Assignment 4: Motor Feedback Control

Due 04/10/2022

In this assignment, we will build on top of assignment 3 and implement feedback control using a DC motor and the ESP32.

Note: You do not need to turn in the solution for every single problem. Only submit what is marked as ***Deliverables****. Combine all your answers into a single PDF in order, including the links to your videos. Submit the PDF to* bCourses.

1 Preparing for the feedback control implementation

1.1 Setting up the hardware

We have already set up the motor last time. However, in order for us to better visualize the rotation of our motor, install the motor shaft hub and the wheel to the motor shaft. You will need

- 1 × motor shaft hub
- 1 × wheel
- 1 × set screw
- 2 × phillips head (cross head) screws

Your assembly should look like this

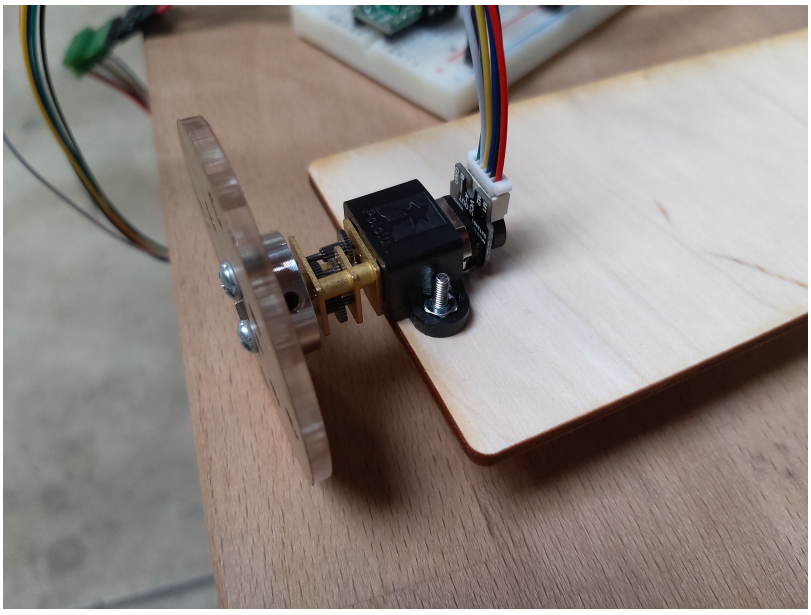

Figure 1: Wheel assembly

1.2 Sending from the Serial Monitor/Plotter

So far we have been using the Serial Monitor/Plotter to read things. Another important feature of the Serial Monitor/Plotter is that it allows us to send messages to our micro-controller. This can be really helpful if you want to change the state of the machine, tune parameters, etc. One thing you often have to do when designing a feedback controller is to tune gains. Instead of having to flash the ESP32 every single time we change the gains, let's write a serial reading code that allows us to change them in real-time.

There are several commands we can use to read from the serial channel. The most common ones are "Serial.read()" and "Serial.readStringUntil()". Open **serial_read_template.ino**, try out both of these functions and see what they print. When you use the Serial Monitor, switch between "Newline" and "No line ending" and see what difference it makes.

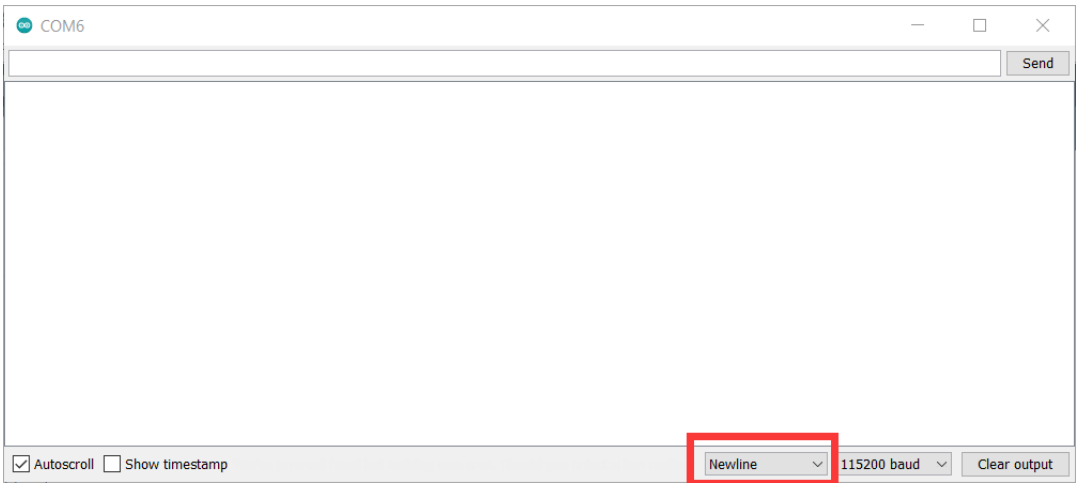

Figure 2: Where to change the Newline setting

We will be using the "Serial.readStringUntil()" function to tune our gains and change the states. Open **serial_processing_template.ino**. Change the code such that by sending "which_gain_to_change + desired_gain", the gain will be updated. For example, by sending "p15.5", the "kp" should be updated to 15.5. I recommend you use the switch expression to select which gain to change.

Deliverables **Written:** Explain why "Serial.read()" and "Serial.readStringUntil()" give you different results despite the same serial input. Check the Arduino documentation for more information.
Deliverables **Video:** Take a video of you entering a new gain and the board updating it correspondingly.

1.3 Open-loop speed control

From what we have learned in the lecture, the load-free speed of a motor should be proportional to the voltage applied in theory.

$$K_v V = \omega_{load-free}$$

Based on this principle, do an experiment to find the motor velocity constant. Make a copy of **serial_processing_template.ino** and name it as **basic_speed_control.ino**. In the new file, write a function

"int commandOpenLoopSpeed(float speed_desired)" that maps the desired speed in rad/s to a PWM output. Merge your assignment 3 motor control file with this file so that you can command the motor. Change your code such that by sending "'s' + desired_speed", the motor changes its speed correspondingly. For example, if you send "s2", the motor should be rotating at 2 rad/s.

$$y(\omega_{des}) = k\omega_{des}$$

Deliverables **Photo:** Take a screenshot of the Serial Plotter showing you updating the desired speed and the actual motor speed converging to the new desired speed. The Serial Plotter should plot both the desired speed and the actual speed.
Deliverables **Written:** What are the drawbacks of an open-loop control? Give three examples.

1.4 Bang-bang speed control

Let's now look at rudimentary closed-loop control. Unlike open-loop control, closed-loop controllers use knowledge about the state of the system in order to produce a control effort. For our system, this will allow our motor to respond to variable shaft loads. One of the most simple closed-loop control methodologies is called on-off or bang-bang control. Bang-bang control is simple: when a target parameter is below the desired reference, the control effort is fully engaged. When the parameter is above the desired reference, the control effort is fully disengaged.

$$y(\omega_{des}) = \begin{cases} y_0 & \text{if } \omega \leq \omega_{des} \\ 0 & \text{if } \omega > \omega_{des} \end{cases}$$

Now, write a function "int commandBangBangControl(float speed_desired, float speed_actual, int y0_des)" that does the switching described. Change your y0_des and observe its effect on the motor.

Deliverables **Photo:** Take two screenshots, one showing the actual speed and the desired speed of the motor when y0_des is small, one showing the speeds when y0_des is large.

2 Feedback control

Let's now begin to move towards more sophisticated closed-loop control schemes.

2.1 Proportional (P) speed control

Just like its name suggests, proportional control produces a control effort proportional to a measured error. The error e(k) and control effort D(k) are defined by the following equations

$$e = \omega_{des} - \omega,$$
$$y(e) = K_p e = K_p(\omega_{des} - \omega)$$

K_p is known as the proportional gain and is selected in order to optimize system performance. For now, let's fiddle with K_p to develop an intuition for proportional control. Make a copy of **basic_speed_control.ino** and name it as **speed_control.ino**. In the new file, write a function "int PControlSpeed(float speed_desired)" that computes the P controller output. Experiment with your motor.

Start with a small K_p and increase it until you feel satisfied with the performance.

Deliverables **Photo:** Take a screenshot of the Serial Plotter showing you updating the desired speed and the actual motor speed converging to the new desired speed. The Serial Plotter should plot both the desired speed and the actual speed. Report your K_p in the screenshot.
Deliverables **Written:** Briefly compare the performance of the P controller to the open-loop and bang-bang controllers. What are the effects of differing K_p values?

2.2 Proportional-integral (PI) speed control

Let's attempt to improve upon our proportional controller through the addition of an integral (I) term. The integral term sums previous errors. The overall control output is defined by the following equation

$$y(e) = K_p e + K_i \sum e$$

K_i is known as the integral gain, as is selected in order to optimize system performance. As with K_p, we will first experiment with this value in order to build intuition. In the same file, write a function "int PIControlSpeed(float speed_desired)" that computes the PI controller output. Experiment with your motor.

Tune K_p and K_i until you feel satisfied with the performance. Your K_i should be much smaller than K_p.

2.3 Proportional-integral (PI) anti-windup I

Next, we must first account for integrator windup, a phenomenon that occurs when a controller saturates. Saturation refers to a situation where control effort increases (or decreases) are no longer capable of influencing the behavior of the system, often due to physical limitations.

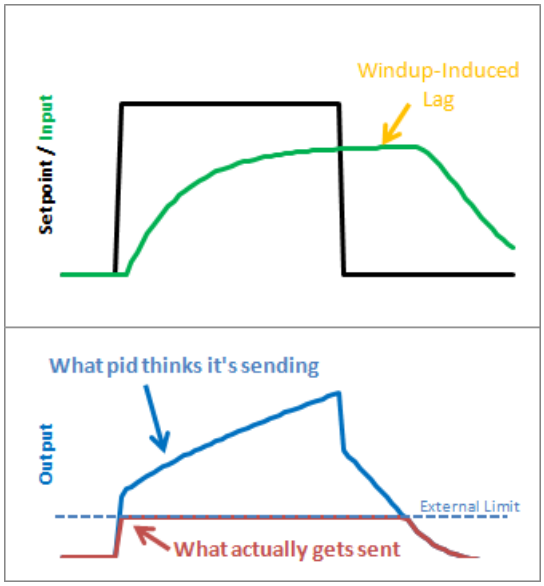

Figure 3: Integrator windup

Refer to this link (http://brettbeauregard.com/blog/2011/04/improving-the-beginner%E2%80%99s-pid-reset-windup/) for its implementation. The reference is a very simple read that briefly introduces the cause (and effect) of integrator windup, and offers potential solutions. Implement this technique

in your PI controller function. Make sure to do both steps in the reference.

Deliverables **Photo:** Take a screenshot of the Serial Plotter showing you updating the desired speed and the actual motor speed converging to the new desired speed. The Serial Plotter should plot both the desired speed and the actual speed. Report your K_p and K_i in the screenshot.
Deliverables **Written:** Briefly compare the performance of the PI controller to the open-loop, bang-bang, and P controllers. What are the effects of differing K_i values?

2.4 Proportional–Integral–Derivative (PID) position control

Finally, let's implement a PID position controller on our motor. The derivative term defines the rate of change in the error. The overall control output is defined by the following equation

$$y(e) = K_p e + K_i \sum e + K_d \dot{e}$$

One thing we have to notice here, that we are implementing a position controller instead of a speed controller. This means the error term above is the error in the position, not the error in the speed. By expanding the equation for our system, we should get

$$y(e) = K_p(\theta_{des} - \theta) + K_i \sum (\theta_{des} - \theta) + K_d(\dot{\theta}_{des} - \dot{\theta})$$

For position control, we want the desired speed to be 0. If the position target is not too far, we often simplify our equation to the following

$$y(e) = K_p(\theta_{des} - \theta) + K_i \sum (\theta_{des} - \theta) + K_d(-\dot{\theta}),$$
$$y(e) = K_p(\theta_{des} - \theta) + K_i \sum (\theta_{des} - \theta) - K_d\omega$$

For a nearby position target, you can update θ_{des} in a single step, and the controller should converge to it very fast. However, if your target is very far, you would want to do tracking instead, that you constantly update your θ_{des} and ω_{des} along the time horizon. Otherwise, your terms may blow up.

In this assignment, we will only look at a close target position. Make a copy of **speed_control.ino** and name it as **position_control.ino**. In the new file, write a function "int PIDControlPosition(float position_desired)" that computes the PID controller output. In addition, change your code such that by sending "'p' + desired_position", the motor moves to the desired position. For example, if you send "p1", the motor should go to 1 rad. You can change the unit to degree if you are more comfortable with it. Make your position at startup the origin.

Deliverables **Video:** Take a video of your entering a new desired position and the motor moving to the new target.

Helpful resources

Arduino Programming Language: https://www.arduino.cc/reference/en/
HUZZAH32 Documentation: https://learn.adafruit.com/adafruit-huzzah32-esp32-feather
HUZZAH32 Pinouts: https://learn.adafruit.com/adafruit-huzzah32-esp32-feather/pinouts
The Lab Kit: https://microkit.berkeley.edu/blank-page/
Installing Arduino Libraries: https://www.arduino.cc/en/Guide/Libraries
Arduino Serial Plotter: https://arduinogetstarted.com/tutorials/arduino-serial-plotter

Deliverables checklist

Deliverables **Written:** Explain why "Serial.read()" and "Serial.readStringUntil()" give you different results despite the same serial input. Check the Arduino documentation for more information. (10%)
Deliverables **Video:** Take a video of your entering a new gain and the board updating it correspondingly. (10%)
Deliverables **Photo:** Take a screenshot of the Serial Plotter showing you updating the desired speed and the actual motor speed converging to the new desired speed. The Serial Plotter should plot both the desired speed and the actual speed. (10%)
Deliverables **Written:** What are the drawbacks of an open-loop control? Give three examples. (10%)
Deliverables **Photo:** Take two screenshots, one showing the actual speed and the desired speed of the motor when y0_des is small, one showing the speeds when y0_des is large. (10%)
Deliverables **Photo:** Take a screenshot of the Serial Plotter showing you updating the desired speed and the actual motor speed converging to the new desired speed. The Serial Plotter should plot both the desired speed and the actual speed. Report your K_p in the screenshot. (10%)
Deliverables **Written:** Briefly compare the performance of the P controller to the open-loop and bang-bang controllers. What are the effects of differing K_p values? (10%)
Deliverables **Photo:** Take a screenshot of the Serial Plotter showing you updating the desired speed and the actual motor speed converging to the new desired speed. The Serial Plotter should plot both the desired speed and the actual speed. Report your K_p and K_i in the screenshot. (10%)
Deliverables **Written:** Briefly compare the performance of the PI controller to the open-loop, bang-bang, and P controllers. What are the effects of differing K_i values? (10%)
Deliverables **Video:** Take a video of your entering a new desired position and the motor moving to the new target. (10%)
No Checkoff Please take the time to focus on the project.