EE C128 Tello Final Report

Braeden Benedict, Kathleen Chang, Irving Fang

12/8/20

Contents

1 Concept

Our final project has the Tello navigate a set of slalom poles using PID control and Aruco tags for position measurements. The performance goal was to minimize the amount of time it took for the Tello to complete the course. Our initial performance goal was to do this and make tight turns around the Slalom poles. We implemented this by tuning PID controllers for the forward/back, left/right, up/down, and yaw control to make the Tello quickly fly to intermediate checkpoints through the course.

2 Control Methods

2.1 Strategy

Four PID controllers for yaw, up/down, left/right, and forward/backward motion are used. We retrieve state information from the Tello data queue for the yaw and Aruco tag position vector for the up/down, left/right, and forward/backward and use these to perform feedback control. For each controller, the derivative and integral portion are calculated based on the current and previous time-step information. The contribution of the P, I, D components were summed and used as the control signal.

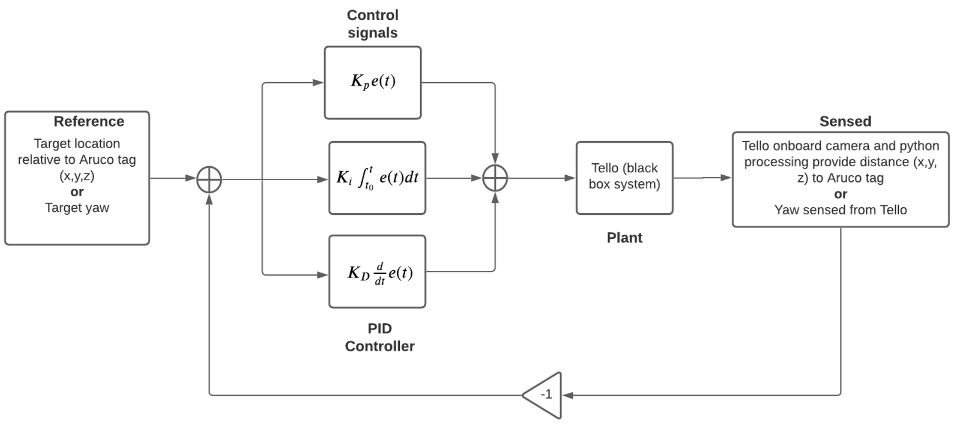
The PID gain parameters are listed in the table below. The yaw uses the same controller as was designed in lab 5b, since that worked and we are only using the yaw control to keep the yaw at 0. This was originally directly based on the measured frequency response of the black box system, but ultimately required manual tuning. For that reason, we chose to just use a manual tuning procedure to determine the left/right, forward/back, up/down parameters for each control direction. First, we increased the Proportional gain until there was overshoot in response to a step input and the control values were approaching their saturation values. We then increased the Integral gain to reduce steady state error. Finally, we increased the derivative gain to reduce overshoot.

	P	I	D
Yaw	2.5	1	0.1
Left/Right	100	15	50
Forward/Back	90	15	50
Up/Down	100	20	10

Navigation through the course was essentially achieved by having the Tello control respond to a series of 3 step responses, with the step position located in front of and slightly to the side of each slalom pole/Aruco tag. We will call this intermediate position a checkpoint. We use only the Aruco tag from the nearest pole to calculate current position. When the Tello has come sufficiently close to the checkpoint (0.15m), the current Tello position is now calculated using the distance from the next Aruco tag, and the reference position also changes so the Tello will navigate to the proper orientation relative to the next tag.

In order to successfully navigate around the pole, the Tello has to fly forward before it can begin flying left/right. For that reason, we temporarily disable the left/right control for 1.4sec after the next Aruco tag is identified. This allows the Tello to fly past the first pole and not crash into it.

2.2 Control Block Diagram



2.3 Information from other blocks

The Tello camera streamed a video feed to the computer, which allowed our Python code using OpenCV to identify Aruco tags and calculate the distance of the Tello relative to each tag. There was some noticeable latency in this stream. Since our understanding was that the Tello has a 0.1s update rate, this was the rate at which we read this state information and updated the controller values.

3 Results

3.1 Results Reliability

Once the PID parameters were tuned and some issues that will be discussed later were worked out, the Tello was able to complete its journey through the course about 75% of the time. It passes to the left of the first pole, to the right of the second pole, stops in front of the last pole, and lands.

3.2 Performance Metrics Proposed

Our primary performance metric proposed was speed through the course. A similar metric that we initially proposed was how tight we could make turns around the slalom poles, which would serve to reduce time. Ultimately, we did not aggressively pursue this because doing so with our control scheme would have required us to have a reliable estimate of the drone to pole distance when we were close to the pole. Due to to the limited viewing angle of the camera, we could only see the Aruco tag from a certain distance away, which made this impractical. Therefore, we worked to optimize our primary performance metric of speed.

3.3 Plots

The plots below show the left/right, up/down, and forward/back position, reference, and error signals. The first vertical line indicates when the Tello starts the course, and the vertical lines afterward indicate when the Tello has reached each checkpoint. Both the current position and the reference can change when a checkpoint is reached since we are now using a new Aruco tag for position estimates.

We observe that we are using the full range of the controller values, especially for the forward/back control, without major saturation. In Figure 1, the 1.4 second dead period in the control signal can be seen during the time when the drone is only flying forward so that it can clear the slalom pole. Finally, we also show the velocity of the Tello as measured by the Tello itself in Figure 4. These correspond well with the position changes given by the Aruco tag system. Note that the x,y,z in this plot coresopnd the the same x,y,z and in the previous plots.

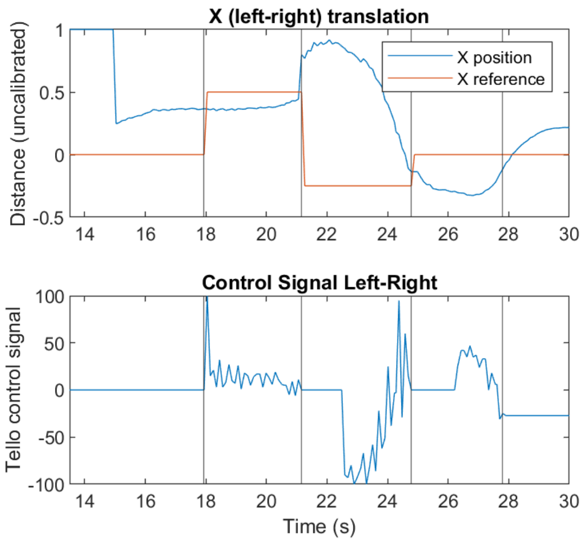


Figure 1: Left-right

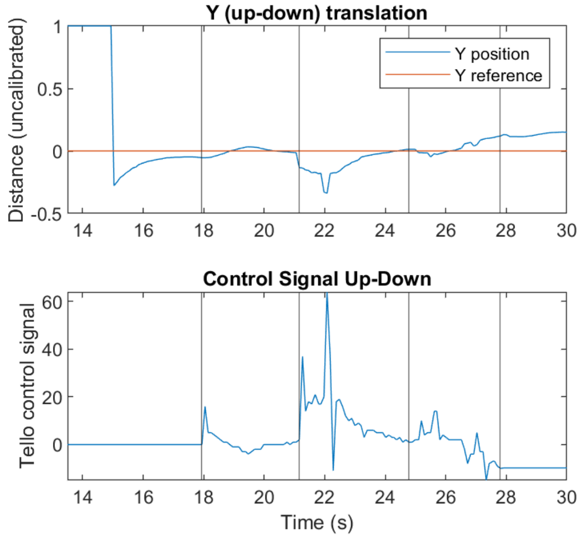


Figure 2: Up-down

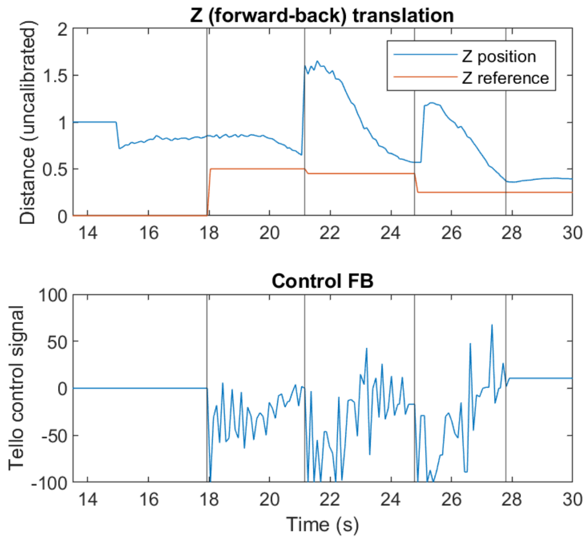


Figure 3: Forward-back

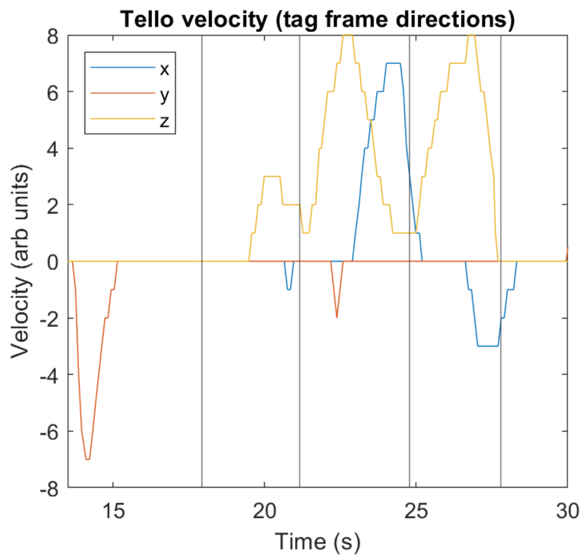


Figure 4: Translational Velocity

3.4 Performance Metrics

Before tuning the PID parameters, we were able to complete the course, but very slowly. Ultimately, we were able to complete the course from the beginning of the control to the landing signal in 9.9 seconds. We did not set an initial goal because we weren't really sure what the course would look like or what was realistically possible, but we will retroactively set a 10 second goal, which we clearly hit perfectly!

4 Youtube Video

https://youtu.be/OK7-oIf_UHQ
Drone Perspective: **0:00 - 0:27**
0:00 - 0:14 Takeoff and pass the first pole, t = 14-21 on graphs in section 3
0:14 - 0:18 Move sideways, t = 21
0:18 - 0:22 Pass the second pole, t = 21-25
0:22 - 0:23 Center with the final slalom pole, t = 25
0:23 - 0:27 Land, t = 25:end

Viewer Perspective: **0:27 - 0:47**

5 Discussion

5.1 Performance flaws

We were fairly happy with the performance achieved since we completed the course fairly quickly. As previously mentioned, we did not attempt to optimize the turn radius around the poles due to not being able to track the position of the drone relative to the pole at that point.

One flaw in our design that became apparent was simply using these checkpoints as step response inputs to fly the drone through the course. It is particularly evident in Figure 1 and 3 that the time to reach the first checkpoint takes essentially as long as the time to reach the next two checkpoints even though it has a much shorter distance to travel. Since the rise time for a step response is not dependent on the actual magnitude of the step, it takes basically the same amount of time as the next two steps. Although it works and puts to use some of the concepts we have learned in class, there are definitely better ways that the drone could be navigated through the course more quickly.

5.2 Issues encountered

The largest issues encountered had to do with getting the software up and running to stream camera data and get data from the Aruco tags. This left us with relatively little time to actually dive into making the controls better. We had discussed implementing state feedback since we do have access to both position and velocity, but ultimately we did not have time to do that. While these are the trials and tribulations of a real project as opposed to a lab, working out how to use threads for camera streaming, Aruco tag ID, etc seems like it should be outside the scope of this class.

Another issue with the Tello is its short battery life. It makes it difficult to work for more than 30 minutes at a time, even when turning of the drone and plugging it in briefly between runs.

5.3 Final Thoughts

This was a fun project, even if it perhaps was not as educationally useful as designing the inverted pendulum controller. A more advanced access with direct access to the controllers would be nice to have assuming that the code to interface with them was already in place.

6 Software System Description

6.1 Control architecture

Yaw, left/right, forward/back, up/down each had a controller. The controller calculates the error based on the reference and current measurement, the integral based on the update interval and previous integrated error, and the derivative based on the previous time step's error and current error. Previous values for yaw, left-right, forward-back, up-down in variables. The update rate for state information was 0.05 seconds. The queue that collects state data and the Aruco tag code that detects position is used in the controller. In order to track which aruco tag should be targeted, we had a list of unique tag ID numbers in order of passing and once the Tello sensed it was close enough to the tag, we moved on to the next tag. Once all the tag checkpoints are reached, the drone lands.

6.2 Software packages

We are using OpenCV to track aruco tags from the tongplw [1] repository shared by Professor Fearing. The tracking algorithm is 3D and identifies the x, y, and z axis based on the orientation of the aruco code. To get the Tello to livestream its position, we are using f41ardu [2]'s repository and TelloKeyboardCommands.py from Professor Fearing. The TelloKeyboardCommands.py lets us send messages to the Tello, and f41ardu's repository has code that sends the camera output to a window on the computer running the code.

7 Code

```
# This script is part of our course on Tello drone programming
# https://learn.droneblocks.io/p/tello-drone-programming-with-python/

# Import the necessary modules
import socket
import threading
import time
from time import sleep
import sys
import numpy as np
from queue import Queue
from queue import LifoQueue
import os
import cv2 as cv
import math


State_data_file_name = \'statedata.txt\'
index = 0
reference_yaw = 0.0     # Reference yaw signal
reference_x = 0.0
reference_y = 0.0
reference_z = 0.0
control_LR = 0      # Control input for left/right
control_FB = 0      # Cotnrol input for forward/back
control_UD = 0      # Control input for up/down
control_YA = 0      # Control input for yaw
INTERVAL = 0.05  # update rate for state information
start_time = time.time()
dataQ = Queue()
stateQ = LifoQueue() # have top element available for reading present state by control loop

tvec_GLOBAL = [1.0,1.0,1.0]
rvec_GLOBAL = [1.0,1.0,1.0]
target_num_GLOBAL = 0
tag_list = [1,10,33]

# IP and port of Tello for commands
tello_address = ('192.168.10.1', 8889)
# IP and port of local computer
local_address = ('', 8889)
# Create a UDP connection that we'll send the command to
CmdSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
CmdSock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# Bind to the local address and port
CmdSock.bind(local_address)

###################
# socket for state information
local_port = 8890
StateSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # socket for sending cmd
StateSock.bind(('', local_port))
CmdSock.sendto('command'.encode('utf-8'), tello_address)   # command port on Tello

```

[1] https://github.com/tongplw/OpenCV-ArucoDetection/blob/master/src/ 7%20Get%20ID%2C%20tvec%20and%20rvec.py
[2] https://gist.github.com/f41ardu/b75da46dae383e5c835295600deef604

```
def writeFileHeader(dataFileName):
    fileout = open(dataFileName,'w')
    #write out parameters in format which can be imported to Excel
    today = time.localtime()
    date = str(today.tm_year)+'/'+str(today.tm_mon)+'/'+str(today.tm_mday)+'  '
    date = date + str(today.tm_hour) +':' + str(today.tm_min)+':'+str(today.tm_sec)
    fileout.write('Data file recorded ’ + date + ’\n')
    # header information
    fileout.write('  index,   time,    ref,ctrl\_LR,ctrl\_FB,ctrl\_UD,ctrl\_YA,  pitch,   roll,    yaw,
        vgx,    vgy,    vgz,   templ,   temph,    tof,      h,    bat,   baro,   time,    agx,    agy,
        agz\n\r')
    fileout.close()


def writeDataFile(dataFileName):
    fileout = open(State_data_file_name, 'a')  # append
    print('writing data to file')
    while not dataQ.empty():
        telemdata = dataQ.get()
        np.savetxt(fileout , [telemdata], fmt='%7.3f', delimiter = ',')  # need to make telemdata a list
    fileout.close()


def report_tag(str,index):
    telemdata=[]
    telemdata.append(index)
    telemdata.append(time.time()-start_time)
    telemdata.append(reference_yaw)
    telemdata.append(control_LR)
    telemdata.append(control_FB)
    telemdata.append(control_UD)
    telemdata.append(control_YA)
    data = str.split(';')
    data.pop() # get rid of last element, which is \\r\\n
    for value in data:
        temp = value.split(':')
        if temp[0] == 'mpry': # roll/pitch/yaw
            temp1 = temp[1].split(',')
            telemdata.append(float(temp1[0]))     # roll
            telemdata.append(float(temp1[1]))     # pitch
            telemdata.append(float(temp1[2]))     # yaw
            continue
        quantity = float(value.split(':')[1])
        telemdata.append(quantity)
    telemdata.append(tvec_GLOBAL[0])
    telemdata.append(tvec_GLOBAL[1])
    telemdata.append(tvec_GLOBAL[2])
    telemdata.append(rvec_GLOBAL[0])
    telemdata.append(rvec_GLOBAL[1])
    telemdata.append(rvec_GLOBAL[2])
    telemdata.append(reference_x)
    telemdata.append(reference_y)
    telemdata.append(reference_z)
    dataQ.put(telemdata)
    stateQ.put(telemdata)
    if (index %100) == 0:
        print(index, end=',')

# Send the message to Tello and allow for a delay in seconds
def send(message):
  # Try to send the message otherwise print the exception
  try:
    CmdSock.sendto(message.encode(), tello_address)
    # print(Sending message:   + message)
  except Exception as e:
    print(Error sending:   + str(e))
```

```

# receive state message from Tello
def rcvstate():
    print('Started rcvstate thread')
    index = 0
    while not stateStop.is_set():

        response, _ = StateSock.recvfrom(1024)
        if response == 'ok':
            continue
        report_tag(str(response),index)
        sleep(INTERVAL)
        index +=1
    print('finished rcvstate thread')

def recordingSetup(filename, cap):
    width = int(cap.get(cv.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv.CAP_PROP_FRAME_HEIGHT))
    fps = cap.get(cv.CAP_PROP_FPS)
    fourcc = cv.VideoWriter_fourcc(*'DIVX')
    return cv.VideoWriter(filename + '.avi',fourcc, fps, (width,height))

def camera():
    global rvec_GLOBAL
    global tvec_GLOBAL
    global target_num_GLOBAL
    global tag_list

    print('Started camera thread')
    path = os.path.abspath('..')
    fname = path + /slalomTello/res/calibration_parameters.txt #NEED ONE HERE
    cap = cv.VideoCapture(udp://@0.0.0.0:11111)

    #calibration parameters
    f = open(fname, r)
    ff = [i for i in f.readlines()]
    f.close()
    from numpy import array
    parameters = eval(''.join(ff))
    mtx = array(parameters['mtx'])
    dist = array(parameters['dist'])

    filename = Outputs/output_vid #NEED ONE HERE
    print(filename)
    vidRecorder = recordingSetup(filename, cap)

    tvec = [[[0.0, 0.0, 0.0]]]
    rvec = [[[0.0, 0.0, 0.0]]]

    aruco_dict = cv.aruco.Dictionary_get(cv.aruco.DICT_4X4_250)
    parameters = cv.aruco.DetectorParameters_create()
    parameters.adaptiveThreshConstant = 10

    print('beginning camera thread loop')
    ret = False

    dist_threshold = 0.4 # if tello is within this distance, we no longer look at the tag
    while(True):
        ret, frame = cap.read()
        if(ret):
            gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
            corners, ids, rejectedImgPoints = cv.aruco.detectMarkers(gray, aruco_dict, parameters=parameters)
            font = cv.FONT_HERSHEY_SIMPLEX
            if np.all(ids != None):
                rvec, tvec, _ = cv.aruco.estimatePoseSingleMarkers(corners, 0.05, mtx, dist)
```

```
                for i in range(0, ids.size):
                    cv.aruco.drawAxis(frame, mtx, dist, rvec[i], tvec[i], 0.1)

                    # show translation vector on the corner
                    font = cv.FONT_HERSHEY_SIMPLEX
                    text = str([round(i,5) for i in tvec[i][0]])
                    position = tuple(corners[i][0][0])
                    cv.putText(frame, text, position, font, 0.4, (0, 0, 0), 1, cv.LINE_AA)
                cv.aruco.drawDetectedMarkers(frame, corners)

                curr_id = tag_list[target_num_GLOBAL] #target tag
                ids_list = np.ndarray.tolist(ids.flatten()) #turn it into a list

                try:
                    curr_index = ids_list.index(curr_id) #index of the target tag
                    curr_dist = np.linalg.norm(tvec[curr_index][0]) #get the distance

                    tvec_GLOBAL = tvec[curr_index][0] #
                    rvec_GLOBAL = rvec[curr_index][0]
                except:
                    pass

                if len(tag_list) == 0:
                        break
            else:
                pass

            vidRecorder.write(frame)
            cv.imshow('frame',frame)
            if cv.waitKey(1) & 0xFF == ord('q'):
                break

    print('finished stream thread')
    cap.release()
    cv.destroyAllWindows()
    print('Ended stream thread')

# Receive the message from Tello
def receive():
  # Continuously loop and listen for incoming messages
  while True:
    # Try to receive the message otherwise print the exception
    try:
      response, _ = CmdSock.recvfrom(128)
      print(Received message:   + response.decode(encoding='utf-8'))
    except Exception as e:
      # If there's an error close the socket and break out of the loop
      CmdSock.close()
      print(Error receiving:   + str(e))
      break

send('command')
send('streamon')
send('battery?')
# Create and start a listening thread that runs in the background
# This utilizes our receive function and will continuously monitor for incoming messages
receiveThread = threading.Thread(target=receive)
receiveThread.daemon = True
receiveThread.start()

writeFileHeader(State_data_file_name)  # make sure file is created first so don't delay
stateThread = threading.Thread(target=rcvstate)
stateThread.daemon = False  # want clean file close
stateStop = threading.Event()
stateStop.clear()
```

```
stateThread.start()

stateThreadStream = threading.Thread(target=camera)
stateThreadStream.daemon = True  # want clean file close
stateThreadStream.start()


print('Type in a Tello SDK command and press the enter key. Enter quit to exit this program.')

# Loop infinitely waiting for commands or until the user types quit or ctrl-c
while True:
  try:
    # Read keybord input from the user
    if (sys.version_info > (3, 0)):
      # Python 3 compatibility
      message = input('')
    else:
      # Python 2 compatibility
      message = raw_input('')

    # If user types quit then lets exit and close the socket
    if 'quit' in message:
      print(Program exited)
      stateStop.set()  # set stop variable
      stateThread.join()   # wait for termination of state thread before closing socket
      writeDataFile(State_data_file_name)
      CmdSock.close()  # sockete for commands
      StateSock.close()  # socket for state
      print(sockets and threads closed)
      break

    # Send the command to Tello
    send(message)
    sleep(5.0) # wait for takeoff and motors to spin up
    # height in centimeters
    print('takeoff done')

    # Controller Variables

    kp_yaw = 2.5
    ki_yaw = 1.0
    kd_yaw = 0.1

    kp_ud = 100
    ki_ud = 20
    kd_ud = 10

    kp_lr = 100
    ki_lr = 15
    kd_lr = 50

    kp_fb = 90
    ki_fb = 15
    kd_fb = 50

    killNext = 0

    # Control stores
    integratedError_yaw = 0.0
    errorDerivative_yaw = 0.0
    errorStore_yaw = 0.0

    integratedError_fb = 0.0
    errorDerivative_fb = 0.0
    errorStore_fb = 0.0
```

```
    integratedError_lr = 0.0
    errorDerivative_lr = 0.0
    errorStore_lr = 0.0

    integratedError_ud = 0.0
    errorDerivative_ud = 0.0
    errorStore_ud = 0.0

    # to prevent hickups
    lastTime = 0.0
    lastYaw = 0.0
    last_tvec_x = 0.0
    last_tvec_y = 0.0
    last_tvec_z = 0.0

    print('Starting control loop')
    for i in range(0,600):
        # Get data (read sensors)
        presentState = stateQ.get(block=True, timeout=None)  # block if needed until new state is ready
        ptime = presentState[1]     # present time
        if i==0:
            print(Start time: +str(ptime))
        yaw = presentState[9]       # current yaw angle
        tvec_x = presentState[23]
        tvec_y = presentState[24]
        tvec_z = presentState[25]
        if lastTime > ptime:
            ptime = lastTime
            yaw = lastYaw
            tvec_x = last_tvec_x
            tvec_y = last_tvec_y
            tvec_z = last_tvec_z

        #Yaw control
        reference_yaw = 0
        if 1:
            error_yaw = reference_yaw - yaw
            integratedError_yaw = integratedError_yaw + INTERVAL*error_yaw
            errorDerivative_yaw = (error_yaw - errorStore_yaw) / INTERVAL
            errorStore_yaw = error_yaw
            control_YA = kp_yaw*error_yaw + ki_yaw*integratedError_yaw + kd_yaw*errorDerivative_yaw # + k3*
                integrated2Error

        lastTime = ptime
        lastYaw = yaw

        #UD control
        reference_y = 0.0
        if tvec_y!=0.0:
            #print('UD control active')
            error_ud = reference_y - tvec_y
            integratedError_ud = integratedError_ud + INTERVAL*error_ud
            errorDerivative_ud = (error_ud - errorStore_ud) / INTERVAL
            errorStore_ud = error_ud
            control_UD = kp_ud*error_ud + ki_ud*integratedError_ud + kd_ud*errorDerivative_ud


        #LR control
        xtargets = [0.5, -0.25, 0.0]
        reference_x = xtargets[target_num_GLOBAL]
        if tvec_x!=0.0:
            #print('LR control active')
            error_lr = -(reference_x - tvec_x)
            integratedError_lr = integratedError_lr + INTERVAL*error_lr
            errorDerivative_lr = (error_lr - errorStore_lr) / INTERVAL
            errorStore_lr = error_lr
```

```
            control_LR = kp_lr*error_lr +  ki_lr*integratedError_lr + kd_lr*errorDerivative_lr
        if killNext!=0:
            control_LR = 0
            integratedError_lr = 0
            killNext = killNext - 1

        #FB control
        ztargets = [0.5, 0.45, 0.25]
        reference_z = ztargets[target_num_GLOBAL]
        if tvec_z!=0.0:
            #print('FB control active')
            error_fb = -(reference_z - tvec_z)
            integratedError_fb = integratedError_fb + INTERVAL*error_fb
            errorDerivative_fb = (error_fb - errorStore_fb) / INTERVAL
            errorStore_fb = error_fb
            control_FB = kp_fb*error_fb +  ki_fb*integratedError_fb + kd_fb*errorDerivative_fb

        if (target_num_GLOBAL==len(tag_list)-1) and abs(error_lr)<0.15 and abs(error_fb)<0.15:
            print(Finished course)
            break

        #verify you are within the threshold
        if abs(error_lr)<0.15 and abs(error_fb)<0.15 and target_num_GLOBAL<(len(tag_list)-1) and tvec_z!=0.0:
            print('Removing tag: '+str(target_num_GLOBAL))
            integratedError_fb = 0.0
            integratedError_lr = 0.0
            target_num_GLOBAL += 1
            control_LR = 0
            killNext = 14

        # Send Control to quad
        control_LR = int(np.clip(control_LR,-100,100))
        control_FB = int(np.clip(control_FB,-100,100))
        control_UD = int(np.clip(control_UD,-100,100))
        control_YA = int(np.clip(control_YA,-100,100))
        message = 'rc '+str(control_LR)+' '+str(control_FB)+' '+str(control_UD)+' '+str(control_YA)
        print(message)
        send(message)

        # Wait so make sample time steady
        sleep(0.1)

    message='rc 0 0 0 0' # stop motion
    control_input = 0
    send(message)
    sleep(1.5)
    message ='land'
    send(message)
    print('landing')

    # Handle ctrl-c case to quit and close the socket
  except KeyboardInterrupt as e:
    message ='land'
    send(message)
    print('landing')
    stateStop.set()  # set stop variable
    stateThread.join()   # wait for termination of state thread
    writeDataFile(State_data_file_name)
    CmdSock.close()
    StateSock.close()  # socket for state
    print(sockets and threads closed)
    break
```