```python
'''goals3democode.py

  Demo code for Goals 3
'''

# Import useful packages
import hebi
import numpy as np              # For future use
import matplotlib.pyplot as plt

from math import pi, sin, cos, asin, acos, atan2, sqrt, inf
from time import sleep, time
from keycheck import kbhit, getch


#
#  HEBI Initialization
#
#  Create the motor group, and pre-allocate the command and feedback
#  data structures.  Remember to set the names list to match your
#  motor.
#
names = ['4.6', '6.2']
group = hebi.Lookup().get_group_from_names(['robotlab'], names)
if group is None:
  print("Unable to find both motors " + str(names))
  raise Exception("Unable to connect to motors")

command  = hebi.GroupCommand(group.size)
feedback = hebi.GroupFeedback(group.size)

dt = 0.01                       # HEBI feedback comes in at 100Hz!


#
#   PARAMATERS
#

feedback = group.get_next_feedback(reuse_fbk=feedback)
pinit_pan = feedback.position[0]
pinit_tilt = feedback.position[1]
p0=[pinit_pan, pinit_tilt]

v0 = [0.0, 0.0]
v_max = [2.72, 2.0]

#
#FUNCTIONS
#
def movetime(p0, pf, v_max, v0, vf):
    """Computes the time required to move between p0 and pf."""
    distance = abs(pf - p0)
    tm = ((6*distance) / ((4*v_max)+ (v0+vf)))
    tm = tm + ((abs(v0)*0.4)/v_max)
    if tm < 0.1:
        return 0.1
    return tm

def calcparams(t0, tf, p0, pf, v0, vf):
    """Computes the cubic spline parameters a, b, c, d."""
    a = p0
    b = v0
    c = (3 * (pf - p0) / (tf-t0)**2) - (2 * v0 + vf) / (tf-t0)
    d = (-2 * (pf - p0) / (tf-t0)**3) + (v0 + vf) / (tf-t0)**2
    return a, b, c, d
def splinecmds(t,a,b,c,d):
    pcmd = a + b * (t) + c * t**2 + d * (t)**3
    vcmd = b + 2 * c * (t) + 3 * d * (t)**2
    return pcmd, vcmd
#
#

t = 0.0  # Current time
t0 = 0.0  # Start time of the current segment

pf = [p0[0], p0[1]]
vf = [0.0, 0.0]
tm = max(movetime(p0[0], pf[0], v_max[0], v0[0], vf[0]), movetime(p0[1], pf[1], v_max[1], v0[1], vf[1]))

tf = tm + t0
a_p, b_p, c_p, d_p = calcparams(t0, tf, p0[0], pf[0], v0[0], vf[0])
a_t, b_t, c_t, d_t = calcparams(t0, tf, p0[1], pf[1], v0[1], vf[1])

# Data for plotting
Time, PAct_Pan, PCmd_Pan, VAct_Pan, VCmd_Pan, Verror_Pan, Perror_Pan = [], [], [], [], [], [], []
PAct_Tilt, PCmd_Tilt, VAct_Tilt, VCmd_Tilt, Verror_Tilt, Perror_Tilt = [], [], [], [], [], []

# Main control loop
while True:
    # Compute the current position and velocity commands
    pcmd_pan, vcmd_pan = splinecmds((t - t0), a_p, b_p, c_p, d_p)
    pcmd_tilt, vcmd_tilt = splinecmds((t - t0), a_t, b_t, c_t, d_t)

    # Send commands to the motor
    feedback = group.get_next_feedback(reuse_fbk=feedback)
    pact_pan = feedback.position[0]
    pact_tilt = feedback.position[1]
    vact_pan = feedback.velocity[0]
    vact_tilt = feedback.velocity[1]
```

```python

    command.position = [pcmd_pan, pcmd_tilt]
    command.velocity = [vcmd_pan, vcmd_tilt]
    group.send_command(command)

    # Store data for plotting
    Time.append(t)
    PAct_Pan.append(pact_pan)
    PCmd_Pan.append(pcmd_pan)
    VAct_Pan.append(vact_pan)
    VCmd_Pan.append(vcmd_pan)

    PAct_Tilt.append(pact_tilt)
    PCmd_Tilt.append(pcmd_tilt)
    VAct_Tilt.append(vact_tilt)
    VCmd_Tilt.append(vcmd_tilt)

    Perror_Pan.append(pact_pan - pcmd_pan)
    Verror_Pan.append(vact_pan - vcmd_pan)

    Perror_Tilt.append(pact_tilt - pcmd_tilt)
    Verror_Tilt.append(vact_tilt - vcmd_tilt)

    if abs(vcmd_pan) > v_max[0] or abs(vcmd_tilt) > v_max[1]:
        print("Exceeding max vel!")
    # Check for key presses
    if kbhit():
        ch = getch()
        if ch == 'a':
            t0 = t
            p0 = [pcmd_pan, pcmd_tilt]
            v0 = [vcmd_pan, vcmd_tilt]

            pf = [1.0, 0.0]

            if p0[0] == pf[0]:
                v0[0] = 0.0
            if p0[1] == pf[1]:
                v0[1] = 0.0

            vf = [0.0, 0.0]
            tm = max(movetime(p0[0], pf[0], v_max[0], v0[0], vf[0]), movetime(p0[1], pf[1], v_max[1], v0[1], vf[1]))

            tf = tm + t0
            a_p, b_p, c_p, d_p = calcparams(t0, tf, p0[0], pf[0], v0[0], vf[0])
            a_t, b_t, c_t, d_t = calcparams(t0, tf, p0[1], pf[1], v0[1], vf[1])

        elif ch == 'b':
            t0 = t
            p0 = [pcmd_pan, pcmd_tilt]
            v0 = [vcmd_pan, vcmd_tilt]

            pf = [-1.0, 0.0]

            if p0[0] == pf[0]:
                v0[0] = 0.0
            if p0[1] == pf[1]:
                v0[1] = 0.0

            vf = [0.0, 0.0]
            tm = max(movetime(p0[0], pf[0], v_max[0], v0[0], vf[0]), movetime(p0[0], pf[0], v_max[0], v0[0], vf[0]))

            tf = tm + t0
            a_p, b_p, c_p, d_p = calcparams(t0, tf, p0[0], pf[0], v0[0], vf[0])
            a_t, b_t, c_t, d_t = calcparams(t0, tf, p0[1], pf[1], v0[1], vf[1])


        elif ch == 'i':
            t0 = t
            p0 = [pcmd_pan, pcmd_tilt]
            v0 = [vcmd_pan, vcmd_tilt]

            pf = [pinit_pan, pinit_tilt]

            if p0[0] == pf[0]:
                v0[0] = 0.0
            if p0[1] == pf[1]:
                v0[1] = 0.0

            vf = [0.0, 0.0]
            tm = max(movetime(p0[0], pf[0], v_max[0], v0[0], vf[0]), movetime(p0[1], pf[1], v_max[1], v0[1], vf[1]))

            tf = tm + t0
            a_p, b_p, c_p, d_p = calcparams(t0, tf, p0[0], pf[0], v0[0], vf[0])
            a_t, b_t, c_t, d_t = calcparams(t0, tf, p0[1], pf[1], v0[1], vf[1])
        elif ch == 'c':
            t0 = t
            p0 = [pcmd_pan, pcmd_tilt]
            v0 = [vcmd_pan, vcmd_tilt]

            pf = [1.0, pi/4]
            vf = [0.0, 0.0]
            tm = max(movetime(p0[0], pf[0], v_max[0], v0[0], vf[0]), movetime(p0[1], pf[1], v_max[1], v0[1], vf[1]))

            tf = tm + t0
            a_p, b_p, c_p, d_p = calcparams(t0, tf, p0[0], pf[0], v0[0], vf[0])
            a_t, b_t, c_t, d_t = calcparams(t0, tf, p0[1], pf[1], v0[1], vf[1])
        elif ch == 'd':
            t0 = t
```

```
            p0 = [pcmd_pan, pcmd_tilt]
            v0 = [vcmd_pan, vcmd_tilt]

            pf = [0.0, -pi/6]
            if p0[0] == pf[0]:
                v0[0] = 0.0
            if p0[1] == pf[1]:
                v0[1] = 0.0
            vf = [0.0, 0.0]
            tm = max(movetime(p0[0], pf[0], v_max[0], v0[0], vf[0]), movetime(p0[1], pf[1], v_max[1], v0[1], vf[1]))

            tf = tm + t0
            a_p, b_p, c_p, d_p = calcparams(t0, tf, p0[0], pf[0], v0[0], vf[0])
            a_t, b_t, c_t, d_t = calcparams(t0, tf, p0[1], pf[1], v0[1], vf[1])
        elif ch == 'e':
            t0 = t
            p0 = [pcmd_pan, pcmd_tilt]
            v0 = [vcmd_pan, vcmd_tilt]

            pf = [-pi/4, pi/6]

            if p0[0] == pf[0]:
                v0[0] = 0.0
            if p0[1] == pf[1]:
                v0[1] = 0.0

            vf = [0.0, 0.0]
            tm = max(movetime(p0[0], pf[0], v_max[0], v0[0], vf[0]), movetime(p0[1], pf[1], v_max[1], v0[1], vf[1]))

            tf = tm + t0
            a_p, b_p, c_p, d_p = calcparams(t0, tf, p0[0], pf[0], v0[0], vf[0])
            a_t, b_t, c_t, d_t = calcparams(t0, tf, p0[1], pf[1], v0[1], vf[1])
        elif ch == 'z':
            t0 = t
            p0 = [pcmd_pan, pcmd_tilt]
            v0 = [vcmd_pan, vcmd_tilt]

            pf = [0.0, 0.0]

            if p0[0] == pf[0]:
                v0[0] = 0.0
            if p0[1] == pf[1]:
                v0[1] = 0.0

            vf = [0.0, 0.0]
            tm = max(movetime(p0[0], pf[0], v_max[0], v0[0], vf[0]), movetime(p0[1], pf[1], v_max[1], v0[1], vf[1]))

            tf = tm + t0
            a_p, b_p, c_p, d_p = calcparams(t0, tf, p0[0], pf[0], v0[0], vf[0])
            a_t, b_t, c_t, d_t = calcparams(t0, tf, p0[1], pf[1], v0[1], vf[1])

        elif ch == 'q':
            # Quit the program
            break

    if t+dt > tf:
        t0 = t
        p0 = [pcmd_pan, pcmd_tilt]
        v0 = [0.0, 0.0]

        pf = [pcmd_pan, pcmd_tilt]



        vf = [0.0, 0.0]
        tm = inf

        tf = tm + t0
        a_p, b_p, c_p, d_p = calcparams(t0, tf, p0[0], pf[0], v0[0], vf[0])
        a_t, b_t, c_t, d_t = calcparams(t0, tf, p0[1], pf[1], v0[1], vf[1])






    # Advance time
    t += dt

    # Stop if the segment is complete
    if t >= tf:
        t0 = t
        p0 = [pcmd_pan, pcmd_tilt]
        v0 = [0.0, 0.0]

        pf = [pcmd_pan, pcmd_tilt]


        vf = [0.0, 0.0]
        tm = inf

        tf = tm + t0
        a_p, b_p, c_p, d_p = calcparams(t0, tf, p0[0], pf[0], v0[0], vf[0])
        a_t, b_t, c_t, d_t = calcparams(t0, tf, p0[1], pf[1], v0[1], vf[1])
```

```python
#Have TA check graph
# Plot the results
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
ax1.plot(Time[0:len(Time)], PAct_Tilt[0:len(PAct_Tilt)], color='green', linestyle='-',  label='Act_Tilt')
ax1.plot(Time[0:len(Time)], PCmd_Tilt[0:len(PCmd_Tilt)], color='green', linestyle='--', label='Cmd_Tilt')
ax2.plot(Time[0:len(Time)], VAct_Tilt[0:len(VAct_Tilt)], color='green', linestyle='-',  label='Act_Tilt')
ax2.plot(Time[0:len(Time)], VCmd_Tilt[0:len(VCmd_Tilt)], color='green', linestyle='--', label='Cmd_Tilt')
ax1.plot(Time[0:len(Time)], Perror_Tilt[0:len(Perror_Tilt)], color='purple', linestyle='-',  label='Error_Tilt')
ax2.plot(Time[0:len(Time)], Verror_Tilt[0:len(Verror_Tilt)], color='purple', linestyle='--', label='Error_Tilt')

ax1.plot(Time[0:len(Time)], PAct_Pan[0:len(PAct_Pan)], color='blue', linestyle='-',  label='Act_Pan')
ax1.plot(Time[0:len(Time)], PCmd_Pan[0:len(PCmd_Pan)], color='blue', linestyle='--', label='Cmd_Pan')
ax2.plot(Time[0:len(Time)], VAct_Pan[0:len(VAct_Pan)], color='blue', linestyle='-',  label='Act_Pan')
ax2.plot(Time[0:len(Time)], VCmd_Pan[0:len(VCmd_Pan)], color='blue', linestyle='--', label='Cmd_Pan')
ax1.plot(Time[0:len(Time)], Perror_Pan[0:len(Perror_Pan)], color='red', linestyle='-',  label='Error_Pan')
ax2.plot(Time[0:len(Time)], Verror_Pan[0:len(Verror_Pan)], color='red', linestyle='--', label='Error_Pan')

ax1.set_title('Robot Data - Step 6')
ax1.set_ylabel('Position (rad)')
ax2.set_ylabel('Velocity (rad/s)')
ax2.set_xlabel('Time (s)')

ax1.grid()
ax2.grid()
ax1.legend()
ax2.legend()

plt.show()
```

Maria and Kaliyah's Robotics Report

Introduction

The purpose of this system is to translate real-time keyboard input into the rotation of two motors. Both motors are attached to a camera such that the pan motor controls the horizontal movement and the tilt motor controls the vertical movement of the camera. The system must be able to run indefinitely until the 'q' key is pressed. The velocity required to move the motor requires recalibration at every keyboard command.

System implementation was organized into different code sections, identified through comments (i.e. parameters, functions, main control loop). The system utilizes functions written to minimize the length of the code. Function names are concise descriptions of their purpose. Variables follow similar naming conventions, with '_tilt' or '_pan' at the end of motor-specific variables.

General Structure and Organization of the Code

Flow Chart of Goals 4 Step 6 code

1) **Start and Initialization**
    a) *Define Variables*
        i) Set control loop time (dt), and capture initial positions.

2) **Parameter and Function Definitions**
    a) *Define Functions*
        i) movetime(p0, pf, v_max, v0, vf): Calculate time (tm) to move from start to end position based on velocities.
            (1) <u>Decision:</u> If calculated tm is less than 0.1, set it to 0.1.
        ii) calcparams(t0, tf, p0, pf, v0, vf): Compute cubic spline parameters (a, b, c, d).
        iii) splinecmds(t, a, b, c, d): Generate position and velocity commands based on spline parameters.
    b) *Set Initial Parameters*
        i) Initial position (p0) and velocity (v0) are captured from feedback, and maximum velocity (v_max) is defined.
        ii) Set the current time (t) and the start time of the segment (t0).
        iii) Calculate tm, tf, and spline parameters (a, b, c, d) for both pan and tilt motors based on new pf.

3) **Main Control Loop**
    a) *Calculate Position and Velocity Commands*
        i) Call splinecmds to compute commands for pan and tilt motors.
    b) *Capture Feedback*
        i) Retrieve current position (pact_pan and pact_tilt) and velocity (vact_pan and vact_tilt) feedback.

c) *Send Commands*
    i) *Update and send position and velocity commands to motors.*
d) *Data Logging*
    i) Store time, actual and commanded positions and velocities, and error values for plotting.
e) *Check Velocity Constraints*
    i) Decision: If commanded velocity exceeds max velocity, print a warning.
f) *Check for Key Press*
    i) Decision: If a key is pressed:
        (1) a key: Set target position pf to [1.0, 0.0], update initial and final velocities.
        (2) b key: Set target position pf to [-1.0, 0.0].
        (3) c key: Set target position pf to [1.0, π/4].
        (4) d key: Set target position pf to [0.0, -π/6].
        (5) e key: Set target position pf to [-π/4, π/6].
        (6) z key: Set target position pf to [0.0, 0.0].
        (7) q key: Break the loop and exit program.
g) *Recalculate Parameters after each position update:*
    i) Update tm, tf, and spline parameters (a, b, c, d) for both pan and tilt motors based on new pf.
h) *Segment Completion Check*
    i) Decision: If time t + dt exceeds tf:
        (1) Update initial position and velocity to current commands.
        (2) Set pf to current commands, reset final velocity (vf) and adjust segment end time tf.
i) *Increment Time:*
    i) Update t by adding dt.
j) *Loop End: Continue until user input q ends the loop.*

4) Data Plotting and End

a) *Plotting*
    i) Generate two subplots for position and velocity over time for both pan and tilt motors.
    ii) Display plot, showing command vs. actual values and error metrics.

Summary

The code's organization is designed to ensure smooth and precise control over the robot's pan and tilt motors while maintaining a consistent command frequency of 10 milliseconds (100 Hz). This frequency is achieved by implementing a control loop with a fixed time increment (dt = 0.01 seconds), which advances time consistently with each iteration, ensuring that commands are sent to the motors every 10 milliseconds.

The structure is built to handle real-time events through a non-blocking keyboard input check (using functions like kbhit and getch). This allows the robot to respond immediately to user commands during operation. Modular functions, such as movetime, calcparams, and splinecmds, support this objective by dynamically calculating and updating position and velocity commands.

Additionally, the logging and plotting sections provide insights into performance without interrupting the control flow. This organization ensures that the code is reactive and capable of meeting strict timing requirements by continually recalculating control parameters in response to inputs. This way, it establishes a real-time control structure that adapts to unexpected user-driven changes.

Detailed Description of the Code Sections/Elements/Aspects.

Continual 100Hz "Heartbeat" and Signals

Function	Arguments	Return Values	Global Variables (not set as parameter)
movetime	t0, p0, pf, v0, vf	tm	None
calcparams	t0, tf, p0, pf, v0, vf	a, b, c, d	None
splinecmds	t, a, b, c, d	pcmd, vcmd	None

Variable	Definition	Set	Used
pcmd_pan and pcmd_tilt	Position command for pan motor and tilt motor	Line 91, Line 92	Sent to motor in command (Line 101, Line 91)
vcmd_pan and vcmd_tilt	Velocity command for pan motor and tilt motor	Line 91, Line 92	Sent to motor in command (Line 102, Line 107)
v_max	Maximum velocity [pan, tilt]	Line 45	Used in movetime calculations (Lines 78, 141) and in velocity constraint check (Line 123)
p0, pf, v0, vf, t0	(Initial, final) position and velocity [pan, tilt], Start time for the current segment	Line 42, Line 76, Line 44, Line 77, Line 74	Used in movetime calculation (Line 78), Used in movetime calculations (Lines 78,

			141), Used in movetime and calcparams calculations (Lines 78, 144), Used in calcparams (Lines 144, 145)
t	Current time	Line 73	Used to compute splinecmds and advance in loop (Lines 91, 276)
tm	Time to complete movement between positions	Line 78	Used to calculate tf (Line 80) and in movetime function
tf	Final time for current segment	Line 80	Used to determine end of segment (Lines 254, 279)
dt	Time interval (10ms, corresponding to 100Hz)	Line 32	Used to advance time in control loop (Line 276)
a, b, c, d (for pan (_p) and tilt (_t))	Cubic spline parameters for pan and tilt motor	Line 81, Line 82	Used in splinecmds to compute pcmd_tilt and vcmd_tilt (Lines 91 and 92, 66–69)

The code in goals4step6.py is structured to generate precise position and velocity commands for the robot's pan and tilt motors, updating these commands every 10ms for smooth, continuous motion. Key signals used in each control cycle include position commands (pcmd_pan and pcmd_tilt), velocity commands (vcmd_pan and vcmd_tilt), and the feedback signals from the motors that represent actual position and velocity (pact_pan, pact_tilt, vact_pan, and vact_tilt). Signals such as the position and velocity commands are recalculated at each cycle, using the current time and spline parameters, but initial and final positions (p0 and pf) as well as initial and final velocities (v0 and vf) persist across cycles until they are updated by a user input.

The core calculations use cubic spline formulas, which define position and velocity commands over time. The spline parameters—a, b, c, and d—are determined based on initial and final conditions using helper functions. The movetime function calculates the required movement time based on the distance to be traveled and max velocities using the equation:

$$tm = \frac{6(pf-p0)}{4\times vmax+(vo+vf)} + \frac{|v0|\times 0.4}{vmax}.$$

Calcparams uses cubic spline formulas to compute the parameters needed to generate smooth trajectories. These helper functions manage and validate parameters by ensuring non-zero tm values and valid splines, supporting consistent command accuracy. This organization allows for persistent variables across cycles and dynamically updated parameters, achieving the code's objectives for real-time control and responsiveness to user inputs.

Logical Behavior / Event Handling

Function	Arguments	Return Values	Global Variables (not set as parameter)
movetime	t0, p0, pf, v0, vf	tm	None
calcparams	t0, tf, p0, pf, v0, vf	a, b, c, d	None

Variable	Definition	Set	Used
p0, pf, v0, vf, t0	(Initial, final) position and velocity [pan, tilt], Start time for the current segment	Line 42, Line 76, Line 44, Line 77, Line 74	updated with each key press, recalculated with each new target position
tm, tf, [a, b, c, d (for pan (_p) and tilt (_t))], t0,	Time to complete movement between positions	Line 78	updated with each key press, recalculated with each new target position

The current code manages several key user input events, each designed to modify the robot's pan and tilt positions or velocities in real-time. These events include pressing the keys a through e, z, and q, which correspond to specific target movements or commands, such as moving to predefined positions or quitting the program. Each event triggers calculations that generate smooth motor commands, recalibrating spline parameters to ensure smooth and stable transitions between positions.

The parameters p0 (initial position), pf (final position), v0 (initial velocity), and vf (final velocity) are dynamically set for each event. They are verified to prevent abrupt changes in position or speed, resulting in predictable and smooth motor movements. Below is a table summarizing the events, their responses, and the critical parameter updates:

Event Key	Action	Target position	Initial velocity Reset

a	Move to [1.0, 0.0]	[1.0, 0.0]	Set to 0 if initial == target, else set to last vcmd
b	Move to [-1.0, 0.0]	[-1.0, 0.0]	Set to 0 if initial == target, else set to last vcmd
c	Move to [1.0, pi/4]	[1.0, pi/4]	Set to 0 if initial == target, else set to last vcmd
d	Move to [0.0, -pi/6]	[0.0, -pi/6]	Set to 0 if initial == target, else set to last vcmd
e	Move to [-pi/4, pi/6]	[-pi/4, pi/6]	Set to 0 if initial == target, else set to last vcmd
z	Move to [0.0, 0.0]	[0.0, 0.0]	Set to 0 if initial == target, else set to last vcmd
q	Quit the program	-	-

The consequences of each event result in smooth motion through recalculated spline parameters (a, b, c, d). The timing, denoted as tm, is calculated to avoid abrupt changes by considering the maximum velocity (v_max). These safeguards ensure signal computation leads to smooth and continuous behavior, adapting seamlessly to real-time input.

Initialization/Setup/Cleanup/Support Functionality

Function	Arguments	Return Values	Global Variables (not set as parameter)
Lookup().get_group_from_names()	[‘robotlab’], names	motor nums	None
plot()	Time[0:len(Time)], One of: (PAct_Tilt[0:len(Pact_Tilt], PCmd_Tilt[0:len(Pcmd_Tilt)], VAct_Tilt[0:len(VAct_Tilt)], VCmd_Tilt[0:len(VCmd_Tilt], Perror_Tilt[0:len(Perror_Tilt)], Verror_Tilt[0:len(Verror_Tilt)])	Lines on a subplot corresponding to the provided data and design specifications.	None

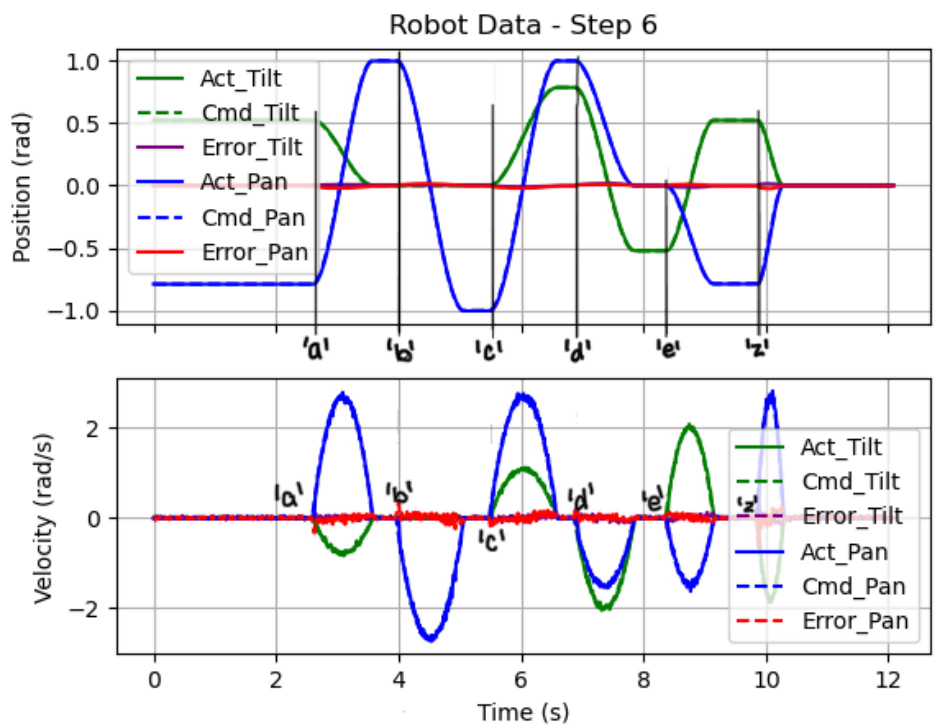
	*Same structure for Pan motor arguments, color, linestyle, label		
set_title(), set_ylabel(), set_xlabel()	strings	None	None
show()	None	Figure	None

Variable	Definition	Set	Used
names	List of motor names	Line 23	To connect to motors during setup.
command	Motor command with .position and .velocity properties	Line 29	Used to send commands to motors.
feedback	List of feedback information for both motors	Line 30, Lines 39-41, Lines 95-99	In the main control loop base calibrations on current position and velocity.
pinit_pan, pinit_tilt	Initial motor positions	Line 40-42, Line 172	In the setup to document the initial motor positions
p0, v0	Lists of initial motor positions and velocity	Lines 42-44, 50, 52, 54, 59, 61-64, 76, 78, 81-82	Used to calibrate commands based on the current position and velocity.
ax1, ax2	Subplots showing the position and velocity of the motors.	Lines 300-306, Lines 308-313, Lines 315-318, Lines 320-323	Implemented after the user presses ‘q’, exiting the main control loop.

The software is connected to the motors using the names variable and the Lookup().get_group_from_names() function. The command and feedback variables are used to control the motors and base calibrations off of current positions and velocities. The pinit_pan and pinit_tilt variables are needed in the setup to begin the program based off of initial positions. The p0 and v0 lists are used for data collection and are organized such that the 0th index is for the pan motor and the 1st index is for the tilt motor.

After the robot stops, two subplots are created (ax2 and ax2) to show the position and velocity of the motors as a function of time. The plot(), set_title(), set_ylabel, set_xlabel, and show() functions are used to design the plots.

Performance and Tuning



The graph displayed above, which is a key component of our motor control system, illustrates various types of key hits, providing insight into how the motor responds to each specific input. The constant lines seen in the graph indicate periods during which the function maintains a hold, waiting for the next event to occur. This next event could either be a key hit that prompts the execution of a spline or a command that instructs the program to terminate. This detailed graph serves as a comprehensive demonstration of each key hit and the corresponding reactions of the motors, allowing for a thorough understanding of their behavior and the significance of this graph in our system.

One notable observation is that the maximum velocity achieved from different key hits can vary, even when applied to the same motor. This discrepancy arises because we account for the maximum move time, ensuring that the motors end simultaneously so that only one motor reaches its peak velocity.

Furthermore, when a segment's final position matches its initial position, we intentionally set both the motor's initial and final velocities to zero. This adjustment occurs during the hold time, especially when the key hit preceding one event is identical to the subsequent hit. If we allowed even the slightest velocity to remain in the final command, the motor would be at risk of drifting, which could compromise precision. We ensure that the motors remain stable and accurately positioned by enforcing a zero velocity in only these scenarios.

Features, Limitations, Options for Improvement

The system's different functions and comments made the code concise and clear. The system avoided making implicit assumptions by including all necessary variables in function parameters. The system is limited in that the pan and tilt motors utilize lists and variables in different cases, instead of generalizing the a,b,c,d variables into the lists. The end position for the motors had to be adjusted due to hardware issues that did not align with the motor's default 0 position. In the future, the code may be adapted to process commands made to both motors simultaneously. The commands may also be processed at smaller time intervals for even smoother output.