Mini Project: Motor Control*

EENG450AB: Systems Exploration, Engineering, and Design Laboratory

Vibhuti Dave and Tyrone Vincent

Department of Electrical Engineering
Colorado School of Mines

Spring 2018

1 Introduction

The purpose of this exercise is to design a control system that regulates the speed of a motor based using an Arduino and Rasberry Pi, along with an ultrasonic sensor and camera. The system should operate as follows:

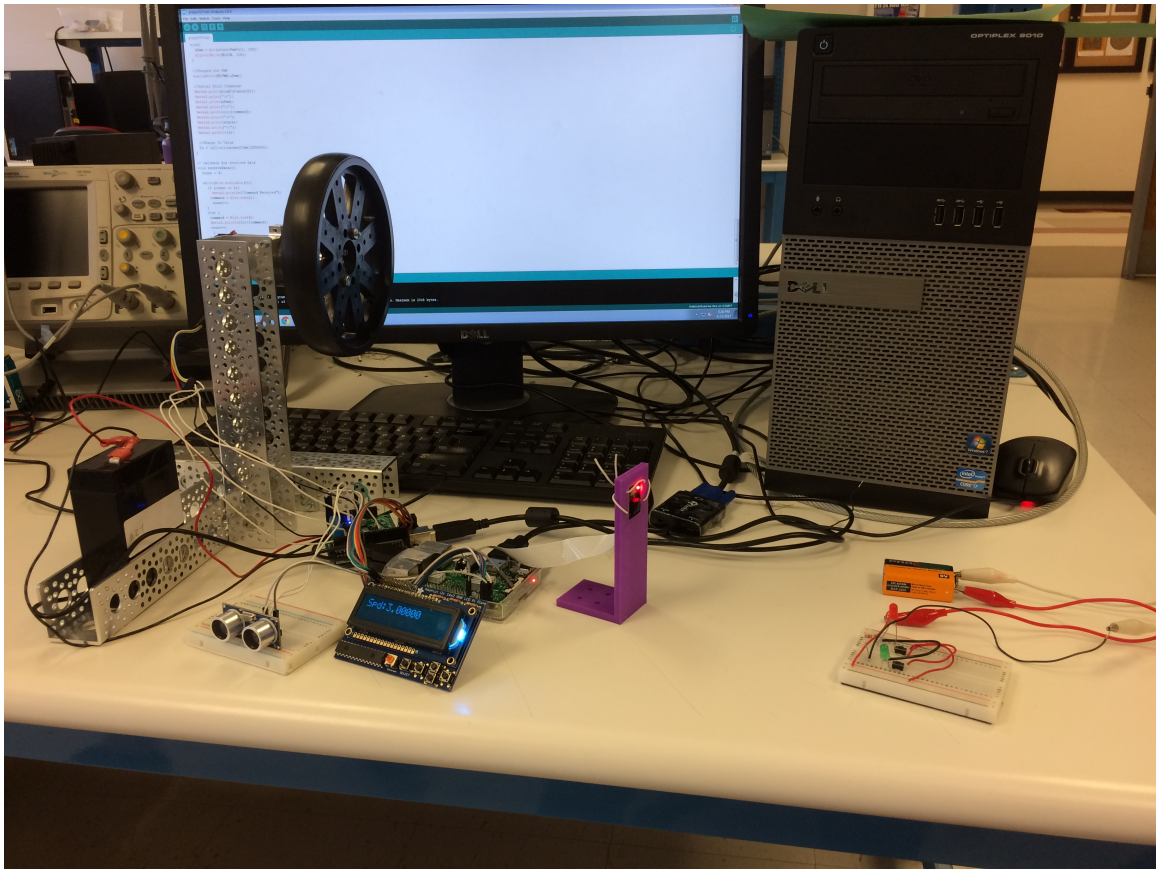
- When started the motor does not run.
- When the camera detects that a green LED is lit, the motor runs with the speed determined by the ultrasonic sensor. If no object is detected, the motor runs at maximum speed. If an object is detected, the distance d in centimeters is calculated, and the motor runs so that the wheel rotates at $d/10$ radians per second. The desired speed should also be displayed on an LCD screen.
- When the camera detects that a red LED is lit, the motor stops, and does not start again until green is LED is detected.
- If no LED is detected, the system should operate in the mode corresponding to the most recently seen LED (i..e the wheel spins if the last LED seen was green, it is stopped if the last LED seen was red).

2 Work Process

You are responsible as a team for completion of all elements of the mini-project. You should have weekly team meetings, with minutes taken and recorded. At the first meeting, elect team leader who will organize the meetings and ensure that they are effective. 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 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 mount the ultrasonic sensor to the base, and secure the camera and Pi using the supplied brackets. Put the LEDs on a breadboard, with switches that are used to turn them on. Don't forget to include a resistor to keep the current within limits. You can use the Arduino or Pi power pints to supply power to these LEDs.



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 Use Open CV to implement the color detection scheme

Open CV can be used to detect rectangles of appropriate dimensions and colors as in the problem description. 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 will want to take many images of the target LEDs, and process these images to determine the appropriate ranges for target color. Imaging or drawing programs can be used to determine the numerical values of the color at specific pixels that you select with a cursor. HINT: the LEDs put out a lot of light, so you can detect them more easily by using a very fast shutter speed and low iso. This is an example of changing these settings, although note this example is doing the opposite of what you want as it is making a very long exposure.

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

A description of the motor driver that you have been supplied can be found here. 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. Add the motor driver to the Arduino, hook up the motor driver to the motor with wheel attached and use the Arduino to spin the motor. 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. Use a 6V battery as the power supply for the motor driver. There is a jumper that allows this 6V battery to also be the power supply for the Arduino, but it is not necessary if you have the Arduino connected via USB. If later you want to use this functionality, you can find jumpers in the cabinet, or ask a TA. The Arduino is able to supply a pulse-width-modulated signal on the digital lines 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` and the pulse width.

4.3 Use the Arduino to read the rotary encoder

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 should say "*New* A high performance Encoder library is now available, 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

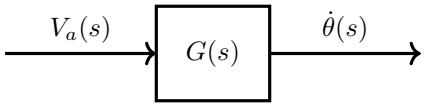
4.4 Detect distance using the Ultrasonic sensor and setup communication between the Raspberry Pi, Arduino and LCD

Hook up the ultrasonic sensor to the Arduino. Have the Arduino detect the distance from an object, divide the distance in cm by 10, and send this information to the Pi. This will be the setpoint for the speed of the wheel. Contrary to what we did before, write your code so that the ultrasonic sensor *does not* use an interrupt pin (i.e. pins 2 or 3). This can be done if the ultrasonic ping test is done within the `main()` function, so the code will wait until the reflected ping is returned. Because the data sent from Arduino to Pi is floating point number, you will first need to find a way to encode this information as bytes, send the bytes over I2C, and then decode this information to change it back to a floating

point number. Display the speed setpoint on the LCD. Be sure to print relevant messages on the Pi and Arduino side so you can keep track of data being sent and received.

4.5 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 speed (as measured by the encoder) as output. You will find a transfer function to model this response from input voltage V_a to angular velocity $\dot{\theta}$, $G(s) = \frac{\dot{\theta}(s)}{V_a(s)}$.



1. Set up an Arduino program that will perform a step response experiment with an interface to Matlab. 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. You will also want to signal the user or display an error message if the main function takes longer than the desired sampling rate. 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.

   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**. Use Matlab to read the experiment data over the serial line and plot the results. Choose a sampling rate between 1 and 10ms.

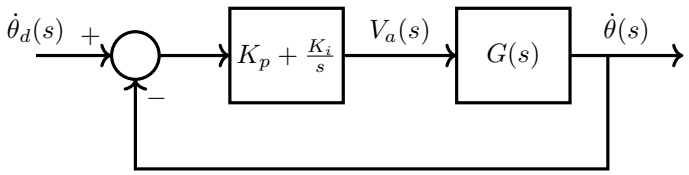
2. Perform a step response experiment, and estimate the transfer function from motor voltage command to angular velocity. Although the motor is theoretically second order, when the motor inductance is small, the response can be well approximated as first order, so the transfer function for the motor 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

3. Create a simulation that replicates your step response experiment, using the transfer function you identified in the step above.

4.6 Design a controller for your motor

Using the transfer function $G(s) = \frac{\dot{\theta}(s)}{V_a(s)}$ as the system to be controlled, design a Proportional-Integral (PI) controller to regulate the motor speed.



That is, you should be able to specify a desired rotational velocity $\dot{\theta}_d$ in radians per second that the wheel will turn, even if there are disturbances.

1. Design a PI controller that achieves closed loop step response specifications of a rise time of 250 ms and an overshoot of less than 5%, with zero steady state error. Document your design procedure. (You can check your design against the autotune function of the Simulink PID block, but you should not use this as your primary design mechanism.) Simulate the closed loop system in Simulink.
    - For review, read the EENG307 Notes: Lecture 19, PID Control
2. Implement the controller using the Arduino. Don't forget about units!
    - Implementation is discussed in the Appendix of EENG307 Notes: Lecture 19, PID Control, Appendix.
3. Create an Arduino program that will find the closed loop step response (that is, changing the setpoint from 0 to 1 rad/s), with serial communication to Matlab, so that the results are compared to a Simulink simulation of your design.

4.7 Put everything together

Put everything together. The Pi monitors the camera and tells the Arduino when to start and stop. The Arduino will measure the distance to an object with the ultrasonic sensor in centimeters and uses this value divided by 10 as the desired speed setpoint $\dot{\theta}_d$ in radians per second. The Arduino also sends the distance back to the Pi. The Arduino implements the Proportional/Integral controller which regulates the motor/wheel speed. Thus, the wheel should spin at 5 rad/s if the measured distance is 50 cm. Display the speed of the wheel and the measured distance on the LCD.

5 Documentation

The following documentation should be uploaded to the Canvas assignment "Demo 1 Team Documentation".

1. A short document that describes your process for detecting red and green LEDs.
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 simulated motor response, and
    - the actual motor response.
4. Your team charter and agenda/minutes

For future use, store 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 (speed 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

As a group, you should be prepared to demonstrate the following. Each member of the group should be comfortable demonstrating the overall system and one other item.

- Demonstrate the function of the overall system - i.e. regulation of the the speed of a motor based on the distance of the users hand from an ultrasonic sensor, with stop and go regulated by the camera.
- Demonstrate the computer vision processing to detect the LEDs, 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.
- Demonstrate how the Arudino and motor driver drives the motors using a pulse width modulated output. You should show the motor voltage on an oscilloscope and predict how they change with different values for the analogWrite and/or digitalWrite commands. Demonstrate how the Arduino reads the encoder positions. You should be able to show the raw encoder signals on an oscilloscope and discuss how they are decoded. 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. Use the oscilloscope to show the data and clock pins and illustrate how the the I^2C protocol works, referencing the information about the protocol on this page. Demonstrate the logic for transmitting floating point information.
- 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.

7 Appendix: Verifying a Discrete Time Controller Using Simulink

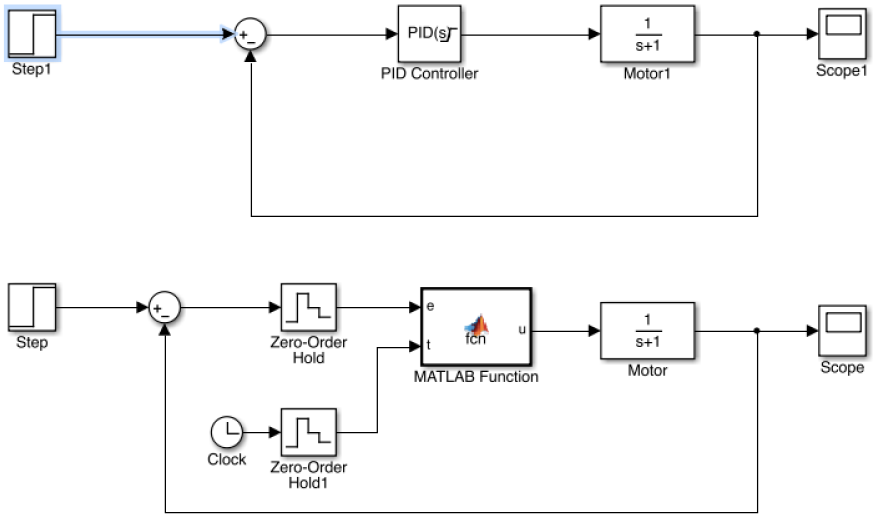
An easy way to simulate your embedded processor in Simulink, is to use a Matlab function block. The Matlab function block can be found in the User-Defined Function library. If you drag it into your Simulink block diagram, and double click on the function, it will open the Matlab editor. The following 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.



8 Appendix : Set up remote access to the Rasberry Pi from your laptop

Remote access is convenient so you don't have to go back and forth between keyboards and monitors. It will also be helpful when your Pi is on the robot and you want to change its code without having a wire connected to it.

Set up the Raspberry Pi with a monitor, desktop and keyboard. Make sure it is connected to the internet. Using the terminal, type hostname -I. This will give you the IP address for the Pi. Install VNC on your Pi. Then install a VNC client on your PC/laptop. You should be able to see the desktop for the Pi on your laptop screen. Now you can use the laptop's keyboard, mouse and monitor to work on the Pi and still have access to your actual laptop. Another way of using the Pi remotely is to use SSH. But this will not show you the GUI for the OS. You will need to be comfortable working with a command line interface. You can use Putty to SSH from a Windows based laptop. For a MAC OS, you don't need any additional software.

Regardless of how you choose to connect to the Pi remotely, you will need to know the IP address of the Pi. I have luckily managed to get the same IP address for my Pi every time I connect at Mines over wifi. But since this IP address is dynamically assigned, it could change. Another option is to submit a work order with CCIT to obtain a static IP address for the Pi. This way, you will always know what the address is. However, you will need to change the hostname from the default "raspberrypi" to something else to avoid conflicts with other Raspberry Pis (such as mine) on the same network. The static IP address will work over Ethernet. This means you will need to have access to an active Ethernet port as well as an Ethernet cable at Mines. If you choose to go this route, wifi gets disabled. CCIT followed these instructions to set up the Pi with a static IP. I personally stuck with VNC and a wifi connection.

Keep in mind, the IP address will change when connecting remotely to the Pi at home or on a different network. If you are going headless (no monitor), finding the IP address can be more complicated. Once you have an IP address,

follow the same instructions described above.