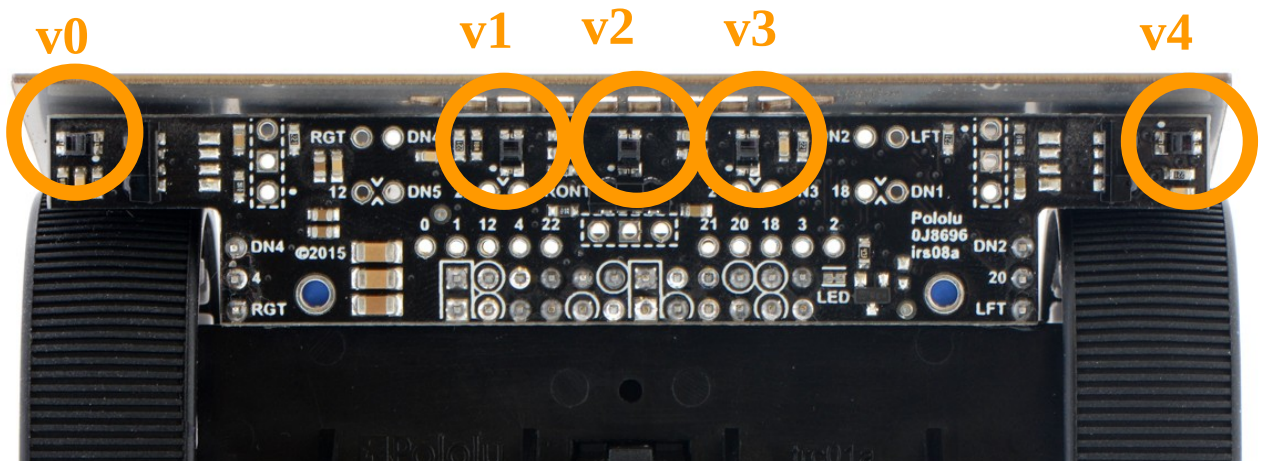
ENGR 207: Programming Robots and Sensors
Lab 5: Line-following and Closed-Loop Control

Assigned: 10-April-2017
Due: 17-April-2017

Introduction

The Zumo robot has five reflectance sensors on its underside. These reflectance sensors each have a single infrared (IR) LED and a photo-transistor that are used to measure the amount of IR light reflected back from a surface directly below it. The sensor returns a value that is proportional to how dark the surface is below the sensor – for example, the sensor will return an intermediate value if the surface is half dark and half light.



In this lab we will use this array of five reflectance sensors to drive the robot along a dark line using feedback control. The Zumo robot has a class called "*Zumo32U4LineSensors*" that handles the calibration and reading of the reflectance sensors.

The values that will be returned by each of the five sensors may vary due to manufacturing differences and environmental conditions (e.g., the lighting in the room). To compensate for this the Zumo runs a calibration routine that proceeds as follows: the user places the Zumo on the line it must follow and the Zumo spins around until the dark line has passed underneath each of the five sensors. As it is spinning it keeps track of the minimum and maximum sensor readings encountered by each sensor. These min/max values are then used to scale future readings so that each sensor outputs a "calibrated" value from 0-1,000. Finally, a simple formula is used to determine (roughly) where the line is relative to the Zumo robot:

$$position = \frac{v_0(0) + v_1(1000) + v_2(2000) + v_3(3000) + v_4(4000)}{v_0 + v_1 + v_2 + v_3 + v_4}$$

where v_i is the the value (from 0-1,000) output by the i-th sensor. For example:

- if the vehicle is exactly on the line, and sensor 2 outputs its full value (1000) while the two nearby sensors (1 and 3) output half their values (500), then the position output by the sensor is:

$$position = \frac{0\,v_0(0) + 500(1000) + 1000(2000) + 500(3000) + 0(4000)}{0 + 500 + 1000 + 500 + 0} = \frac{4000000}{2000} = 2000$$

This position is obtained by invoking the following C++ code in the `void loop ()`

```
int position = lineSensors.readLine(lineSensorValues);
```

Since we want to keep the robot at the center of the line, this value of 2,000 is our setpoint. The error signal we will use for control is then:

$$error = setpoint - position = 2000 - position$$

and a positive error implies that the leftmost reflectance sensors are active and therefore the line is to the left of the robot. We will explore two different control strategies – "bang-bang" control and PID control.

Bang-Bang Control

The "bang-bang" control strategy is to adjust the control abruptly by switching between two (or more) controls when a certain condition is met. A common example is the thermostat in a home adjusting the heat on a cold winter day– it is either "on" or "off" and there are no intermediate possible controls. In this case the condition determining whether the heating is on/off is the temperature set by the home owner. For the purposes of line-following we will implement a controller that has three possible values: going straight, turning left, or turning right. The criteria to switch between these three values will be the sign and magnitude of the error signal. The bang-bang control law we will implement in this lab is:

$$u = (leftSpeed, rightSpeed) = \begin{cases} (50, 400) & \text{if } error > 200 \ (\text{turn left}) \\ (400, 50) & \text{if } error < -200 \ (\text{turn right}) \\ (400, 400) & \text{else} \ (\text{go straight}) \end{cases}$$

PID Control

As we have discussed during our previous lectures, when using PID control the control signal is a function of the current error value, the rate of change of the error, and the integral of the error that has accumulated over time. In this lab we will only use PD control (it turns out the integral term is not very useful for line following). Since we are implementing this on a digital computer we will approximate the derivative of the error by simply computing the difference between the current error and the error at the previous timestep (the previous time the void loop() was called).

The control law is then

$$u = kp * error + kd * (error - lastError)$$

which is used to adjust the left and right motor speed as follows

$$(leftSpeed, rightSpeed) = (400 - u, 400 + u)$$

In the above equation `lastError` is a variable defined at the beginning of the Arduino sketch (outside) of the void loop. It is updated with the new error measurement each time the loop runs so that during the next iteration the previous error is available.

Note that the maximum motor speed is 400. Thus, so that we do not command a motor speed greater than this value we use the "constrain" Arduino function to ensure leftSpeed and rightSpeed are always between 0-400. The constrained values are then passed to the motor.setSpeeds function.

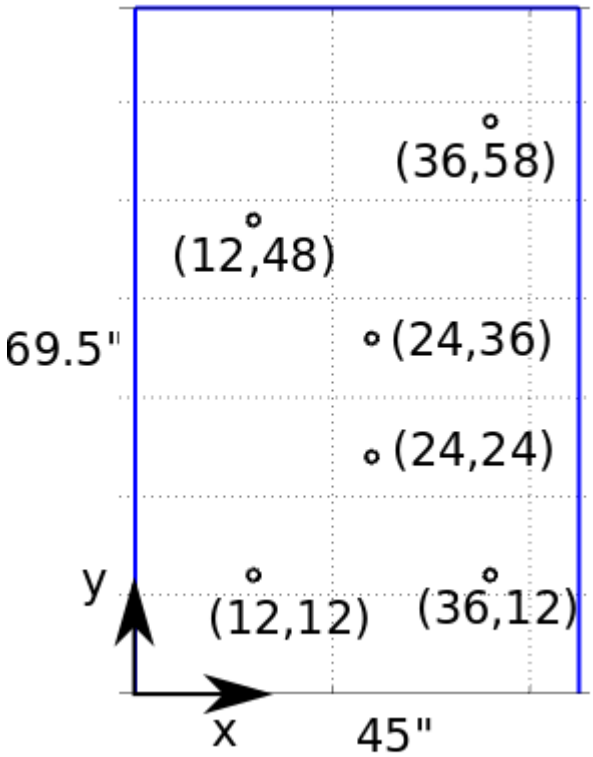
```
leftSpeed = constrain(leftSpeed, 0, 400);
rightSpeed = constrain(rightSpeed, 0, 400);
motors.setSpeeds(leftSpeed, rightSpeed);
```

Lab Instructions:

1) Follow the instructions provided in "L4_Installing_Zumo_Libraries.pdf" to install and test the Zumo 32U4 libraries if necessary.

2) Download the ZumoLab5.ino sketch from Blackboard and move it into your working directory.

3) Using a measuring tape, draw with a dry erase marker the following points (goal regions) on your Zumo whiteboard stage.



4) Use electrical tape and/or whiteboard marker to construct a smooth path for the robot to follow that visits all of the six goal regions and forms a closed loop. Note that sharp turns and corners may be difficult for the robot to follow.

5) Implement the bang-bang control law to follow the path. Everytime you start the Zumo you will need to perform the calibration by placing it on dark line and pressing the "A" button. The Zumo will then spin and calibrate. When that is complete you can place it back on the line and press "A" again to execute the void loop() portion of your code.
   - Increase the error threshold (given as 200 in the control law above) to 1000 and observe how the robot behaves.
   - Decrease the error threshold to 1 and observe how the robot behaves.

Required Submissions:

- (5 pts) Two (2) videos showing the robot completing the course with the bang-bang controller using the large and small threshold as described above. Also submit only the void loop() portion of the corresponding code.
- (4 pts) Write a paragraph in which you:
  - Describe and explain why the robot behaves as it does in each of the two cases.

6) Implement the PD control law to follow the path
   - P-control: Begin by setting kd = 0 and increase the kp gain from 0 until the robot can follow the course in stable manner but with significant oscillations. At this stage do not be concerned with path following performance: our aim is to find the kp gain at which the robot is just barely able to stay on the path.
   - P-D control: Keeping the kp gain you determined in the previous step fixed, tune the kd gain by trial and error until the robot is able to smoothly follow the path.

Required Submissions:

- (5 pts) Two (2) videos showing the robot completing the course with both P-control and PD control. Also submit only the void loop() portion of the corresponding code.
- (6 pts) Write a lengthier paragraph in which you:
  - give the kp and kd gains you determined
  - discuss how the PD controller compared to the bang-bang control. Are there any advantages or disadvantages of one over the other?
  - discuss any difficulties that were encountered and how they were overcome
  - summarize what you learned in this lab

Bonus Points: Construct a path for the robot to follow that (roughly) spells out the acronym “CUA”. Tune your PD controller to follow this path and submit a video of the Zumo completing this task. Any group that completes this will receive 5 bonus points that will automatically be added to their lowest exam or homework grade.