ME 461 Laboratory #9
Wall Following and RC Driving

Goals:

1. Use two of the myRIO's ADC channels to read the voltage output of two IR distance sensors.
2. Implement an IR rangefinder-based wall following algorithm.
3. Using the I2C serial port, communicate between the myRIO (master) and the MSP430 chip (slave) on the myRIO robot board.
4. Setup the MSP430's DTC to sample multiple ADC10 channels and communicate those values back to the myRIO.

Exercise 1:

Two Sharp IR distance sensors are mounted to the top of your robot. They output an analog voltage that is inversely related to the distance from the reflective surface. The operating range is approximately 10cm – 80cm. The relationship between voltage and distance is nonlinear, the details of which we are not concerned with. They are connected to myRIO ADC channels A/AI0 and A/AI1.

Starting with the labview code you developed for Lab 8, add two myRIO ADC blocks inside your simulation loop to read the IR sensors. These ADC blocks output a floating point value between the ADC low reading of 0 volts and the ADC high value of 5 volts. Display the output of both of these ADC channels with either a SimTime Waveform or just a numeric indicator. Run this program and notice that when you place your hand in front of the IR sensors you receive a large voltage reading when your hand is close to the sensor (but not closer than 10cm) and when you move your hand away from the sensor the voltage reading gets smaller. This is opposite from what would make more sense for a control algorithm where a close distance would be a small voltage reading and a large distance would correspond to a large voltage reading. We will invert this below. Also notice what voltage reading you get when your hand is closer than 10cm to the sensor. The reading is not valid when the obstacle is closer than 10cm and this can cause us issues when we use this sensor for wall following and the robot gets too close to the wall. We will have to keep this in mind.

So as a first step invert the sensor reading from the IR sensors. Move your hand in front of the sensor until you can determine what the approximate maximum voltage the sensor is outputting for its shortest distance measurement. Then to invert the sensor reading simply subtract the sensor reading from this maximum voltage that you found. This way a short measurement will correspond to a small reading and a long measurement will correspond to a large reading. Display this inverted measurement and **show it working to your TA.**

Notice above that it is never mentioned to calibrate the voltage output to a distance in cm. This could be done but it is messy and nonlinear and really not worth the time. Instead we will leave the

measurement in units of volts and just remember that this voltage reading is related to a distance measurement.

Our next step is to develop a control algorithm that will keep the robot moving forward and a fixed distance away from a right wall. Later we will add to our algorithm the ability to turn left when a front wall is encountered. To determine an appropriate right-wall setpoint simply place the robot a fixed distance from a right wall and record the inverted ADC conversion result. Use this as the set-point or reference for the right wall control. Now, implement a proportional control law to regulate the right-wall distance. The control effort produced by this law will be the "turn" value of the PI controller you developed in lab 8. Set the "vref" value to 1 ft/s. Hand-tune this proportional gain while the application is running and the robot is on the floor following the right wall.

Introduce another proportional control law that is activated when the front-wall distance is smaller than some threshold. The goal of this controller is to turn the robot left when a front wall is near so that right-wall following may resume on the front wall. Use a front-wall setpoint that is large so that the controller will turn the robot until the front wall is on its right. Keep in mind that some dead zone (hysteresis) is necessary to avoid rapid switching between the two modes. Tune by trial-and-error the Kpfront value (and its sign), the threshold that switches between right wall and front wall control, the threshold that switches back to right wall control and the "vref" velocity of the robot car when the algorithm is in the front wall mode. The full wall-following controller is summarized by the following pseudo-code.

Wall-following Controller:

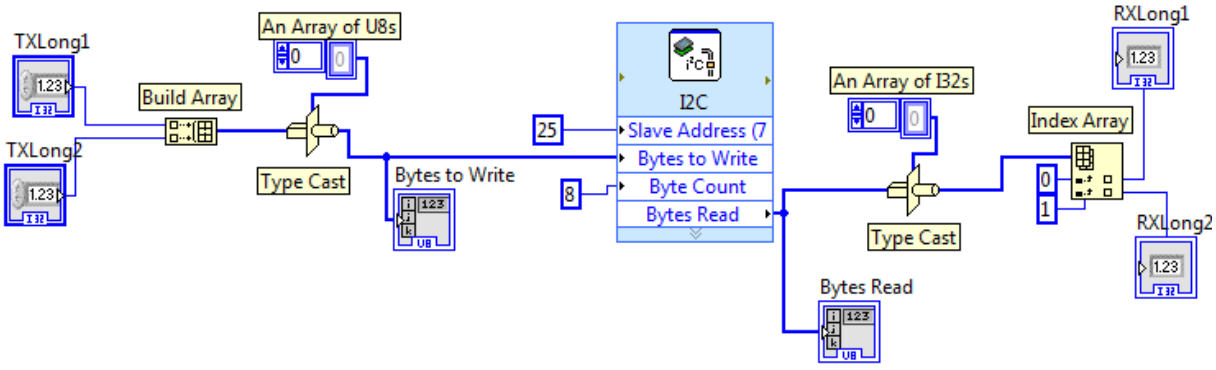
if **right wall follow** *then* (right-wall following controller)
    $t_{ref} = K_{p,right} \times (ref_{right} - dist_{right})$
    $v_{ref} = v_{right}$
    if **$dist_{front}$ < threshold 1** *then* (front wall is near)
        right wall follow = **FALSE** (activate left turn)
    end if
else (left turn controller)
    $t_{ref} = K_{p,front} \times (ref_{front} - dist_{front})$
    $v_{ref} = v_{front}$
    if **$dist_{front}$ > threshold 2** *then* (front wall is far away)
        right wall follow = **TRUE** (resume right-wall following)
    end if
end if

The robot should be able to drive around inside the arena with the wall on its right. Keep in mind that when hand-tuning controller gains that they may be positive or negative. **Demonstrate your wall-following controller to the TA.**

Exercise 2:

For this last exercise, we are going to develop an I2C protocol to communicate 4 16bit integers from the myRIO down to the MSP430 and then the MSP430 will send the myRIO 4 16bit integers. Most of the MSP430 code will be given to you. You will just need to enable and use the data transfer controller (DTC) of the ADC10 peripheral to allow the MSP430 to sample A3, A2, A1 and A0 in sequence. The code given to you will just sample one ADC10 channel (A0) and send back its value in all four of the 16bit integers. This starter code is located at N:\me461\Fall14\lab9\user_lab9_msp430_starter.c. As a first step have your TA show you how to connect the debugger in order that you can program the MSP430F2272 chip on the myRIO robot break out board. Create a new CCS project and cut and paste the entire starter C file over the default C file created with the project. Debug and Run this program when you have your Labview code developed below. After you are finished developing the LabView code we will come back to the MSP430 and modify this C code to have the MSP430 use it's DTC.

In LabView use the myRIO I2C block to setup communication with the MSP430. Configure the I2C block to use Channel: B/I2C, Mode: Write/Read, Speed: Standard Mode (100 kbps). In the block diagram set the Slave Address to 0x25. (To display a constant in Hexidecimal create a constant and then right-click on the constant and select Display Format.) We want to send 4 16bit integers to the MSP430 and read back 4 16bit integers. So set the Byte Count to 8. Since you set the I2C to Write/Read mode this is telling the I2C to first write 8 bytes and then read 8 bytes. Use the LabView code below as a guide, but you will have to modify it. The Labview code given here transmits two 32bit integers and receives two 32bit integers. Your code needs to send 4 16bit integers and receive 4 16bit integers. Put this code in your Lab 8 code inside the simulation loop so that it runs each 5ms. Also note that for this lab we are not going to do anything with the data coming back from the MSP430 except for viewing it in an indicator making sure it works properly. This code will be very useful for groups that choose to use the myRIO in their final project.



The "Type Cast" block and arrays are a bit confusing at first in LabView. Our goal for our code is to take 4 I16 Control variables and convert them to their 2 byte elements and send an 8 byte array of U8s to the I2C block. Then the I2C block will send those 8 U8s to the MSP430 and then read 8 more U8s from the MSP430. Then we need to take those 8 U8s in Labview and put them into I16 variables. The Type Cast

block takes the array to convert in its left port. The top port of the Type Cast block is the desired variable to convert to and the right port is the converted value. To create a constant array in Labview go to Array->Array Constant. Initially this array does not have a type. You need to place the desired type in the large and empty square of the array. To do this get a Numeric Constant from the Numeric panel and place it somewhere in your block diagram. Then right click on the constant and select Properties. In the Data Type tab click on the box that specifies the type. Then a window will pop up asking you to select the type for this constant. Once the constant is set to the correct type you can drag it into the empty block of the array. Now the array has the type you chose.

Complete this Labview code and test it out using the default code given for the MSP430F2272. As stated above, the default code sends the raw ADC value converted for channel A0 back in all four I16 values. So you should see the same value returned in all four indicators. One addition to your Labview code: convert the raw ADC values to a voltage value using VCC equals 3.3volts.

Once your Labview code is working properly, the final task is to modify the F2272 code to use the Data Transfer Control (DTC) of the ADC10 to enable a sequence of ADC channels to be sampled. Setup the DTC to sample channels A3, A2, A1 and A0 in that order into a four element integer array that you create. Then inside the ADC interrupt send each of these converted values to the myRIO over I2C. Have your TA show you how to setup the benches function generator to output a constant voltage and use that voltage to test that the ADC channels are reading the correct voltage. Note the pins of A0 through A3: A0 – P2.0, A1 – P2.1, A2 – P2.2, A3 – P2.3.