**ME EN 3230 Fall 2021**

**Memo: Basic Tutorial for Line/Wall Following**

For PM7, you are required to demonstrate line and wall following control of your robot using your reflectance array and proximity sensors. It is actually quite simple to achieve smooth line/wall following if you do it properly.

Let's say you define ***d*** to be the position of your reflectance sensor over the line or the distance of your proximity sensor from the wall (in whatever units you like). Once you've properly calibrated your reflectance sensor or proximity sensor, then you can compute ***d*** from your sensor readings. You can then define the ***error*** to be the difference between the actual position/distance ***d*** and the desired position/distance $d_0$.

$$error = d_0 - d$$

Then we just need to define a feedback law that determines the motor commands to your two wheels based on the error. The simplest idea would just be to turn left if the error is positive and turn right if the error is negative. However, this leads to jerky line/wall following since the robot is always turning one way or the other. A better way is to define a ***proportional feedback law*** such that the degree to which you turn depends on the magnitude of the error. To do this, you define a ***base_speed*** to set how fast you want the robot to move forward (if you're using the motor driver libraries, this would be a number between 0 and 400, which is the range of motor commands), and then add/subtract a correction term to each motor based on the error:

$$Motor1command = base\_speed + Kp*error$$

$$Motor2command = base\_speed - Kp*error$$

where ***Kp*** is a proportional gain factor that you need to experimentally tune. Depending on how you wire your two motors and calculate your error, you may need to flip signs on the terms in these equations. Ideally you would wire up your motors so that a positive ***base_speed*** drives both motors forward. Once you implement these equations in your code, test it out by holding the robot with the wheels slightly off the ground and see if the wheels change speed appropriately as you move the robot back and forth with respect to the line/wall. If the robot is centered over the desired position, the two wheels should spin at the same speed. If you move the robot to the left, the left wheel should speed up and the right wheel should slow down. If you move the robot to the right, the opposite should happen. Start with a relatively low value for ***Kp*** (think about how big the ***error*** will be and how much you need to penalize it to make an impact on the ***motor command***s relative to the ***base_speed***). You'll find that as you increase the ***base_speed***, you'll need a larger ***Kp*** in order to closely follow the line/wall. If you want high-speed line following, you may need to add a derivative term to your feedback law, and if you want your robot to stay exactly centered over the line, you may need to add an integral term to your feedback law, however a simple proportional feedback law should be sufficient for PM7.

For experimentally tuning your feedback law, it can be helpful to wirelessly transmit your control parameters (e.g. base_speed, Kp) to your Arduino Mega, and then wirelessly transmit data back (e.g. time, error, motor commands). That way you can quickly and repeatedly adjust your control parameters to tune your performance. To assist you with this, I have prepared a set of three files:

- *ArduinoPIDtest.m* (MATLAB script)
- *UnoRelayData.ino* (Arduino UNO sketch)
- *MegaPIDtemplate.ino* (Arduino MEGA sketch)

The MATLAB script will allow you to specify a set of control parameters (e.g. base_speed, PID gains) and transmit them to an Arduino via serial USB port. It will then receive an array of data back from the Arduino (e.g. error, motor commands) and plot it. You could use this MATLAB script directly with your Arduino MEGA (but this would require your robot to be tethered via USB while you line/wall follow). The more convenient configuration is to have the MATLAB script talk to your Arduino UNO via USB, and then have the UNO relay the data back and forth wirelessly to/from the Mega using your Xbees. The provided Arduino UNO sketch can be used without modification, and will relay any number of control parameters to the Mega and then relay any number of variables back to MATLAB for plotting. The Arduino Mega sketch is just a template that has the serial comms programmed to wirelessly receive a set of control parameters and then transmit 3 seconds worth of data back. You can configure it to match the serial channel of your Xbee and run it without any other modifications to test the serial comms, but then you'll need to add your own code to read your sensors, compute the feedback law, and command the motors. You can then experimentally tune your line/wall following. To recap the procedure:

1. Connect your Arduino UNO to your PC via USB and download the UNO sketch *UnoRelayData.ino*
2. Configure the MATLAB script *ArduinoPIDtest.m* with the correct serial port for your UNO (e.g. 'com3').
3. Configure the sketch *MegaPIDtemplate.ino* to make sure it uses the serial channel (e.g. Serial2 or Serial3) corresponding to your Xbee and download to your MEGA
4. Run the MATLAB script to test your serial comms
5. Modify the Arduino MEGA sketch *MegaPIDtemplate.ino* (add control code) and download to your MEGA
6. Repeatedly run the MATLAB script with different values for the control parameters and observe the performance. **Don't forgot you need to manually reset your MEGA before each trial.** The UNO should reset automatically every time the MATLAB script opens the serial port.

Note: For debugging purposes, you can bypass MATLAB and just communicate the control parameters to the UNO via the serial monitor (type them in a row with spaces in between each parameter and then hit enter). If you also have the MEGA connected via USB, you can use the serial monitor on the MEGA to verify that it received the control parameters. The relayed data should then appear on the UNO serial monitor. In this case, you will need to reset both the UNO and MEGA before each trial.