MEMORANDUM

To: Professor Charlie Refvem, Department of Mechanical Engineering, Cal Poly SLO
crefvem@calpoly.edu

From: Jack Butler
jbutle10@calpoly.edu

Tarun Ganamur
tsreesai@calpoly.edu

Phillip Shafik
pashafik@calpoly.edu

Date: November 1st, 2022

RE: Lab 0x04 – Closed-Loop Motor Control

Program Overview

Our program is comprised of three main tasks – UI, Motor, and Encoder. UI is the brain of the system, dealing with all of the logic if the system. It sets flags and makes sure user input gets passed to the other tasks if necessary. It also does some simple value fetching, like the encoder position or motor speed. Motor controls the motors – mostly, it sets duty cycles either based off the open or closed loop controller. Encoder manages the encoders and measurement duties – its main jobs are running the step response and the 30-second data collection, and sending the gathered data to UART. The task diagram for our program is as follows:

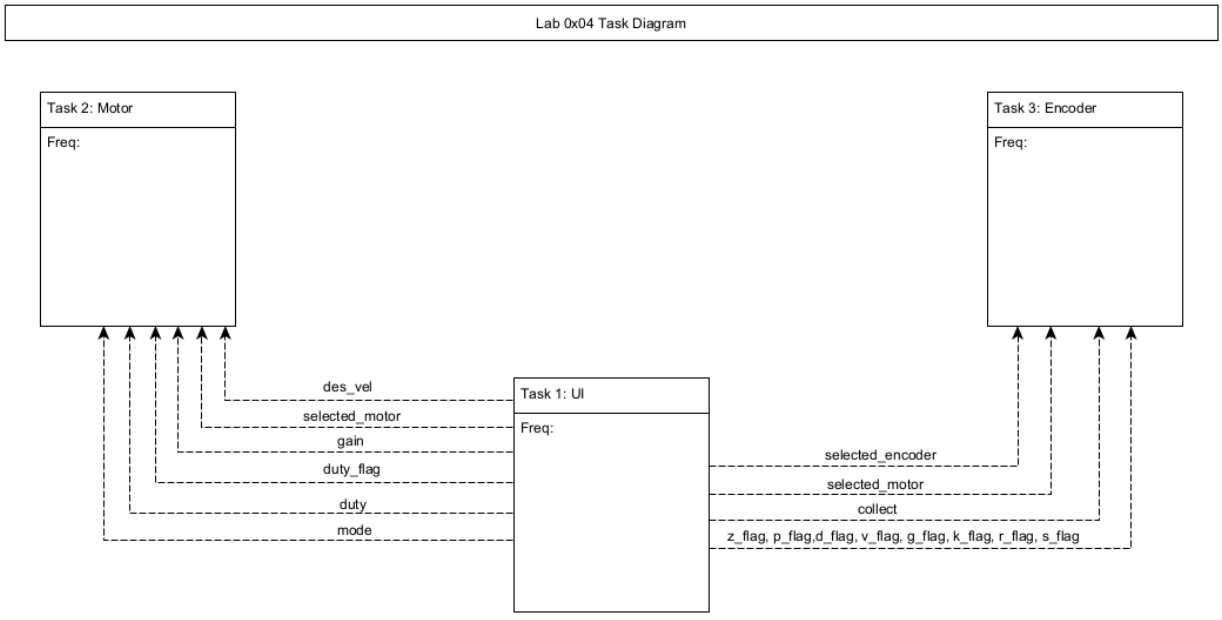

Figure 1. Task Diagram

The shares/queues in the program are as follows:

Table 1: Shares and queues in the program

Name	**Type**	**Contents**
Des_vel	Share	Proportional controller desired velocity
Gain	Share	Proportional controller gain
Collect	Share	Flag that tells Encoder to begin 30-second collection
Mode	Share	Flag for if controller is open or closed loop
Duty_flag	Share	Flag telling Motor to update open loop duty cycle
Duty	Share	Flag containing user duty cycle input
Selected_motor	Share	Flag signifying selected motor
Selected_encoder	Share	Flag signifying selected encoder
Prompted	Share	Flag signifying if user has been prompted for input
Numinput	Queue	Holds numerical user input
Z_flag	Share	Flag for when user input a z
P_flag	Share	Flag for when user input a p
D_flag	Share	Flag for when user input a d
V_flag	Share	Flag for when user input a y
G_flag	Share	Flag for when user input a g
K_flag	Share	Flag for when user input a k
R_flag	Share	Flag for when user input an r
S_flag	Share	Flag for when user input an s
Collectionpos	Queue	Contains positions collected
Colelctionvel	Queue	Contains velocities collected

Task Details

UI

UI is the only task allowed to use VCP, but is not allowed to use UART. Using VCP, it reads user input and writes any necessary messages to the terminal. It parses user input in order to set the relevant flags for other tasks, and takes care of blocking user input when the system is busy (when recording data or performing a step response). The state-transition diagram for UI is as follows:

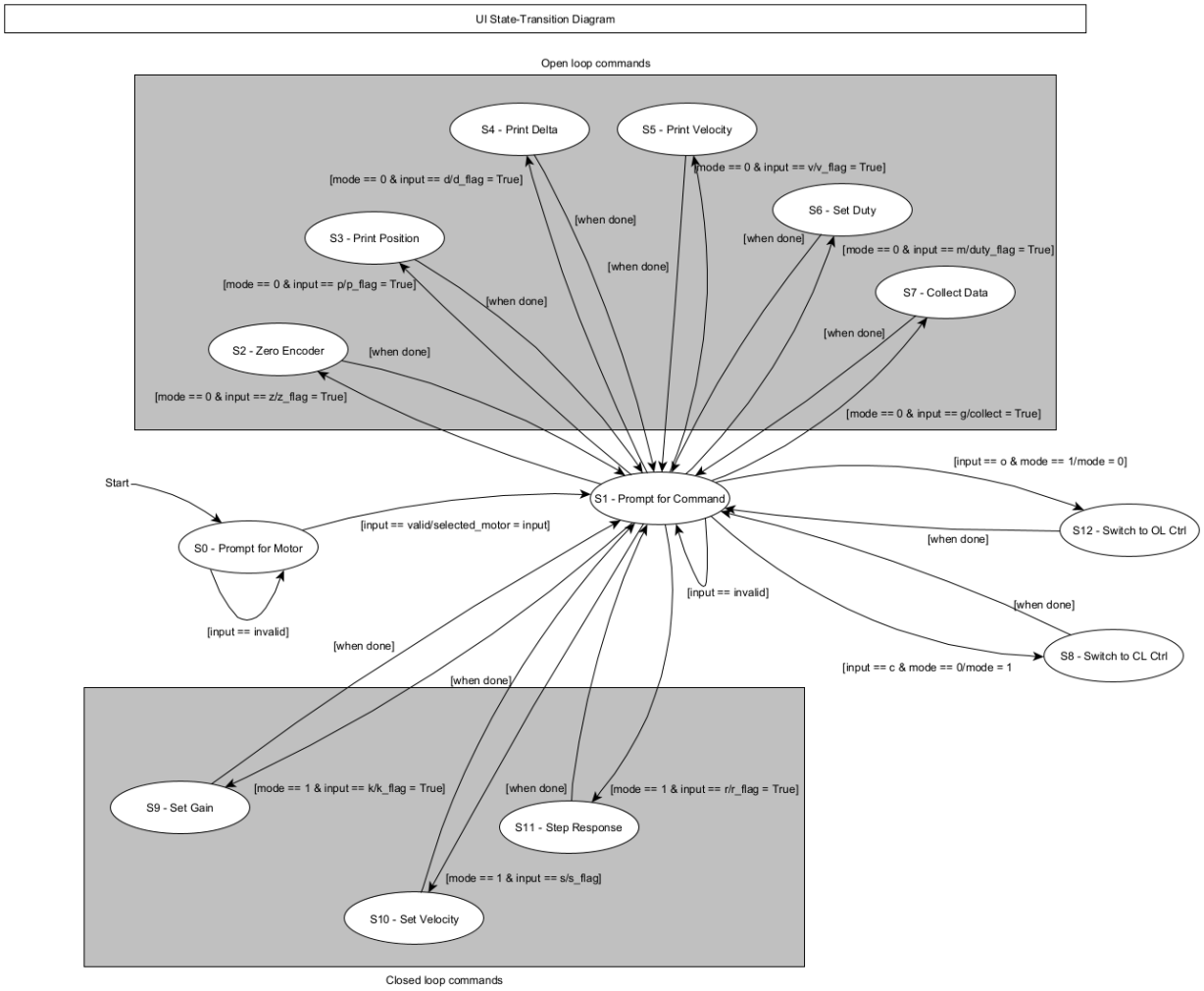

Figure 2. State-transition diagram for UI task

Motor

Motor does the actual controlling of the motor, in both open and closed loop configurations. The state-transition diagram for Motor is as follows:

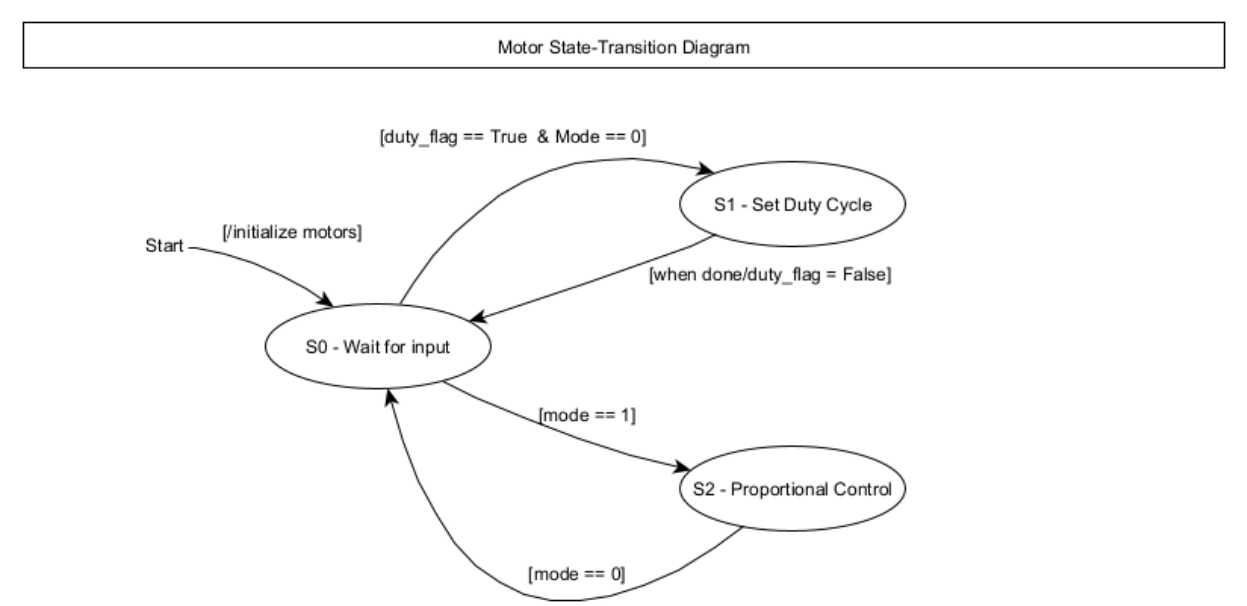

Figure 3. State-transition diagram for Motor task

Encoder

Encoder takes care of the heavier data collection jobs, like the step response and the 30-second data collection. It manages writing the data that was collected to UART, and sets a flag to let UI know to block user input while it's collecting data. The state-transition diagram for Encoder is as follows:

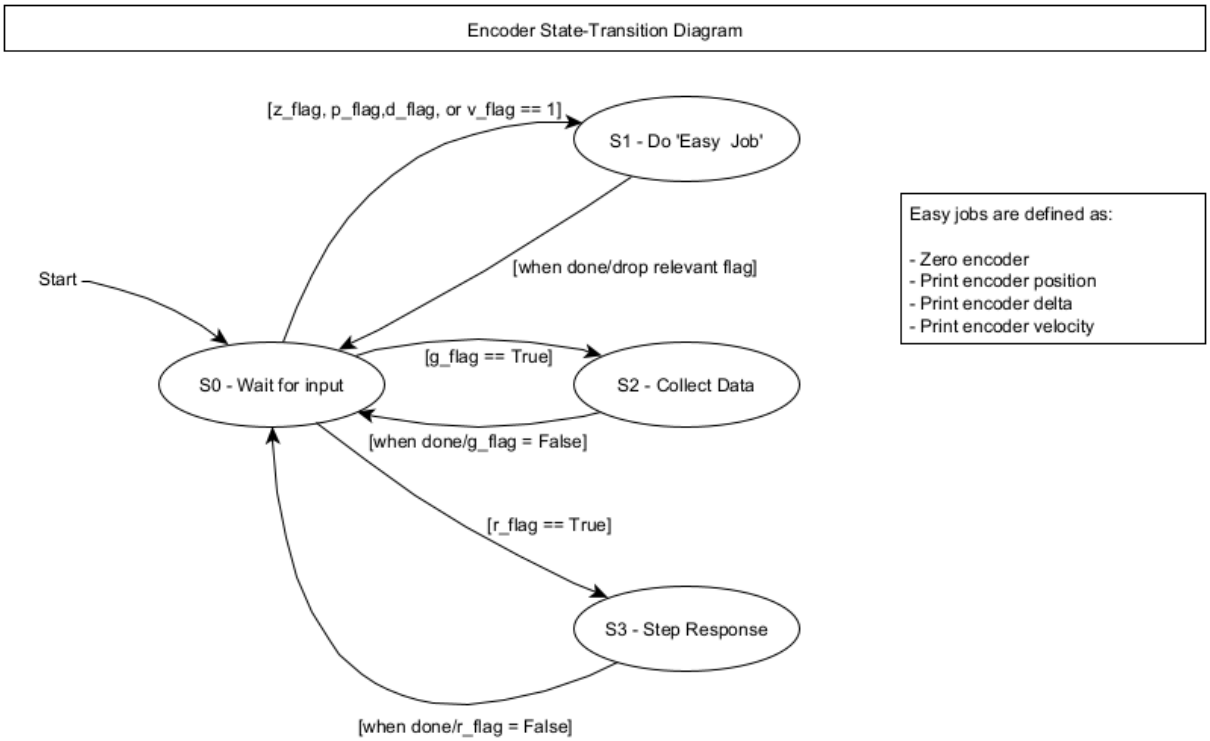

Figure 4. State-transition diagram for Encoder task

Step Response Tuning
To tune, we set the desired velocity to 20 rad/s and varied K_p.

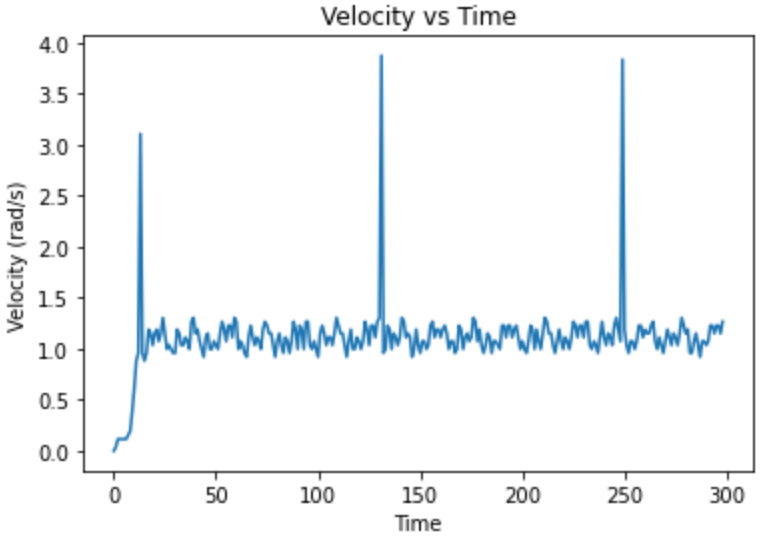

Figure 5. Step response at K_p = 1

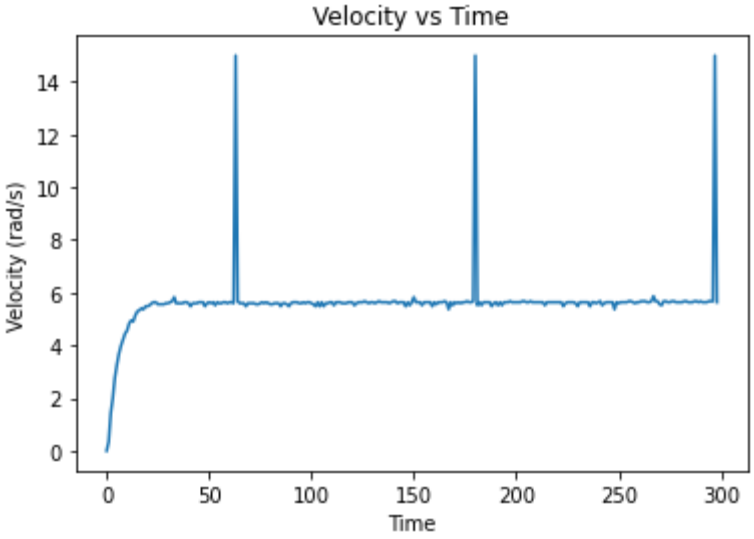

Figure 6. Step response at K_p = 3

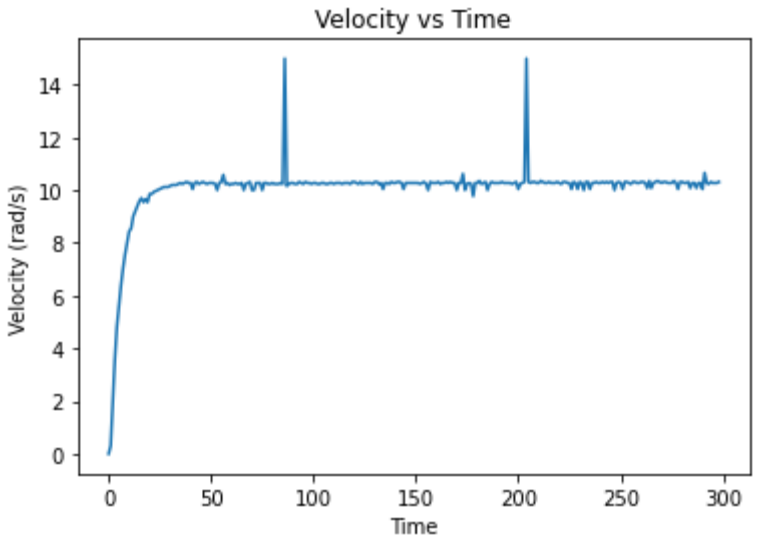

Figure 7. Step response at K_p = 10

After our *very* brief tuning, we settled on a K_p of 10 percent*s/rad.

Conclusion

This lab laid a lot of the groundwork for things we'll be doing on our term project – we created a controller class, made a solid user interface, and laid the groundwork for position control. In a sense, a big part of the term project will be scaling up this lab.

Main

```
from pyb import UART
from pyb import Pin
from pyb import Timer
import micropython
import task_share
import task_share
import cotask
import Encoder
import L6206
import gc
import cl

pyb.repl_uart(None)
ui_help = '|-------------------------------------------------------------------------
--|\r\n|------------- Motor Controller Command Help --------------------------
|\r\n|-----------------------------------------------------------------------
|\r\n|------------- Open Loop Commands ---------------------------------------
|\r\n|___z or Z___| zero the pos. of encoder 1 or
2                          |\r\n|___p or P___| print pos. of encoder 1 or
2                             |\r\n|___d or D___| print delta for encoder 1 or
2                           |\r\n|___v or V___| print vel. for encoder 1 or
2                            |\r\n|___m or M___| user prompt for duty cycle for
motors 1 or 2             |\r\n|___g or G___| collect data (vel. & pos.) for 30
sec from enc 1 or 2    |\r\n|___c or C___| switch modes to closed
loop                                |\r\n|---------------------------------------
------------------------------|\r\n|------------ Closed Loop Commands -----------
-------------------------|\r\n|___k or K___| choose closed loop gains for motor
1 or motor 2          |\r\n|___s or S___| choose a velocity setpoint for motor 1
or motor 2        |\r\n|___r or R___| trigger a step response on motor 1 or 2,
send data to PC |\r\n|___o or O___| switch modes to open
loop                                  \r\n|------------ Switch Motors ------------
------------------------------|\r\n|___a or A___| change selected motor to
motorA                           |\r\n|___b or B___| change selected motor to
motorB                           |\r\n|---------------------------------------
---------------------------|\r\n'

"""Methods"""
#encoder a's callback function
def encoder_A_CB(cb_source):
    if not collectionpos.full():
        enc1.update()
        encoder_pos[index] = enc1.position
        times[index] = index
```

```
        index += 1

#encoder b's callback function
def encoder_B_CB(cb_source):
    if not collectionpos.full():
        enc2.update()
        encoder_pos[index] = enc2.position
        times[index] = index
        index += 1

#method to take valid numerical numinput, numinput stays empty until valid input
has been given
def numerical_input(prompt: str, lowerbound: int, higherbound: int, fullin):
    if not prompted.get():
        prompted.put(True)
        numinput.clear()
        charin = ''
        fullin.clear()
        vcp.write('\r\n' + prompt)
    else:
        if vcp.any():
            charin = str(vcp.read(1).decode())
            if charin in {'\n','\r'}:
                try:
                    if (int(''.join(fullin))>=lowerbound and
int(''.join(fullin))<=higherbound):
                        prompted.put(False)
                        numinput.put(int(str(''.join(fullin))))
                    else:
                        vcp.write('\r\n> Please give a number in the valid range
' + str(lowerbound) + ' to ' + str(higherbound) + '\n\r> ')
                        fullin.clear()
                except:
                    vcp.write('\r\n> Please give a valid numerical input\r\n> ')
                    fullin.clear()
            else:
                fullin.append(charin)
                vcp.write(charin)
        else:
            pass
        return fullin

#performs a step response of the motor
def step_response(mot, enc, counter):
    #store position, calculate velocity (rad/s), store velocity
```

```
    if counter < 100:
        prop_controller.set_kp(50)
        prop_controller.set_velTarget(0)
        set_dc(mot, 0)
    elif counter == 100:
        prop_controller.set_velTarget(50)
        new_dutyC = prop_controller.p_eff(0)
        set_dc(mot, new_dutyC)
    elif counter < 500: #4 seconds (count/task period)
        vel = ((enc.delta/(16*256*4))/0.01)*6.283
        pos = (enc.position/(16*256*4))*6.283
        uart.write(f'{pos},{vel}\r\n')
        counter += 1
    #stop recording, print position and velocity queues to uart
    else:
        collect.put(0)
        prop_controller.set_kp(10)
        prop_controller.set_velTarget(0)
        set_dc(mot, 0)

#writes the encoder position to uart IN RAD
def print_encoder(enc):
    vcp.write('\r\n> Encoder position is: ' +
str((enc.position/(16*256*4))*6.283) + ' rad')

#zeros the encoder
def zero_encoder(enc):
    enc.zero()

#sets the motor duty cycle
def set_dc(mot,duty):
    mot.set_duty(duty)

#returns the encoder delta
def enc_delta(enc):
    vcp.write('\r\n> Encoder delta is: ' + str(enc.delta))

#collects speed and position data for 30 seconds then sends it to be plotted
def collect_data(enc, counter):
    if collect.get() == 1:
        #store first position point (rad), we won't have enough to calculate
velocity yet
        if counter == 0:
            collectionpos.put((enc.position/(16*256*4))*6.283)
```

```
        #store position, calculate velocity (rad/s), store velocity
        elif counter <= 9300: #30 seconds/task period
            vel = ((enc.delta/(16*256*4))/0.01)*6.283
            pos = (enc.position/(16*256*4))*6.283
            uart.write(f'{pos},{vel}\r\n')
        #stop recording, print position and velocity queues to uart
        else:
            collect.put(0)
            pass
    else:
        pass

#opens or closes the loop - mode 1 = cl, mode 0 = ol
def toggle_mode(mode):
    if mode.get() == 1:
        mode.put(0)
    elif mode.get() == 0:
        mode.put(1)

#sets the closed loop gain of the controller
def cl_gain(clgain):
    gain.put(clgain)

#sets the closed loop velocity of the controller
def cl_vel(clvel):
    des_vel.put(clvel)

"""
MOTOR TASK
Controls motors based off of what UI tells it to do
    - Does not interact with VCP or UART
    - Reads shares for UI orders
    - Writes motor data to shares
"""
def motor():
    state_motor = 0
    while True:
        # intialization of the motors
        if(state_motor == 0):
            mot_A.enable()
            mot_B.enable()
            state_motor = 1
        # motor running in OL mode
        elif(state_motor == 1):
            mot_duty = duty.get()
```

```
            # if selected motor is motor A and OL is true
            if(selected_motor.get() == 0 and mode.get() == 0):
                if(duty_flag.get()):
                    set_dc(mot_A, mot_duty)
                    state_motor = 1
            # if selected motor is motor B and OL is true
            elif(selected_motor.get() == 1 and mode.get() == 0):
                if(duty_flag.get()):
                    set_dc(mot_B, mot_duty)
                    state_motor = 1
            #CL is true
            elif(mode.get() == 1):
                state_motor = 2

        # motor running in CL mode
        elif(state_motor == 2):
            if(selected_motor.get() == 0):
                #vcp.write('motor a active in cl')
                enc1.update()
                measured_vel = ((enc1.delta/(16*256*4))/0.01)*6.283
                new_dutyA = prop_controller.p_eff(measured_vel)
                set_dc(mot_A,new_dutyA)
                state_motor = 2

            # if selected motor is motor B and CL is true
            elif(selected_motor.get() == 1):
                #vcp.write('motor b active in CL')
                enc2.update()
                measured_vel = ((enc2.delta/(16*256*4))/0.01)*6.283
                new_dutyB = prop_controller.p_eff(measured_vel)
                set_dc(mot_B, new_dutyB)
                state_motor = 2

            # if either motor is selected and OL is true
            elif(mode.get() == 0):
                state_motor = 1
                new_duty = 0
                set_dc(mot_B, new_duty)
                set_dc(mot_A, new_duty)
        yield(0)

"""
ENCODER TASK
Does encoder stuff
    - Does not interact with VCP or UART
```

```
    - Reads shares for UI orders
    - Writes encoder data to shares
"""
def encoder():
    state_encoder = 0
    while True:
        # intialize the both encoders
        if(state_encoder == 0):
            state_encoder = 1

        # wait state for the encoder, check if UI needs position or velocity
        elif(state_encoder == 1):
            # if enc. A is selected, wait
            #vcp.write('encoder state 1')
            if(selected_encoder.get() == 0):
                enc1.update()
                #check if it's an easy task flag, if so do the easy task
                if(g_flag.get() == 0 and r_flag.get() == 0):
                    if(z_flag.get() == 1):
                        zero_encoder(enc1)
                        z_flag.put(0)
                        state_encoder = 1
                    elif(p_flag.get() == 1):
                        print_encoder(enc1)
                        p_flag.put(0)
                        state_encoder = 1
                    elif(d_flag.get() == 1):
                        enc_delta(enc1)
                        d_flag.put(0)
                        state_encoder = 1
                    elif(v_flag.get() == 1):
                        vcp.write('\r\n> Velocity: ' +
str(((enc1.delta/(16*256*4))/0.01)*6.283) + ' rad/s')
                        v_flag.put(0)
                        state_encoder = 1
                    else:
                        state_encoder = 1
                #if not an easy flag, check if it's a data collection or SR query
                else:
                    if g_flag.get() == 1:
                        state_encoder = 2
                    else:
                        state_encoder = 3
            # if enc. B is selected, wait
            else:
```

```
                enc2.update()
                #check if it's an easy task flag, if so do the easy task
                if(g_flag.get() == 0 and r_flag.get() == 0):
                    if(z_flag.get() == 1):
                        zero_encoder(enc2)
                        z_flag.put(0)
                        state_encoder = 1
                    elif(p_flag.get() == 1):
                        print_encoder(enc2)
                        p_flag.put(0)
                        state_encoder = 1
                    elif(d_flag.get() == 1):
                        enc_delta(enc2)
                        d_flag.put(0)
                        state_encoder = 1
                    elif(v_flag.get() == 1):
                        vcp.write('\r\n> Velocity: ' +
str(((enc2.delta/(16*256*4))/0.01)*6.283) + ' rad/s')
                        v_flag.put(0)
                        state_encoder = 1
                    else:
                        state_encoder = 1
                #if not an easy flag, check if it's a data collection or SR query
                else:
                    if(g_flag.get() == 1):
                        state_encoder = 2
                    else:
                        state_encoder = 3
        #if collection query, collect data for 30 seconds (blocking)
        elif(state_encoder == 2):
            counter = 0
            if(selected_encoder.get() == 0):
                while(counter <= 9300):
                    enc1.update()
                    collect_data(enc1,counter)
                    counter += 1
                g_flag.put(0)
                state_encoder = 1
                vcp.write('Done Collecting Data\r\n')
            else:
                while(counter <= 9300):
                    enc2.update()
                    collect_data(enc2,counter)
                    counter += 1
                vcp.write('Done Collecting Data\r\n')
```

```
                g_flag.put(0)
                state_encoder = 1
        #if SR query, do an SR (blocking)
        elif(state_encoder == 3):
            counter = 0
            if(selected_encoder.get() == 0):
                while(counter <= 400):
                    enc1.update()
                    step_response(mot_A,enc1,counter)
                    counter += 1
                r_flag.put(0)
                state_encoder = 1
            else:
                while(counter <= 400):
                    enc2.update()
                    step_response(mot_B,enc2,counter)
                    counter += 1
                r_flag.put(0)
                state_encoder = 1
        yield(0)

"""
UI TASK

Takes user input, parses it, and tells the other tasks what to do
    - Receives user input through VCP
    - Reads user input
    - Sets flags/shares telling other tasks what to do, based off of user input
    - Prints things to UART and VCP
    - Blocks user input while something important is happening
    - NOT SURE IT MAKES SENSE TO DO SIMPLE MOTOR/ENCODER STUFF HERE (LIKE CHECK
MOTOR POSITION, SPEED, ZERO ENCODER ETC)
"""
def ui(vcp, fullin):
    state = 1
    while(True):
        # state in UI to choose the motor that is to be run (A or B)
        if state == 1:
            if(vcp.any()):
                vcp.write('Choose Motor A or B\r\n> ')
                input = vcp.read(1).decode()
                if (input in {'a','A'}):
                    vcp.write(input + '\r\n')
                    vcp.write(ui_help + '> ')
                    selected_motor.put(0)
```

```
                    selected_encoder.put(0)
                    state += 1
                elif (input in {'b','B'}):
                    vcp.write(input + '\r\n')
                    vcp.write(ui_help + '> ')
                    selected_motor.put(1)
                    selected_encoder.put(1)
                    state += 1
                else:
                    vcp.write(input + '\r\n')
                    vcp.write('> Invalid Motor selected\r\n> ')
                    state = 1

        # state in UI to prompt the user to select one function out the avaliable
functions from the table
        elif(state == 2):
                if(vcp.any()):
                    ui_input = vcp.read(1).decode()
                    vcp.write(ui_input + '\r\n')
                    if (ui_input in {'h','H'}):
                        vcp.write(ui_help)
                        vcp.write('> ')
                    elif (ui_input in {'z','Z'}):
                        vcp.write('> Zeroing position of the encoder\r\n')
                        vcp.write('\r\n> ')
                        state = 3
                    elif (ui_input in {'p','P'}):
                        vcp.write('> Printing pos. of encoder to UART\r\n')
                        vcp.write('\r\n> ')
                        state = 4
                    elif (ui_input in {'d', 'D'}):
                        vcp.write('> Printing delta of encoder \r\n')
                        vcp.write('\r\n> ')
                        state = 5
                    elif (ui_input in {'v', 'V'}):
                        vcp.write('> Printing velocity \r\n')
                        vcp.write('\r\n> ')
                        state = 6
                    elif (ui_input in {'m','M'}):
                        vcp.write('> Set the duty cycle for the motor\r\n')
                        state = 7
                    elif (ui_input in {'g','G'}):
                        vcp.write('> Collecting data from open loop\r\n')
                        vcp.write('\r\n> ')
                        state = 8
```

```
                    elif (ui_input in {'c','C'}):
                        vcp.write('> Changed Mode to Closed Loop\r\n')
                        vcp.write('\r\n> ')
                        state = 9
                    elif (ui_input in {'k','K'}):
                        vcp.write('> Choose kp for the controller\r\n')
                        vcp.write('\r\n> ')
                        state = 10
                    elif (ui_input in {'s','S'}):
                        vcp.write('> Choose velocity target\r\n')
                        vcp.write('\r\n> ')
                        state = 11
                    elif (ui_input in {'r','R'}):
                        vcp.write('> Trigger a step response\r\n')
                        vcp.write('\r\n> ')
                        state = 12
                    elif (ui_input in {'o','O'}):
                        vcp.write('> Changed mode to Open Loop\r\n')
                        vcp.write('\r\n> ')
                        state = 13
                    elif (ui_input in {'a', 'A'}):
                        selected_encoder.put(0)
                        selected_motor.put(0)
                        vcp.write('> Motor switched to Motor A\r\n')
                    elif (ui_input in {'b', 'B'}):
                        selected_encoder.put(1)
                        selected_motor.put(1)
                        vcp.write('> Motor switched to Motor B\r\n')
                    else:
                        vcp.write('> Invalid input! Press h/H for help\r\n')
                        state = 2

        # state to zero the encoder reading
        elif(state == 3):
            if(selected_motor.get() == 0):
                z_flag.put(1)
                vcp.write('Zeroed the Encoder for Motor A\r\n> ')
            else:
                z_flag.put(1)
                vcp.write('Zeroed the Encoder for Motor B\r\n> ')
            state = 2

        # state to print the current position of the encoder for the selected
motor
        elif(state == 4):
```

```
            if(selected_motor.get() == 0):
                p_flag.put(1)
                vcp.write('Printed the Encoder Pos. to UART for Motor A\r\n> ')
            else:
                p_flag.put(1)
                vcp.write('Printed the Encoder Pos. to UART for Motor B\r\n> ')
            state = 2

        # state to print the current encoder delta to for the selected motor
        elif(state == 5):
            if(selected_motor.get() == 0):
                d_flag.put(1)
                vcp.write('Printed the delta of Encoder to UART for Motor A\r\n>
')
            else:
                d_flag.put(1)
                vcp.write('Printed the delta of Encoder to UART for Motor B\r\n>
')
            state = 2

        # state to print the current angular velocity of the selected motor (need
to implement bttr method)
        elif(state == 6):
            if(selected_motor.get() == 0):
                v_flag.put(1)
                vcp.write('Printed current angular velocity of Motor A\r\n> ')
            else:
                v_flag.put(1)
                vcp.write('Printed current angular velocity of Motor B\r\n> ')
            state = 2

        # state to set the duty cycle for the selected motor, this task sets a
flag to the motor subtask which then sets the DC
        elif(state == 7):
            numerical_input('> Input your desired duty cycle\r\n> ',-100,100,
fullin)
            if numinput.any():
                duty.put(numinput.get())
                vcp.write('\r\nMotor ' + str(selected_motor.get()) + ' duty cycle
set to: ' + str(duty.get()) + '\r\n> ')
                duty_flag.put(1)
                state = 2
            else:
                state = 7
```

```
        # state to collect data in the open loop mode (Need to implement method
to write one line of data from vel and pos to UART at schduler freq, as a
subtask)
        elif(state == 8):
            if(selected_motor.get() == 0):
                g_flag.put(1)
                collect.put(1)
            else:
                g_flag.put(1)
                collect.put(1)
            state = 2
        # state to change mode to closed loop
        elif(state == 9):
            mode.put(1)
            vcp.write('Mode switched to closed loop\r\n> ')
            state = 2

        # state to allow the user to select a gain for the proportional
controller
        elif(state == 10):
            numerical_input('Input your desired proporitonal gain\r\n>
',0,500,fullin)
            if numinput.any():
                kp_gain = numinput.get()
                cl_gain(kp_gain)
                vcp.write('\r\n> Kp set to: ' + str(kp_gain) + '\r\n')
                prop_controller.set_kp(gain.get())
                state = 2
            else:
                pass

        # state to allow the user to select a target velocity for the motor
        elif(state == 11):
            numerical_input('Input your desired motor speed (rad/s)\r\n> ', -50,
50,fullin)
            if numinput.any():
                    target_vel = numinput.get()
                    cl_vel(target_vel)
                    vcp.write('\r\n> Target Velocity set to: ' + str(target_vel)
+ ' rad/s\r\n> ')
                    state = 2
                    measVel = des_vel.get()
                    prop_controller.set_velTarget(measVel)
            else:
                pass
```

```
        # state in which step response is done and data is send to uart
        elif(state == 12):
            vcp.write('Step Response triggered, data will be sent to UART\r\n> ')
            if selected_motor.get() == 0:
                r_flag.put(1)
            else:
                r_flag.put(1)
            state = 2

        # state in which controller switches to OL
        elif(state == 13):
            mode.put(0)
            vcp.write('Mode switched to open loop\r\n> ')
            state = 2
        yield(0)

micropython.alloc_emergency_exception_buf(100)
if __name__ == "__main__":

    #make da uart
    uart = pyb.UART(2,115200)
    uart.init(115200, bits=8, parity=None, stop=1)
    #make shares/queues
    #start = task_share.Share ('h', thread_protect = False, name =
"start_flag")                    #flag telling motor to go (((verify this)))
    #kp = task_share.Share ('h', thread_protect = False, name =
"kp")                              #proportional controller effort (kp)
    des_vel = task_share.Share ('h', thread_protect = False, name =
"vel")                             #desired velocity
    gain = task_share.Share ('h', thread_protect = False, name =
"gain")                            #closed loop controller gain
    collect = task_share.Share ('h', thread_protect = False, name =
"collect_flag")                 #flag telling collect_data task to go
    mode = task_share.Share('h', thread_protect = False, name =
"mode_flag")                       #flag for telling which mode we are in OL/CL
(default OL)
    duty_flag = task_share.Share('h',  thread_protect = False, name =
"duty_cycle_flag")
    duty = task_share.Share('h', thread_protect = False, name =
"duty")                            #duty cycle for motor
    selected_motor = task_share.Share('h',  thread_protect = False, name =
"selected_motor")        #flag for telling which motor was selected
    selected_encoder = task_share.Share('h',  thread_protect = False, name =
"selected_encoder")    #flag for telling which encoder was selected
```

```
    #eff = task_share.Share('h',  thread_protect = False, name =
"eff")                              #controller effort
    prompted = task_share.Share('h',  thread_protect = False, name =
"prompted")                     #flag for telling if the user was prompted for num
input yet
    numinput = task_share.Queue('b', 4, thread_protect = False, name =
"numinput")                   #numerical user input
    mode.put(0)
    collect.put(0)
    # open loop control flags
    z_flag = task_share.Share('h', thread_protect = False, name =
"z_flag")                         #flag to check if zeroing the encoder is true
or false
    p_flag = task_share.Share('h', thread_protect = False, name =
"p_flag")                         #flag to check if pos. needs to be printed to
UART
    d_flag = task_share.Share('h', thread_protect = False, name =
"d_flag")                         #flag to check if delta needs to be printed to
UART
    v_flag = task_share.Share('h', thread_protect = False, name =
"v_flag")                         #flag to check if vel. needs to be printed to
UART
    g_flag = task_share.Share('h', thread_protect = False, name =
"g_flag")                         #flag to collect data from encoder for 30
seconds
    # closed loop control flags
    k_flag = task_share.Share('h', thread_protect = False, name = "k_flag")
    s_flag = task_share.Share('h', thread_protect = False, name = "s_flag")
    r_flag = task_share.Share('h', thread_protect = False, name = "r_flag")
    # UI blocking flag
    #working = task_share.Share('h', 4, thread_protect = False, name = "working")
    #working.put(False)

    collectionpos = task_share.Queue ('f', 3000, thread_protect = False,
overwrite = False, name = "C_pos")     #stores position data collected in
collect_data
    collectionvel = task_share.Queue ('f', 3000, thread_protect = False,
overwrite = False, name = "C_vel")     #stores velocity data collected in
collect_data

    # Create the tasks. If trace is enabled for any task, memory will be
    # allocated for state transition tracing, and the application will run out
    # of memory after a while and quit. Therefore, use tracing only for
    # debugging and set trace to False when it's not needed
    fullin = list()
```

```
    vcp = pyb.USB_VCP()
    task_ui = cotask.Task(ui(vcp,fullin), name = 'Task_Ui', priority = 1, period
= 5, profile = True, trace = False)
    task_motor = cotask.Task(motor(), name = 'Task_Motor', priority = 2, period =
10, profile = True, trace = False)
    task_encoder = cotask.Task(encoder(), name = 'Task_Encoder', priority = 2,
period = 10, profile = True, trace = False)
    cotask.task_list.append(task_ui)
    cotask.task_list.append(task_motor)
    cotask.task_list.append(task_encoder)

    #making objects:
    #encoders
    print('Encoder Enabled')
    tim_enc1 = Timer(4,period = 65535, prescaler = 0)
    enc1 = Encoder.Encoder(tim_enc1,Pin.cpu.B7,Pin.cpu.B6)
    tim_enc2 = Timer(8,period = 65535, prescaler = 0)
    enc2 = Encoder.Encoder(tim_enc2,Pin.cpu.C7,Pin.cpu.C6)
    #motors
    print('Motor Enabled')
    tim_A = Timer(3, freq = 20000)
    tim_B = Timer(2, freq = 20000)
    mot_A = L6206.L6206(tim_A, Pin.cpu.B4, Pin.cpu.B5,Pin.cpu.A10)
    mot_B = L6206.L6206(tim_B, Pin.cpu.A0, Pin.cpu.A1,Pin.cpu.C1)
    #misc
    prop_controller = cl.cl()
    '''
    step_A = stepresponse(mot_A,enc1)
    step_B = stepresponse(mot_B,enc2)
    '''

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

    vcp.write('> Type A for Motor A, Type B for Motor B\r\n')
    try:
        while(True):
            cotask.task_list.pri_sched()
    except KeyboardInterrupt:
        print('\n\r> Program Terminated')
```