#!/usr/bin/python

#Working SRC 2017 Code for 4 Servo Control

**#2 Drive wheels, 1 Gripper and 1 Arm**

#Revised 11.29.16 - Kevin Pace

**import sys # System-specific parameters and functions**

# This module provides access to some variables used or maintained by the

# interpreter and to functions that interact strongly with the interpreter. It is always available.

**import threading # Thread-based parallelism**

# This module constructs higher-level threading interfaces on top of

# the lower level [\_thread](https://docs.python.org/3/library/_thread.html#module-_thread) module. See also the [queue](https://docs.python.org/3/library/queue.html#module-queue) module.

**from Adafruit\_PWM\_Servo\_Driver import PWM**

**import time # Time access and conversions**

#This module provides various time-related functions. For related functionality, see also

# the datetime and calendar modules.

**try: # Errors and Exceptions try to import the gpio libraries (need to download) and throw an exception if there is an error**

**import RPi.GPIO as gpio**

**except RuntimeError:**

**print "error importing the gpio library which is probably because you need to run this program with sudo"**

# Until now error messages haven’t been more than mentioned, but if you have tried out the

# examples you have probably seen some. There are (at least) two distinguishable kinds of

# errors: syntax errors and exceptions.

## # 8.3. Handling Exceptions

# The try statement works as follows.

# First, the try clause (the statement(s) between the try and except keywords) is executed.

# If no exception occurs, the except clause is skipped and execution of the try statement is finished.

# If an exception occurs during execution of the try clause, the rest of the clause is skipped. Then if

# its type matches the exception named after the except keyword, the except clause is executed, and

# then execution continues after the try statement.

# If an exception occurs which does not match the exception named in the except clause, it is passed on

# to outer try statements; if no handler is found, it is an unhandled exception and execution stops with a

# message as shown above.

# ===========================================================================

# Example Code

# ===========================================================================

**# Initialise the PWM device using the default address**

**pwm = PWM(0x40) # The I2C base address for each board is 0x40**

# Note if you'd like more debug output you can instead run:

#pwm = PWM(0x40, debug=True)

**servoMin = 150 # Min pulse length out of 4096 # 12 bits of resolution, only standard servo**

**servoMax = 600 # Max pulse length out of 4096 # 12 bits of resolution, only standard servo**

**servoZero = (servoMax-servoMin)/2 + servoMin # only standard servo**

**servoLeft = 0 # Pin number 0 Red**

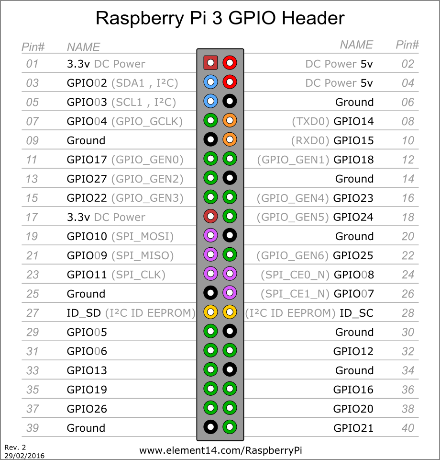
**servoRight = 1 # Pin number 1 Green**

**servoLift = 2 # Pin number 2**

**servoGrip = 3 # Pin number 3 Orange**

#gpio setup here # *A module to control Raspberry Pi GPIO*

*# channels*

#outPin = 38 # GPIO20

**inPin = 40 # assign the gpio pins to variables # GPIO21**

**inPin2 = 38 # GPIO20**

#define inPin3 here below

""" inPin3 = """

**gpio.setmode(gpio.BOARD)**

# The **GPIO**.**BOARD** option specifies that you are

# referring to the pins by the number of the pin on the plug.

# gpio.setup(outPin, gpio.OUT, initial=gpio.HIGH) #set the **output pin to a permanent high, this will go directly into the input pin once the button is pressed**

# GPIO.setup(Port\_or\_pin, GPIO.IN)

#…changing Port\_or\_pin to the number of the GPIO port or pin you want to use. I’m going to use the

# BCM GPIO numbering and port GPIO25, so it becomes…

# You can also set the initial value of the output at the time of setting up the port

# with initial=x optional extra argument…

# GPIO.setup(port\_or\_pin, GPIO.OUT, initial=1)  
# or  
# GPIO.setup(port\_or\_pin, GPIO.OUT, initial=0)

# And that’s really (almost) all there is to it. You can use GPIO.HIGH or GPIO.

# LOW and True or False

**gpio.setup(inPin, gpio.IN) #setup pin 21 as input**

**gpio.setup(inPin2, gpio.IN)**

#uncomment below to set the inPin3 to GPIO input

""" gpio.setup(inPin3, gpio.IN) """

# GPIO.setup(channel, GPIO.IN)   # input channel

  # or

# GPIO.setup(channel, GPIO.OUT)  # output channel

**def setServoPulse(channel, pulse):**

**pulseLength = 1000000 # 1,000,000 us per second = 1 Million**

**pulseLength /= 60 # 60 Hz [object] = [object] or 60**

**print "%d us per period" % pulseLength**

**pulseLength /= 4096 # 12 bits of resolution 60/4096**

# [object] = [object] \* [another\_object] 60Hz / 4096 12 bits of resolution

**print "%d us per bit" % pulseLength**

**pulse \*= 1000 # pulse = 1000**

**pulse /= pulseLength # pulse = pulseLength # 12 bits of resolution 60/4096**

# That is 110/4096\*20000 (50 Hz) so 537 µs

**pwm.setPWM(channel, 0, pulse)**

**. pwm.setPWMFreq(60) # Set frequency to 60 Hz**

# Simple Servo Calls

**def ServoClockwise(channel):**

**pwm.setPWM(channel, 0, servoMin)**

# Arguments

**# channel**: The channel that should be updated with the new values (0..15)

**# on**: The tick (between 0..4095) when the signal should transition from low to high

**# off**:the tick (between 0..4095) when the signal should transition from high to low

# **servoMin** = 150

**def ServoCounterClockwise(channel):**

**pwm.setPWM(channel, 0, servoMax)**

# **servoMax** = 600

**def ServoStop(channel):**

**pwm.setPWM(channel, 0, servoZero)**

#for the turn left and turn right functions, edit the sleep values for back up and turn to get the servo timing correct

**def turnLeft():**

**print "right button pressed!"**

**#back up**

**ServoClockwise(servoLeft) # servoLeft = Pin 0**

**ServoCounterClockwise(servoRight) # servoRight = Pin 1**

**time.sleep(1.2) #edit this**

#turn left

**print "second part of left turn"**

**ServoClockwise(servoLeft) servoLeft = Pin 0**

**ServoClockwise(servoRight) # servoRight = Pin 1**

**time.sleep(0.8) #edit this**

**def turnRight():**

**print "left button pressed"**

**#back up**

**ServoClockwise(servoLeft) servoLeft = Pin 0**

**ServoCounterClockwise(servoRight) # servoRight = Pin 1**

**time.sleep(1.2) #edit this**

**#turn right**

**ServoCounterClockwise(servoLeft) servoLeft = Pin 0**

**ServoCounterClockwise(servoRight) # servoRight = Pin 1**

**time.sleep(0.8) # edit this**

""" Here is the heart of the autonomous mode! """

**def autoMode():**

**print "got here"**

**while True:**

**ServoCounterClockwise(servoLeft)**

**ServoClockwise(servoRight)**

**print gpio.input(inPin), " ", gpio.input(inPin2)**

**if gpio.input(inPin) == 1: #if the button is pressed, back off the wall and turn left**

**print "break button pressed"**

**break**

**elif gpio.input(inPin2) == 1: #where inPin2 should be the button on the left side of the robot**

**turnRight() #function I defined above**

#uncomment below **(“” block comment)** but make sure to define inPin3 as whichever GPIO pin you intend to have the right switch hooked up to

**""" elif gpio.input(inPin3) == 1**

**turnleft() """**

# ULTRASONIC MODE

**"""**

**1) stop at a certain distance measured by the sensor from the maze wall**

**2) pause**

**3) look left, pause, then measure the distance and store that distance in a temporary variable**

**4) look right, pause, measure the dist and store in a temp variable**

**5) look straight**

**6) compare the temporary variables and if the rightDistance > leftDistance, turn right. If leftDistance >= rightDistance, turn left.**

**7) execute the turn and continue movement in that direction.**

**8) repeat or loop these steps until user breaks out of the autonomous loop**

**"""**

# Keyboard stuff

**import Tkinter as tk** # The **[Tkinter](https://docs.python.org/2/library/tkinter.html" \l "module-Tkinter" \o "Tkinter: Interface to Tcl/Tk for graphical user interfaces)** module (“Tk interface”) is the standard Python interface to the # Tk GUI toolkit. Both Tk and **[Tkinter](https://docs.python.org/2/library/tkinter.html" \l "module-Tkinter" \o "Tkinter: Interface to Tcl/Tk for graphical user interfaces)** are available on most Unix platforms, as well as on

# Windows systems. (Tk itself is not part of Python; it is maintained at ActiveState.)

**class MyFrame(tk.Frame):**

**def \_\_init\_\_(self, master):**

**tk.Frame.\_\_init\_\_(self, master)**

**# method call counter**

**self.pack()**

**self.afterId = None**

**root.bind('<KeyPress>', self.key\_press)**

**root.bind('<KeyRelease>', self.key\_release)**

**def key\_press(self, event):**

**if self.afterId != None:**

**self.after\_cancel( self.afterId )**

**self.afterId = None**

**else:**

**print 'key pressed %s' % event.char**

**if event.char == "w":**

**text.insert('end', ' FORWARD ')**

**ServoCounterClockwise(servoLeft)**

**ServoClockwise(servoRight)**

**elif event.char == "s":**

**text.insert('end', ' RIGHT\_TURN ')**

**ServoCounterClockwise(servoLeft)**

**ServoCounterClockwise(servoRight)**

**elif event.char == "K":**

**text.insert('end', ' Quit ')**

**pwm.setPWM(0, 0, servoZero)**

**root.destroy()**

**elif event.char == "z":**

**text.insert('end', ' BACKWARD ')**

**ServoClockwise(servoLeft)**

**ServoCounterClockwise(servoRight)**

**elif event.char == "a":**

**text.insert('end', ' LEFT\_TURN ')**

**ServoClockwise(servoLeft)**

**ServoClockwise(servoRight)**

**elif event.char == "u":**

**text.insert('end', ' UP ')**

**ServoClockwise(servoLift)**

**elif event.char == "d":**

**text.insert('end', ' DOWN ')**

**ServoCounterClockwise(servoLift)**

**elif event.char == "c":**

**text.insert('end', ' CLOSE\_GRIP ')**

**ServoClockwise(servoGrip)**

**elif event.char == "o":**

**text.insert('end', ' OPEN\_GRIP ')**

**ServoCounterClockwise(servoGrip)**

**elif event.char == "l":**

**text.insert('end', ' Stop Auto Mode ')**

**autoMode() #refer to the defined function above**

**def key\_release(self, event):**

**self.afterId = self.after\_idle( self.process\_release, event )**

**def process\_release(self, event):**

**ServoStop(servoLeft)**

**ServoStop(servoRight)**

**ServoStop(servoLift)**

**ServoStop(servoGrip)**

**print 'key release %s' % event.char**

**self.afterId = None**

# Program

#pwm.setPWMFreq(60) # Set frequency to 60 Hz moved up

**root = tk.Tk()**

**root.geometry('800x600')**

**root.attributes('-fullscreen', False)**

**text = tk.Text(root, background='black', foreground='white', font=('Comic Sans MS', 12))**

**text.pack()**

**text.insert('end', 'STEM TRI-Fecta 2017')**

**app1 = MyFrame(root)**

**root.mainloop()**

**print("done")**