# GE423 Laboratory Assignment 6
# Robot Sensors and Wall-Following
## Recommended Due Date: By your lab time the week of March 16th

**Possible Points: If checked off before 4:00PM Mar. 20th … 14 points**
**If checked off after 4:00PM Mar. 20th and before your lab time the week of Mar. 30th … 11 points**
**If work not finished before your lab time the week of Mar. 30th … 0 points**

**Goals for this Lab Assignment:**

1. Learn about the sensors available on the robot for environment sensing.
2. Learn about wall-following to navigate an unknown area.

**DSP/BIOS Objects Used:**

HWI, PRD

**Library Functions Used:**

fire_UltrasonicSensor, read_UltraSonicLight, read_UltraSonicRange, StartIRs, ReadSharpIR

**Lecture Topics:** More about TSKs, IR and Ultrasonic distance sensors, Compass and Rate Gyro Sensors.

**Prelab:** If you have not done so already, update your VB GUI to be able to display both the X and Y position of where the robot is located in the course, and add a display of the sensor readings for this lab. One half of the screen should display the robot's position in the course, the other half should display sensor readings in textboxes or labels. Additionally, setting up your VB application to download gains entered into text boxes will be very useful when tuning the wall-following gains/parameters.

**Laboratory Exercise**

**Big Picture:** This lab has a number of tasks to get you to the final goal so a general overview is in order. The goal is to have your robot follow a wall that is on its right and when it comes to a corner turn to the left and continue right wall-following. At the same time, continuously upload coordinate data to your VB program displaying the robot's location relative to a start position.

I highly recommend you READ THE LAB COMPLETELY BEFORE YOU START CODING so you get the full picture of the assignment.

**Exercise 1: Reading sensors**

Three of the four sensors that we will work with in this lab; the IR distance sensor, the ultrasonic distance sensor and the compass, have very slow response times. Unlike the optical encoders and ADC's from the previous labs, we will not be able to get a reading from these sensors every 1 ms. The sample period of these sensors are on the order of 100ms and can vary plus or minus a few milliseconds. The code interfacing these sensors will have to be written differently then a simple PRD or HWI used to sample our fast sensors.

On the other hand the fourth sensor, the rate gyro, is a fast sensor. It outputs a 1.0 to 4.0 Volt signal (correlating to -300º/s to 300º/s) that is sampled by ADC channel 3. We will use the same code used in lab 4 to sample ADC channel 3's reading.

The IR distance sensor, part number GP2D02, will be the main sensor used in the wall-following algorithm. Specifications for this part can be found in its datasheet at the web link www.ece.uiuc.edu/coecsl/ge423/SharpGP2D02_2003_02_07.pdf. The output from this sensor is an eight bit value with 255 approximately the distance of 2 inches and 0 greater than 30 inches. Figure 1 shows a typical plot of the IR sensor's reading to distance curve. As you can see from the plot, the sensor reading is non-linear and also drops out when the reading gets around the 70 to 80 mark. The important thing to know about these sensors is that each sensor is different.

The curve in Figure 1 does not match all the IR sensors in the lab. Many of the sensors have a drop off point around 50 and other have a drop off point around 100. We found this out to our disappointment during a previous semester teaching this course. We knew they varied some but not these large amounts. You can see in Figure 1 that the sampled data points were fit to a curve that we thought would give us "good enough" results from all the IR sensors. That was not the case so instead this semester we will ignore the conversion between IR reading and inches and just use the "raw" IR reading in our control. Therefore you must experiment with each IR sensor on your robot to figure out the approximate IR reading at the distance that you want to control the robot. We will discuss this issue more during lab time.

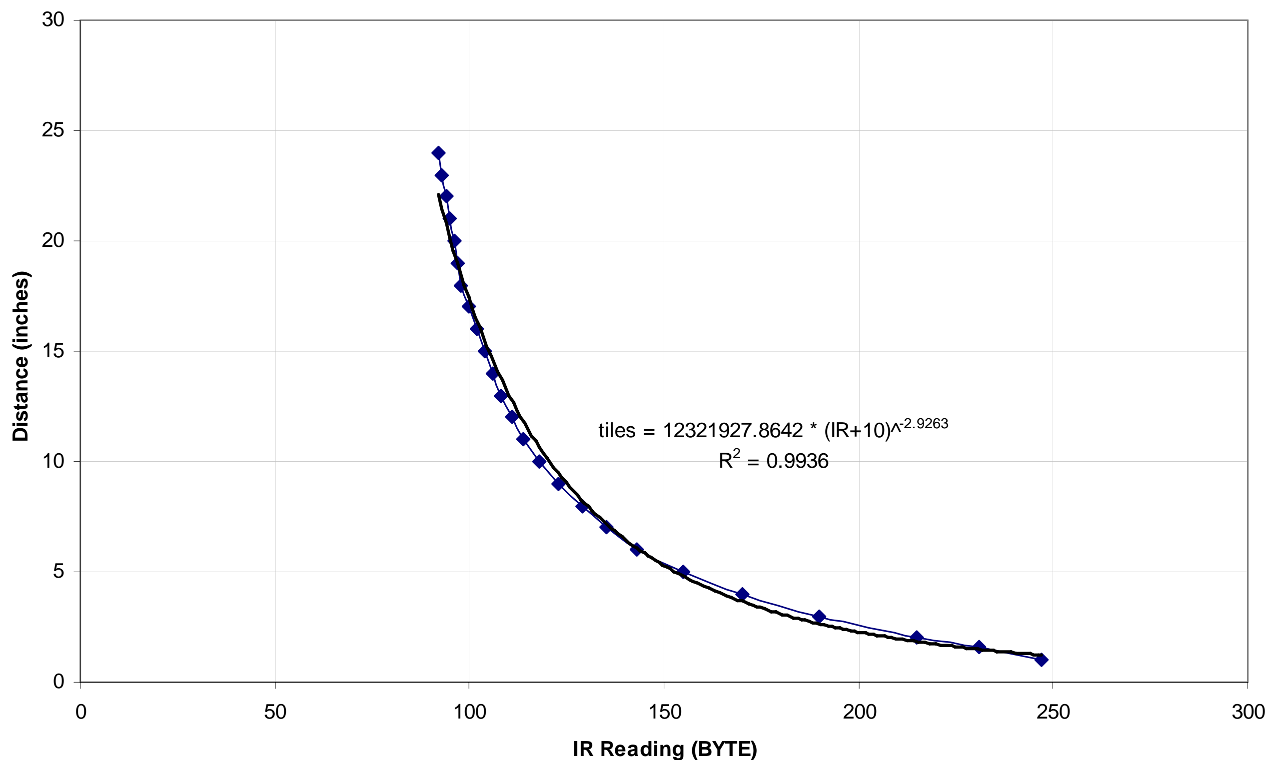


**Figure 1: Plot of a typical IR Distance Sensor's Reading vs. Distance. This curve may NOT represent your IR sensor necessarily. Individual sensor experimentation is recommended with the devices.**

The ultrasonic distance sensors are slightly more user friendly then the IR distance sensors but they also have their issues. See the specifications on this sensor at www.ece.uiuc.edu/coecsl/ge423/DevantechSRF08UltraSonicRanger.pdf. The good news about Ultrasonic sensors is that generally they each behave in the same fashion. As the IR sensor, they return an 8 bit value (0-255) but the units have already been converted to centimeters. The other neat thing about this sensor is that it gives you multiple distance readings per measurement acquisition. You command the sensor to take a distance reading and it first sends out a known sound frequency. After firing it sits and waits for the echo of the sound. The time of that echo is correlated to distance. It not only times the first echo but also another 4 echoes. This allows the sensor to sense not only the close object but also objects further away. The functions we provide return only the first two echoes because that should suffice for most projects. For the final project you may change the code to read the remaining three measurements. The biggest problem with these ultrasonic sensors is that they can only be fired one at a time because the sound from one sensor can affect the reading of another sensor. This means we will have to alternate firing the two ultrasonic sensors on the robot.

The digital compass on the robot is the most problematic sensor on the robot. Not because of noise or nonlinearity but instead because of all the metal in the room and underneath the floor in the mechatronics lab. Metal plays havoc with this sensor. Care must be taken when using the compass reading. Depending where you are in the course, a different set of calibration numbers should be used. Read more about this sensor at coecsl.ece.uiuc.edu/ge423/HM55BModDocs.pdf.

**Reading the robot's 3 IR distance sensors:** The IR sensor can take anywhere from 55-65ms to calculate its distance measurement. This is an incredibly slow sample rate when compared to the rates that the c6713 DSP can handle. We definitely do not want to waste time on the DSP waiting for this sensor to produce a measurement. The perfect scenario would have the IR sensor interrupt the DSP when the distance measurement is complete. Unfortunately this sensor does not have this capability so we will have to read the sensor at a time interval guaranteed to have a completed measurement. We have found that a 70ms interval produces good results. The sensor's serial interface used to communicate the measurement reading to the DSP is also very slow for the DSP and if not programmed correctly would become a huge bottleneck for your program's performance. We will discuss the issues with the serial interface further in lecture, but our solution to these issues is to perform all interface code to the IR sensors in the TSK level. This, of course, can cause varying latencies in our measurements, but since the measurements are so slow we will not notice these relatively small latencies.

Create a task function in the file **user_IR_UltraFuncs.c** to read the three IR sensors. Use the shell below as a guide.

```
void getIRs(void) {
    //  put all lines of code in an infinite loop so that the task does not exit.

    if (new_irdata == 0) {   // new_irdata is a flag that communicates with your control
loop
                           //function indicating that new IR data is ready to be copied.
                           //This check is seeing if the control loop is ready for new data.
        // Start an IR measurement

        // Put this task to sleep for 70ms, allowing the measurement to complete

        // Read the IR measurements

        new_irdata = 1;        // tell control loop function that there is new data

        //Put this task to sleep for 5ms, allowing a small delay before the next
        //measurement.

    } else {
        // This condition should not happen unless there is an error between your
        // communication with this task and your control loop task using the IR data.
        // This just guards against that possibility

        // In this else just put the task to sleep for 70ms
    }
}  // end of getIRs
```

Add this task to your DSP/BIOS configuration. To test if your new code is working, create a program to print to the LCD the IR readings each time an IR measure is complete. To do this we are going to think a bit ahead and use the ADC interrupt function as our printing function. We will need to use the ADC to measure the rate gyro later in this lab. Just as in lab 4, create a PRD to start the ADC every 1 ms. Then update HWI7 (don't forget HWI_Dispatcher) to call your

function to print the IR values to the LCD. This function should check the `new_irdata` flag to see if new data is ready. If there is new data, the data should be copied to a new set of global variables to be used in the control loop function. Next set the `new_irdata` variable indicating data has been received and finally print the new values to the LCD screen.

Again looking ahead to the control loop code, we will want to modify the sensor reading slightly so it can be used easily in the control calculations. If you remember from the above discussion the IR sensor returns a value between 255 and 0. 255 is a short distance and 0 a long distance. In your code reverse this and change the value to a floating point number by adding the following line of code with your variables replacing the ones here:

```
front_wall_distance = (float) (255 - current_IR2);  // reverse the direction of the reading
// This below line should only be used in your code if you find that your sensor gives noisy
// readings at far distances.  Most of the time this is NOT needed.
//if (front_wall_distance > 160) front_wall_distance = 160;
// 160 is just an example, saturate reading where it becomes noisy.
```

The second line of code is commented out because it is normally not needed. It limits the reading to a maximum value. This maximum value is found by experimenting with the sensor to determine where the reading gets too noisy. Do this for each of the IR sensors only if you find the far distance readings bouncing around a lot.

Build and run your code. Test each IR by moving your hand in front of the sensor. Demo this code for your TA.

**Reading the ultrasonic distance sensors.** The ultrasonic sensors communicate to the DSP through the $I^2C$ serial interface. The $I^2C$ serial interface will be discussed more in class but in short it is a serial interface that allows multiple devices to share the same transmit/receive line. Each device on the $I^2C$ bus has a unique address, allowing the DSP to specify which device to command. The two ultrasonic sensors have the addresses 0x70 and 0x71. The sampling period of the ultrasonic sensor is around 85 to 90 ms. To be safe we will write our code to sample the ultrasonic every 100 ms. An added feature of the ultrasonic sensor is that it also has a photo-resistor sensing light intensity. The light intensity value will not be needed for the wall-following algorithm but we will read it just to learn how it works.

Create a task function in the file **user_IR_UltraFuncs.c** to read the ultrasonic sensors. Use the shell below as a guide.

```
void getI2CSensors(void) {
       // Initially put the Task to sleep for 1 second to allow I2C bus some initialization
time.

       //  put remaining lines of code in an infinite loop so that the task does not exit.

       if (new_i2cdata == 0) {// new_i2cdata is a flag that communicates with your control loop
                             //function indicating that new I2C data is ready to be copied.
                             //This check is seeing if the control loop is ready for new data.
              // in a while loop continuously fire the ultrasonic sensor at address 0x70 until
              //no error is returned.  Most of the time errors do not occur but this guards
              // against the possibility of an I2C error.

              // Put this task to sleep for 100ms, allowing the measurement to complete

              // read the ultrasonic ranges
              // read the ultrasonic sensor's light intensity

              // in a while loop continuously fire the ultrasonic sensor at address 0x71 until
              //no error is returned.  Most of the time errors do not occur but this guards
              // against the possibility of an I2C error.
```

```
                // Put this task to sleep for 100ms, allowing the measurement to complete

                // read the ultrasonic ranges
                // read the ultrasonic sensor's light intensity

                new_i2cdata = 1;  // tell control loop function that there is new data
        } else {
                // This condition should not happen unless there is an error between your
                // communication with this task and your control loop task using the I2C data.
                // This just guards against that possibility

                // In this else just simply put the task to sleep for 100ms
        }
} // end of getI2CSensor
```

Now in the same fashion as you tested the IR sensors, add this task to your DSP/BIOS configuration and add to your code global variables and instructions to print the ultrasonic sensor values to the LCD screen. Demo this code for your TA.

**Reading the compass angle.** The reading of the compass has already been taken care of for you in the default project-creator generated project. This is due to the fact that both the color LCD and the compass communicate to the DSP over the same SPI serial port. Two separate chip select signals are used to communicate to both devices on the same serial bus. The compass can only be read when the color LCD is not being written to. For that reason we had to add the code for reading the compass in the color LCD code. See "`color_lcd.c`".

To use the value read from the compass in your source code, you just need to define an extern variable. Add the following towards the top of your code (after the #includes): `extern float CompassAngle;` Print this value to the LCD screen to see that is working. Note that the compass's value ranges from -180 to 180.

**Exercise 2: Right Wall-following**

Our next task is to use the IR sensor's distance reading to control the robot in such a fashion that it follows a wall on its right. By the end of lab 5 we had implemented a coupled closed-loop PI control for the speed of the robot's wheels. That control algorithm had two input variables that you could change to make the robot speed up and turn. We defined those variables `vref` and `turn`. Our wall-following algorithm is going to control these two variables to make the robot perform as we desire.

First of all we should describe where the IR sensors will be located on the robot. The right IR sensor will be mounted on the right side of the robot pointing at approximately a 45° towards the right and the front. The front IR sensor will be mounted on the front of the robot pointing straight in the front direction.

Using these two sensors we will have two situations that the robot will encounter. The highest priority (i.e. the situation that should be checked for first) case is when the front sensor detects an object within a certain distance. In this case we need to tell the robot to turn to the left until the front sensor no longer detects the object. To do this we are going to define an error state `front_wall_error`. `front_wall_error` will be defined as the error between the maximum distance the IR can detect and its current distance reading. Then a simple proportional control law can be defined that is *turn = Kp_front*front_wall_error*. Also, set the reference speed of the robot slow to allow time for the turn. These two adjustments will cause the robot to continue turning to the left until `front_wall_error` is again outside of the turning threshold.

The second situation is when no obstacles are in front of the robot. In this case we want to tell the robot to follow the object (or wall) seen by the right IR sensor with a desired gap between the robot and the right object. For this situation we define an error term `right_wall_error` that is the error between the desired gap distance and the actual measured distance. Use here the proportional control law *turn = Kp_right*right_wall_error* and set the robot's speed to the desired speed. This will servo the robot close to the right wall or obstacle. To set the desired speed of the robot, use the handheld optical encoder as you did in lab 5 or—even better and more time efficient—download values from your VB program. This way you can experiment with different speeds, gains, etc. while wall-following. Try to see how *smooth* of a control you can create.

Now we are ready to code the wall-following algorithm. Below is a code outline you should use in the ADC's interrupt to generate your control loop. You have already started this code in the above sections.

```
// Add declarations for tunable global variables to include:
//      ref_right_wall - desired distance from right wall, approx 0.8 tiles.  You will have
//              to figure out what that is in IR sensor units
//      front_error_threshold - maximum distance from a front wall before you stop
//              wall-following and switch to a front wall-avoidance mode, start with 0.5 tiles
//      Kp_right_wall - proportional gain for controlling distance of robot to wall,
//              start with 0.015
//      Kp_front_wall - proportional gain for turning robot when front wall error is high,
//              start with 0.018
//      front_turn_velocity - velocity when the robot starts to turn to avoid
//              a front wall, use 0.4 to start
//      turn_command_saturation - maximum turn command to prevent robot from spinning quickly if
//              error jumps too high, start with 1.0
// These are all 'knobs' to tune in lab!

// declare other globals that you will need

extern int new_i2cdata;
extern int new_irdata;
// declare as extern all other global variables you will be using from user_IR_UltraFuncs.c

ADC_INT7_Func(void) {
        // Add code here to increment time

        // Read the converted ADC values.  Exercise 3 will ask you to add code to integrate the
        // rate gyro to determine the robot's current angle
        // Add code here to read encoders 1,2,3 and 4.  Use encoders 1 and 2 to find your X,Y
        // position along with the gyro's angle(wait until exercise 3)and use enc3 to input vref

        if (new_i2cdata == 1) {
                // Add code here to copy measured sensor values from the global variables you
                //filled in the task to another set of global variables you can use in this
                //function.  The reason you cannot use the global variables the task fills is
                //because they are changed before new_i2cdata is set to 1.
                new_i2cdata = 0;
        }
        if (new_irdata == 1) {
                // same as above but copy IR data.
                new_irdata = 0;
        }
```

```
        // Add code here to calculate distance to front wall and error on front wall sensor.

        // Add code here to calculate distance to right wall and error between
        // a reference distance, ref_right_wall, and your measurement

        if (fabsf(front_wall_error) > front_error_threshold){
                // Change turn command according to proportional feedback control on front error
                // will use Kp_front_wall here…

                vref = front_turn_velocity;
        }
        else {
                // Change turn command according to proportional feedback control on right error
                // will use Kp_right_wall here

                // Set vref to something higher... start with 1.5.
                // We have also tried setting vref according to right error and/or front error
                // with good results
        }

        // Add code here to saturate the turn command so that it is not larger
        // than turn_command_saturation

        PIVelControl(vref,turn);

}
```

Above we called a function `PIVelControl`. This will be a function that you write and needs to be located in the C file `user_PIFuncs.c`. You must move your code from Lab 5 that implemented coupled closed-loop PI velocity control to a function called `PIVelControl`. This function should have the following structure – it is the same structure given to you in Lab 5 so only very minor modification should be needed to remove the switch statement, etc.

```
void PIVelControl(float vref, float turn) {

        // Read encoders 1 and 2

        // Calculate position in tile units from encoder readings

        // Calculate velocity (tiles/sec) from position

        // Fix the roll-over problem with the encoders...

        // Calculate PI Coupled Control errors e1, e2, e1sum, e2sum and control effort u1 and u2

        // Check for integral windup by seeing if control effort larger than 10

        // Add friction compensation to control signal

        // Send PWM command to motors

        // Save old positions and velocities
}
```

**Exercise 3: Rate Gyro**

As our final task for this lab we will add the rate gyro to our mix of sensors. The rate gyro, part number ADXRS300, produces a voltage proportional to its angular velocity. This sensor has a range of +/- 300º/s with an analog output of 1 Volt for -300º/s and 4 Volts for 300º/s. The rate gyro's signal is brought into the DSP through ADC channel 3. Thus this ADC voltage reading can be scaled to the units of rad/s by first subtracting the sensor's offset voltage (approximately 2.5 Volts) and then multiplying by the gain (600*pi/180 rad/s) / (3 Volts).

With our robot application we are not too interested in how fast or slow the robot is spinning. Instead, what we are more interested in is our angle or heading. With that information and our wheel's optical encoder position readings we can approximate the XY coordinates of the robot. To find our heading with the rate gyro we will need to integrate the angular velocity value each sample period. Below you are supplied with a shell of source code to get you started integrating this signal to find the robot's angle.

Before you implement this code, there is a large problem with this method of finding the heading of your robot. The integrator naturally drifts due to an inexact signal produced by the rate gyro. Even a single bit (5mV) of error from the ADC can cause, in a short amount of time, a large error in the angle measurement. The analog signal from the rate gyro does have some noise in it causing the angle measurement to drift. You will see this drift when you implement your code. Unfortunately there is no way to totally fix this drift problem. Even the super expensive orientation sensors have noticeable drift. Some way of resetting the angle calculation every so often needs to be implemented in the system when this type of angle sensing is used. One example would be to use other sensors to find a home position for the robot that would have a known angle of orientation. This home position would have to be found every so often to reset the angle measurement. Another method that we have tried is comparing the compass reading to the rate gyro angle measurement. This works somewhat but, as we discussed earlier, the compass also has its problems. Implement the code below and then see if you can think of some ways to reduce the drift.

```
// global variables
float gyro_zero = 2.5  // 2.5 is just a starting position.  You are going to change it below to
                       // a more accurate value
// add here all other floating point variables needed.

// inside your adc interrupt function add your code

// you need an integer value keeping track of time.

// Read all four ADC values.  ADC3 is the rate gyro's value

// for the first 3 seconds of the program's run find the zero offset voltage for the rate gyro.
// During these first three seconds the robot should not be allowed to move.

// Then after 3 seconds have expired start calculating the angle measurement
// 1. Find the angular velocity value by first subtracting the zero offset voltage from ADC3's
//    reading.  Then multiply this value by the sensor gain given above.
// 2. Calculate the integral of this signal by using the trapezoidal method of integration.
//    This value is your angle measurement in units of Radians.
// 3. Display this angle value to the LCD every 100ms or so.  (You can do this in a separate
//    PRD if you wish.
// 4. Think (and implement) about some ways to stop the drift of the angle measurement when the
//    robot is sitting still.  Do this after you have implemented your code for 1 and 2.
```

**Lab Check Off:** Show your TA the final VB program as your robot follows the wall. Use the rate gyro measurement and/or the compass measurement to determine the orientation of your robot. Upload this data to the VB program and update the robot's position on a XY grid. Your robot should run smoothly as it follows the wall inside the course. You will need to adjust your wall-following gains if your robot is jerking a lot.

**Post-Lab:**

In this lab, you used only the IR sensor to detect wall position. If you wish, you may integrate the readings of multiple sensors. One method might be to use the ultrasonic sensor to 'look ahead' for wall errors. You may wish to try experimenting with different techniques of measuring your wall error or different sensor mounting positions. You have three IR sensors to measure whatever you want!