Lab 7 - Closed Loop Control

Overview

This lab introduces the topic of closed-loop control. In previous labs, you used open-loop control to drive the robot without feedback and in another you retrieved measurements from encoders which described the position of the wheels. Now you will combine both of these concepts to control the SCUTTLE with feedback from the encoders.

A closed-loop control system uses the actuating error, or the difference between a target and measured feedback, to reduce error over time and bring a system's output closer to a target value. This scheme is common even outside of robotics, in a car's cruise control for example. The driver sets a target speed and feedback from measuring the wheel speeds allows the car's computer to decide to apply or release the throttle. In our case: the SCUTTLE uses its encoders to measure wheel speeds, determines the error between the target wheel speed and the measured speed, and finally scales its effort to approach the target speed.

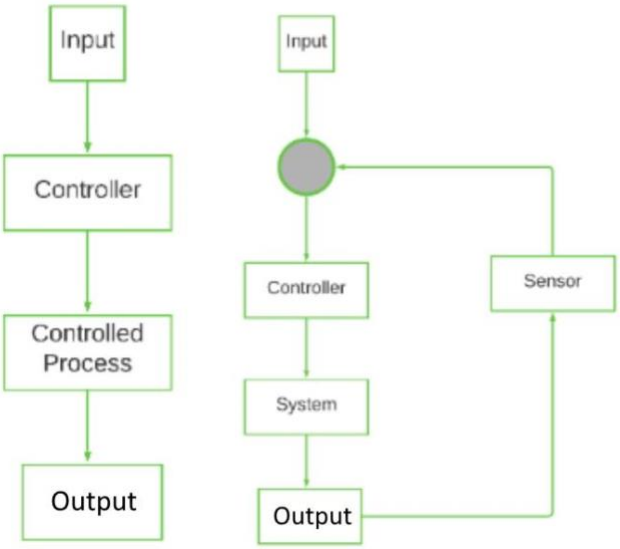

Figure 1: Open-loop (left) and closed-loop (right)

There are three basic feedback control actions: proportional (k_p), integral (k_i), and derivative (k_d). Each term can also be referred to as control "effort". The proportional control effort is generated based on the current error. Integral control effort is based on past error. Finally, derivative control effort is based on the anticipated future error. These control efforts can all be combined to create closed-loop controllers of different types, such as a proportional (P) controller, integral controller (I) proportional-integral (PI) controller, proportional-derivative (PD) controller, or proportional-integral-derivative (PID) controller.

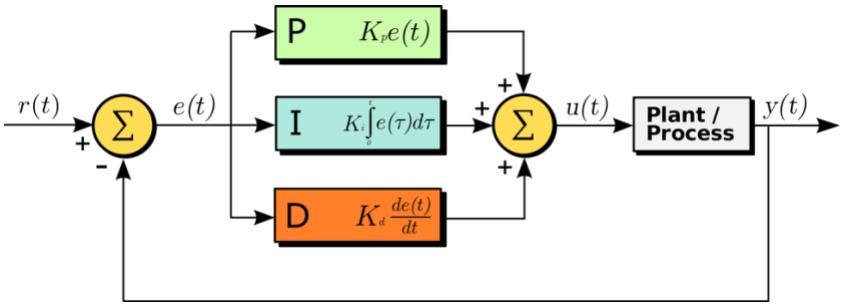

Figure 2: A block diagram of a PID controller in a feedback loop.
r(t) is the desired process variable (PV) or setpoint (SP), and y(t) is the measured PV.

Figure 2 demonstrates the equation of a PID controller. The total control output u(t) is a summation of three control efforts which are proportional, integral, and differential control efforts. The control effort only engages when its control parameter is non-zero. For example, a proportional controller has a non-zero proportional parameter whereas the integral and differential parameters are zero.

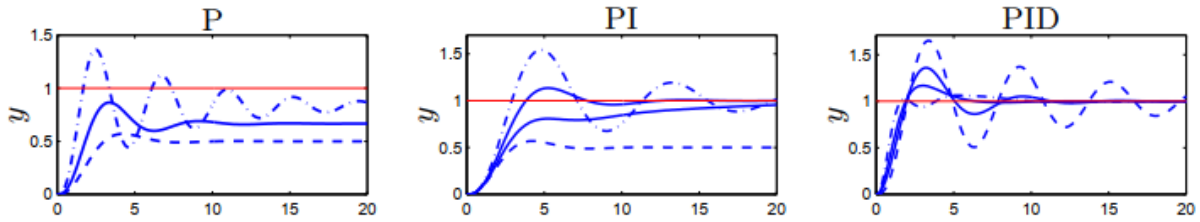

Figure 3: Plots of P (left), PI (center), and PID (right) controller error response;
when k constants are too small (dashed), too large (dot-dashed), and correctly tuned (solid)

Figure 3 shows three example plots of error response in a system with closed-loop feedback. The P controller includes curves where the constant k_p was too small (dashed), too large (dot-dashed), and correctly tuned (solid). Even when correctly tuned, the P controller does not reach the target in this case due to steady-state error. Steady-state error is caused by the control effort approaching zero as error reduces, which prevents the system from fully reaching and holding a target output.

Not all systems have steady-state error, but those that do can reduce it using the integral term in a PI controller. This yields an effort for proportional error as well as integrated error. Now, steady-state error will accumulate over time and drive the actuator towards the target output. The dashed curve in the PI plot shows a pure P controller, which includes steady-state error. The dot-dashed curve is an example of k_i being too large, which causes the integral effort to grow too quickly. The lower solid line is an example where k_i is set, but the constant is too small and only slowly moves the output to the target. The second solid line shows a properly-tuned PI controller response, where error slightly overshoots and then quickly stabilizes around the target value.

Finally, the PID controller utilizes all three control efforts: proportional, integral, and derivative. The added derivative term allows the closed-loop controller to anticipate error from sudden changes in feedback. The plot on the right is an example of this type of controller. The dashed curve shows a PI controller with no derivative effort, which overshoots the target multiple times. Notice how once the derivative term is added (solid curves) the response quickly eliminates the sharp peaks present in the PI control curve. In a real system, derivative control can sometimes create instability, especially if k_d is too large.

Resources:
The following are links to useful resources for this lab:

- SCUTTLE Homepage
- SCUTTLE GitHub
- Guides and resources

Software Needed:

- Python scripts:
  - Lab7Template.py

Proportional Control

The simplest closed-loop controller only uses proportional effort (k_p). This effort is directly proportional to the error between your target value and the measured values. For this exercise, wheel speeds represented as phi dot are measured to provide feedback for reaching targeted wheel speeds.

1. First verify that your SCUTTLE's motors and encoders function correctly.
    a. Run *L1_motors.py* and ensure the wheels move in the correct directions.
    b. Run *L1_encoder.py* and verify the sign and magnitude of measurements.
    c. Make sure your SCUTTLE is charged. This lab uses the motors extensively and your battery needs to be at least 10 volts.
2. Download *Lab7Template.py* from GitHub. This script is in the labs folder, not basics.
    a. This script demonstrates a closed-loop controller that drives the SCUTTLE forward at its maximum wheel speed: 9.7 rad/s.
    b. The script will also log the left wheel's speed to a file called *excel_data.csv* in the /tmp/ directory.
    c. The actual closed-loop control code is in *L2_speed_control.py*. Take a moment to find the function this template calls from that module and read it to understand how the script calculates control efforts.
3. In *L2_speed_control.py*, set the k_p constant to 0.04 and set k_i and k_d to 0.0.
4. Run *Lab7Template.py* and verify that the wheels move as they should. Allow the program to run for ten seconds before using Ctrl+C to stop it.
    a. Ensure your SCUTTLE is on its stand first.
    b. If the wheels do not move at all, increase k_p in increments of 0.01 until they turn.
    c. Use the command `sudo mv /tmp/excel_data.csv .` to move the logged wheel speed data to your current directory. Make sure to include the period at the end.
    d. Export this file to your laptop and open it in Excel, then verify it has at least sixty samples.
5. Once you have exported the wheel speed data, create a plot in Excel for the target speed and measured speed for each sample. Follow the example in figure 4 and make sure to include labels, titles, units, and a legend. Set a fixed axis range for every graph since we need to compare different control responses.

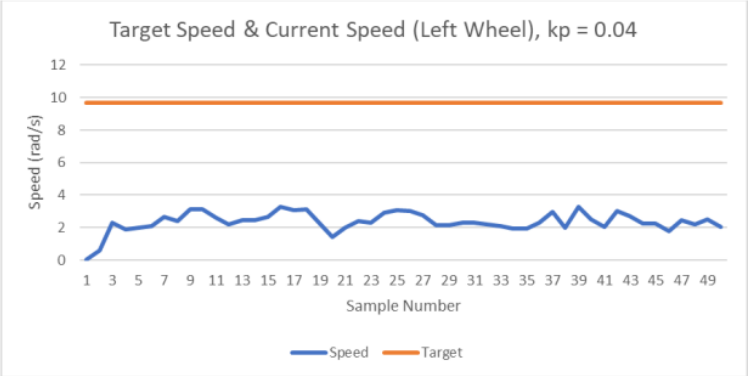

Figure 4: Example error response for P control

6. Repeat steps 4 and 5 for k_p set to 0.09 and 0.5. Consider why the performance changes and make sure to provide an explanation in the report. Keep everything in the same Excel file and create new tabs for each trial to keep your data organized.

Proportional and Integral Control

Next you will demonstrate a PI controller, which uses an accumulation of error over time to correct steady-state error. First you will observe the performance of an I controller, which only corrects errors over time without proportional effort. I controllers are common for systems that need speed control, but don't care about how long it takes to reach that target. A real application for this might be a speed controlled fan, which only needs to maintain a speed but not reach it quickly.

1. First you will run the I controller.
    a. Modify *L2_speed_control.py* and set k_i to 0.01 and k_p to 0.0.
    b. Run *Lab7Template.py* again and repeat the same steps to acquire ten seconds of data, then export it to Excel.
    c. Repeat this process for k_i set to 0.02 and 0.03.
2. What do you notice about the performance of the I controller? Make note of this for your report.
3. Now you will run a PI controller.
    a. Set k_p and k_i to 0.04.
    b. Run the experiment again and record the results in Excel.
4. Compare the new results with the I and P controller results.

Derivative Control

In this final exercise you will use the derivative term to create a PID controller. The derivative effort is meant to anticipate future error by applying effort opposing sudden changes in error. Using the k_d term reduces the influence of noisy measurements and impulses but can make a system unstable in certain conditions. Therefore, k_d should be determined experimentally using the maximum expected change in error.

1. In Excel, find the trial where k_p and k_i were both 0.04.
2. Locate the two adjacent samples with the largest change.
    a. Ignore the first ten samples, which are expected to have large errors with the robot starting at zero.
    b. Find the errors for both adjacent samples. Error is defined as input reference minus measured output.
    c. Find the largest error difference from two adjacent errors. The errors difference (*de*) is calculated from the current error deducts the previous error.
    d. Calculate the gradient of the error, *de/dt*. Since measurements are with respect to samples and not time specifically, you can leave *dt* in *de/dt* as 1.
    e. Find control effort u_d from multiplying de/dt by k_d. Assume k_d is 0.04.
    f. Next find u_p, the proportional control effort for the current timestep. Proportional control effort is calculated by the current error times k_p (0.04).
3. These two values, u_d and u_p are the maximum derivative effort and proportional effort that would be applied at that time. This could be dangerous since the total control effort becomes too large in a sudden which could harm the interactive user.

4. Now apply the constants k_p, k_i, k_d = 0.04 to the PID loop and observe the performance.
    a. What effect does the k_d term influence on the output? Describe this in your report.
5. Now, you should have 8 charts with various PID constant combinations (three P control, three I control, one PI control, and one PID control). Make each chart the same X axis length so that we can easily compare the delay from all figures.

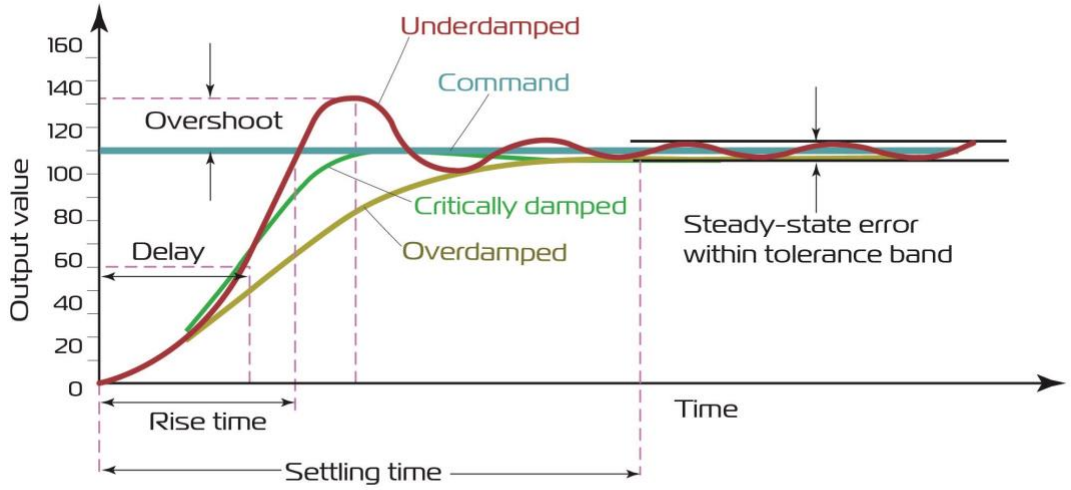

Figure 5: Control system response

6. Though this lab did not involve writing your own Python scripts or NodeRED flows, please upload any modified files to your GitHub repo. **You will need to upload your Excel document with the lab report and include all charts in the lab report.**

Post-Lab Questions

1. Explain the difference between a P and PI controller.

2. What does an increase in the proportional gain (k_p) do to the control system response?

3. There are three different and well-known step response waveform types for control systems: over-damped, under-damped, and critically-damped. Briefly explain each.