```
"""
Created on Fri Feb 16 16:53:17 2018

@privatesection - Stuff in this file doesn't need to be Doxygen-ed

@author: JasonGrillo
"""

import pyb
import micropython
import gc

import encoder_task_func
import IMU_task_func
import motor_task_func
import nerf_task_func
import turret_hub_task_func

import cotask
import task_share



# Allocate memory so that exceptions raised in interrupt service routines
# generate useful diagnostic printouts
micropython.alloc_emergency_exception_buf (100)


# ============================================================================
# ======================== Run the Turret Code ===============================
# ============================================================================

if __name__ == "__main__":

    ############################################################################
    ############################## VARIABLES ###################################
    ############################################################################

    #Pan Coordinates Queue is used to deliver target pan encoder value to
    #Pan Motor Task from Turret Hub Task
    pan_coords = task_share.Queue ('f', 2, thread_protect = False,
                                   overwrite = False, name = "Pan_Coords"

    #Tilt Coordinates Queue is used to deliver target tilt IMU value to
    #Tilt Motor Task from Turret Hub Task
    tilt_coords = task_share.Queue ('f', 2, thread_protect = False,
```

```
                                    overwrite = False, name = "Tilt_Coord

#Pan Position Share is used to deliver current encoder value to
#to the Turret Hub and Pan Motor tasks from the Encoder Task
pan_position = task_share.Share ('f', thread_protect = False,
                                     name = "Pan_Position")

#Tilt Angle Share is used to deliver current IMU pitch value to
#to the Turret Hub and Tilt Motor tasks from the Encoder Task
tilt_angle = task_share.Share ('f', thread_protect = False,
                                    name = "Tilt_Position")

#Share Sent from Turret Hub Task to Nerf Gun Task to Start Feeding Bu
FEED_BULLETS = task_share.Share('f', thread_protect = False,
                                     name = "Feed_Bullets")

#Share sent from Turret Hub Task to Nerf Gun Task to
WINDUP_GUN = task_share.Share ('f', thread_protect = False,
                                    name = "Windup_Gun")

######################################################################
########################### TASK OBJECTS #############################
######################################################################

pan_encoder = encoder_task_func.Encoder_Task(pan_position, 4,
                                     pyb.Pin.board.PB6, pyb.Pin.board.PB7

tilt_IMU = IMU_task_func.IMU_Task(tilt_angle) #what to put here 0 for

#0.02
pan_motor = motor_task_func.Motor_Task(pan_position,
                                       pan_coords, 3,
                                       pyb.Pin.board.PA10,
                                       pyb.Pin.board.PB4,
                                       pyb.Pin.board.PB5, 0.01, .0125
#1.2
tilt_motor = motor_task_func.Motor_Task(tilt_angle,
                                        tilt_coords, 5,
                                        pyb.Pin.board.PC1,
                                        pyb.Pin.board.PA0,
                                        pyb.Pin.board.PA1, 2.0, 0.75,

turret_hub = turret_hub_task_func.Turret_Hub_Task(pan_position, tilt_
                                                  tilt_coords, FEED_B

nerf_gun = nerf_task_func.Nerf_Task(WINDUP_GUN, FEED_BULLETS, pyb.Pin
```

```
##############################################################
############################## TASKS ##########################
##############################################################

#Turret Hub Timing => Timing: 100 ms, Priority 1 (Lowest)
task0 = cotask.Task (turret_hub.turret_hub_fun, name = 'Task_0', prio
                     period = 100, profile = True, trace = False)
#Pan Encoder => Timing 2 ms, Priority 5(Highest)
task1 = cotask.Task (pan_encoder.enc_fun, name = 'Task_1', priority =
                     period = 2, profile = True, trace = False)
#Tilt IMU => Timing 5 ms (minimum 10 ms, applied 2x SF), Priority 5(H
task2 = cotask.Task (tilt_IMU.IMU_fun, name = 'Task_2', priority = 5,
                     period = 5, profile = True, trace = False)
#Pan Motor => Timing 20 ms, Priority 3 (Medium)
task3 = cotask.Task (pan_motor.mot_fun, name = 'Task_3', priority = 3
                     period = 20, profile = True, trace = False)
#Tilt Motor => Timing 20 ms, Priority 3 (Medium)
task4 = cotask.Task (tilt_motor.mot_fun, name = 'Task_4', priority =
                     period = 20, profile = True, trace = False)
#Nerf Gun => Timing 200 ms, Priority 1 (Lowest)
task5 = cotask.Task (nerf_gun.gun_fun, name = 'Task_5', priority = 1,
                     period = 200, profile = True, trace = False)

cotask.task_list.append (task0)
cotask.task_list.append (task1)
cotask.task_list.append (task2)
cotask.task_list.append (task3)
cotask.task_list.append (task4)
cotask.task_list.append (task5)

# Run the memory garbage collector to ensure memory is as defragmente
# possible before the real-time scheduler is started
gc.collect ()

##############################################################
############################## RUN ############################
##############################################################

# Run the scheduler with the chosen scheduling algorithm
while True:
    cotask.task_list.pri_sched()
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Mar  7 00:00:39 2018

@author: JasonGrillo
"""

import pyb
import micropython

class Turret_Hub_Task:
    ''' This defines the task function method for a nerf turret hub.
    '''

    def __init__(self, pan_position, tilt_angle, pan_coords, tilt_coords,
        ''' Construct a turret hub task function by initilizing any share
            variables and objects
            @param pan_position The shared variable for the pan position
            @param tilt_angle The shared variable for the tilt position
            @param pan_coords The queue of coordinates for the pan axis
            @param tilt_coords The queue of coordinates for the tilt axis
            @param FEED_BULLETS The shared variable flag for the nerf gun
            @param WINDUP_GUN The shared variable flag for the nerf gun m
        '''
        self.pan_position = pan_position
        self.tilt_angle = tilt_angle
        self.pan_coords = pan_coords
        self.tilt_coords = tilt_coords
        self.FEED_BULLETS = FEED_BULLETS
        self.WINDUP_GUN = WINDUP_GUN

        self.TARGET_CMD = False

        self.CALIBRATION_FLG = False

        self.pan_centroids = [0.0, 0.0, 0.0, 0.0, 0.0]
        self.tilt_centroids = [0.0, 0.0, 0.0, 0.0, 0.0]

# ----------------------------------------------------------------------
# ----------------------------------------------------------------------

    def read_GUI(self):
        ''' Reads the serial port for incoming commands and executes the
        '''
        if self.vcp.any():
```

```
            self.GUI_input = float(self.vcp.read(2).decode('UTF-8'))
            self.GUI_Lookup_Table(self.GUI_input)

# ------------------------------------------------------------------------
# ------------------------------------------------------------------------

    def turret_hub_fun(self):
        ''' Defines the task function method for a turret hub object.
        '''
        self.vcp = pyb.USB_VCP ()

        STATE_0 = micropython.const(0)
        STATE_1 = micropython.const(1)
        STATE_2 = micropython.const(2)
        STATE_3 = micropython.const(3)
        STATE_4 = micropython.const(4)
        STATE_5 = micropython.const(5)
        STATE_6 = micropython.const(6)
        STATE_7 = micropython.const(7)
        STATE_8 = micropython.const(8)
        STATE_9 = micropython.const(9)
        STATE_10 = micropython.const(10)
        STATE_11 = micropython.const(11)

        self.state = STATE_0

        self.pan_coords.put(0)
        self.tilt_coords.put(0)

        while True:

            ##################################

            ## STATE 0: CALIBRATE POINT A
            if self.state == STATE_0:
                self.read_GUI()
                yield (self.state)

                if self.CALIBRATION_FLG:
                    # input location A into pan centroids
                    self.calibrate_point(0, self.pan_position.get(), self
                    self.CALIBRATION_FLG = False
                    self.state = STATE_1

            ##################################
```

```
        ## STATE 1: CALIBRATE POINT B
        elif self.state == STATE_1:
            self.read_GUI()
            yield (self.state)

            if self.CALIBRATION_FLG:
                # input location B into pan centroids
                self.calibrate_point(1, self.pan_position.get(), self
                self.CALIBRATION_FLG = False
                self.state = STATE_2

        ##################################

        ## STATE 2: CALIBRATE POINT C
        elif self.state == STATE_2:
            self.read_GUI()
            yield (self.state)

            if self.CALIBRATION_FLG:
                # input location C into pan centroids
                self.calibrate_point(2, self.pan_position.get(), self
                self.CALIBRATION_FLG = False
                self.state = STATE_3

        ##################################

        ## STATE 3: CALIBRATE POINT D
        elif self.state == STATE_3:
            self.read_GUI()
            yield (self.state)

            if self.CALIBRATION_FLG:
                # input location D into pan centroids
                self.calibrate_point(3, self.pan_position.get(), self
                self.CALIBRATION_FLG = False
                self.state = STATE_4

        ##################################

        ## STATE 4: CALIBRATE POINT E & 1
        elif self.state == STATE_4:
            self.read_GUI()
            yield (self.state)

            if self.CALIBRATION_FLG:
                # input location E into pan centroids
```

```
            self.calibrate_point(4, self.pan_position.get(), self
            # input location 1 into tilt centroids
            self.calibrate_point(0, self.pan_position.get(), self
            self.CALIBRATION_FLG = False
            self.state = STATE_5

    ##################################

    ## STATE 5: CALIBRATE POINT 2
    elif self.state == STATE_5:
        self.read_GUI()
        yield (self.state)

        if self.CALIBRATION_FLG:
            # input location 2 into tilt centroids
            self.calibrate_point(1, self.pan_position.get(), self
            self.CALIBRATION_FLG = False
            self.state = STATE_6

    ##################################

    ## STATE 6: CALIBRATE POINT 3
    elif self.state == STATE_6:
        self.read_GUI()
        yield (self.state)

        if self.CALIBRATION_FLG:
            # input location 3 into tilt centroids
            self.calibrate_point(2, self.pan_position.get(), self
            self.CALIBRATION_FLG = False
            self.state = STATE_7

    ##################################

    ## STATE 7: CALIBRATE POINT 4
    elif self.state == STATE_7:
        self.read_GUI()
        yield (self.state)

        if self.CALIBRATION_FLG:
            # input location 4 into tilt centroids
            self.calibrate_point(3, self.pan_position.get(), self
            self.CALIBRATION_FLG = False
            self.state = STATE_8

    ##################################
```

```
## STATE 8: CALIBRATE POINT 5
elif self.state == STATE_8:
    self.read_GUI()
    yield (self.state)

    if self.CALIBRATION_FLG:
        # input location 5 into tilt centroids
        self.calibrate_point(4, self.pan_position.get(), self
        print('Calibration complete.')
        self.state = STATE_9

###################################

## STATE 9: STOPPED, NOT SHOOTING
elif self.state == STATE_9:
    self.read_GUI()
    yield (self.state)

    if self.TARGET_CMD:
        if self.WINDUP_GUN.get():
            self.FEED_BULLETS.put(1)
            self.state = STATE_10
        else:
            print('Windup the Gun!!')

###################################

## STATE 10: MOVING, SHOOTING
elif self.state == STATE_10:
    # clear the target cmd flag for state 9 next time
    self.TARGET_CMD = False
    self.state = STATE_11

###################################

## STATE 11: STOPPED, SHOOTING
elif self.state == STATE_11:
    self.read_GUI()
    yield (self.state)

    if not self.FEED_BULLETS.get():
        self.state = STATE_9

###################################
```

```
            yield (self.state)

# ------------------------------------------------------------------------
# ------------------------------------------------------------------------

    def target_cmd(self, pan, tilt, target_cmd = True):
        ''' Defines what to do when target cmd is entered through the GUI
        self.pan_coords.put(pan)
        self.tilt_coords.put(tilt)
        print(pan)
        print(tilt)
        if target_cmd:
            self.TARGET_CMD = True
        else:
            self.TARGET_CMD = False

# ------------------------------------------------------------------------
# ------------------------------------------------------------------------

    def calibrate_point(self, index, pan_coor, tilt_coor, pan = False, ti
        ''' enters the calibrated point into the proper centroid list.
        @param index The index of the point in the centroid list
        @param pan_coor The pan coordinate of the point
        @param tilt_coor The tilt coordinate of the point
        @param pan Indicate if it's a pan calibration point
        @param tilt Indicate if it's a tilt calibration point
        '''
        if pan:
            self.pan_centroids[index] = pan_coor - 700
            self.pan_coords.put(pan_coor)

        if tilt:
            self.tilt_centroids[index] = tilt_coor + 3.5
            self.tilt_coords.put(tilt_coor)

# ------------------------------------------------------------------------
# ------------------------------------------------------------------------

    def GUI_Lookup_Table(self, command):
        ''' Decodes GUI commands based on a defined list of commands

        GUI Layout:
         _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
        | A1  B1  C1  D1  E1   Wind_on    Up       Calibration  |

        | A2  B2  C2  D2  E2   Feed_on    Down                  |
```

```
        | A3  B3  C3  D3  E3   Wind_off   Left                  |
        | A4  B4  C4  D4  E4   Feed_off   Right                 |
        | A5  B5  C5  D5  E5     Home                           |
         - - - - - - - - - - - - - - - - - - - - - - - - - - - -
         GUI Command Numbers:
         _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
        | 1   6   11  16  21      26      31          36        |
        | 2   7   12  17  22      27      32                    |
        | 3   8   13  18  23      28      33                    |
        | 4   9   14  19  24      29      34                    |
        | 5   10  15  20  25      30                            |
         - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        @param command The incoming GUI command to decode
        '''

# --- A TARGETS ---

       # A1 Target
        if(command == 1):
            self.target_cmd(self.pan_centroids[0], self.tilt_centroids[0]

        # A2 Target
        elif(command == 2):
            self.target_cmd(self.pan_centroids[0], self.tilt_centroids[1]

        # A3 Target
        elif(command == 3):
            self.target_cmd(self.pan_centroids[0], self.tilt_centroids[2]

        # A4 Target
        elif(command == 4):
            self.target_cmd(self.pan_centroids[0], self.tilt_centroids[3]

        # A5 Target
        elif(command == 5):
            self.target_cmd(self.pan_centroids[0], self.tilt_centroids[4]
```

```
# --- B TARGETS ---

        # B1 Target
        elif(command == 6):
            self.target_cmd(self.pan_centroids[1], self.tilt_centroids[0]

        # B2 Target
        elif(command == 7):
            self.target_cmd(self.pan_centroids[1], self.tilt_centroids[1]

        # B3 Target
        elif(command == 8):
            self.target_cmd(self.pan_centroids[1], self.tilt_centroids[2]

        # B4 Target
        elif(command == 9):
            self.target_cmd(self.pan_centroids[1], self.tilt_centroids[3]

        # B5 Target
        elif(command == 10):
            self.target_cmd(self.pan_centroids[1], self.tilt_centroids[4]

# --- C TARGETS ---

        # C1 Target
        elif(command == 11):
            self.target_cmd(self.pan_centroids[2], self.tilt_centroids[0]

        # C2 Target
        elif(command == 12):
            self.target_cmd(self.pan_centroids[2], self.tilt_centroids[1]

        # C3 Target
        elif(command == 13):
            self.target_cmd(self.pan_centroids[2], self.tilt_centroids[2]

        # C4 Target
        elif(command == 14):
            self.target_cmd(self.pan_centroids[2], self.tilt_centroids[3]

        # C5 Target
        elif(command == 15):
            self.target_cmd(self.pan_centroids[2], self.tilt_centroids[4]

# --- D TARGETS ---
```

```
        # D1 Target
        elif(command == 16):
            self.target_cmd(self.pan_centroids[3], self.tilt_centroids[0]

        # D2 Target
        elif(command == 17):
            self.target_cmd(self.pan_centroids[3], self.tilt_centroids[1]

        # D3 Target
        elif(command == 18):
            self.target_cmd(self.pan_centroids[3], self.tilt_centroids[2]

        # D4 Target
        elif(command == 19):
            self.target_cmd(self.pan_centroids[3], self.tilt_centroids[3]

        # D5 Target
        elif(command == 20):
            self.target_cmd(self.pan_centroids[3], self.tilt_centroids[4]

    # --- E TARGETS ---

        # E1 Target
        elif(command == 21):
            self.target_cmd(self.pan_centroids[4], self.tilt_centroids[0]

        # E2 Target
        elif(command == 22):
            self.target_cmd(self.pan_centroids[4], self.tilt_centroids[1]

        # E3 Target
        elif(command == 23):
            self.target_cmd(self.pan_centroids[4], self.tilt_centroids[2]

        # E4 Target
        elif(command == 24):
            self.target_cmd(self.pan_centroids[4], self.tilt_centroids[3]

        # E5 Target
        elif(command == 25):
            self.target_cmd(self.pan_centroids[4], self.tilt_centroids[4]

    # --- SHOOT ---

        # WINDUP ON
        elif(command == 26):
```

```
            self.WINDUP_GUN.put(1)

        # FEED ON
        elif(command == 27):
            self.FEED_BULLETS.put(1)

        # WINDUP OFF
        elif(command == 28):
            self.WINDUP_GUN.put(0)

        # FEED OFF
        elif(command == 29):
            self.FEED_BULLETS.put(0)

# --- MOVE ---

        # UP
        elif(command == 31):
            self.tilt_coords.put(self.tilt_angle.get() + 1)

        # DOWN
        elif(command == 32):
            self.tilt_coords.put(self.tilt_angle.get() - 1)

        # LEFT
        elif(command == 33):
            self.pan_coords.put(self.pan_position.get() + 100)

        # RIGHT
        elif(command == 34):
            self.pan_coords.put(self.pan_position.get() - 100)

# --- CALIBRATE LOCATIONS ---

        # CALIBRATION POINT
        elif(command == 36):
            self.CALIBRATION_FLG = True


# --- Home Button ---

        # HOME
        elif(command == 30):
            self.target_cmd(self.pan_centroids[2], self.tilt_centroids[0]
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Mar  7 00:00:39 2018

@author: JasonGrillo
"""

import pyb
import micropython

class Turret_Hub_Task:
    ''' This defines the task function method for a nerf turret hub.
    '''

    def __init__(self, pan_position, tilt_angle, pan_coords, tilt_coords,
        ''' Construct a turret hub task function by initilizing any share
            variables and objects
            @param pan_position The shared variable for the pan position
            @param tilt_angle The shared variable for the tilt position
            @param pan_coords The queue of coordinates for the pan axis
            @param tilt_coords The queue of coordinates for the tilt axis
            @param FEED_BULLETS The shared variable flag for the nerf gun
            @param WINDUP_GUN The shared variable flag for the nerf gun m
        '''
        self.pan_position = pan_position
        self.tilt_angle = tilt_angle
        self.pan_coords = pan_coords
        self.tilt_coords = tilt_coords
        self.FEED_BULLETS = FEED_BULLETS
        self.WINDUP_GUN = WINDUP_GUN

        self.TARGET_CMD = False

        self.CALIBRATION_FLG = False

        self.pan_centroids = [0.0, 0.0, 0.0, 0.0, 0.0]
        self.tilt_centroids = [0.0, 0.0, 0.0, 0.0, 0.0]

        self.printer_counter = 0
        self.target_pan = ''
        self.target_tilt = ''

# ----------------------------------------------------------------------
# ----------------------------------------------------------------------
```

```
    def read_GUI(self):
        ''' Reads the serial port for incoming commands and executes the
        '''
        if self.vcp.any():
            self.GUI_input = float(self.vcp.read(2).decode('UTF-8'))
            self.GUI_Lookup_Table(self.GUI_input)

# ------------------------------------------------------------------------
# ------------------------------------------------------------------------

    def turret_hub_fun(self):
        ''' Defines the task function method for a turret hub object.
        '''
        self.vcp = pyb.USB_VCP ()

        STATE_0 = micropython.const(0)
        STATE_1 = micropython.const(1)
        STATE_2 = micropython.const(2)
        STATE_3 = micropython.const(3)
        STATE_4 = micropython.const(4)
        STATE_5 = micropython.const(5)
        STATE_6 = micropython.const(6)
        STATE_7 = micropython.const(7)
        STATE_8 = micropython.const(8)
        STATE_9 = micropython.const(9)
        STATE_10 = micropython.const(10)
        STATE_11 = micropython.const(11)

        self.state = STATE_0

        self.pan_coords.put(0)
        self.tilt_coords.put(0)

        while True:
#           print(self.tilt_angle.get())
            ################################

            ## STATE 0: CALIBRATE POINT A
            if self.state == STATE_0:
                self.read_GUI()
                yield (self.state)

                if self.CALIBRATION_FLG:
                    # input location A into pan centroids
                    self.calibrate_point(0, self.pan_position.get(), self
                    self.CALIBRATION_FLG = False
```

```
            self.state = STATE_1

        ##################################

        ## STATE 1: CALIBRATE POINT B
        elif self.state == STATE_1:
            self.read_GUI()
            yield (self.state)

            if self.CALIBRATION_FLG:
                # input location B into pan centroids
                self.calibrate_point(1, self.pan_position.get(), self
                self.CALIBRATION_FLG = False
                self.state = STATE_2

        ##################################

        ## STATE 2: CALIBRATE POINT C
        elif self.state == STATE_2:
            self.read_GUI()
            yield (self.state)

            if self.CALIBRATION_FLG:
                # input location C into pan centroids
                self.calibrate_point(2, self.pan_position.get(), self
                self.CALIBRATION_FLG = False
                self.state = STATE_3

        ##################################

        ## STATE 3: CALIBRATE POINT D
        elif self.state == STATE_3:
            self.read_GUI()
            yield (self.state)

            if self.CALIBRATION_FLG:
                # input location D into pan centroids
                self.calibrate_point(3, self.pan_position.get(), self
                self.CALIBRATION_FLG = False
                self.state = STATE_4

        ##################################

        ## STATE 4: CALIBRATE POINT E & 1
        elif self.state == STATE_4:
            self.read_GUI()
```

```
            yield (self.state)

            if self.CALIBRATION_FLG:
                # input location E into pan centroids
                self.calibrate_point(4, self.pan_position.get(), self
                # input location 1 into tilt centroids
                self.calibrate_point(0, self.pan_position.get(), self
                self.CALIBRATION_FLG = False
                self.state = STATE_5

        ##################################

        ## STATE 5: CALIBRATE POINT 2
        elif self.state == STATE_5:
            self.read_GUI()
            yield (self.state)

            if self.CALIBRATION_FLG:
                # input location 2 into tilt centroids
                self.calibrate_point(1, self.pan_position.get(), self
                self.CALIBRATION_FLG = False
                self.state = STATE_6

        ##################################

        ## STATE 6: CALIBRATE POINT 3
        elif self.state == STATE_6:
            self.read_GUI()
            yield (self.state)

            if self.CALIBRATION_FLG:
                # input location 3 into tilt centroids
                self.calibrate_point(2, self.pan_position.get(), self
                self.CALIBRATION_FLG = False
                self.state = STATE_7

        ##################################

        ## STATE 7: CALIBRATE POINT 4
        elif self.state == STATE_7:
            self.read_GUI()
            yield (self.state)

            if self.CALIBRATION_FLG:
                # input location 4 into tilt centroids
                self.calibrate_point(3, self.pan_position.get(), self
```

```
                self.CALIBRATION_FLG = False
                self.state = STATE_8

            ##################################

            ## STATE 8: CALIBRATE POINT 5
            elif self.state == STATE_8:
                self.read_GUI()
                yield (self.state)

                if self.CALIBRATION_FLG:
                    # input location 5 into tilt centroids
                    self.calibrate_point(4, self.pan_position.get(), self
                    print('Calibration complete.')
                    print(self.tilt_centroids)
                    self.state = STATE_9

            ##################################

            ## STATE 9: STOPPED, NOT SHOOTING
            elif self.state == STATE_9:
                self.read_GUI()
                yield (self.state)

                if self.TARGET_CMD:
                    self.state = STATE_10

            ##################################

            ## STATE 10: MOVING, SHOOTING
            elif self.state == STATE_10:
                # clear the target cmd flag for state 9 next time
                self.TARGET_CMD = False
                self.state = STATE_11

            ##################################

            ## STATE 11: STOPPED, SHOOTING
            elif self.state == STATE_11:
                # print the pan and tilt coordinates less frequently...
                self.print_coords()

                self.read_GUI()
                yield (self.state)

#                if not self.FEED_BULLETS.get() and not self.WINDUP_GUN.g
```

```
#                   self.state = STATE_9

            ##################################

            yield (self.state)

# ------------------------------------------------------------------------
# ------------------------------------------------------------------------

    def print_coords(self, counter = 10):
        ''' Prints the pan and tilt coordinates based on a decremented co
        @param counter The counter value that gets decremented. Once zero
        '''
        if not self.printer_counter:
            print('Pan: ' + str(self.pan_position.get() - self.pan_centro
            print('Tilt: ' + str(self.tilt_angle.get() - self.tilt_centro
            self.printer_counter = counter
        else:
            self.printer_counter -= 1

# ------------------------------------------------------------------------
# ------------------------------------------------------------------------

    def target_cmd(self, pan, tilt, target_cmd = True):
        ''' Defines what to do when target cmd is entered through the GUI
        self.pan_coords.put(pan)
        self.tilt_coords.put(tilt)
        print(pan)
        print(tilt)
        if target_cmd:
            self.TARGET_CMD = True
        else:
            self.TARGET_CMD = False

# ------------------------------------------------------------------------
# ------------------------------------------------------------------------

    def calibrate_point(self, index, pan_coor, tilt_coor, pan = False, ti
        ''' enters the calibrated point into the proper centroid list.
        @param index The index of the point in the centroid list
        @param pan_coor The pan coordinate of the point
        @param tilt_coor The tilt coordinate of the point
        @param pan Indicate if it's a pan calibration point
        @param tilt Indicate if it's a tilt calibration point
        '''
        if pan:
```

```
            self.pan_centroids[index] = pan_coor - 700
            self.pan_coords.put(pan_coor)

        if tilt:
            self.tilt_centroids[index] = tilt_coor + 3.5
            self.tilt_coords.put(tilt_coor)

# ----------------------------------------------------------------------
# ----------------------------------------------------------------------

    def GUI_Lookup_Table(self, command):
        ''' Decodes GUI commands based on a defined list of commands

        GUI Layout:
         _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
        | A1  B1  C1  D1  E1   Wind_on    Up       Calibration  |
        | A2  B2  C2  D2  E2   Feed_on    Down                  |
        | A3  B3  C3  D3  E3   Wind_off   Left                  |
        | A4  B4  C4  D4  E4   Feed_off   Right                 |
        | A5  B5  C5  D5  E5     Home                           |
         - - - - - - - - - - - - - - - - - - - - - - - - - - - -

         GUI Command Numbers:
         _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
        | 1   6   11  16  21      26      31          36        |
        | 2   7   12  17  22      27      32                    |
        | 3   8   13  18  23      28      33                    |
        | 4   9   14  19  24      29      34                    |
        | 5   10  15  20  25      30                            |
         - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        @param command The incoming GUI command to decode
        '''

# --- A TARGETS ---

       # A1 Target
        if(command == 1):
```

```
            self.target_cmd(self.pan_centroids[0], self.tilt_centroids[0]
            self.target_pan = 0
            self.target_tilt = 0

        # A2 Target
        elif(command == 2):
            self.target_cmd(self.pan_centroids[0], self.tilt_centroids[1]
            self.target_pan = 0
            self.target_tilt = 1

        # A3 Target
        elif(command == 3):
            self.target_cmd(self.pan_centroids[0], self.tilt_centroids[2]
            self.target_pan = 0
            self.target_tilt = 2

        # A4 Target
        elif(command == 4):
            self.target_cmd(self.pan_centroids[0], self.tilt_centroids[3]
            self.target_pan = 0
            self.target_tilt = 3

        # A5 Target
        elif(command == 5):
            self.target_cmd(self.pan_centroids[0], self.tilt_centroids[4]
            self.target_pan = 0
            self.target_tilt = 4

# --- B TARGETS ---

        # B1 Target
        elif(command == 6):
            self.target_cmd(self.pan_centroids[1], self.tilt_centroids[0]
            self.target_pan = 1
            self.target_tilt = 0

        # B2 Target
        elif(command == 7):
            self.target_cmd(self.pan_centroids[1], self.tilt_centroids[1]
            self.target_pan = 1
            self.target_tilt = 1

        # B3 Target
        elif(command == 8):
            self.target_cmd(self.pan_centroids[1], self.tilt_centroids[2]
            self.target_pan = 1
```

```
            self.target_tilt = 2

        # B4 Target
        elif(command == 9):
            self.target_cmd(self.pan_centroids[1], self.tilt_centroids[3]
            self.target_pan = 1
            self.target_tilt = 3

        # B5 Target
        elif(command == 10):
            self.target_cmd(self.pan_centroids[1], self.tilt_centroids[4]
            self.target_pan = 1
            self.target_tilt = 4

# --- C TARGETS ---

        # C1 Target
        elif(command == 11):
            self.target_cmd(self.pan_centroids[2], self.tilt_centroids[0]
            self.target_pan = 2
            self.target_tilt = 0

        # C2 Target
        elif(command == 12):
            self.target_cmd(self.pan_centroids[2], self.tilt_centroids[1]
            self.target_pan = 2
            self.target_tilt = 1

        # C3 Target
        elif(command == 13):
            self.target_cmd(self.pan_centroids[2], self.tilt_centroids[2]
            self.target_pan = 2
            self.target_tilt = 2

        # C4 Target
        elif(command == 14):
            self.target_cmd(self.pan_centroids[2], self.tilt_centroids[3]
            self.target_pan = 2
            self.target_tilt = 3

        # C5 Target
        elif(command == 15):
            self.target_cmd(self.pan_centroids[2], self.tilt_centroids[4]
            self.target_pan = 2
            self.target_tilt = 4
```

```
# --- D TARGETS ---

        # D1 Target
        elif(command == 16):
            self.target_cmd(self.pan_centroids[3], self.tilt_centroids[0]
            self.target_pan = 3
            self.target_tilt = 0

        # D2 Target
        elif(command == 17):
            self.target_cmd(self.pan_centroids[3], self.tilt_centroids[1]
            self.target_pan = 3
            self.target_tilt = 1

        # D3 Target
        elif(command == 18):
            self.target_cmd(self.pan_centroids[3], self.tilt_centroids[2]
            self.target_pan = 3
            self.target_tilt = 2

        # D4 Target
        elif(command == 19):
            self.target_cmd(self.pan_centroids[3], self.tilt_centroids[3]
            self.target_pan = 3
            self.target_tilt = 3

        # D5 Target
        elif(command == 20):
            self.target_cmd(self.pan_centroids[3], self.tilt_centroids[4]
            self.target_pan = 3
            self.target_tilt = 4

# --- E TARGETS ---

        # E1 Target
        elif(command == 21):
            self.target_cmd(self.pan_centroids[4], self.tilt_centroids[0]
            self.target_pan = 4
            self.target_tilt = 0

        # E2 Target
        elif(command == 22):
            self.target_cmd(self.pan_centroids[4], self.tilt_centroids[1]
            self.target_pan = 4
            self.target_tilt = 1
```

```
        # E3 Target
        elif(command == 23):
            self.target_cmd(self.pan_centroids[4], self.tilt_centroids[2]
            self.target_pan = 4
            self.target_tilt = 2

        # E4 Target
        elif(command == 24):
            self.target_cmd(self.pan_centroids[4], self.tilt_centroids[3]
            self.target_pan = 4
            self.target_tilt = 3

        # E5 Target
        elif(command == 25):
            self.target_cmd(self.pan_centroids[4], self.tilt_centroids[4]
            self.target_pan = 4
            self.target_tilt = 4

    # --- SHOOT ---

        # WINDUP ON
        elif(command == 26):
            self.WINDUP_GUN.put(1)

        # FEED ON
        elif(command == 27):
            self.FEED_BULLETS.put(1)

        # WINDUP OFF
        elif(command == 28):
            self.WINDUP_GUN.put(0)

        # FEED OFF
        elif(command == 29):
            self.FEED_BULLETS.put(0)

    # --- MOVE ---

        # UP
        elif(command == 31):
            self.tilt_coords.put(self.tilt_angle.get() + 0.25)

        # DOWN
        elif(command == 32):
            self.tilt_coords.put(self.tilt_angle.get() - 0.25)
```

```
        # LEFT
        elif(command == 33):
            self.pan_coords.put(self.pan_position.get() + 10)

        # RIGHT
        elif(command == 34):
            self.pan_coords.put(self.pan_position.get() - 10)

# --- CALIBRATE LOCATIONS ---

        # CALIBRATION POINT
        elif(command == 36):
            self.CALIBRATION_FLG = True


# --- Home Button ---

        # HOME
        elif(command == 30):
            self.target_cmd(self.pan_centroids[2], self.tilt_centroids[0]
```

```
"""
Created on Fri Feb  9 23:53:47 2018

@author: JasonGrillo
"""

import encoder
import micropython

class Encoder_Task:
    ''' This defines the task function method for an encoder. The encoder
        passes its data via a shared variable with another task.
        To create an instance of this task class (example):
            # create encoder position shared variable
            pan_position = task_share.Share ('i', thread_protect = False,
                                             name = "Share_0_pan_positi
            # create encoder 1 task object
            pan_encoder = Encoder_Task(pan_position, 4,
                                       pyb.Pin.board.PB6, pyb.Pin.board.PB7
            # create task1 function
            task1 = cotask.Task (pan_encoder.enc_fun(), name = 'Task_1',
                         period = 2, profile = True, trace = False)
            # append task1 to list of scheduled tasks
            cotask.task_list.append (task1)
    '''

    def __init__(self, pan_position, timer, pin1, pin2):
        ''' Construct an encoder task function by initilizing any shared
            variables and initialize the encoder object
            @param pan_position The shared variable between tasks that co
            @param timer The Encoder's timer channel
            @param pin1 The Encoder's first pin, Pin A
            @param pin2 The Encoder's second pin, Pin B
        '''
        self.pan_position = pan_position
        self.Encoder = encoder.Encoder(timer, pin1, pin2)

    def enc_fun(self):
        ''' Defines the task function method for an Encoder object.
        '''

        STATE_0 = micropython.const(0)
        STATE_1 = micropython.const(1)

        self.state = STATE_0
```

```
while True:
    ## STATE 0: ZERO REFERENCE
    if self.state == STATE_0:
        self.Encoder.zero_encoder()
        self.state = STATE_1

    ## STATE 1: UPDATING
    elif self.state == STATE_1:
        # Read encoder and update the shared variable
        self.pan_position.put(self.Encoder.read_encoder())
    yield (self.state)
```

```
"""
Created on Fri Feb  9 23:53:47 2018

@author: JasonGrillo
"""

import BNO055
import pyb
import micropython

class IMU_Task:
    ''' This defines the task function method for an IMU. The IMU
        passes its data via a shared variable with another task.
        To create an instance of this task class (example):
            # create run shared variable
            Run = task_share.Share('i', thread_protect = False,
                                   name = "Run_Intertask_Comm_Variable")
            # create IMU position shared variable
            IMU_position = task_share.Share ('i', thread_protect = False,
                                             name = "IMU_position")
            # create IMU 1 task object
            IMU_1 = IMU_Task(Run, IMU_position, 4,
                                      pyb.Pin.board.PB6, pyb.Pin.board.PB7
            # create task1 function
            task1 = cotask.Task (IMU_1.IMU_fun, name = 'Task_1', priority
                         period = 10, profile = True, trace = False)
            # append task1 to list of scheduled tasks
            cotask.task_list.append (task1)
    '''

    def __init__(self, tilt_angle):
        ''' Construct an IMU task function by initilizing any shared
            variables and initialize the IMU object
            @param tilt_angle The shared variable between tasks that cont
        '''
        self.tilt_angle = tilt_angle
        self.imu = BNO055.bno055 (pyb.I2C (1, pyb.I2C.MASTER, baudrate =

    def IMU_fun(self):
        ''' Defines the task function method for an IMU object.
        '''

        STATE_0 = micropython.const(0)
        STATE_1 = micropython.const(1)

        self.state = STATE_0
```

```
while True:
    ## STATE 0: Initialize State Machine
    if self.state == STATE_0:
        # Calibrate the IMU against the hardstop
        # ... must be against hardstop upon system boot
        self.imu.zero_Euler_vals()
        self.state = STATE_1

    ## STATE 1: Get IMU Values
    elif self.state == STATE_1:
        # Read IMU and update the shared variable with Euler pitc
        self.tilt_angle.put(self.imu.get_euler_roll())

    yield (self.state)
```

```
"""
Created on Fri Feb  9 23:53:47 2018

@author: JasonGrillo
"""
import motor
import controller
import micropython

class Motor_Task:
    ''' This defines the task function method for a motor. The motor
        utilizes shared data from an encoder to know where it is.
        To create an instance of this task class (example):
            # have run shared variable declared
            Run = task_share.Share('i', thread_protect = False,
                                   name = "Run_Intertask_Comm_Variable")
            # have encoder position shared variable declared
            enc_1_position = task_share.Share ('i', thread_protect = Fals
                                               name = "Share_0_enc_1_posi
            # create motor 1 task object
            Motor_1 = motor_task_func.Motor_Task(Run, enc_1_position, 4,
                                     pyb.Pin.board.PB6, pyb.Pin.board.PB7
            # create task2 function, adjust parameters for implementation
            task2 = cotask.Task (Motor_1.enc_fun(), name = 'Task_2', prio
                         period = 2, profile = True, trace = False)
            # append task2 to list of scheduled tasks
            cotask.task_list.append (task2)
    '''

    def __init__(self, position, coordinate, timer, EN_Pin, pin1, pin2, K
        ''' Construct an encoder task function by initilizing any shared
            variables and initialize the encoder object
            @param position The shared variable between tasks that contai
            @param coordinate The desired coordinate to which to move the
            @param timer The Motor's timer channel
            @param EN_pin The Motor's __?__ pin
            @param pin1 The Motor's first pin, Pin A
            @param pin2 The Motor's second pin, Pin B
            @param Kp The Motor Controller's proportional gain
            @param Ki The Motor Controller's integral gain
            @param Setpoint Where the motor is desired to go
            @param saturation The anti wind up saturation limit
        '''
        self.position = position
        self.coordinate = coordinate
        self.Motor = motor.MotorDriver(timer, EN_Pin, pin1, pin2)
```

```
        self.Controller = controller.Controller(Kp, Ki, Kd, saturation)

    def mot_fun(self):
        ''' Defines the task function method for a Motor object.
        '''
        STATE_0 = micropython.const (0)
        STATE_1 = micropython.const (1)

        self.state = STATE_0

        while True:
            ## STATE 0: STOPPED
            if self.state == STATE_0:
                # Stop motor
                self.Motor.set_duty_cycle(0)
                self.state = STATE_1

            ## STATE 1: CONTROLLING MOTOR
            elif self.state == STATE_1:
                if self.coordinate.any():
                    print('new coordinate to move to!')
                    self.Controller.clear_controller()
                    self.Controller.set_setpoint(self.coordinate.get())
                # Use controller object to get appropriate duty cycle for
                self.Duty_Cycle = self.Controller.repeatedly(self.positio
                # Set duty cycle to motor
                self.Motor.set_duty_cycle(self.Duty_Cycle)

            yield(self.state)
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar  4 00:52:43 2018

@author: JasonGrillo
"""

import pyb
import micropython

class Nerf_Task:
    ''' This defines the task function method for a nerf gun.
    '''

    def __init__(self, WINDUP_GUN, FEED_BULLETS, pin1, pin2):
        ''' Construct an encoder task function by initilizing any shared
            variables and initialize the encoder object
            @param WINDUP_GUN The shared variable flag indicating when to
            @param FEED_BULLETS The shared variable flag indicating when
            @param pin1 The motor windup pin (connected to mosfet)
            @param pin2 The bullet feeder pin (connected to mosfet)
        '''
        self._WINDUP_GUN = WINDUP_GUN
        self._FEED_BULLETS = FEED_BULLETS
        self._pin1 = pin1
        self._pin2 = pin2

    def gun_fun(self):
        ''' Defines the task function method for an NERF GUN object.
        '''
        WindUp_Pin = pyb.Pin(self._pin1, pyb.Pin.OUT_OD, pull=pyb.Pin.PUL
        Fire_Pin = pyb.Pin(self._pin2, pyb.Pin.OUT_OD, pull=pyb.Pin.PULL_

        STATE_0 = micropython.const(0)
        STATE_1 = micropython.const(1)
        STATE_2 = micropython.const(2)

        self.state = STATE_0

        while True:
            ## STATE 0: NOT WINDUP AND NOT SHOOT
            if self.state == STATE_0:
                WindUp_Pin.low()
                Fire_Pin.low()
                if self._WINDUP_GUN.get():
```

```
            WindUp_Pin.high()
            self.state = STATE_1

    ## STATE 1: WINDUP AND NOT SHOOT
    elif self.state == STATE_1:
        if not self._WINDUP_GUN.get() and not self._FEED_BULLETS.
            WindUp_Pin.low()
            Fire_Pin.low()
            self.state = STATE_0
        elif self._WINDUP_GUN.get() and self._FEED_BULLETS.get():
            WindUp_Pin.high()
            Fire_Pin.high()
            self.state = STATE_2

    ## STATE 2: WINDUP AND SHOOT
    elif self.state == STATE_2:
        if not self._WINDUP_GUN.get() and not self._FEED_BULLETS.
            WindUp_Pin.low()
            Fire_Pin.low()
            self.state = STATE_0
        elif self._WINDUP_GUN.get() and not self._FEED_BULLETS.ge
            Fire_Pin.low()
            self.state = STATE_1

    yield (self.state)
```

```
"""
Created on Thu Jan 11 21:19:40 2018

@author: mecha10, JGrillo, TGoehring, TPeterson
"""
import pyb

class MotorDriver:
    ''' This class implements a motor driver for the
    ME405 board.
    either  MotorDriver(3, pyb.Pin.board.PA10, pyb.Pin.board.PB4, pyb.Pin
            MotorDriver(5, pyb.Pin.board.PC1, pyb.Pin.board.PA0, pyb.Pin.
    '''

    def __init__ (self, timer, EN_Pin, Pin_1, Pin_2):
        ''' Creates a motor driver by initializing GPIO
        pins and turning the motor off for safety. '''
        print ('Creating a motor driver')
        ## Set Pin PA10 toas open-drain output with pull up resistors
        self.EN_Pin=pyb.Pin(EN_Pin,pyb.Pin.OUT_OD, pull=pyb.Pin.PULL_UP)
        ## Set Pin PB4 as push-pull with the correct alternate function (
        self.Pin_1=pyb.Pin(Pin_1, pyb.Pin.AF_PP,af=2)
        ## Set Pin PB5 as push-pull with the correct alternate function (
        self.Pin_2=pyb.Pin(Pin_2, pyb.Pin.AF_PP,af=2)
        self.timer= pyb.Timer(timer, freq=20000)
        self.ch1 = self.timer.channel(1, pyb.Timer.PWM, pin=self.Pin_1) #
        self.ch2 = self.timer.channel(2, pyb.Timer.PWM, pin=self.Pin_2) #
        self.EN_Pin.low()                                               # Set P
        self.Pin_1.low()
        self.Pin_2.low()

    def set_duty_cycle (self, level):
        ''' This method sets the duty cycle to be sent
        to the motor to the given level. Positive values
        cause torque in one direction, negative values
        in the opposite direction.
        @param level A signed integer holding the duty cycle of the volta
        '''
        if (level >= 0):
            self.ch1.pulse_width_percent(0)
            self.ch2.pulse_width_percent(level)
        else:
            self.ch2.pulse_width_percent(0)
            self.ch1.pulse_width_percent(-level)
        self.EN_Pin.high()
```

```
"""
Created on Thu Jan 25 18:30:59 2018

@author: Jason Grillo, Thomas Goehring, Trent Peterson
"""

class Controller:
    ''' This class implements a generic proportional gain controller'''

    def __init__(self, Kp, Ki, Kd, saturation):
        '''
        Initializes the proportional gain and defines the
        initial setpoint for the controller.
        @param Kp: Sets proportional gain value
        @param Ki: Sets integral gain value
        @param saturation: The anti-wind up integration saturation limit
        '''
        print('Creating a controller')
        ## Proportional gain for a control loop
        self.Kp = Kp
        ## Integral gain for a control loop
        self.Ki = Ki
        ## Derivative gain for a control loop
        self.Kd = Kd
        ## Desired output target variable
        self.setpoint = 0
        self.saturation = saturation
        ## Actuation signal sent to the plant
        self.__actuate_signal = 0
        ## Current output value of feedback from plant
        self.__current_value = 0
        ## Set the start variable to true to begin integral gain term
        self.__start = True
        self._esum = 0
        self._perror = 0
        self._deriv = 0

        print('Controller sucessfully created')

    def set_setpoint(self, new_setpoint):
        '''
        Method to enable the user to define a new setpoint that the
        control loop will use as a reference value.
        @param new_setpoint: User-defined setpoint that the controller us
        '''
        self.setpoint = new_setpoint
```

```
    def set_Kp(self, new_Kp):
        '''
        Method to enable the user to define a new proportional gain
        that the control loop will use to multiply the error signal
        and output an actuation signal.
        @param new_Kp: User-defined propotional gain that is multiplied b
        '''
        self.Kp = new_Kp

    def set_Ki(self, new_Ki):
        '''
        Method to enable the user to define a new integral gain
        that the control loop will use to multiply the error sum signal
        and output an actuation signal.
        @param new_Ki: User-defined integral gain that is multiplied by e
        '''
        self.Ki = new_Ki

    def set_newSat(self, new_Sat):
        '''
        Method to enable the user to define a integral saturation
        and output a saturated error signal if saturated
        @param newSat: User-defined saturation that is used to prevent in
        '''
        self.saturation = new_Sat

    def set_Kd(self, new_Kd):
        '''
        Method to enable the user to define a new derivative gain
        that the control loop will use to multiply the error signal
        and output an actuation signal.
        @param new_Kd: User-defined derivative gain that is multiplied by
        '''
        self.Kd = new_Kd

    def repeatedly(self,current_value):
        '''
        Method that repeatedly runs the control algorithm. Compares
        setpoint to actual signal value. This error is multiplied by
        the proportional gain and sent to the plant.
        @param current_value: Received current value from feedback loop
        @return actuate_signal: % duty cycle sent to device driver
        '''
        #Define current value variable to be used in control algorithm
        self.__current_value = current_value
```

```
        self._error = self.setpoint - self.__current_value

        # Calculate integral of error (Esum)
        self._esum += self._error

        if self._esum > self.saturation:
            self._esum = self.saturation
        elif self._esum < -self.saturation:
            self._esum = -self.saturation

        #Calculate Derivative of Error
        self._deriv = self._error - self._perror
        self._perror = self._error

        #Creates actuation signal from the proportional gain mulitplied b
        self.__actuate_signal = self._error*self.Kp + self._esum*self.Ki+
        return self.__actuate_signal

    def percent_completion(self):
        ''' Returns the completion calculation for the controlled path.
        @return error The error from the desired position and current pos
        '''
        try:
            percent = (self._error/self.setpoint)*100
        except ZeroDivisionError:
            return 0
        else:
            return percent

    def clear_controller(self):
        ''' Clears the esum for a new target location.
        '''
        self._esum = 0
        self._error = 0
        self.__actuate_signal = 0
        self._perror = 0
        self._deriv = 0
```

```
import pyb

class Encoder:
    ''' This class implements a motor driver for the
    ME405 board. '''

    def __init__ (self, timer, pin1, pin2):
        '''
        Initializes the pins and timer channels for an encoder object.
        To create PB6 and PB7 Encoder reader: \n
            pin1 = pyb.Pin.board.PB6 # Pin A \n
            pin2 = pyb.Pin.board.PB7 # Pin B \n
            timer = 4 \n\n
        To create PC6 and PC7 Encoder reader: \n
            pin1 = pyb.Pin.board.PC6 # Pin A \n
            pin2 = pyb.Pin.board.PC7 # Pin B \n
            timer = 8 \n
        @param timer: Specifies the timer for the encoder
        @param pin1: First pin (A) used to read the encoder
        @param pin2: Second pin (B) used to read the encoder
        '''
        print ('Creating an encoder')
        ## First encoder pin associated with the pin1 (A) input parameter
        self.pin_object_1 = pyb.Pin(pin1)
        ## Second encoder pin associated with the pin2 (B) input paramete
        self.pin_object_2 = pyb.Pin(pin2)
        ## Timer number associated with the assigned pins
        self.timer_val = pyb.Timer(timer)
        ## Timer number associated with the assigned pins (Set timer to h
        self.timer_val.init(prescaler=0,period=0xFFFF)
        ## Initializes channel 1 for pin1 (A) timer input
        self.ch1 = self.timer_val.channel(1,pyb.Timer.ENC_AB,pin=self.pin
        ## Initializes channel 2 for pin2 (B) timer input
        self.ch2 = self.timer_val.channel(2,pyb.Timer.ENC_AB,pin=self.pin
        # Instantaneous encoder reading at time of call
        self.__current_count = 0
        # Difference between last count and current count
        self.__delta_count = 0
        # Encoder reading from previous call
        self.__last_count = 0
        # Current absolute position of the encoder
        self.__encoder_val = 0
        print('Encoder object successfully created')

    def read_encoder(self):
        '''
```

```
        Reads the current encoder value
        @return encoder_val
        '''
        # Read current encoder count
        self.__current_count = self.timer_val.counter()
        # Subtract previous reading from current reading
        self.__delta_count = self.__current_count - self.__last_count
        # Account for 16-bit discontinuity phenomena
        if self.__delta_count > 32767:
            self.__delta_count -= 65535
        elif self.__delta_count < -32768:
            self.__delta_count += 65535
        # Add delta to absolute encoder position
        self.__encoder_val += self.__delta_count
        # Store current encoder reading into previous reading
        self.__last_count = self.__current_count
        #print(self.__encoder_val)
        return self.__encoder_val

    def zero_encoder(self):
        '''
        Resets all encoder parameters to zero
        @return encoder_val
        '''
        self.timer_val.counter(0)   # Reset all encoder parameters to zer
        self.__current_count = 0
        self.__last_count = 0
        self.__delta_count = 0
        self.__encoder_val = 0
        return self.__encoder_val
```

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

@author Jason Grillo, Thomas Goehring, Trent Peterson
"""

import micropython
import ustruct

# BNO055 Registers
# - Referenced registers from Radomir Dopieralski's Circuit Python module

_CHIP_ID = micropython.const(0xa0)

CONFIG_MODE = micropython.const(0x00)
ACCONLY_MODE = micropython.const(0x01)
MAGONLY_MODE = micropython.const(0x02)
GYRONLY_MODE = micropython.const(0x03)
ACCMAG_MODE = micropython.const(0x04)
ACCGYRO_MODE = micropython.const(0x05)
MAGGYRO_MODE = micropython.const(0x06)
AMG_MODE = micropython.const(0x07)
IMU_MODE = micropython.const(0x08)
COMPASS_MODE = micropython.const(0x09)
M4G_MODE = micropython.const(0x0a)
NDOF_FMC_OFF_MODE = micropython.const(0x0b)
NDOF_MODE = micropython.const(0x0c)

_POWER_NORMAL = micropython.const(0x00)
_POWER_LOW = micropython.const(0x01)
_POWER_SUSPEND = micropython.const(0x02)

_MODE_REGISTER = micropython.const(0x3d)
_PAGE_REGISTER = micropython.const(0x07)
_TRIGGER_REGISTER = micropython.const(0x3f)
_POWER_REGISTER = micropython.const(0x3e)
_ID_REGISTER = micropython.const(0x00)

class bno055:
    """ This class implements a simple driver for the BNO055 Adafruit
    IMU. This IMU talk to the CPU over I<sup>2</sup>C.

#    An example of how to use this driver:
#    @code
#    imu = BNO055.bno055 (pyb.I2C (1, pyb.I2C.MASTER, baudrate = 100000),
```

```
#    imu.sys_status ()
#    imu.sys_error ()
#    imu.get_euler_pitch ()     # or other data...
#    @endcode
    The example code works for a BNO055 on a Adafruit<sup>TM</sup> breako
    board. """

    def __init__ (self, i2c, address):
        """ Initialize a BNO055 driver on the given I<sup>2</sup>C bus.
        @param i2c An I<sup>2</sup>C bus already set up in MicroPython
        @param address The address of the IMU on the I<sup>2</sup>C bus
        """
        self._address = address
        self._i2c = i2c
        #Select NDOF mode, @IMU Hard address, Set NDOF_Mode to MODE_REGIS
        self._i2c.mem_write(IMU_MODE, self._address, _MODE_REGISTER)
        self._i2c.mem_write(_POWER_NORMAL, self._address, _POWER_REGISTER
        # Define calibration values... init as zero to have no effect
        self._zeroes = [0,0,0]

    def get_euler_pitch(self):
        """ Get the absolute euler pitch of the IMU. (_zeroes[0])
        @return _pitch_value The calibrated absolute pitch of the IMU
        """
        #Read 2 Pitch start at pitch lsb
        self._pitch = self._i2c.mem_read(2, self._address, 0x1E)
        #Unpact struct to get pitch value
        self._pitch_decode = ustruct.unpack('<h',self._pitch)
        self._pitch_value = float(self._pitch_decode[0]/16)
        return (self._pitch_value - self._zeroes[0])

    def get_euler_roll(self):
        """ Get the absolute euler roll of the IMU. (_zeroes[1])
        @return _roll_value The calibrated absolute roll of the IMU
        """
        self._roll = self._i2c.mem_read(2, self._address, 0x1C)
        self._roll_decode = ustruct.unpack('<h',self._roll)
        self._roll_value = float(self._roll_decode[0]/16)
        return (self._roll_value - self._zeroes[1])

    def get_euler_yaw(self):
        """ Get the absolute euler yaw of the IMU. (_zeroes[2])
        @return _raw_value The calibrated absolute yaw of the IMU
        """
        self._yaw = self._i2c.mem_read(2, self._address, 0x1A)
        self._yaw_decode = ustruct.unpack('<h',self._yaw)
```

```
        self._yaw_value = float(self._yaw_decode[0]/16)
        return (self._yaw_value - self._zeroes[2])

    def sys_status(self):
        """ Get the IMU status to see if it is running or if there are er
        @return _status_value The absolute pitch of the IMU
        """
        self._status = self._i2c.mem_read(1,self._address,0x39)
        self._status_decode  = ustruct.unpack('b',self._status)
        self._status_value = int(self._status_decode[0])
        return self._status_value

    def sys_error(self):
        """ Obtain the error, if the IMU is not returning values.
        @return _error_value The
        """
        self._error = self._i2c.mem_read(1,self._address,0x3A)
        self._error_decode = ustruct.unpack('b',self._error)
        self._error_value = int(self._error_decode[0])
        return self._error_value

    def zero_Euler_vals(self):
        """ Zero the IMU for calibration purposes.
        @return _zeroes The calibration list for Euler angle outputs
        """
        self._zeroes[0] = self.get_euler_pitch()
        self._zeroes[1] = self.get_euler_roll()
        self._zeroes[2] = self.get_euler_yaw()
        return self._zeroes
```