SE423 Laboratory Assignment 5 (One Week Lab)

DC Motor PI Speed Control and Robot Steering

Goals for this Lab Assignment:

1. Merge code from previous labs to produce code that samples both ADC channels and optical encoders for feedback and output's PWM to control and drive the DC motors of the robot.
2. Learn to implement the PI control law to control the speed of the robot's wheels.
3. Couple the PI controller for the left and right wheels in order to control both the speed and turn rate of the robot car.

Reading before Lab

1. Read through this entire lab.
2. Review your notes on the PID controller from the Continuous Control class you have taken, (SE320, ME360, ECE486, …). We will be implementing a PI control in this lab.

Laboratory Exercises

Exercise 1: Start from Labstarter but Merge ADCC code from Lab 4.

First start out by creating a new project using LABstarter and rename in LAB5<yourinitials>. Then merge the ADCC code from Lab 4 into this labstarter. There are two ways you can accomplish this. One way would be to go through Lab 4's Exercise 3 code and copy all the parts needed for sampling ADCC's four gyro channels every 1ms. The second way would be to copy the entire Lab 4 C file and paste over all the code for this new Lab 5 labstarter. Then you would need to go through this pasted code and remove all the code associated with ADCD (function generator) and ADCB (microphone). Either way you choose, take your time and go through your C code and make sure you have included all parts to sample ADCC channels C2, C3, C4 and C5 and have removed all the parts for ADCB and ADCD. Before you go on to Exercise 2, make sure to run this code and make sure it is working to sample the four ADCC channels every 1ms. Print every 100ms to the robot's text LCD the degrees/second readings of ADCINC2, ADCINC3, ADCINC4 and ADCINC5 after their respective "zeros" have been subtracted from the readings. Use the below item list to remind you what should be included: (All items are not necessarily listed here. Add other steps as needed.)

1. Setup EPWM4 and ADCC so that EPWM4 triggers four ADCC SOCs in order that channels C2, C3, C4 and C5 are sampled every 1ms.
2. In ADCC's ISR function convert the four channel's values to volts.
3. For the first 2 seconds in ADCC_ISR do nothing with the voltage readings allowing the IMU to settle in the case this was a power on cycle.

4. For the next 2 seconds sum up each channel's reading and on the 2000th sample divide by 2000 to find the channel's zero approximation.
5. Then for the remainder of calls to ADCC_ISR take the channel's voltage reading and subtract its zero value. Then multiply by the correct factor that converts the voltage reading to degrees/second.
6. Print these four readings to the text LCD every 100 ms.
7. Remember in main() that you need to initialize:
    a. PieVectTable with the ADCC_ISR function and remove ADCD_ISR and ADCB_ISR if needed.
    b. PIEIER1 enabling the correct minor interrupt and disable the minor interrupt for ADCD and ADCB if needed.

Demo your code working to your TA.

Exercise 2: Now Merge Lab3 Code to Achieve Open Loop Driving of the Robot Motors and Calculate Speed.

Exercise 2 is also a code merging exercise that will add all the PWM motor driving and optical encoder velocity calculations we performed in Lab 3. This code merge should be a bit easier as we do not have to worry about new interrupt functions.

1. Copy from your Lab 3 code to the bottom of your Lab 5 code the functions, setEPWM1A, setEPWM2A, init_eQEPs, readEncLeft, readEncRight, readEncWheel. Also make sure to copy the predefinitions of these functions towards the top of your Lab 5 code.
2. In main() of your Lab 3 code find and copy your initializations of EPWM1 and EPWM2 and paste after the init_serial functions in Lab 5's code. Also make sure the call init_eQEP() after these initializations of EPWM1 and EPWM2.
3. Copy all the global variables that you created in Lab 3 to calculate velocity of the left and right motor and command the PWM to the motors.
4. In Lab 3, you setup one of the CPU Timers to call its interrupt function every 1ms. In this lab I would like you to use the ADCC's ISR function instead which is already being called every 1ms. Copy all the code you wrote in Lab 3 that was called every 1 ms is the CPU Timer into your ADCC_ISR function. The ADCC_ISR code should be close to the following order: (again I may not list every step here, you add as needed)
    a. For the first 2 seconds when the ADCC readings are being ignored, simply call your setEPWM1A and setEPWM2A functions with zero passed to them so the motors do not move during those first 2 seconds.

b. In the same way for the 2nd 2 seconds also call the setEPWM1A and 2A functions passing zero so the motors do not move.
c. Then from 4 seconds on
    i. Read the left and right wheel encoder values and scale to units of feet.
    ii. Use these wheel encoder values along with the previous encoder reading, 1 ms. prior, to calculate both the left and right wheel velocity in units of feet/second.
    iii. Read the Encoder Wheel's (Encoder 3) value. Here we are just using this value as command to the motors, so set uLeft and uRight equal to the Encoder Wheel value like you did in Lab 3.
    iv. If you would like to copy your friction compensation code, you can here, but we are going to leave it commented out until exercise 3 when we implement PI speed control.
    v. Copy your code that makes sure uLeft and uRight are not greater than 10 or less than -10.
    vi. Call the setEPWM1A and 2A functions passing them either uLeft or uRight as you did in Lab 3. Don't forget the negative sign on one of those values.
    vii. Make sure to save your previous encoder positions to be used the next 1 ms. to calculate velocity.
    viii. Print every 100ms to the text LCD your Left and Right velocities and the value of the Encoder Wheel.

5. Compile and run your code. Using the third encoder dial in different commands and make sure the robot wheels spin in the correct direction and the velocities displayed are the correct units. **Show this working to your TA.**

Exercise 3: Closed-Loop Speed Control with Decoupled Control Loops (See Figure 1).

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

1. In this exercise you will implement the decoupled PI controller shown in Figure 1. NOTE here that we will change the function of the encoder wheel (encoder 3). Instead of being used to set uLeft and uRight directly as in exercise 2, the value read from the encoder wheel will set the "desired velocity" shown in Figure 1. From here forward we will call this desired velocity "Vref" (Velocity Reference). uLeft and uRight will be set to the PI control equation. For example, uLeft = Kp*errKLeft + Ki*IKLeft.
    a. Modify the exercise 2 code that after 4 seconds drives the motors in an open-loop fashion to a PI closed-loop controller. After your code that calculates the speed of each wheel, set a Vref

float variable to the encoder wheel's value (Encoder 3) divided by 20 and calculate errLeft (e1) and errRight (e2) using Vref and the wheel speeds.

b. Then using the current error, a previous error and the previous calculated integral, use the trapezoidal rule to find both IKleft and IKright.

c. Now that you have the error and integral states, you can calculate your uLeft and uRight from the equations shown in Figure 1. For friction compensation use 60% of the columbic and viscous 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.

d. Change what prints to the text LCD to the speeds of each wheel and your Vref value.

e. Build and run your code. Use the encoder wheel to dial in a Vref command. You should see your wheel's velocities track this Vref value. Try both positive and negative speeds.

f. Before going onto step 2, observe integral windup. Start your code, but leave the motor amplifier off. Dial in a Vref of 1 foot/second and 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. To see what is happening add your IKLeft and IKRight floating point variables along with uLeft and uRight to Code Composer Studio's Watch Expressions window. What happens to IKLeft and IKRight the longer you wait to turn on the motor amp? **Demo this to your TA.**

2. Correct for integral wind-up. Add two "if" statements after you have calculated uLeft and uRight from the PI equation. If the absolute value (fabs() floating point absolute value) of uLeft/uRight is greater than the maximum PWM output value of 10 then stop integrating by setting IKLeft/IKright = IKoldLeft/IKoldRight. In other words if the control effort is at the maximum value it can output, than there is no reason for the integral to get any larger. NOTE: Make sure your saving of the past states IKoldLeft,IKoldRight is performed after these if statements.

3. Build and run your code. Test the robot on your bench. When you change the desired speed, Vref, by rotating encoder #3, the robot's wheel velocity should, after a small transient, match the desired speed. Check that you have corrected the integral windup issue for both positive and negative Vref set points. Tune the Ki gain slightly to see if you can achieve a quicker transient response in motor speed.

   Demo this working to your TA.

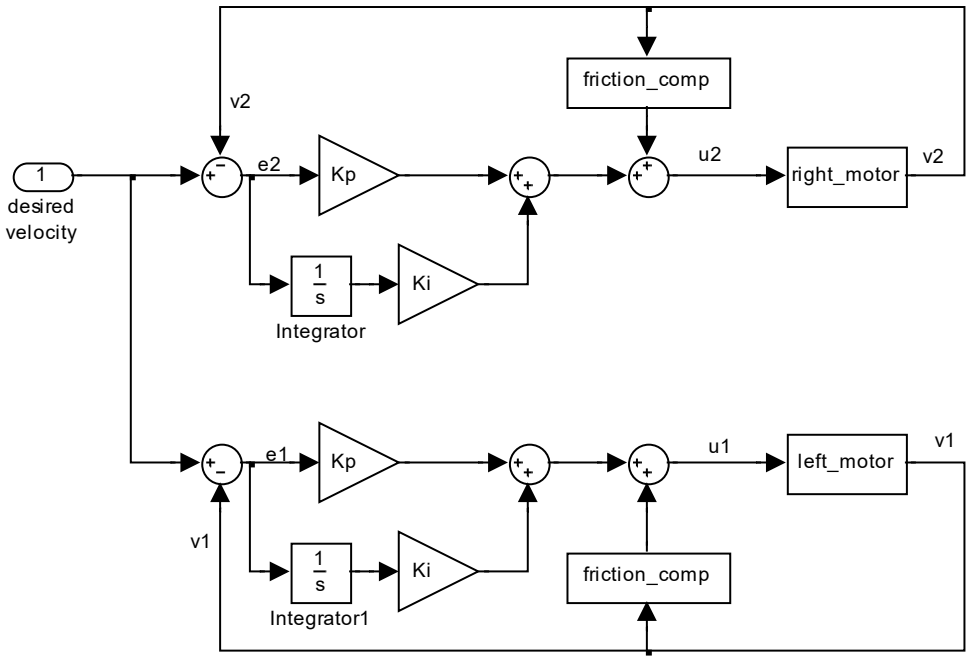


Figure 1: Independent PI control loops for the right and left motors.

Exercise 4: Closed-Loop Speed Control with Coupled Control Loops (See Figure 2).

1. Implement the coupled control algorithm shown in Figure 2. Use a Kp_turn value of 3.0. Use the tuned Kp and Ki gains you found in exercise 3. Pay close attention to the '+' and '-' in the block diagram sum blocks. Most of your PI control law equations remain the same. It is only the errLeft and errRight equations that change and note that they are slightly different from each other.
2. For this exercise keep "turn" equal to zero and Vref equal to the encoder wheel divide by 20.
3. Build and run your code. Grab one of the wheels and see what happens to the wheels on the opposite side.

   Demo this working to your TA.

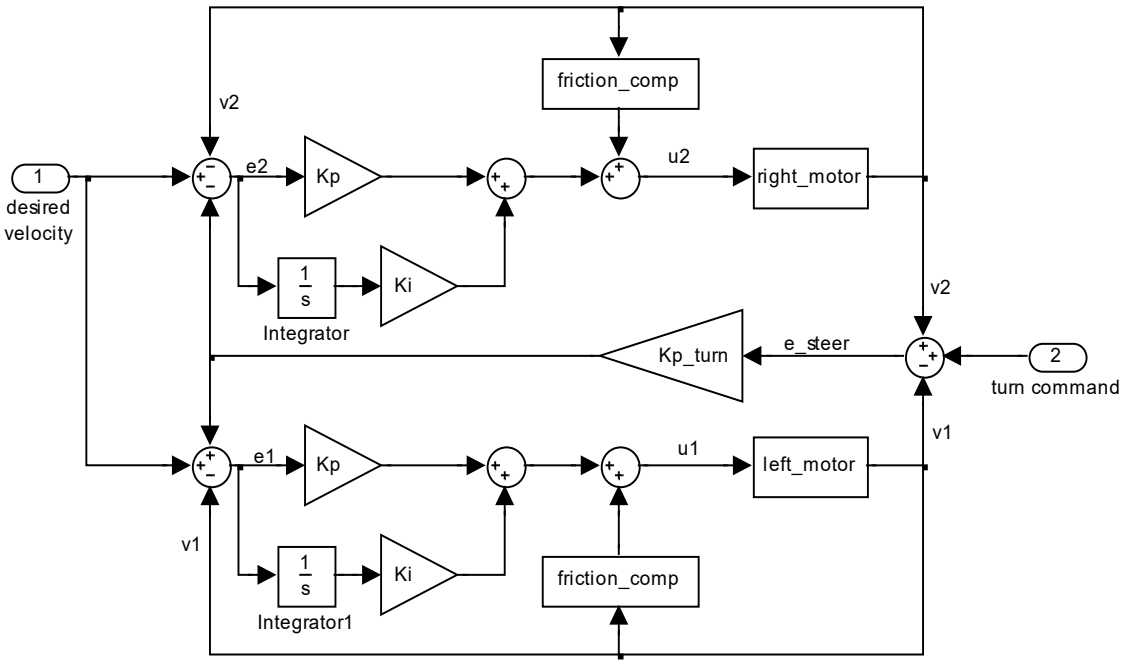


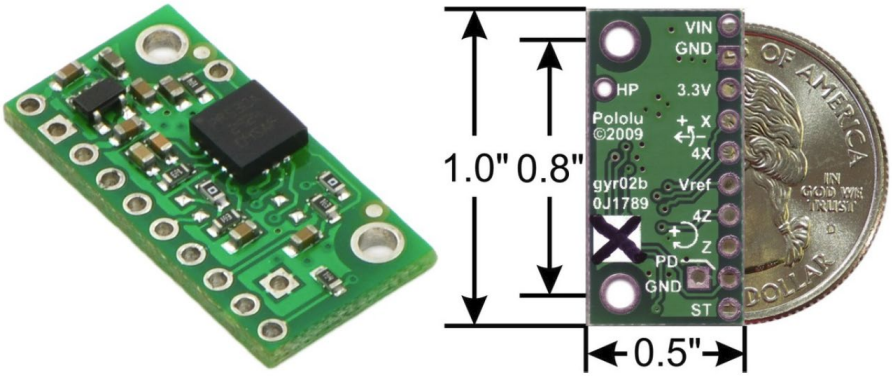
Figure 2: Coupled PI Control Block Diagram

Exercise 5: Driving Around (See Figure 2).

1. The coupled control algorithm has two command inputs, Vref and turn. For the remainder of this semester these are the two commands your C code is going to use to drive the robot around its course. By adjusting the "turn" variable you can control the rate of turn of the robot. It does not have units of degrees or radians per second but it is proportional to the robot's rate of turn. Change your code so that Vref is constant at 1 foot/second and turn is now equal to the encoder wheels value divided by 20.
2. Build and Flash your code to your robot. First run your code on the cake pan at your bench to see that it is working to steer the robot. Then when you know it is working, disconnect the cables to the robot and put it on the floor and power it with a battery. Power on the robot and after four seconds enable the motor amp. Your robot should start going forward at 1 foot/second. Use the encoder wheel to dial in turn values that make the robot steer around its environment.

   Demo this working to your TA.

Exercise 6: Integrate the LPR510 Gyro Rate to Calculate Bearing Angle of Robot.



4Z(ADCINC4) and 4X(ADCINC3) (4 here means 4 times the accuracy) Voltage Range ≈0.23V -> -100^o/s, ≈1.23V -> 0^o/s, ≈2.23V -> 100^o/s

Z(ADCINC5) and X(ADCINC2) Voltage Range ≈0.23V -> -400^o/s, ≈1.23V -> 0^o/s, ≈2.23V -> 400^o/s

1. In this final exercise you will integrate both of the LPR510 Gyro Z readings to find two measurements of the bearing angle of the robot. Recall from your PI controller how you used the trapezoidal integration rule to find the approximation of the integral of the velocity error. Using the trapezoidal integration rule, the current and previous gyro z readings and the previous calculated bearing angle, calculate the current bearing angle in units of radians. Do this for both the Z (+/- 400^o/s range) and 4Z (+/- 100^o/s range 4 times more accurate) Gyro outputs.
2. Build and Run your code. Turn the robot in place back and forth and check that your calculated angles make sense. For example, turn the robot approximately 90 degrees and see that the bearing angle is close to 1.57 radians. Do you notice any difference between the two gyro resolutions? Do you notice a drift in the calculation of the bearing over time? Sit the robot still and watch the rate the bearing drifts from zero.

Lab Check Off:

1. Demonstrate your merged code from Lab 4 and Lab 3 working together in Lab 5's project.
2. Demonstrate your Decoupled PI Speed control working controlling the speed of the motors with robot sitting on its cake pan. It of course would also work on the floor but that will be shown in Exercise 5.
3. Demonstrate your Coupled PI Speed control working controlling the speed of the motors with robot sitting on its cake pan.
4. With the robot on the ground, demonstrate that you can steer your robot around its environment.
5. Demonstrate that you can integrate the LPR510's Z gyro axis to find the robot's bearing. Do you notice a drift in the integral?
6. Submit all your written code after adding comments explaining your code and what you learned. Also be clear what code is for what exercise. There is no "How To" submission for this lab.