Mini Project*

EENG350: Systems Exploration, Engineering, and Design Laboratory

Tyrone Vincent and Darren McSweeny

Department of Electrical Engineering
Colorado School of Mines

Fall 2020

1 Introduction

The purpose of this exercise is to design a control system that regulates the rotational speed of a wheel attached to a motor based using an Arduino and Rasberry Pi, along with a mechanical encoder and camera. The wheel is marked along the circumference with the numbers 0, $\frac{\pi}{2}$, π and $\frac{3\pi}{2}$ representing the angle in radians relative to the 0 marker. You will implement a system such that using a single Aruco marker, the user can direct the system to spin the wheel so that the wheel is at one of the four angles indicated on the wheel. For example, which angle number is to be on top might be determined by which quadrant (NE, NW, SE or SW) the Aruco marker appears in the camera image. The current desired setpoint (determined by the Aruco marker) along with the current position of the wheel, is displayed in the LCD screen. The current position shown on the LCD should update at least every second, and show what the current encoder reading is, converted to radians. The control system will also reject disturbances, so that if someone tries to spin the wheel, the motor will resist and the marker will remain at the top. The system should be fast enough that in a demonstration all four markers can be moved to the top one after the other, with the total time elapsed less than 10 seconds. The marker detection should also be robust enough to work with the lights on or off, and with the marker between 1 foot and 5 feet away from the camera.

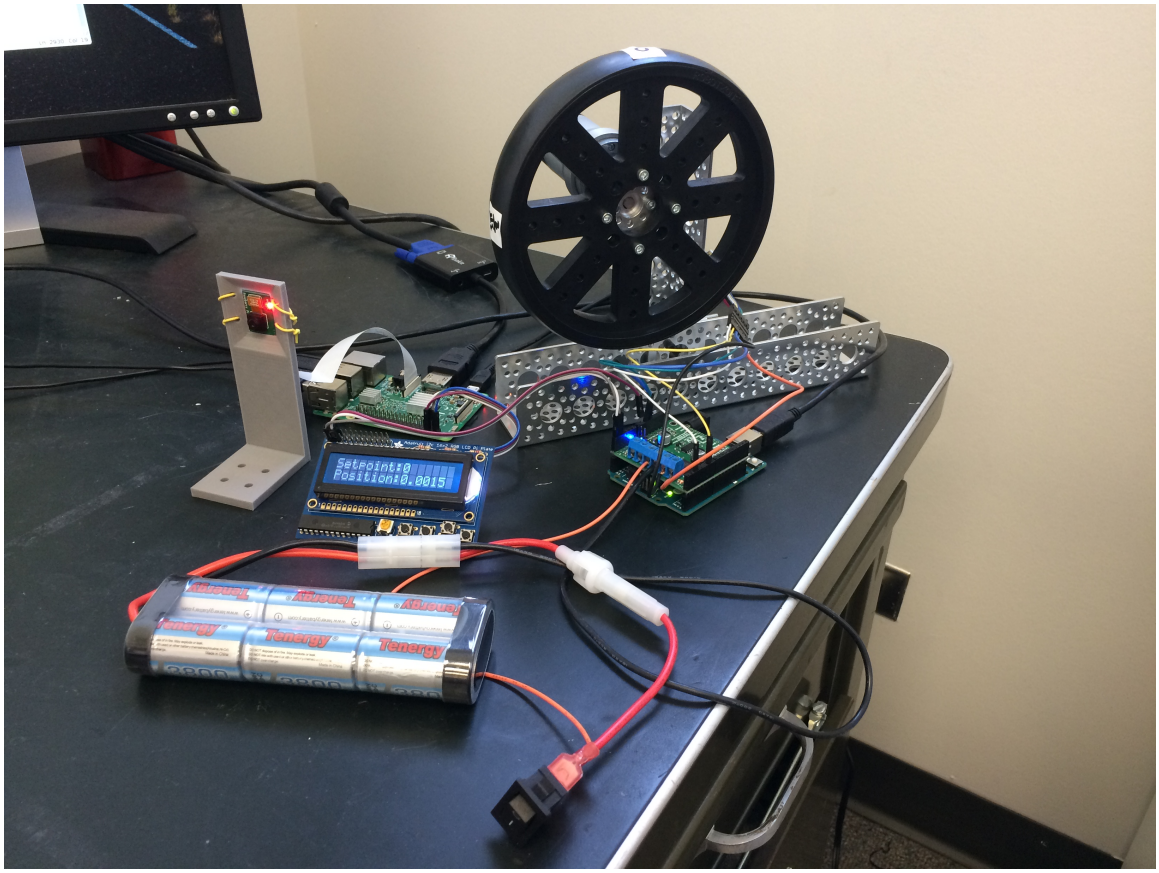
2 Work Process

You are responsible as a team for completion of all elements of the mini-project. You should have weekly team meetings. You can use Zoom to have the meeting via a teleconference - see the link on Canvas. Your team leader will organize the meetings and ensure that they are effective, and that detailed minutes for the meetings are taken. The team should select a method to organize files and materials. You can use the Canvas collaboration tools, but other mechanisms such as Github, Dropbox, Slack, etc. may be more useful.

3 The System

Mount the motor on a frame to make it stable, and add a wheel. You are using Pololu 37D gear motors with a 50 to 1 gear ratio. A description of the motor that you are using can be found here. Note the pin assignments for both the motor and motor encoder, which you will use later. You will want to make sure that the wheel can spin freely, so use several actobotics frame elements to create a solid base with a mast that the motor can be attached to. Also secure the camera and Pi using the supplied brackets. Connect the motor driver shield onto the Arduino. Hook up a battery (via the switch and fuse connector) to the voltage monitor, and from the voltage monitor, connect to the motor driver power input. Hook up the motor power leads to one of the motor driver output channels. Note that there is a jumper on the motor driver board that allows the battery to also be the power supply for the Arduino, but it is not necessary if you have the Arduino connected via USB.

The photo below shows the typical setup, **except for the voltage monitor**. The voltage monitor should go between the battery and the motor driver.



4 Project Elements

The following are elements of the project. You should review all of these elements, and distribute the work among your group members. However, the project deliverables are the joint responsibility of the entire group.

4.1 Create a GitHub repository to hold your code and keep track of your project

If you don't yet have an account, sign up for an account on GitHub. If you aren't familiar with GitHub, Read a short introduction. When working on the Raspberry Pi, you will want to use git at the command line. There are a variety of introduction to the command line commands, such as this one. Github also has project boards to keep track of your project tasks.

Create a GitHub repository for your mini-project. Use this repository to store and mange your software. Since you will be using this repository throughout the semester, create a subfolder called "Miniproject" and other subfolders below that as necessary. Create a reame that describes your project, and the repository organization, Create a project board that describes the tasks that need to be accomplished, and make assignments for those tasks.

4.2 Use Open CV to implement the Aruco detection scheme

You will need to account for changes in lighting, so properly setting the white balance and determining the proper range of color values is key. It will be a good idea to include a calibration step - this would turn the camera on and let the automatic white balance settle to a constant, then the settings are saved and the automatic white balance turned off. Take a look at the classes: awb_mode, and awb_gains. Choose a reference image. This could be a grey piece

of paper, or one with equal amounts of red, blue and green on it. Take a few pictures at a close distance. Print the auto-white-balance gains for those pictures. They should be hovering around the same values. Choose an average or values that closely match and set the gains using awb_gains.

You can use your code from Assignment 2 as a starting point for the Aruco detection. You will need to modify it so that the location of the marker in the image is determined.

4.3 Use a motor driver to spin a motor under control of the Arduino

On the motor web page, under "Resources" you can find the users guide. Download the guide, save it in a place your team can easily access it. Read the guide, skimming the first sections, but reading section 3.c in detail. However, **DO NOT utilize the Arduino library described in section 3.d**. This library contains notation which is confusing and leads students to errors, and in fact is unnecessary once you understand the pin mappings given in section 3.c. In addition, the description of pins 9 and 10 as "Motor speed input" and pins 7 and 8 as "Direction" are **incorrect terminology**. Pins 9 and 10 specify whether the voltage supplied by the h-bridge to the relevant motor is on or off, and thus could better be described as "Motor voltage", while pins 7 and 8 determine the sign of this voltage, and thus would be better labeled "Voltage sign". You will use `AnalogWrite()` to create a pulse width modulated signal on pins 9 and 10. Since `AnalogWrite` uses timer1 for pins 9 and 10, you should not use timer1 as a timer interrupt for anything else.

1. Use the Arduino to spin the motor. The Arduino is able to supply a pulse-width-modulated signal on pins 9 or 10 using using the `analogWrite` command. It is suggested to also read the PWM tutorial.

   In the `setup()` portion of your code, you will want to setup pins 4, 7, 8, 9, and 10 for output, and set the enable pin high. Set pin 12 for input. In the `main()` portion of the code use the `analogWrite` command on pins 9 or 10 to command the motor driver to supply a pulse width modulated signal of a desired pulse width. (The sign of this voltage can be set on pins 7 or 8)

2. Use the oscilloscope to measure the voltage at pin 9 or 10, as well as the voltage across motor. Verify what value of `analogWrite` will correspond to a particular pulse width. Verify that pins 7 and 8 change the voltage sign. Come up with an equation that will relate the value written using `analogWrite` to the pulse width.

4.4 Use the Arduino to read the rotary encoder on the motor

The motors have a rotary quadrature encoder attached to the motor shaft opposite of where the wheel is attached.

1. Read the description of what a rotary encoder is: `http://en.wikipedia.org/wiki/Rotary_encoder#Incremental_rotary_encoder`

2. Find the high performance encoder library on this page, and download it: `http://playground.arduino.cc/Main/RotaryEncoders`. (It is listed under libraries, and described as "High performance with 4X counting and very efficient interrupt routines") From the Arduino program, click on `Sketch -> Include Library -> Add .zip library` to install the library from the file that you downloaded. Read the download page for instructions on using the library.

3. Use the library to read the encoder - verify that it is working by spinning the wheel by hand.

4. Determine the scaling such that the angular position of the wheel in radians can be displayed. Add a mechanism such that the angular position can be re-set to zero. (Some ideas - using a switch with an interrupt input, looking for an input from the serial monitor, or looking for a command via i2c from the Pi.)

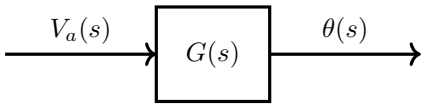
For the final project, you will want to control two motors, Since the Arduino uno has only two interrupt enabled pins, you will need to use one for each motor. Get ready now by setting up the system so that only one channel uses an interrupt pin.

4.5 Setup communication between the Raspberry Pi, Arduino and LCD

The Pi will determined the desired position and send that to the Arduino. The Arduino will send back the current position (determined by the motor encoder) and both of these are displayed on the LCD screen by the Pi.

4.6 Create a simulation of your motor

In order to design your controller, you will need a model of the system to be controlled. In this case, it is the motor and wheel, which as the motor voltage (i.e. command to the motor driver) as input and motor position (as measured by the encoder) as output. You will find a transfer function to model this response from input voltage V_a to angular position θ, $G(s) = \frac{\theta(s)}{V_a(s)}$.



1. Set up an Arduino program that will perform a step response experiment. While our goal will be to find find a transfer function with angular position as output, as an intermediate step, we will perform an experiment with angular velocity as output, since angular velocity will be bounded, but angular position will not be. For the step experiment, the program will initially output a motor voltage command of 0, changing to a desired positive value at 1 second. (By motor voltage command, we mean the value used in the `analogWrite` function). You will want to display the current time, motor voltage command, and angular velocity for each sampling interval. **Make sure you record the units that you are using for all of these items, and be consistent with these units in your simulations and software**.

   This program should have a `main()` function that runs at a **fixed cycle time**, also called the sampling time. Use the function `micros` or `millis` to control the sample rate, by waiting at the end of your main function until the correct time has passed since the last time through. Here is an example of this.You will also want to signal the user or display an error message if the main function takes longer than the desired sampling rate. Choose a desired sampling rate between 5 and 10ms. If you keep getting errors because the main function takes longer than the desired sampling rate to execute, increase the baud rate of the serial link to speed up the `Serial.print` statements (it can go up to 250000 bps).

   Have the program read the motor encoder and calculate the angular position (relative to the starting position), and from this data and the known sampling rate, calculate the angular velocity. **Convert the angular velocity to SI units: radians per second.** You will need to know the number of counts per revolution to make this conversion - this can be found on the motor web page.

   You can copy and past the results from the serial monitor. It will be most convenient if the Arduino program only prints out results for a short period of time. For example, you could have an if statement that sets the motor voltage and prints out the time and angular velocity as long as the elapsed time (measured by `millis`) is between 1 and 2 seconds. If you close and open the serial monitor, the program will start again. Alternately, see the appendix for a method of transferring data to Matlab over the serial link.

2. Perform a step response experiment, and estimate the transfer function from motor voltage command to angular velocity. Although the motor is theoretically second order, when the motor inductance is small, the response can be well approximated as first order, so the transfer function for the motor will be of the form $K\frac{\sigma}{s+\sigma}$. You will probably use a voltage command greater than 1, so don't forget to scale the results appropriately when finding K.

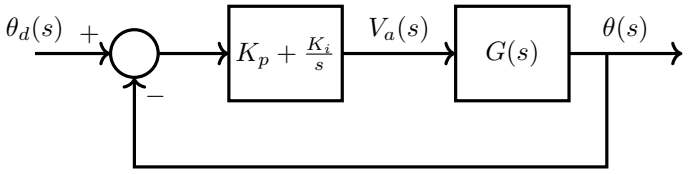
   - To estimate the transfer function, use the techniques discussed in the EENG307 Notes: Lecture 15, System Identification

   You should be able to show your experimental step response and the step response of your identified transfer function (e.g. using the Matlab `step` command) on the same plot to show that they match.

3. At this point, you have an estimate of $\frac{\dot{\theta}(s)}{V_a(s)}$. Since position is the integral of velocity, our desired transfer function is $\frac{\theta(s)}{V_a(s)} = \frac{1}{s}\frac{\dot{\theta}(s)}{V_a(s)}$.

4.7 Design a controller for your motor

Using the transfer function $G(s) = \frac{\theta(s)}{V_a(s)}$ as the system to be controlled, design a Proportional-Integral (PI) controller to regulate the motor position. You will need integral control to reject disturbances such as your instructors finger on the wheel.



That is, you should be able to specify a desired rotational position θ_d in radians that the wheel will turn to, even if there are disturbances.

1. Design a PI controller that achieves closed loop step response specifications of a rise time of 1 second and an overshoot of less than or equal to 12%, with zero steady state error. Document your design procedure. The autotune function of the Simulink PID block may be useful here. Simulate the closed loop system in Simulink.
2. Implement the controller using the Arduino. Don't forget about units!
    - Implementation of PI control in hardware is discussed in the Appendix of EENG307 Notes: Lecture 19, PID Control, Appendix.
3. Create an Arduino program to perform a closed loop step response experiment. In this experiment, the setpoint should initially be zero, and at a specified time the setpoint is changed to 1 rad. The Arduino should display the current time and wheel position in a format that can be copy and pasted in to excel or Matlab, and then plotted. You should be able to plot the step response of your simulated closed loop system and experimental closed loop system on the same graph for comparison. What might account for any differences?

4.8 Put everything together

Integrate your components to achieve the desired behavior as requested in Section 1.

5 Documentation

The following documentation should be uploaded to the Canvas assignment "Mini Project Team Documentation".

1. A short document that describes your marker detection scheme.
2. The equation that will relate the value written using `analogWrite` and the pulse width, along with a plot of this function.
3. A short document (perhaps using the publish function of the Matlab editor) that includes
    - your control design and design method
    - the open loop step response (both simulated and experimental), and
    - the closed loop step response (both simulated and experimental).

The following documentation should be uploaded to the Canvas assignment "Mini Project Team Documents".

1. Your team charter
2. Meeting minutes
3. GitHub repository link

Upload your code to your GitHub repository, in a folder named Mini Project. This should include Arduino and Simulink/Matlab code you can use to verify the motor controller is working correctly. That is, you should be able to quickly simulate either an open loop (voltage command) or closed loop (position command with PI controller implemented) step response, perform an experimental step response, and quickly compare the results in Matlab. Your code should be well documented.

The reflection logs (uploaded on Canvas) and CATME team ratings also need to be completed as part of the documentation.

6 Demonstration

The demonstration will take place over zoom, with the system integration group member in the lab running the experiment, and the other group members participating remotely. Each group member should be comfortable answering questions about the overall system, as well as questions about the specific hardware or software that they were responsible for. The team should be able to do the following:

- Demonstrate the function of the overall system - i.e. regulation of the the position of a motor with the setpoint determined by an Aruco marker.
- Demonstrate the computer vision processing to detect the Aruco markers and show how this information is used to set the proper operating state. This demonstration can use intermediate images after each computer vision processing step.
- Explain in detail how the Arudino and motor driver drives the motors using a pulse width modulated output. Explain in detail how the Arduino reads the encoder positions. Open up Encoder.h and explain how the code counts when one interrupt pin is used vs two interrupt pins.
- Demonstrate the communication between Arduino and Pi.
- Demonstrate a step response experiment in hardware and in simulation. Demonstrate the design and implementation of the speed controller - show that the design specifications are met, and that the controller runs at a fixed, desired sample time.

Appendix: Verifying a Discrete Time Controller Using Simulink

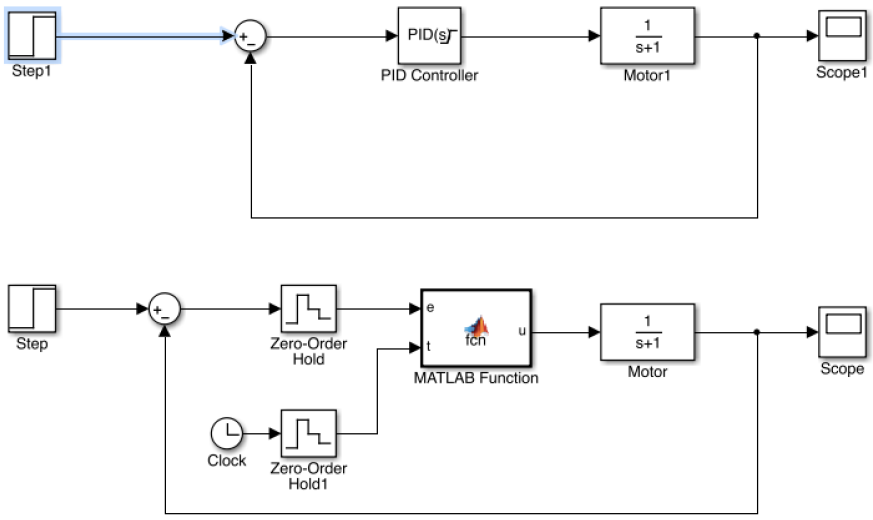
An easy way to simulate your embedded processor in Simulink, is to use a Matlab function block. The Matlab function block can be found in the User-Defined Function library. If you drag it into your Simulink block diagram, and double click on the function, it will open the Matlab editor. The following code can be used to implement a PI controller using the Matlab function block. Variables that need to be remembered between function called are defined as `persistent`.

```
function u = fcn(e,t)
%% Define Variables
% Control Gains
Kp=1;
Ki=1;
umax = 10;
% Memory variables
persistent I_past;
persistent t_past;
if isempty(I_past), I_past=0; end;
if isempty(t_past), t_past=0; end;

%% Calculate Controller output
% sample time
Ts = t - t_past;
```

```
% Integrator
I = I_past+e*Ts;
% PI control calculation
u = Kp*e+Ki*I;
% anti-windup
if abs(u)>umax,
 u = sign(u)*umax;
 e = sign(e)*min(umax/Kp, abs(e));
 I = (u-Kp*e)/Ki;
end;
I_past=I;
t_past=t;
```

The following block diagram shows a Simulink block diagram with one loop implemented using a PI controller, and another implemented using the Simulink block. The zero order hold block is used to choose the sample rate. This can be entered by double clicking on the element. We have also added a clock so the Matlab function knows the elapsed time.



Appendix: Using Matlab to run experiments on the Arduino and collect data

In order to determine if you system is operating correctly, it can be very useful to collect real time data (for example, the position of the wheel as a function of time). Matlab is a very flexible platform for displaying data, so a direct connection between the Arduino and Matlab will be ideal. On Canvas, on the **tips, tricks, and things to remember** page, you can find example code and instructions.