Longitudinal Dynamics: Cruise Control

In Lab 2 you implemented a longitudinal PID controller to track a reference speed in simulation. This week you will design a PI controller for your BARC vehicle, and test it in a cruise control experiment. Your controller will be based on the longitudinal actuator models you identified in Lab 4, and tuned using pole placement. The controller you design in this week's lab will be critical for future labs, so be sure to ask questions if you need help tuning.

Task 1 Longitudinal Model and Estimated Parameters

In Lab 4, you identified two first-order models for your vehicle's longitudinal dynamics; one for braking and one for speeding up. The models should be in the form

$$P(s) := \frac{V(s)}{U(s)} = \frac{b}{s-a},$$

where b and $a \in \mathcal{R}$ are the parameters you identified using least-squares analysis. You will use these transfer functions to represent your vehicle's actuators in this week's velocity control design.

Task 2 State Estimation

In the cruise control experiment, you will provide a reference velocity to your vehicle, and design a controller to track that reference while rejecting disturbances, such as sudden changes in incline angle. The error signal you feed into your controller consists of the difference between this reference and your actual current velocity signal. Thus you will need to be able to estimate your current longitudinal velocity online, during run-time.

- You can write your own velocity estimation code. We suggest the following method, adapted from Professor Packard's ME 132 notes:

  Suppose $\theta(t)$ is the encoder measurement at time t. This function is sampled at fixed sample time $T > 0$, resulting in a discrete-time signal denoted as

  $$\theta_k^{[d]} := \theta(t)|_{t=kT}, \quad k = 1, 2, ...$$

  We now estimate the angular velocity using a 2nd-order, backwards-finite-difference approximation:

  $$\dot{\theta}_k^{[est]} := \frac{3\theta_k^{[d]} - 4\theta_{k-1}^{[d]} + \theta_{k-2}^{[d]}}{2T}$$

  The angular velocity can be converted to longitudinal velocity by multiplying by the wheel radius (5 cm).

Task 3 PI controller design

Recall that the transfer function $C(s)$ of a PI controller is:

$$C(s) := \frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} = \frac{K_p s + K_i}{s},$$

where your error signal $E(s)$ is the difference between the reference $R(s)$ and the current (estimated) velocity $V(s)$. Your PI controller calculates a control input based on a proportional (K_p) and integral (K_i) weight on your error signal.

1. ***Deliverable**: Find the closed-loop transfer function

   $$G(s) = \frac{V(s)}{R(s)}$$

   in terms of a, b, K_i, and K_p.

2. ***Deliverable**: Calculate the gains K_p and K_i required to set your closed-loop eigenvalues corresponding to a control objective of $(\omega_n = 1.8, \xi = 1.5)$.

Task 4 Simulink Experiment

Test and tune your PI controller by running cruise control experiments. You will first test your controller in simulation and then using the BARC car. Your objective in both tasks will be to track a reference velocity of 0.5 m/s.

1. ***Deliverable**: Create a simple Simulink model to use as a simulation environment for your controller. Note that the real BARC vehicle uses a digital controller, so you will need to convert your continuous-time transfer functions for the plant and controller to discrete-time in order to get an accurate simulation. In your conversions, use a sample time of 0.02 seconds. Your model should contain four key components:

   - A block containing your discretized plant model (use the Matlab command `c2d` for an easy conversion tool). Depending on your controller's PWM output, you should choose the appropriate actuator model (either speed or braking). Recommended use of the MATLAB FCN simulink block to set variables *a* and *b* to workspace depending on a switching condition and a discrete plant simulink block that uses those variables (run the sim using a m file so the variables are stored in the workspace).
   - A block containing your discrete-time PI controller (you may want to use the standard Simulink PID control block and edit the parameters to fit your calculated gains).
   - A step input disturbance to evaluate your closed-loop disturbance rejection. Your disturbance should be the sum of three different step inputs:

  - a step input of -10 at time $t = 3s$,
  - a step input of -70 at time $t = 10s$, and
  - a step input of 160 at time $t = 20s$.
- A constant reference input of 0.5 m/s.

Additionally, you may find it useful during the tuning process to use scope blocks at various locations. Submit a screenshot of your Simulink layout, with clear labels on each block.

2. Run simulations of your system, and adjust your calculated gains as necessary until your controller can track the reference well even in the presence of a step disturbance. It might be helpful to first consider a scenario without any disturbance, and begin to add disturbances once the model successfully tracks the reference velocity in these simplified conditions.

3. ***Deliverable**: Submit plots of (a) your controller's calculated motor PWM values over time, (b) your disturbance step inputs, and (b) the actual speed and the reference speed on the same graph.

Task 5 BARC Experiment

Now test your controller on the BARC vehicle. Note that you have to tune the parameters you identified in Simulink to overcome model mismatch.

1. Write a longitudinal cruise controller for your vehicle. We have provided an outline for you in the updated `LongitudinalController.py` file. You may edit this function for your purposes, or write your own from scratch if you prefer.
   Important Note: The K_p and K_i parameters you tuned Simulink will be **significantly** too high for the BARC vehicle. Scale them down by one to two orders of magnitude before testing on BARC.

2. Write a launch file that does the following:

   - Launches your Arduino node (so you have access to the `pwm` and `encoder` topics)
   - Launches your Camera node (If the camera is giving errors, you can record a video with your phone)
   - Launches the `record_experiment.py` node, so that you can save and analyze your data
   - Launches your PI controller node. Make sure limit the PWM commands with a min (1200) and max (1800) right before it's published.
   - (Optional but very helpful) Launch a keyboard node and add a callback function in your PI controller node which listens to the keyboard commands. This is great for making your car start and stop with keys presses so it doesn't move right away when everything runs.

   You should be able to use previous launch files as templates for this.

3. Please note the following:

   - When you want to record your experiments, do not forget to turn on the `data_service` using the command
     `roslaunch data_service service.launch`

in a new terminal, otherwise your experiments will not be recorded. You can choose whether or not you would like to record your data during your tuning process. Ultimately you will only need to save (and submit) your data from your successful, tuned controller.

- Give each of your experiments a unique identifier, so that your recorded data does not get overridden. Do this by calling your written launch file in the following way:
  `roslaunch labs lab5.launch id:="unique_experiment_id_here"`
- While you are only tuning your controller, set the `camera_on` parameter in the `record_experiment.py` node to `false`. This means your video will not be recorded, and your data will upload to the cloud much faster. Once you have chosen final tuning parameters, you can turn the `camera_on` parameter back to `true`, and record video of your best experiment.

4. Find or set up an appropriate testing environment to demonstrate that your controller can track the reference velocity both on flat and inclined ground. During the experiment, your vehicle should start out on level ground, and eventually move up (or down) a slope. Note that the sloped portion of the course should be long enough to give the controller time to track the reference velocity.

5. Test your controller on the BARC, and tune your K_I and K_P parameters until the vehicle tracks its reference velocity both on horizontal and sloped ground. When you have found gains that work, make sure to record your sensor data **with video** for at least one experiment.

6. ***Deliverable**: Demonstrate your controller's capability by submitting a plot of the BARC's velocity compared with the reference speed as a function of time. Also submit a video of a successful experiment (the BARC camera data will suffice).

7. ***Deliverable**: The controller parameters you tuned in Simulink are very different from those you tuned using the BARC vehicle. What are some possible explanations for this inconsistency?
