|  |  |  |  |
| --- | --- | --- | --- |
| **MEMORANDUM** | | A picture containing graphical user interface  Description automatically generated | |
|  |  | | |
| **To:** | Professor Charlie Refvem, Department of Mechanical Engineering, Cal Poly SLO | | |
|  | [crefvem@calpoly.edu](mailto:crefvem@calpoly.edu) | | |
| **From:** | Jack Butler | | Tarun Ganamur |
|  | [jbutle10@calpoly.edu](mailto:jbutle10@calpoly.edu) | | 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:

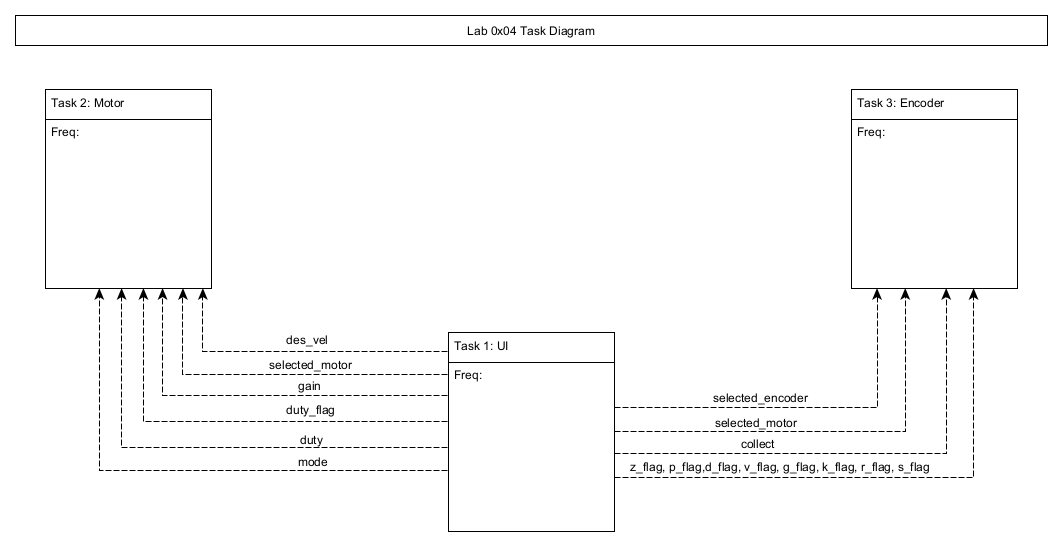


Figure 1. Task Diagram

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:

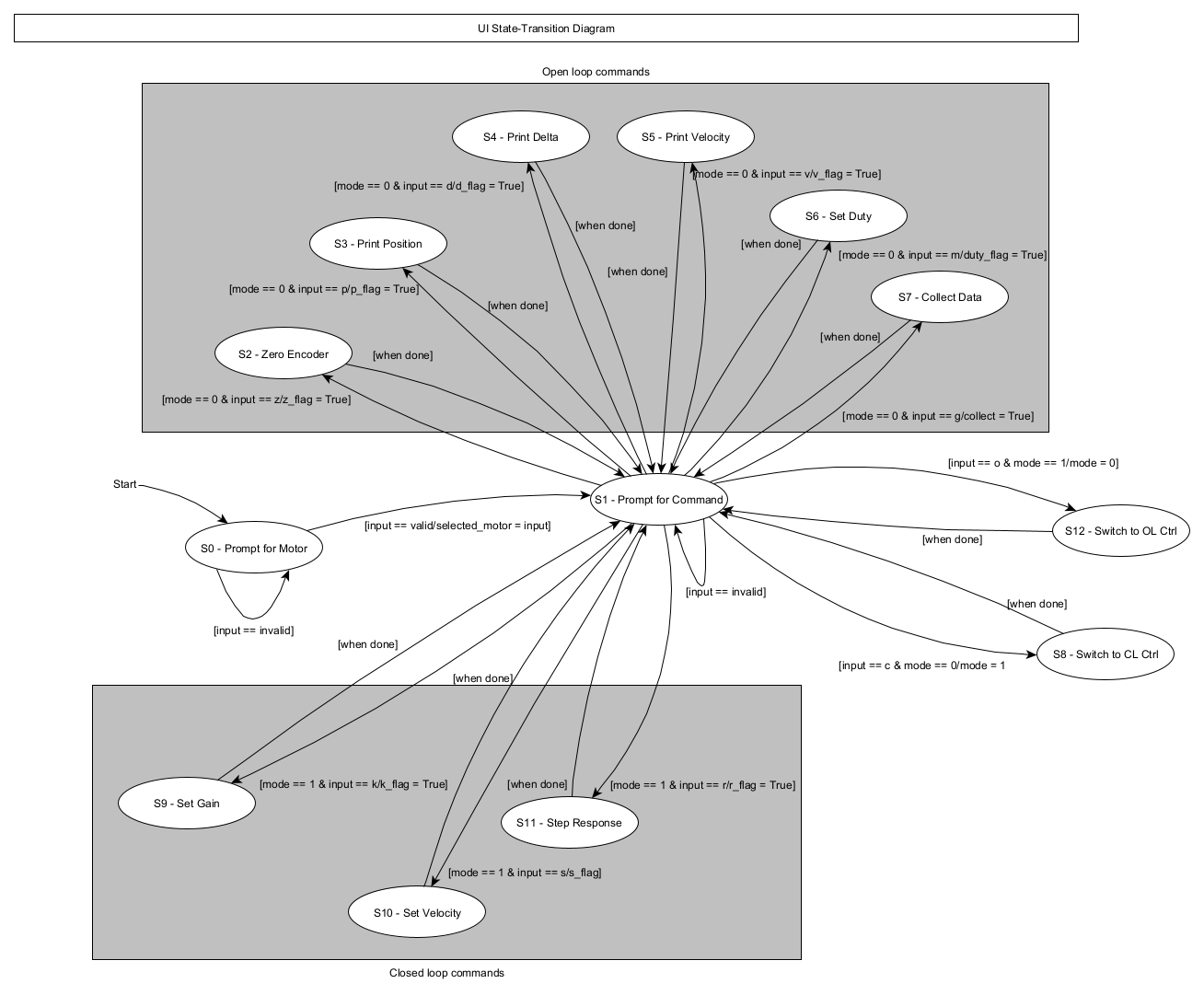


Figure 2. State-transition diagram for UI task

*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:

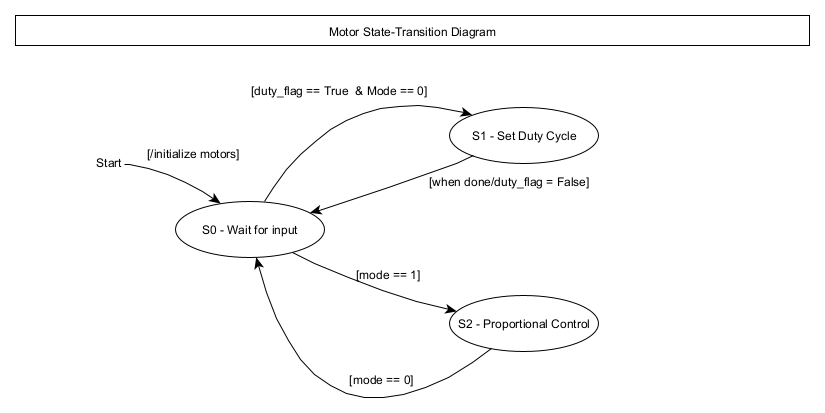


Figure 3. State-transition diagram for Motor task

*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:

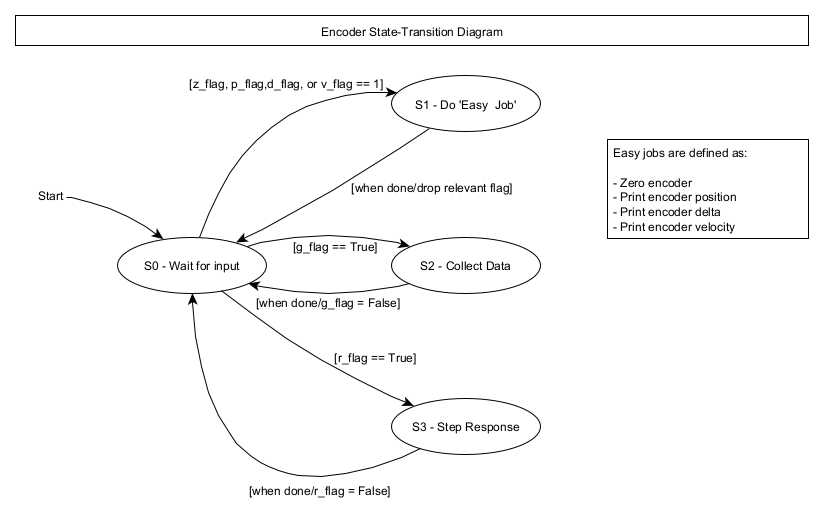


Figure 4. State-transition diagram for Encoder task

**Step Response Tuning**

To tune, we set the desired velocity to 20 rad/s and varied K\_p.

Chart, histogram

Description automatically generated

Figure 5. Step response at K\_p = 1

Chart, box and whisker chart

Description automatically generated

Figure 6. Step response at K\_p = 3

Chart, box and whisker chart

Description automatically generated

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')