

Mechatronics Laboratory Assignment 5

Motor Control and Straight-Line Robot Driving

Recommended Due Date: By your lab time the week of March 9th

Possible Points: If checked off before your lab time the week of Mar. 16th ... 14 points

If checked off after your lab time the week of Mar. 16th and before 4:00PM Mar. 20th ... 11 points

If work not finished before 4:00PM Mar. 20th ... 0 points

Goals for this Lab Assignment:

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

DSP/BIOS Objects Used:

PRD

Library Functions Used:

Init_encoders, read_encoders, Init_pwm, out_PWM, Init_UART1and2, LCDPrintfLine1, LCDPrintfLine2, WirelessSend

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 the PI controllers for the right and left drive sides. You might want to test your code in office hours before class. 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.

Create a new project. You will be creating this entire code yourself, but we suggest a certain organizational structure to help us better debug your code. The following pseudo-code will help guide you through the steps needed.

Inside a PRD function called every 1ms:

1. Increment time;
2. Read the attached encoder to obtain an open-loop value to output to the motors. In later exercises this reading will be the velocity setpoint;
3. Read the drive-motor encoders, and calculate the motor velocity;
4. Check for encoder rollover and correct velocity if needed;
5. Calculate the robot's position from the average of the two motor velocities;
6. Read the dip-switches on the DSP

7. Using a switch-case code structure to implement different control algorithms for different dip-switch settings, calculate control efforts u_1 and u_2 .
8. Modify resulting u_1 and u_2 using friction compensation (use 60% of the values found in Lab 3);
9. Output u_1 and u_2 to PWM signals to motor;
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.

For exercise one 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 UP nothing is output to the motors. Also display "Robot Off" to the LCD screen. When the first dip-switch is down and all others up, run the open-loop controller. Display something like "Open Loop" to the LCD screen.

Always test your code out first on the bench before flashing the code and putting the robot on the ground. When the code is working, flash your code to the robot (see Lab 3 if you forgot how).

With the robot on the floor, drive the robot back and forth and see how straight it follows a line. Your TA may have small pieces of carpet or other objects for you to place in your robot's path. Place them such that the robot must drive over the object with only one side of the tires. Drive over the object at different speeds, and mentally note how the robot turns as it encounters the object.

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 down and all others are up. If so, your case statement will calculate u_1 and u_2 from two different PI control loops. You will be implementing the code equivalent to the diagram in Figure 1. As a start, use $K_p = 3$ and $K_i = 5$. These are just suggestions and the gains assume that all velocities are measured in tiles/second... you may modify them if you wish but you will almost certainly need to tune them better for later projects. Also, print a message to the LCD screen like "PI".
2. Add another 'if' statement inside your case statement that checks if the control effort (u) is 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. Why is this necessary?
3. Build and run your code. Test the robot just like the open-loop case.

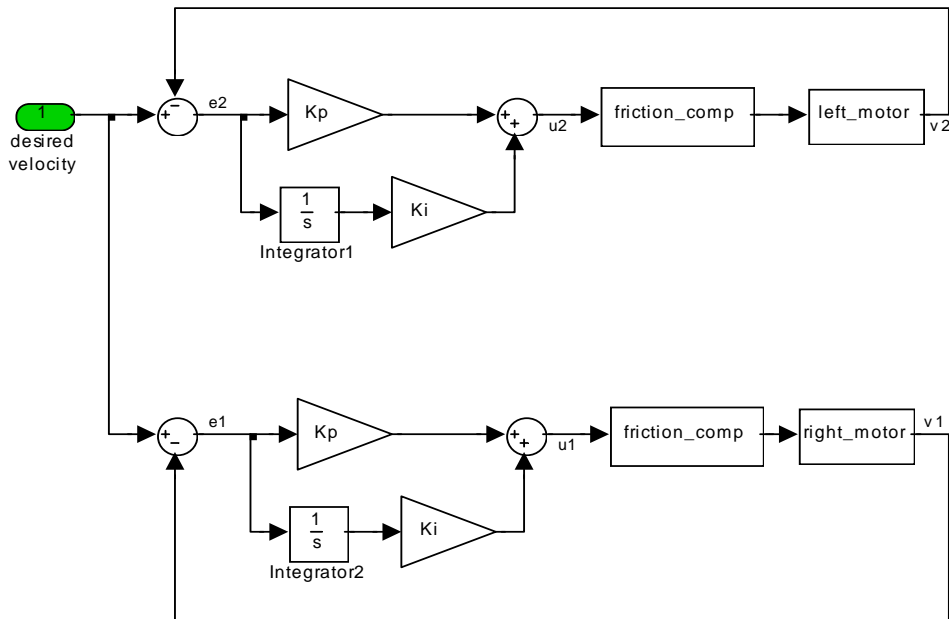


Figure 1: Independent PI control loops for the right and left motors.

Exercise 3: Closed-Loop Robot Driving with Coupled Control Loops (See Figure 2).

4. Add another case statement to see if the third switch is down and all others are up. In this case statement implement the coupled control algorithm shown in figure 2. Also, print a message to the LCD screen like “Coupled PI”. Again, all proportional gains should be approximately 3 and all integral gains should be roughly 5. These are just suggestions... you may modify them if you wish.
5. For this case, “turn” should always be zero and encoder 3 controls the reference velocity.
6. Again, make sure that all integral terms do not wind up using the method given earlier.
7. Build and run your code. Test the robot just as you did earlier.

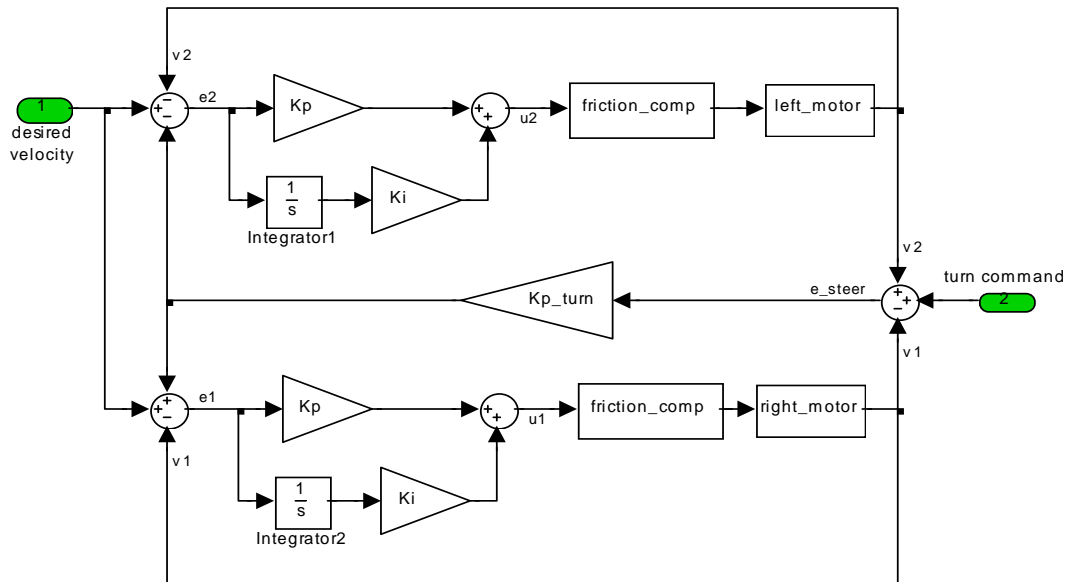


Figure 2: Coupled PI control structure.

Exercise 4: Driving around.

8. Add another case statement to see if the forth switch is down and all others are up. 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”.
9. Build and run your code. When you drive the robot around, you should be able to steer it easily with your attached encoder.

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

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

Exercise 5: Create your initial VB interface. *Same assignment as Lab 3’s extended Prelab.*

Add code to your DSP program so that every 0.25 seconds, 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). *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 VB 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 VB application. Use Lab 4’s VB assignment as a start for this

assignment. Below are steps to guide your development of this VB application. Each lab assignment from here on out will ask you to continue adding functionality to your VB application and demonstrate it working at each of your lab check-offs.

1. Create a VB program whose window uses most of the screen. In this window create an area that will represent the robot course. One way to do this is to use a picture box object. In the picture box draw grid lines (in software using the *Line* method) for a 20 by 20 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. Find in VB's Help (See VB's function reference help section) find the following to help you with these tasks: PictureBox Autoredraw property, PictureBox ScaleTop, ScaleHeight, ScaleLeft and ScaleWidth property, a shape object, a line object, PictureBox Cls, Line and Circle Methods.
2. You will be able to pick a starting location and orientation for you robot in the course.
3. Using the VB code from Lab 2/Lab 4, receive the X, Y, angle, velocity data sent from the DSP over the wireless serial interface. Send your variables in a string just like we sent the intime variable to VB in Lab 2. This way you can determine the number of decimal places to send for each of your variables. A little more code is needed at the VB end for this method to parse the string into your individual variables. VB has a function called "Split" that can parse a delimited string into separate strings. After the string has been parsed you can use the VB function "Val" to convert the separate strings into a number. VB's help shows an example of using the Split function. (See VB's function reference help section). Later, we will be adding sensor readings to the robot; these will be displayed in text boxes in your VB application.

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 VB interface to the TA. He or she will ask to see your robot run, and may change the switches to observe the different behavior. Your VB 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?