Mechatronics Laboratory Assignment 5
Motor Control and Straight-Line Robot Driving

Recommended Due Date: By your lab time the week of March 6th
Possible Points: If checked off before your lab time the week of Mar. 13th … 14 points
If checked off after your lab time the week of Mar. 13th and before 4:00PM Mar. 17th … 11 points
If work not finished before 4:00PM Mar. 17th … 0 points

Goals for this Lab Assignment:

1. Learn why feedback control is used to complete a straight-line driving task by comparing open-loop, decoupled loop, and coupled-loop control methods.
2. Create a LABVIEW interface to communicate with the robot and display the current robot position.

DSP/BIOS Objects Used:

PRD

Library Functions Used:

Global variables: enc1_rad, enc2_rad, enc3_rad and enc4_rad, switchstate
Functions: void PWM_out(enum epwm ep, float u)

Lecture Topics:

PI Control and Integral Wind-up

Prelab:

As a prerequisite for this class, all students should have completed a first course in control theory. We therefore expect that at some time, the student has already learned about basic feedback loops, specifically PI control. Please review your notes on this subject, as you are expected to implement a PI-type controller in lab. Read the lab and begin coding a PI speed controller for the right and left drive sides. This will help you finish the lab in a timely fashion.

Laboratory Exercise: In this lab, we will be attempting to control the motion of the robot and testing this by driving the robot in a straight line. There are many different ways to do this; we will investigate very standard techniques:

Exercise 1: Open-Loop Robot Driving.

Starting with a new project, implement open-loop control of the robot just as you did in Lab 3. The value read from optical encoder 3 divided by 100 should be sent to both motors as an open-loop torque command. We are going to compare the open-loop to the closed-loop controls designed later in lab. Setup your program (in a switch-case statement) so that when all the dip-switches are FORWARD (towards the front of the robot) nothing is output to the motors. Also display “Robot Off” to the LCD screen. When the first dip-switch is back and all others forward, run the open-loop controller. Display something like “Open Loop” to the LCD screen.

*Always test your code out first on the bench before putting the robot on the ground.*

We suggest a certain organizational structure to help you better debug your code. The following pseudo-code will help guide you through the steps needed. Inside the “void control(void)” function just after the call to “updateData()”:

1. Remember that “void control(void)” is the function for the SWI, SWI_control. SWI_control is posted in the given code every 1ms after all the optical encoder channels have been read.
2. Increment a long integer to keep track of elapsed time;
3. Use the drive-motor optical encoders (Encoders #1 & #2) to calculate the motor velocity in tiles/second;
4. Calculate the robot’s position from the average of the two motor velocities;
5. The global variable “switchstate” contains the status of the four white switches on the “shield” board just to the left of the color LCD screen. Since there are 4 switches, the value of switchstate ranges from 0 to 15.
6. Use a switch-case code structure to implement different control algorithms for different dip-switch settings;
7. Inside the open-loop case statements, set control efforts u1 and u2 to Encoder #3’s value divided by 100. In exercises 2, 3 and 4, encoder #3’s value will be used as the desired velocity set point for the control or the desired turn set point.
8. For this open loop case do not add friction compensation. Also make sure NOT to apply friction compensation when the robot is in the OFF state. The OFF state should send 0, with “PWM_out”, to both motors.
9. Output u1 and u2 to the motors with the “PWM_out” command;
10. Save past variables as needed;
11. Print to LCD on line 1 a text message indicating which control algorithm is active and on line 2 the open-loop command (or reference velocity) and measured motor velocities with only 1 decimal resolution. Remember not to print too fast to the LCD.

Right now just test your code on the bench leaving the robot on the cake pan. Just like in Lab 3, you should be able to turn encoder #3 to command robot’s wheel to spin forwards or backwards. After you have completed the 3 controller modes below, you will then place the robot on the floor and compare each of them.

**Exercise 2:** Closed-Loop Robot Driving with Decoupled Control Loops (See Figure 1).

Partners switch so that the other person can do some coding.

1. Add another case statement to check if the second switch is back and all others forward. If so, your case statement will calculate u1 and u2 from two different PI control loops. You will be implementing the code equivalent to the diagram in Figure 1. For friction compensation use 60% of the values you found in lab 3. As a start, use Kp = 3.0 and Ki = 5.0. These are good starting gains assuming that all velocities are measured in tiles/second. You will tune these slightly in lab to achieve a more responsive PI control. Also, print a message to the LCD screen like “PI”. Before going onto step 2, observe integral windup. Start your code, but leave the motor amplifier off. After a few seconds turn on the amp. The wheels should take off spinning at maximum speed. After a few seconds the wheels will die back down to the desired speed.
2. Check for integral wind-up. Add another ‘if’ statement inside your case statement that checks if the control efforts (u1 for motor 1 and u2 for motor 2) are within the range of +/- 10. If the control command is outside this region, make sure that the integral term does not wind up by multiplying the integral sum by 0.99. This will force a decay of the integral term.
3. Build and run your code. Test the robot on your bench. When you change the desired speed (you should name this variable “vref”) by rotating encoder #3, the robot’s wheel velocity should, after a small transient, match the desired speed. Tune the Ki gain slightly to see if you can achieve a quicker transient response in motor speed.
Exercise 3: Closed-Loop Robot Driving with Coupled Control Loops (See Figure 2).

1. Add another case statement to see if the third switch is back and all others are forward. In this case statement, implement the coupled control algorithm shown in figure 2. Use a Kp_turn value of 3.0. Also, print a message to the LCD screen like “Coupled PI”. Use the tuned gains you found in Exercise 2.

2. For this case, “turn” should always be zero and encoder 3 controls the reference velocity.

3. Again, make sure that all integral terms do not wind up using the method given earlier.

4. Build and run your code. Test this code on the bench. Grab one of the wheels and see what happens to the wheels on the opposite side. What is the advantage of this control implementation?
Exercise 4: Driving around.

1. Add another case statement to see if the forth switch is back and all others are forward. In this case statement you will again implement the coupled control algorithm shown in figure 2, but reference velocity should be fixed at 1.0 tiles/second and “turn” updated by the value of optical encoder #3. Also, print a message to the LCD screen like “Coupled PI Turn” and on the other line print the value of “turn”.

2. Build and run your code. When you drive the robot around, you should be able to steer it easily with the attached encoder #3.

3. Also at this time try out your 3 other straight line modes on the floor. Can you see a difference between open loop and closed loop?

In summary, your final program should use the following switch logic:

- If switch 4 is back and all others are forward, run the coupled PI control law with Enc3 changing “turn”
- If switch 3 is back and all others are forward, run the coupled PI control law with Enc3 changing “vref”
- If switch 2 is back and all others are forward, run the two independent PI control laws.
- If switch 1 is back and all others are forward, run the open loop control law
- If all other cases, robot should not move.

Exercise 5: Create your initial LABVIEW interface. Same assignment as Lab 3’s extended Prelab.
Add code to your DSP program so that your DSP sends the PC the following information: its x-location (in tiles), its y-location (in tiles), its velocity (in tiles/second), and its orientation (in radians). See Ex 5 Step 3 below for the method of communicating this data to LabView. Note: you have not been introduced to the rate gyro and the compass sensors which will give you a bearing measurement. For Lab 5 then y and orientation will always be zero. In future labs additional sensor readings may also be sent. On part of your application window, the LABVIEW routine should plot the position and orientation of the robot. As your robot moves around on the floor your depiction of the robot should move around in your LABVIEW application. Use Lab 4’s LABVIEW assignment as a starting point for this assignment. Below are steps to guide your development of this LABVIEW application. Each lab assignment from here on out will ask you to continue adding functionality to your LABVIEW application and demonstrate it working at each of your lab check-offs.

1. Create a LABVIEW program whose window uses most of the screen. In this window create an area that will represent the robot course. Use a “2D Picture” object to create this area. In the 2D picture draw grid lines for a 16 by 16 tile course. Use a circle to indicate where the robot is located in the course and draw a line to indicate the direction and velocity of the robot. The length of the line will indicate the velocity.

2. You will be able to pick a starting location and orientation for your robot in the course.

3. Using the LABVIEW code from Lab 4, receive the X, Y, angle and velocity data sent from the DSP over WiFi. To send your variables to LABVIEW from the F28335 processor, you simply assign the four given variables F28335_Extra1, F28335_Extra2, F28335_Extra3, F28335_Extra4 to your x, y velocity and angle values. In Lab 6 we will be learning more about how all this data transfer is occurring. On the LABVIEW end parse the string containing the four variables and convert the separate strings into a number using the “Scan from String” block.

This ‘dead reckoning’ method of navigation we are using has errors that propagate the farther you travel (i.e. the errors are integrated). While completing the remaining labs, start thinking about how you might use various sensor readings to measure course features that will help correct your robot position estimate. Examples include measuring wall distances to correct (x, y) positions, using visual landmarks to correct angular errors, etc.

Lab Check Off:
Show your 4 control methods and your LABVIEW interface to your instructor. Your LABVIEW interface should indicate the robot’s X position and velocity. You should be able to answer the following questions:

1. Which method worked best? Did you notice any significant behavioral differences in the controllers?

2. What was the point of the coupled loop approach versus the decoupled loop approach? Hint: Grab one of the wheels and hold it to prevent only one motor from moving. What happens to the other motor? What kind of robot driving situations would cause one wheel to stop moving?

3. How would you implement steering commands using the open-loop and independent PI control methods?