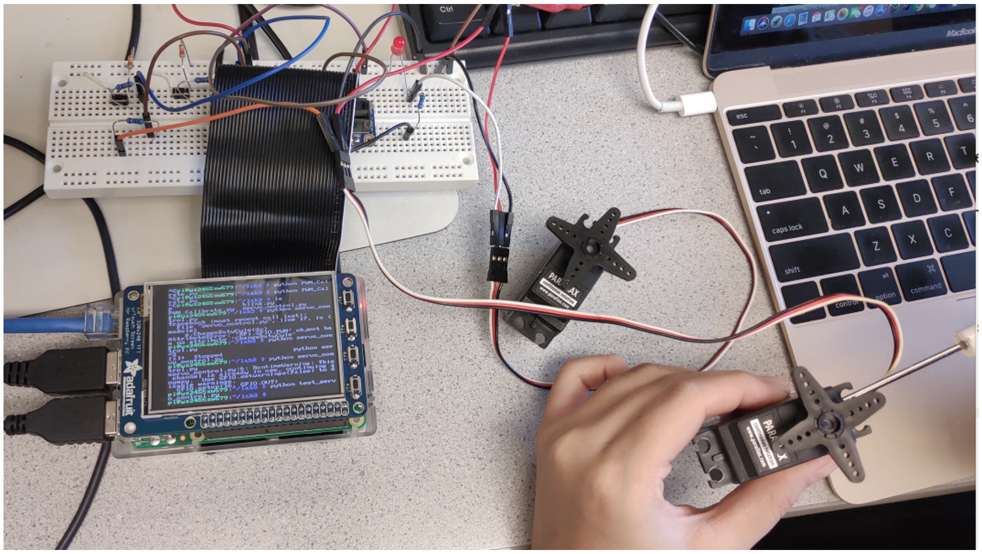
(Need oscilloscope pic here)

2. Calibrate servos with pwm_calibrate.py and provide clean resets of GPIO pins that remain set after failed test code runs.

- Testing:
  Here we show the code for calibrating with frequency of 1/21.5hz (approximately 46.5hz) and duty cycle 1.5/21.5 (approximately 6.98%)

```python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(26, GPIO.OUT)

p = GPIO.PWM(26, 46.5)
try:
    p.start(6.98)
except KeyboardInterrupt:
    pass
p.stop()
GPIO.cleanup()
```

Using a screwdriver to adjust the potentiometer until servos stop completely.



---

3. Develop a python code named servo_control.py to:
    - Start the servo from stop state.
    - Then range the speed of serbo through 10 steps in clockwise direction with each speed increment running for 3 seconds.
    - Then repeat the same step in counter-clockwise direction.
    - Stop the servo.

- Testing:

  Here’s the record of duty cycle, frequency and pulse width of PWM

```
Duty Cycle: 0.06976744186046512  | Frequency:  0.046511627906976744  | Pulse Width: 1.48
Duty Cycle: 0.06890130353817504  | Frequency:  0.04655493482309125  | Pulse Width: 1.46
Duty Cycle: 0.06803355079217148  | Frequency:  0.046598322460391424  | Pulse Width: 1.44
Duty Cycle: 0.06716417910447761  | Frequency:  0.046641791044776115  | Pulse Width: 1.42
Duty Cycle: 0.06629318394024275  | Frequency:  0.046685340802987856  | Pulse Width: 1.4
Duty Cycle: 0.06542056074766354  | Frequency:  0.04672897196261683  | Pulse Width: 1.38
Duty Cycle: 0.06454630495790459  | Frequency:  0.046772684752104776  | Pulse Width: 1.3599999999999999
Duty Cycle: 0.06367041198501872  | Frequency:  0.04681647940074907  | Pulse Width: 1.3399999999999999
Duty Cycle: 0.0627928772258669  | Frequency:  0.046860356138706656  | Pulse Width: 1.3199999999999998
Duty Cycle: 0.061913696060037514  | Frequency:  0.046904315196998124  | Pulse Width: 1.2999999999999998
Duty Cycle: 0.06976744186046512  | Frequency:  0.046511627906976744  | Pulse Width: 1.52
Duty Cycle: 0.07063197026022305  | Frequency:  0.04646840148698885  | Pulse Width: 1.54
Duty Cycle: 0.07149489322191273  | Frequency:  0.04642525533890437  | Pulse Width: 1.56
Duty Cycle: 0.07235621521335808  | Frequency:  0.0463821892393321  | Pulse Width: 1.58
Duty Cycle: 0.07321594068582021  | Frequency:  0.046339202965708995  | Pulse Width: 1.6
Duty Cycle: 0.07407407407407407  | Frequency:  0.046296296296296294  | Pulse Width: 1.62
Duty Cycle: 0.07493061979648474  | Frequency:  0.04625346901017576  | Pulse Width: 1.6400000000000001
Duty Cycle: 0.07578558225508318  | Frequency:  0.04621072088724584  | Pulse Width: 1.6600000000000001
Duty Cycle: 0.07663896583564174  | Frequency:  0.04616805170821791  | Pulse Width: 1.6800000000000002
Duty Cycle: 0.07749077490774908  | Frequency:  0.046125461254612546  | Pulse Width: 1.7000000000000002
```

Here's the list of servo_control.py. It set the servo speed from 0 to clockwise full speed, back to 0 and toward counterclockwise full speed. Interval between speed jump is 3s.

```python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(16, GPIO.OUT)

p = GPIO.PWM(16, 1/21.5)
p.start(100*1.5/21.5)

# for t in range(1.5, 1.28, -0.02):
try:
    t = 1.5
    while t >= 1.3:
        dc = t / (t + 20)
        f = 1 / (t + 20)
        p.ChangeDutyCyle(dc)
        p.ChangeFrequency(f)
        time.sleep(3)
        t -= 0.02

    # for t in range(1.5, 1.72, 0.02):
    t = 1.5
    while t <= 1.7:
        dc = t / (t + 20)
        f = 1 / (t + 20)
        p.ChangeDutyCyle(dc)
        p.ChangeFrequency(f)
        time.sleep(3)
        t += 0.02
except KeyboardInterrupt:
    pass

#stop servo

dc = 1.5 / (1.5 + 20)
f = 1 / (t + 20)
p.ChangeDutyCyle(dc)
p.ChangeFrequency(f)
p.stop()
GPIO.cleanup()
```

4. Control two servos using buttons for full speed clockwise and counter-clockwise.

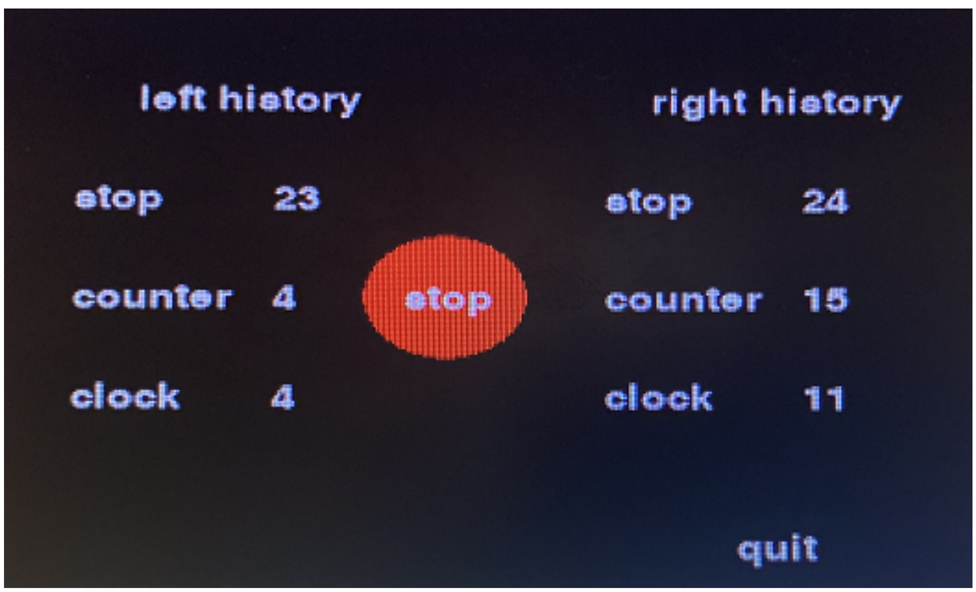
- Testing:

```
import RPi.GPIO as GPIO
import time
import subprocess

GPIO.setmode(GPIO.BCM)
#left
GPIO.setup(16, GPIO.OUT)
#right
GPIO.setup(13, GPIO.OUT)

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)

pLeft = GPIO.PWM(16, 1000/21.5)
pLeft.start(100*1.5/21.5)
pRight = GPIO.PWM(13, 1000/21.5)
pRight.start(100*1.5/21.5)
```

```
#left stop
def GPIO17_callback(channel):
    # pLeft.ChangeDutyCycle(0)
    pLeft.ChangeDutyCycle(100*1.5/21.5)
    pLeft.ChangeFrequency(1000/21.5)
#left clockwise
def GPIO22_callback(channel):
    pLeft.ChangeDutyCycle(100*1.7/21.7)
    pLeft.ChangeFrequency(1000/21.7)
#left counter-clockwise
def GPIO23_callback(channel):
    pLeft.ChangeDutyCycle(100*1.3/21.3)
    pLeft.ChangeFrequency(1000/21.3)
#right stop
def GPIO27_callback(channel):
    # pRight.ChangeDutyCycle(0)
    pRight.ChangeDutyCycle(100*1.5/21.5)
    pRight.ChangeFrequency(1000/21.5)
#right clockwise
def GPIO19_callback(channel):
    pRight.ChangeDutyCycle(100*1.7/21.7)
    pRight.ChangeFrequency(1000/21.7)
#right counter-clockwise
def GPIO26_callback(channel):
    pRight.ChangeDutyCycle(100*1.3/21.3)
    pRight.ChangeFrequency(1000/21.3)

GPIO.add_event_detect(17, GPIO.FALLING, callback=GPIO17_callback, bouncetime = 300)
GPIO.add_event_detect(22, GPIO.FALLING, callback=GPIO17_callback, bouncetime = 300)
GPIO.add_event_detect(23, GPIO.FALLING, callback=GPIO17_callback, bouncetime = 300)
GPIO.add_event_detect(27, GPIO.FALLING, callback=GPIO17_callback, bouncetime = 300)
GPIO.add_event_detect(19, GPIO.FALLING, callback=GPIO17_callback, bouncetime = 300)
GPIO.add_event_detect(26, GPIO.FALLING, callback=GPIO17_callback, bouncetime = 300)

now = time.time()
future = now + 20

while True:
    time.sleep(0.05)
    if time.time() > future:
        break
```

Above program control servos behavior using 6 buttons: left clockwise, left stop, left counter-clockwise, right clockwise, right stop and right counter-clockwise.
There are two ways to stop a servos: using **calibration signal** or send **0 duty cycle PWM** to servo. The **latter** will put servo at a complete stop while the former sometimes does not work well because of the calibration error.

5. Implement a python program rolling_control.py with following functions:
    a. Display motor history on TFT screen.
    b. Show a red panic stop button. If pressed, motor stop at once and stop button changes to green resume button.
    c. Implement quit button to shut down program.
- Testing:
  Here we show the final interface on TFT:



Below we show some key parts of our program and way of modularization.
Here we put button and history log positions in dictionary:

```
pygame.init()
pygame.mouse.set_visible(True)
WHITE = 255, 255, 255
BLACK = 0,0,0
screen = pygame.display.set_mode((320, 240))

my_font= pygame.font.Font(None, 20)
my_buttons= { 'stop':(140,120),'start':(80,220), 'quit':(240, 220), 'left history':(80,40), 'right history':(240, 40)}
new_buttons= { 'resume':(140,120), 'start':(80,220),'quit':(240, 220), 'left history':(80,40), 'right history':(240, 40)}
# q_buttons = { 'stop':(180,120), 'quit':(240, 180), 'left history':(80,40)}

screen.fill(BLACK)
rectList = []
surfaceList = []
newRect = []
newSurf = []

lpos = {0:(40,80), 1:(40,120), 2:(40,160)}
rpos = {0:(200,80), 1:(200,120), 2:(200,160)}
ltpos = {0:(100,80), 1:(100,120), 2:(100,160)}
rtpos = {0:(260,80), 1:(260,120), 2:(260,160)}
```

Then using queue to implement the updating of log information.

```
leftq = [("stop1", "0"), ("stop2", "0"), ("stop3", "0")]
rightq = [("stop1", "0"), ("stop2", "0"), ("stop3", "0")]
start = time.time()
```

In each button callback function, we change the duty cycle and frequency of according PWM output and update the queue by popping a head and appending a tail. Then update screen display.

```
def GPIO22_callback(channel):
    # print("Left clock")

    pLeft.start(100*1.7/21.7)
    pLeft.ChangeFrequency(1000/21.7)
    leftq.pop(0)
    leftq.append(("clock", str(int(time.time()-start))))
    print(leftq)
    screen.fill(BLACK)
    display("stop")
```

Here we show the modularization of display function: it can take an argument to display stop or resume state of screen.

```python
def display(state):
    if state == "stop":
        pygame.draw.circle(screen, pygame.Color(255,0,0), (140,120),25,0)
        for my_text, text_pos in my_buttons.items():
            text_surface = my_font.render(my_text, True, WHITE)
            rect = text_surface.get_rect(center = text_pos)
            surfaceList.append(text_surface)
            rectList.append(rect)
            screen.blit(text_surface, rect)

    elif state == "resume":
        pygame.draw.circle(screen, pygame.Color(0,255,0), (140,120),25,0)
        for my_text, text_pos in new_buttons.items():
            text_surface = my_font.render(my_text, True, WHITE)
            rect = text_surface.get_rect(center = text_pos)
            newSurf.append(text_surface)
            newRect.append(rect)
            screen.blit(text_surface, rect)


    for i in range(3):
        l_surface = my_font.render(leftq[i][0], True, WHITE)
        rect = text_surface.get_rect(center = lpos[2-i])
        lt_surface = my_font.render(leftq[i][1], True, WHITE)
        rect_t = text_surface.get_rect(center = ltpos[2-i])
        screen.blit(l_surface, rect)
        screen.blit(lt_surface, rect_t)

    for i in range(3):
        r_surface = my_font.render(rightq[i][0], True, WHITE)
        rect = text_surface.get_rect(center = rpos[2-i])
        rt_surface = my_font.render(rightq[i][1], True, WHITE)
        rect_t = text_surface.get_rect(center = rtpos[2-i])
        screen.blit(r_surface, rect)
        screen.blit(rt_surface, rect_t)
```

Here we show the modularization of quit function and main function to detect touch screen behaviour.

```python
def quit():
    global code_run
    code_run = False

display("stop")
code_run = True
while code_run:
    #set sleep time to avoid start-up latency
    time.sleep(0.005)
    x = -1
    y = -1
    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 x < 120 and y > 120:
        print("Start Pressed")
    elif (y > 120 and x > 160) :
            print("Quit Pressed")
            start_flag = False
            quit()
    elif 120 < x < 160 and 100 < y < 140:
        pLeft.stop()
        pRight.stop()
        screen.fill(BLACK)
        display("resume")
```

6. Assemble a robot frame with Pi.

- Testing:
  In this step, we followed the lab write-up to assemble the robot. First we construct the frame then install servos on it. Then connect power to servos. We use AAA battery for servos power and another power bank for RPi. Their grounds are connected.

7. Design a python code name run_test.py that allow the robot pi move automatically in following steps:
    a. Move forward;
    b. Stop;
    c. Move Backward;
    d. Turn Left;

e. Stop;
f. Turn Right;
g. Stop
h. Repeat

- Testing:

  On the base of rolling_control.py, we add moving function in order of forward, stop, backward, stop, left and right.

  Here's an example of forward code:

```
if( x < 120 and y > 120):
    if(switch < 7):
        switch = switch + 1
    else: switch = 1
someFlag = True
#Move forward
if(switch == 1):
    now = time.time()
    while time.time() - now < 1.5:
        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 120 < x < 160 and 100 < y < 140:

            pRight.ChangeDutyCycle(0)
            pLeft.ChangeDutyCycle(0)
            someFlag = False
            switch = 1
            screen.fill(BLACK)
            display("resume")
            break
        switch = 2
        pRight.ChangeDutyCycle(100*1.6/21.6)
        pRight.ChangeFrequency(1000/21.6)
        pLeft.ChangeDutyCycle(100*1.4/21.4)
        pLeft.ChangeFrequency(1000/21.4)
    if someFlag :
        leftq.pop(0)
        leftq.append(("Forward", str(int(time.time()-start))))
        rightq.pop(0)
        rightq.append(("Forward", str(int(time.time()-start))))
        screen.fill(BLACK)
        display("stop")
```

  At the start of "forward" code, we check if panic stop button is touched. If touched, servos will be stoped and we break the while loop. We use variable switch to keep track of where we were when panic stop happens. At the end of "forward", TFT will update the servo history.

  Here's another example of "stop" code.

```
            #Stop
        if(switch == 2):
            now = time.time()
            while time.time() - now < 1:
                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 120 < x < 160 and 100 < y < 140:
                    # pLeft.stop()
                    # pRight.stop()
                    pRight.ChangeDutyCycle(0)
                    #pRight.ChangeFrequency(0)
                    pLeft.ChangeDutyCycle(0)
                    #pLeft.ChangeFrequency(0)
                    someFlag = False
                    switch = 2
                    screen.fill(BLACK)
                    display("resume")
                    break
                pRight.ChangeDutyCycle(0)
                #pRight.ChangeFrequency(0)
                pLeft.ChangeDutyCycle(0)
                #pLeft.ChangeFrequency(0)
                switch = 3
            if someFlag :
                leftq.pop(0)
                leftq.append(("Stop", str(int(time.time()-start))))
                rightq.pop(0)
                rightq.append(("Stop", str(int(time.time()-start))))
                screen.fill(BLACK)
                display("stop")
```

8. Enable wifi and configure RPi to launch application at boot.

- Testing:
  We add a line of command to run run_test.py at the end of /home/pi/.bashrc. Then we changed configuration in Boot Options/Desktop/CLI and selected Console Autologin.
  Here we show that when power bank is plugged in, Pi will boot and run the run_test.py at once.

Conclusions:

Things that works smoothly:

a. Used PWM in RPi.GPIO to create a PWM output and check the output in oscilloscope.
b. Designed a python code blink.py to use PWM calls to blink an LED.
c. Calibrate two wheels.
d. Developed a python code servo_control.py to perform following functions by adjusting frequency and duty cycle:

Range the speed of the servo through ten speed steps in the clockwise direction, and each speed increment runs for 3 seconds.

Range the speed of the servo through ten speed steps in the counter-clockwise direction, and each speed increment runs for 3 seconds.

e. Implemented two_wheel.py to control the servos using buttons with the six states: left servo, clockwise; left servo, stop; left servo, counter-clockwise; right servo, clockwise; right servo, stop; right servo, counter-clockwise.
f. Designed rollong_control.py with the following functions:
   Record start time and directions for each motor and display a scrolling history of the most recent motion.
   Add a red 'panic stop' button on the piTFT, if pressed, motors immediately stop and this button changes to a green 'resume' button.
   Implement a quit button on the piTFT. When hit, quit causes the program to end and control return s to the linux console screen.
g. Assembled the robot.
h. Tested two_wheel.py and rolling_control.py on our robot.
i. Designed a python code run_test.py with the following functions:
j. Move robot forward about 1 foot; stop; move robot backwards about 1 foot; pivot left; stop; pivot right; stop; and loop this process. Besides, history should be updated every moment.
k. Enabled wifi, in which way we could remotely control the pi without network cable.
l. Made run_test.py starts at power-up.

Things that works not well at first:

a. For our run_test.py function, we cannot quit the program at first.
   Solution: this problem is caused by logic fault in our program, we changed our program to detect whether quit button has been pressed before coming into the robot moving part .

b. For the run_test.py function, our robot works for a while and then doesn’t work.
   Solution: it is caused by “stop() and start()” in PWM controlling. We changed ‘stop()’ to ‘set duty circle to zero’ then problem solved.

Improvement suggestion:

a. Provide a reference program for students, and we could compare our program with the reference, in which way we could improve our program.

Items could be clearer:

a. Declare what will happen if we pressed the “resume”, whether it will start moving from the first state “moving forward” or continuing the state before stop button pressed.