Robot Arm Documentation
Science Olympiad 2015-2016

By Mayank Mali and Peter Wilson

Acton-Boxborough Regional High School

Motherboard Circuit Wiring

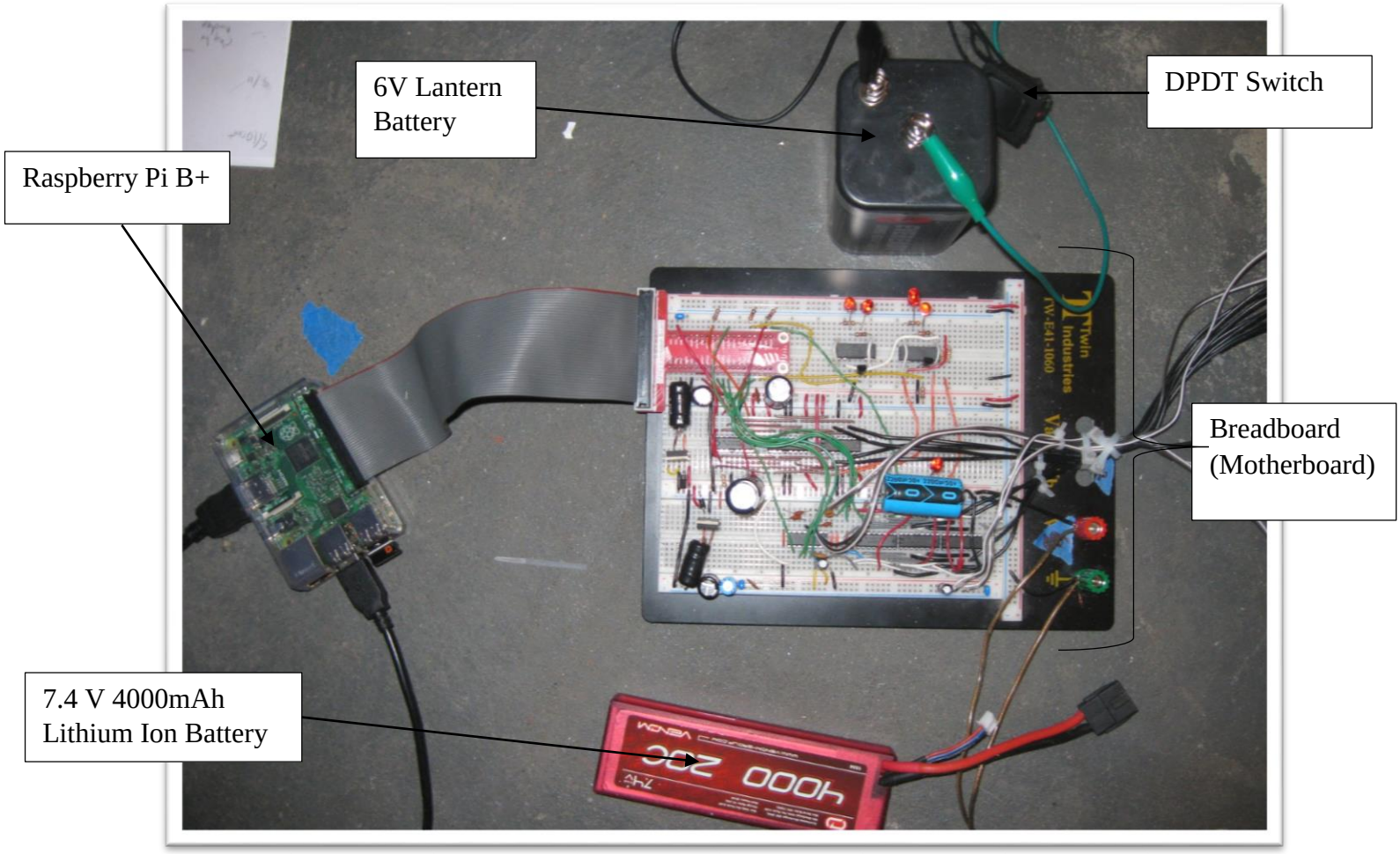

Figure 1 - the Motherboard

(**Figure 1**) The two energy sources are a 4000mAh, 7.4V Li-ion battery (red) and a 6V lantern battery (black). The Raspberry Pi and motherboard setup will read the potentiometers on both arms and, based on their positions, control the speed of the VEX motors with multiple PWM signals and the position of the stepper motor with timed pulses (simple proportional-control). Note the DPDT switch in the top left of the picture (small black blob with wires attached). The DPDT switch, along with the 6V battery, control the claw (VEX motor 5 (**Figure 4**)) independent of the breadboard (motherboard) and potentiometers (**Figure 2**, **Figure 3**, and **Figure 4**).

Master Arm (control module)

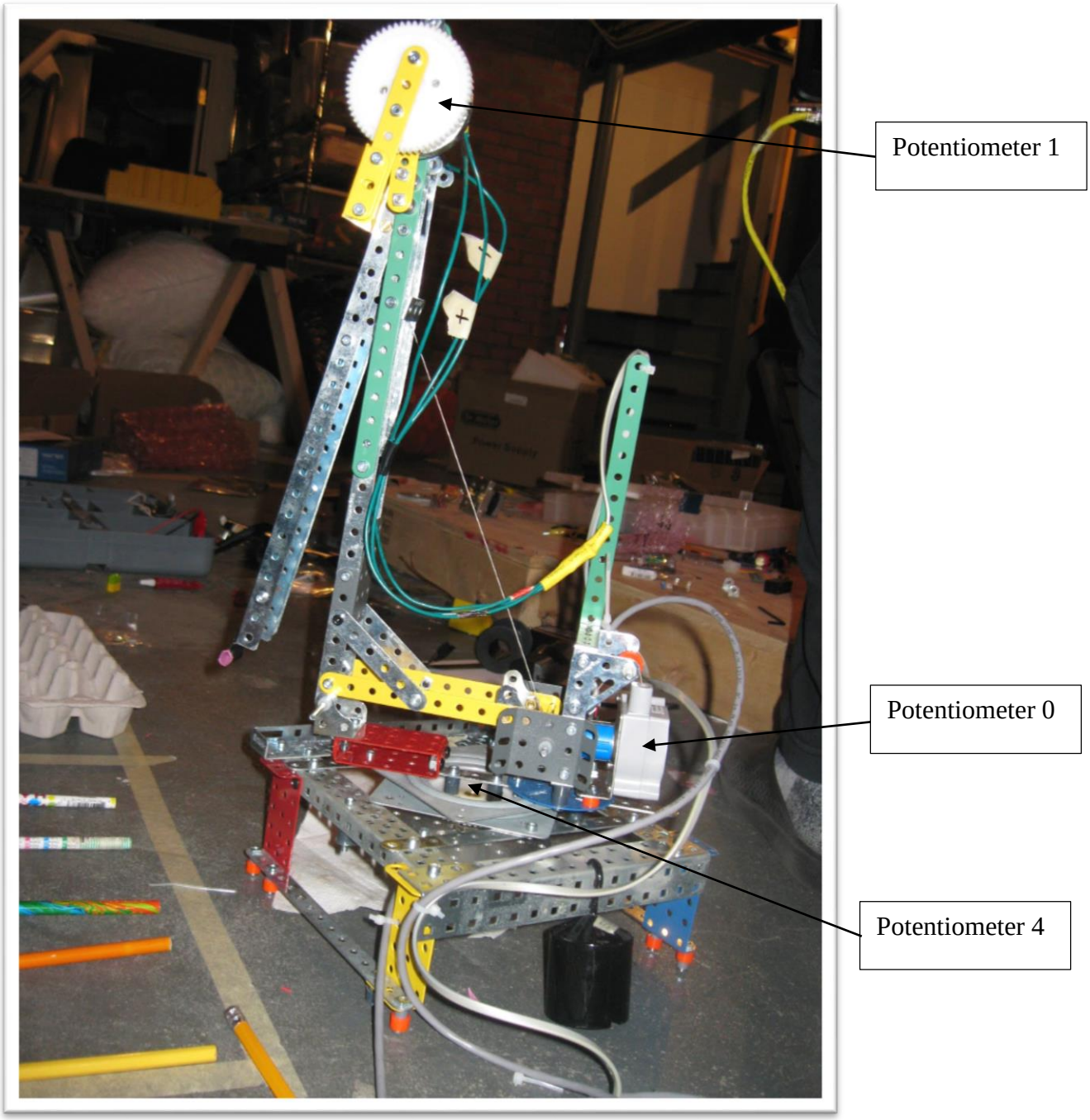

Figure 2 - the Master Arm

Master Arm Potentiometer Numbering

(**Figure 2**) Potentiometer **(0)** is on the master shoulder (grey box on the rotating base with a grey wire). Potentiometer **(1)** is on the master elbow (large, white gear with green wires attached on the joint itself). Potentiometer **(4)** is on the master base (white round piece visible through center of turntable).

Slave Arm

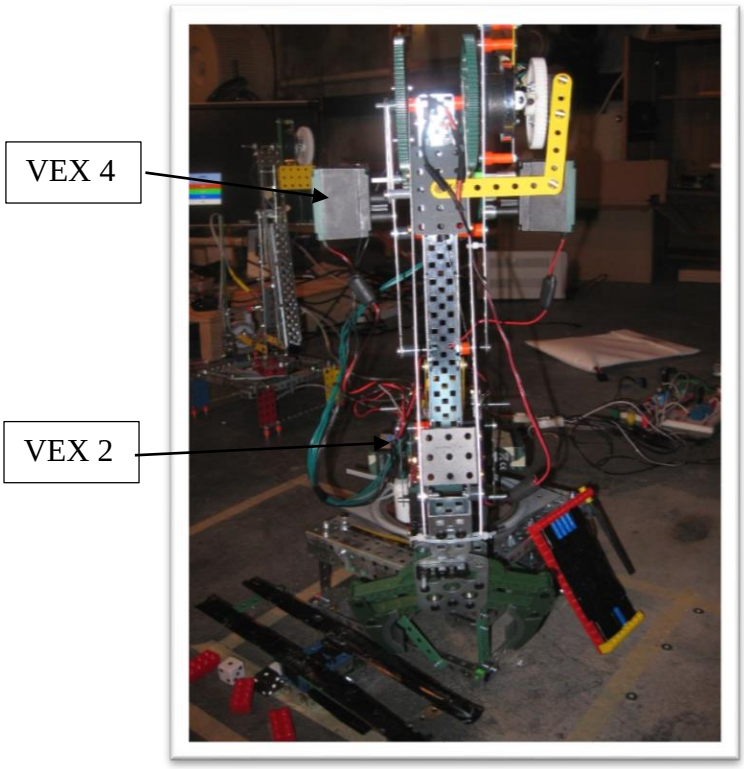

Figure 3 – the Slave Arm (front)

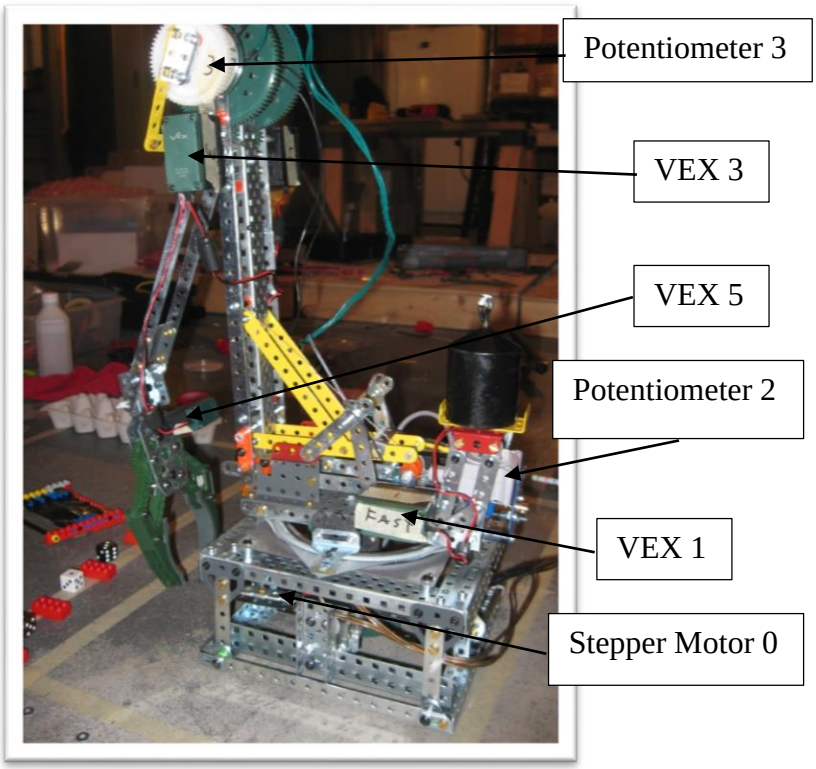

Figure 4 – The Slave Arm (left)

Slave Arm Potentiometer Numbering

Potentiometer **(2)** is located on the slave shoulder (grey box under the large black weights on the back of the arm with a grey wire protruding) in **Figure 4**. Potentiometer **(3)** is located on the elbow (white gear and black box on elbow joint with green wires) in **Figure 4**.

Slave Motor numbering

The stepper motor **(0)** (grey and black rectangular prism driving the base) is partly visible on left bottom of the slave arm in **Figure 4**.
The VEX motor 393's **(1-5)** (2 per joint, numbered on diagram and one on the claw) are (**2, 4)** in **Figure 3** and (**1, 3, 5)** in **Figure 4**.

Operating Description

Robot Reaction to Control Input

The robot arm moves by controlling individual joints by sending a PWM signal through the corresponding controlling DC Vex motors, except for the base which is controlled by a stepper motor with a series of ordered electric pulses with a driver chip. There are also potentiometers on each joint that gives a value based on the joint's radial displacement. The slave robot directly interacts with field elements. However, the master robot is a different build that lies outside the field and is directly controlled by the first human's hand.

Each corresponding error (displacement) in joint position between the master and the slave robot is read and returned to the Raspberry Pi computer by a SPI ADC hardware differential bus. The Raspberry Pi minimizes these errors by driving the DC Vex motors or the stepper motor on the slave arm to match the positions of the joints on the master arm. Motor 0 is controlled by potentiometer 4, motors 1 and 2, which are wired together and control the shoulder, are controlled by potentiometers 0 and 1, and motors 3 and 4, which drive the elbow, are controlled by comparing potentiometers 2 and 3. The exception to motor control is the motorized end effecter claw, which is directly controlled by a double-pull double-throw toggle switch (controlled by the second human) and its own set of batteries (6V lantern). It is important to note that the slave robot does not have a potentiometer for reading base rotations. Instead, a virtual sensor is coded to run with the stepper motor to provide new position readings when the stepper motor moves.

The Raspberry Pi is the RPi model 2 B. It runs on a distribution of Linux called Raspbian. The RPi runs code written in the Python Programming Language and uses a default Python GPIO library to control the GPIO pins on the RPi's hardware. The SPI library is also used to control the SPI hardware pins for reading all the potentiometer values on the slave and master robot.

Tentative plan of movement/Scoring Strategy

Initially, the arm will deploy a passive end effector (black square propped against side in photo) to the North Zone and move North carton east, then move the West carton east (after placing a ping pong ball in a "B"-cell in the West carton) as well to give room for the pencils to stay in the North zone. During the first minute of the competition, the user will use the master

controller arm to control the slave arm in an attempt to drop all the ping-pong balls from the West side of the field to the egg carton cells labeled with a “B”. Each ping-pong ball will be grabbed by a motorized end effecter claw, moved to a small height above the desired cell, and finally dropped into the target cell. Then, during the second minute and part of the third minute of the competition, the user will control the robot to pick up all the Lego pieces in the front of the robot and drop them one-by-one into any arbitrary cell in the North egg carton and the robot will systematically pick up each dice in front of itself, place them onto the passive end effector with the 1 up, and finally flip over the passive end effector with the dice stuck to it. After this, the arm will nudge the rightmost pencil inwards and use its second passive end effector to grab them all and place them on the West goal.

Software

arm_run_diff.py

This code is run by the Raspberry Pi as a sudo python command put in the /etc/rc.local file to run upon startup of the RPi. This code calculates all the potentiometer differences with the SPI interface and then controls each motor group in the slave arm with the GPIO interface. It also uses defined stepper motor pulses to drive the stepper motor. For organizational purposes, it uses code definitions from our own full_lib.py library defined in the section after this.

```
#full robot arm code

from full_lib import *
import time

EL_PIN_FORWARD = 20
EL_PIN_BACKWARD = 25
EL_BRAKE_PIN = 12

#replace later
SH_PIN_FORWARD = 17
SH_PIN_BACKWARD = 27
SH_BRAKE_PIN = 19


PWM_FREQ = 1000
DUTY_FORWARD = 100
DUTY_BACKWARD = 100


#we correct these to true values later
EL_IN_CHANNEL_MASTER = 2
EL_IN_CHANNEL_SLAVE = 3
SH_IN_CHANNEL_MASTER = 0
SH_IN_CHANNEL_SLAVE = 1

IOinit(IO_BCM) #initializes to BCM mode
```

```
el = IOUnit(EL_IN_CHANNEL_MASTER, EL_IN_CHANNEL_SLAVE, EL_PIN_FORWARD,
EL_PIN_BACKWARD, EL_BRAKE_PIN, PWM_FREQ, DUTY_FORWARD, DUTY_BACKWARD)
#no input, backward and forward out enabled, brake enabled, backward and
forward PWM enabled

sh = IOUnit(SH_IN_CHANNEL_MASTER, SH_IN_CHANNEL_SLAVE, SH_PIN_FORWARD,
SH_PIN_BACKWARD, SH_BRAKE_PIN, PWM_FREQ, DUTY_FORWARD, DUTY_BACKWARD)
#no input, backward and forward out enabled, brake enabled, backward and
forward PWM enabled



def setTolSlope():
  el._tolerance = 8
  el._tol_max = 108
  el._slope = 100/(el._tol_max - el._tolerance)
  el._intercept = -el._tolerance*el._slope

  sh._tolerance = 2
  sh._tol_max = 42
  sh._slope = 100/(sh._tol_max - sh._tolerance)
  sh._intercept = -sh._tolerance*sh._slope



def getOffset():
  for x in range (0, 10):
    sh._offset += sh.diff_readadc()
    el._offset += el.diff_readadc()
  sh._offset = round(sh._offset/10, 3)
  el._offset = round(el._offset/10, 3)

def updateElShPots():
  el._potValSlave = el.inADCSlave()
  el._diff = el.diff_readadc() - el._offset
  sh._potValSlave = sh.inADCSlave()
  sh._diff = sh.diff_readadc()- sh._offset

setTolSlope()

el.pwmFStart()
el.pwmBStart()
sh.pwmFStart()
sh.pwmBStart()


getOffset()


#==============STEPPER=====================
BASE_IN_CHANNEL_MASTER = 4
base = IOUnit(BASE_IN_CHANNEL_MASTER)
```

```
COIL_A1 = 5
COIL_A2 = 6
COIL_B1 = 26
COIL_B2 = 13

io.setup(COIL_A1, io.OUT)
io.setup(COIL_A2, io.OUT)
io.setup(COIL_B1, io.OUT)
io.setup(COIL_B2, io.OUT)

STEP = [[1, 0, 1, 0],
        [0, 1, 1, 0],
        [0, 1, 0, 1],
        [1, 0, 0, 1]]

stepperPos = 0
for e in range(0, 5):
  stepperPos += 1.4*base.inavgADCMaster()
stepperPos/=5
stepperPos = int(stepperPos)
basepot = stepperPos


speed = 0
accel = 20
max_speed = 100
delay = 0.5

def step(direction):
    global stepperPos
    if direction == 0:
        io.output(COIL_A1, 0)
        io.output(COIL_A2, 0)
        io.output(COIL_B1, 0)
        io.output(COIL_B2, 0)
        print('release')
    elif direction == -1 or 1:
        stepperPos += direction
        io.output(COIL_A1, STEP[stepperPos%4][0])
        io.output(COIL_A2, STEP[stepperPos%4][1])
        io.output(COIL_B1, STEP[stepperPos%4][2])
        io.output(COIL_B2, STEP[stepperPos%4][3])
    else:
        print('error')

def stepperspeed(old_speed):
  global basepot
  global error
  global speed
  basepot = (1.4*base.inavgADCMaster())/3.0 + basepot/3.0*2
  error = round(stepperPos - basepot, 3)
  if(error > 2):
    speed += accel
  elif(error < -2):
```

```
    speed -= accel
  else:
    speed = 0
  if(speed < -max_speed):
    speed = -max_speed
  elif(speed > max_speed):
    speed = max_speed


print (sh._offset, el._offset)

stringFormat = "SH_M %4.2f SH_S %4.2f SH_D %4.2f || EL_M %4.2f EL_S %4.2f
EL_D %4.2f"


oldTime = time.time()


def run_main():
  try:
    global oldTime
    global speed
    global basepot
    global oldLoopTime
    oldLoopTime=0
    while 1:
      newTime = time.time()
      timeElapsed = newTime - oldTime
      #print((newTime-oldLoopTime)*1000)
      oldLoopTime=newTime


      updateElShPots()

      #=============STEPPER===============

      if(speed!=0):
        delay = 1/abs(speed)
        if(timeElapsed >= delay):
          oldTime = newTime
          step(int(-speed/abs(speed)))
          stepperspeed(speed)
        else:
          pass
      else:
        stepperspeed(speed)


      #print(basepot, error, stepperPos, speed)
      #print(stringFormat %
(sh._potValMaster,sh._potValSlave,sh._diff,el._potValMaster,el._potValSlav
e,el._diff))

      if (el._diff > el._tolerance):
```

```
  el.brakeoff()
      #forward
  el.pwmBChangeDutyCycle(0)
  #print('EL FORWARD')


  if(el._diff>el._tol_max):#max
    el.pwmFChangeDutyCycle(100)
  else:
    el.pwmFChangeDutyCycle(el.dutyFunction())

elif(el._diff < -1*el._tolerance):
      #backward
  #print('EL BACKWARD')
  el.brakeoff()
  el.pwmFChangeDutyCycle(0)

  if(el._diff<-1*el._tol_max):
    el.pwmBChangeDutyCycle(100)
  else:
    el.pwmBChangeDutyCycle(el.dutyFunction())
else:
      #still
  #print('EL STILL')
  el.brakeon()
  pass

if (sh._diff > sh._tolerance):
  sh.brakeoff()
      #forward
  sh.pwmBChangeDutyCycle(0)
  #print('sh FORWARD')


  if(sh._diff>sh._tol_max):#max
    sh.pwmFChangeDutyCycle(100)
  else:
    sh.pwmFChangeDutyCycle(sh.dutyFunction())

elif(sh._diff < -1*sh._tolerance):
      #backward
  #print('sh BACKWARD')
  sh.brakeoff()
  sh.pwmFChangeDutyCycle(0)

  if(sh._diff<-1*sh._tol_max):
    sh.pwmBChangeDutyCycle(100)
  else:
    sh.pwmBChangeDutyCycle(sh.dutyFunction())
else:
      #still
  #print('sh STILL')
  sh.brakeon()
  pass
```

```
  except KeyboardInterrupt:
    el.pwmFStop()
    el.pwmBStop()
    sh.pwmFStop()
    sh.pwmBStop()
    IOquit()

run_main()
```

full_lib.py

The full_lib.py file is located in the same scope as the arm_run_diff.py file, and provides function and class definitions for the SPI, GPIO, and the custom generic IOUnit class for input and output functions for all types, such as PWM, except stepper motor functionality. One IOUnit object is made to be assigned to a basic motor/sensor joint group, such as the shoulder, or elbow joints, excluding the base.

```
#IO lib for Raspberry Pi in Robot Arm, Science Olympiad, December 2015

#Authors: Mayank Mali and Peter Wilson
#Description: Uses hardware-fixed spi interface pins, so no GPIO imports
are required

import spidev
import RPi.GPIO as io

#remember to setup SPI PINS
spi = spidev.SpiDev()   #make spi object
spi.open(0,0)    #open spi port 0 on device 0
spi.max_speed_hz=(3000000) #set SPI speed

#io modes
IO_BCM = 0b101
IO_BOARD = 0b011

def IOinit(mode):      #IO_BCM or IO_BOARD
    if(mode==IO_BCM):
        io.setmode(io.BCM)
    elif(mode==IO_BOARD):
        io.setmode(io.BOARD)
def IOquit():
    io.cleanup()

# read SPI data from MCP3008 chip, 8 possible channels (0 thru 7)
def readadc(channel):
    adc = spi.xfer2([12+((6&channel)>>1),(1&channel)<<7,0])
    data = ((adc[1]&15) << 8) + adc[2]
```

```
    return data

def readadc_diff(channel):
    adc = spi.xfer2([8+((6&channel)>>1),(1&channel)<<7,0])
    data = ((adc[1]&31) << 8) + adc[2]
    return data


def avg_readadc(adcnum, times=5):
    total = 0
    for x in range (0, times):
        total += readadc(adcnum)
    return total*1.0/times

def avg_diff_readadc(adcnum, times=5):
    total = 0
    for x in range (0, times):
        total += readadc_diff(adcnum)
    data = total*1.0/times
    if data > 4096:
        data = data-8192
    return data

def rolling_readadc(adcnum, previous_value, times=5):
    return previous_value/2 + avg_readadc(adcnum, times)/2




class IOUnit:
    _inChannelMaster = None
    _inChannelSlave = None
    _outF = None #GPIO pin
    _outB = None #GPIO pin
    _brake = None #GPIO pin
    _pwmF=None
    _dutyF = None
    _pwmB=None
    _dutyB = None
    _potValMaster = 0
    _potValSlave = 0
    _offset = 0 #initial difference between the slave and the master
    _tolerance = 0 #bound on difference, less than this -> no movement
    _tol_max = 0 #bound on pot difference, less than this -> PWM movement,
mofre than this -> full power
    _slope = 0 #for PWM accel/decel function
    _intercept = 0 #for PWM accel/decel function

    def __init__(self, inChannelMaster=None, inChannelSlave=None,
outF=None, outB=None, brake=None, freq=None, dutyF=None, dutyB=None):
        if((inChannelMaster!=None) | (inChannelSlave!=None)):
            self._inChannelMaster = inChannelMaster
            self._inChannelSlave = inChannelSlave
```

```
            #spi pins already setup by this point to read
        if(outF!=None):
            self._outF = outF
            io.setup(self._outF, io.OUT)
        if(outB!=None):
            self._outB = outB
            io.setup(self._outB, io.OUT)
        if(brake!=None):
            self._brake = brake
            io.setup(self._brake, io.OUT)
        if((freq!=None) & (dutyF!=None)& (dutyB!=None)):
            self._pwmF = io.PWM(self._outF, freq)
            self._dutyF = dutyF
            self._pwmB = io.PWM(self._outB, freq)
            self._dutyB = dutyB

    #for out
    def outF(self, state):
        io.output(self._outF, state)
    def outB(self, state):
        io.output(self._outB, state)

    #for in
    def inADCMaster(self):
        return readadc(self._inChannelMaster)
    def inADCSlave(self):
        return readadc(self._inChannelSlave)
    def diff_readadc(self):
        return avg_diff_readadc(self._inChannelMaster)
    def inavgADCMaster(self):
        return avg_readadc(self._inChannelMaster)
    def inavgADCSlave(self):
        return avg_readadc(self._inChannelSlave)
    def rolling_readadc_Master(self, previous_value):
        return rolling_readadc(self._inChannelMaster, previous_value)
    def rolling_readadc_Slave(self, previous_value):
        return rolling_readadc(self._inChannelSlave, previous_value)
    def pwmFStart(self):
        self._pwmF.start(0)
    def pwmBStart(self):
        self._pwmB.start(0)
    def pwmFChangeDutyCycle(self, duty):

        self._dutyF = duty
        self._pwmF.ChangeDutyCycle(duty)

        return 1
    def pwmBChangeDutyCycle(self, duty):

        self._dutyB = duty
        self._pwmB.ChangeDutyCycle(duty)

        return 1
    def dutyFunction(self):
```

```
        return self._slope*abs(self._diff)+self._intercept
    def brakeon(self):

        self._pwmF.ChangeDutyCycle(0)
        self._pwmB.ChangeDutyCycle(0)

        io.output(self._brake, 1)

        return 1
    def brakeoff(self):
        io.output(self._brake, 0)
        return 1
    def pwmFStop(self):
        self._pwmF.stop()
    def pwmBStop(self):
        self._pwmB.stop()
#end
```

Written Practice Log

Run Number	Time [minutes : seconds]	Score [points]	Number of die with 6's facing up
1	3:00	102	3
2	2:57	110	4
3	3:00	103	3
4	2:57	105	3
5	3:00	92	3
6	2:53	105	4
7	2:53	110	4
8	2:30	110	4
9	2:43	110	4
10	2:50	104	3

Table 1 – Run Summaries

National Score

The National Science Olympiad competition for 2016 was held at the University of Wisconsin, Stout. The score for Acton-Boxborough Regional High School's Robot Arm was a perfect 110

points, and the robot received 6th place for the ABRHS (**Figure 5** below). The perfect score was achieved under the three-minute limit. No game elements (ping-pong balls, Lego pieces, dice, or pencils) slipped or were positioned incorrectly or repositioned after a mistake with respect to the Tentative Plan of Movement/Scoring Strategies section described before. All ping-pong balls were in the correct egg-carton cells, all Lego pieces were in the north egg carton, all pencils were supported by weight on the west egg carton, and all dice were positioned with the six-side facing up.

National Tournament: Division C

	School	Anatomy & Physiology	Astronomy	Cell Biology	Chemistry Lab	Disease Detective	Dynamic Planet	Experimental Design	Forensics	Fossils	Game On	GeoLogic Mapping	Green Generation	Hydrogeology	Invasive Species	Its About Time	Protein Modeling	Wind Power	Write It, Do It	Air Trajectory	Bridge Building	Electric Vehicle	Robot Arm	Wright Stuff	Code Busters	Trial	Mystery Design	Trial	Remote Sensing	Trial	Total	Place
C04	Mira Loma H.S. (CA)	5	5	3	22	12	6	3	28	7	32	3	5	14	11	4	5	8	16	4	7	30	1	3	44		6		3		234	1
C11	Harriton H.S. (PA)	2	3	18	4	10	10	27	1	17	9	29	3	5	1	1	3	14	23	7	14	6	2	26	5		39		9		235	2
C20	Seven Lakes H.S. (TX)	11	1	7	2	7	1	33	7	4	5	1	16	12	12	42	4	16	4	16	8	55	7	1	19		8		22		272	3
C03	Troy H.S. (CA)	16	6	5	3	2	25	8	13	18	8	5	2	3	14	2	14	12	53	8	5	3	5	49	6		1		1		279	4
C14	Centerville H.S. (OH)	10	20	10	11	25	3	22	8	5	3	24	20	1	16	3	19	21	3	17	17	13	4	20	12		16		11		295	5
C09	Adlai E. Stevenson H.S. (IL)	15	19	22	9	6	30	6	3	13	31	12	11	2	3	29	7	15	7	19	1	8	15	14	31		2		13		297	6
C13	Solon H.S. (OH)	19	9	19	21	18	13	15	29	15	24	2	1	16	8	13	6	6	18	14	18	4	11	2	4		35		15		301	7
C07	Northville H.S. (MI)	7	24	8	8	4	12	4	5	16	19	23	13	10	6	5	32	19	9	30	12	20	20	4	—		—		—		310	8
C19	Liberal Arts and Science Academy (TX)	12	14	15	29	28	9	20	11	3	14	9	8	9	5	22	11	17	20	11	2	11	27	12	22		24		20		319	9
C28	Mounds View H.S. (MN)	21	12	2	6	3	22	17	14	11	29	14	14	27	7	8	13	1	48	9	15	1	21	17	30		38		4		332	10
C30	West Windsor-Plainsboro H.S. North (NJ)	1	8	9	19	13	17	13	22	12	1	15	9	18	2	9	1	30	32	54	20	22	25	32	3		23		2		384	11
C37	Acton-Boxborough Regional H.S. (MA)	20	21	6	1	16	21	37	2	14	23	8	46	32	9	15	27	23	19	3	23	12	6	34	24		42		6		418	12
C12	Bayard Rustin H.S. (PA)	33	11	16	12	40	5	18	21	2	15	21	10	40	48	12	17	7	37	6	34	10	10	16	38		20		17		441	13
C10	New Trier H.S. (IL)	14	2	23	23	9	36	10	26	10	26	10	6	29	20	11	24	9	42	26	4	19	12	60	13		61		18		451	14
C16	Chattahoochee H.S. (GA)	28	17	26	32	32	23	23	57	6	7	26	22	6	38	6	12	3	54	12	9	14	9	31	8		—		14		493	15
C01	Columbia H.S. (NY)	8	25	12	36	1	7	7	15	19	10	17	17	45	25	18	25	31	2	23	40	43	55	24	9		21		10		505	16
C35	Fossil Ridge H.S. (CO)	17	43	30	42	36	37	5	12	28	12	11	15	17	17	24	20	13	8	20	42	5	29	30	43		9		19		513	17
C23	Carmel H.S. (IN)	23	38	34	13	11	29	48	4	21	38	19	4	11	32	14	10	34	1	41	33	35	8	13	17		34		30		514	18
C42	Iolani School (HI)	3	16	13	26	23	27	28	43	1	4	4	45	7	28	30	16	22	12	13	39	34	53	29	1		4		—		516	19
C17	Boca Raton Community H.S. (FL)	13	10	17	39	55	16	32	9	38	36	37	37	25	27	19	2	2	30	56	6	2	3	23	40		22		21		534	20
C21	Ladue Horton Watkins H.S. (MO)	18	7	20	24	20	28	43	16	9	41	7	40	42	21	21	36	35	17	2	35	44	16	7	32		43		39		549	21
C24	Menomonie H.S. (WI)	4	26	25	51	52	8	29	19	25	54	18	19	33	31	23	29	5	11	5	19	26	23	15	10		15		24		550	22
C08	Troy H.S. (MI)	9	41	4	7	41	14	44	27	43	33	6	36	8	37	17	31	29	38	39	11	9	24	5	21		7		61		553	23

Figure 5 – Concatenated national ranking for the 2016 National Science Olympiad taken from https://www.soinc.org/sites/default/files/uploaded_files/2016_FInal_Results_Div_C.PDF