Institute for Aerospace Studies
UNIVERSITY OF TORONTO

LABORATORY III

Path-Following Robot

ROB301 Introduction to Robotics
Fall 2015

1 Introduction

This is the third lab of ROB301 Introduction to Robotics. The focus for this week is a simple path-following robot. We will investigate introductory control theory for mobile robotic applications. For in-depth analyses of control theory, refer to AER372 or ECE557. For in-depth analyses of mobile robotics and path-planning, refer to AER521. In addition to mandatory attendance during the lab session, Lab III also requires the demonstration of functionality to the TAs as well as a short written report of observations due on 26 October 2015.

2 Purpose

This lab introduces the foundational concepts of control theory and its practical use in mobile robotics. The simple, yet extremely practical, PID controller is investigated with its application to the line-following problem.

3 Equipment & Software

The hardware platform used for ROB301 are LEGO Mindstorms EV3 kits. Included on each workstation is a folder labelled "Lab 3" which contains a copy of these lab instructions as well as a PDF of the step-by-step LEGO assembly instructions for the educator mobile platform. This apparatus must be constructed by at least one member of the team in order to complete the tasks of the day. Approximate build time is 30–60 min. Please note that the educator mobile platform instructions are found on pages 1–46 of the PDF. The downward-facing Color sensor is necessary for the lab implementation, once the base is complete, follow the instructions on pages 47–53 of the PDF. The remainder of the LEGO instructions can be ignored for now.

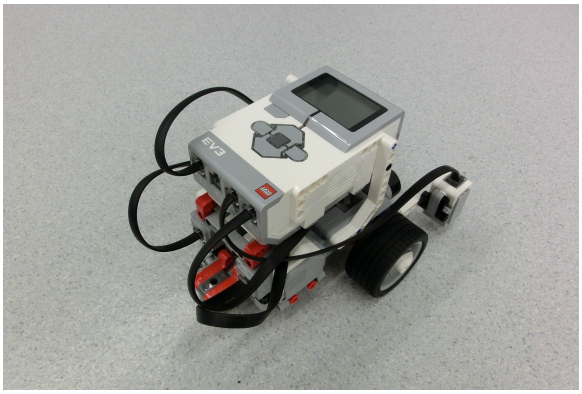

Figure 1: LEGO educator robot

The educator mobile platform consists of two actuators which are two large servo motors for driving. The downward-facing Color sensor will be used in Reflected Light Mode (refer to Lab I instructions).

4 Task Programming

A quick set of tips for actuator driving is done before the calibration procedure of the Color sensor is discussed.

Four different controllers applicable to the line-following problem will be investigated in this section. First, Bang-Bang Control sets the basis of comparison for solution methods. Next, the introduction of proportional, integrative, and derivative gains are introduced to improve the tracking ability of the robot.

A final closed-circuit course is used as a proving-grounds for the performance of the different controller types.

4.1 Motor Driving

The two large servo motors, according to the LEGO instructions, should be connected to Ports B & C. This platform can pivot-turn by either holding one wheel stationary while the other drives or by having the wheels turn in opposite directions.

Potentially Useful Syntax. The following code may be useful for driving:

```
Motor.B.rotate(360,true);
Motor.C.rotate(360);
```

where the addition of ",true" causes the program to execute the current line while simultaneously moving on to execute the next line without delay; or

```
Motor.B.forward();
Motor.C.forward();
```

where the "forward()" causes the motor to turn in the positive direction indefinitely. Use "backward()" to achieve the reverse effect.

Before Moving On. . . Write a program to ensure you can drive the robot in a straight line using both motors at once and that it can turn left/right. No need to demonstrate to the TA.

4.2 Color Sensor Calibration

The Color sensor will be used to detect the line on the ground. Countless methods could be used to observe a line (*e.g.*, two Color sensors straddling a line to try and keep the line between them at all times). As there is only a single Color sensor per kit, we will use the Reflected Light Mode to return a range of 0.00 to 1.00 with the extremes corresponding to *no line is visible* and *only the line is visible*. If the sensor is partly over the line and partly not, it will return a number between the extremes. The exact value when the sensor is halfway over the line and halfway not is 0.5 in an ideal world but this can vary in real-world situations (lighting conditions, proximity of sensor to the surface, cleanliness of the floor, etc.).

Before Moving On. . . Calibrate the Color sensor in Reflected Light Mode. Find and record the values when the sensor is observing only the floor, only the line, and calculate the midpoint in-between. Record your findings and include them in your report. No need to demonstrate to the TA.

4.3 Bang-Bang Control

With no knowledge of control theory, Bang-Bang control is often the first method considered in stabilizing a system. As the name implies, the intent is to observe the environment for the actual state of the system and move toward the desired state.

Pseudo-Code. The following is a general approach to the implementation of Bang-Bang control:

```
desired = set desired position
begin loop
   actual = measure actual position
   error = desired - actual
   if error < 0 then correction = positive
   elseif error > 0 then correction = negative
   else correction = zero
   drive + correction
end loop
```

Before Moving On. . . Write a program to implement the pseudo-code above such that the robot drives forward trying to keep the actual sensor reading around the desired mid-point Color sensor value. Test your robot on a line or curve that you set up on the lab floor. Record your observations and include them in your report. Demonstrate the functionality to the TA.

4.4 P Controller

One of the most basic controllers is the proportional-gain controller. The intent of this controller is to observe the environment for the actual state of the system and move toward the desired state more smoothly than the Bang-Bang type. As the error increases, the proportional gain has greater influence.

Pseudo-Code. The following is a general approach to the implementation of proportional-gain control:

```
desired = set desired position
k_p = set proportional gain
begin loop
   actual = measure actual position
   error = desired - actual
   correction = (k_p × error)
   drive + correction
end loop
```

The value of k_p can be determined in multiple ways. If a mathematical model of the plant is known, then a controller can be designed to stabilize it. Alternatively, when the plant model is not known, too complex, or uncertain, an iterative approach to gain selection can be used. With an iterative approach, the proportional gain value is, at first, set very low and tested. It is then increased until the system becomes oscillatory. Half of this value of k_p is usually used as the final gain as in accordance with the Ziegler-Nichols method. The typical effect of increasing the proportional gain is qualitatively listed in Table 1.

Table 1: Effect of increasing the proportional gain

Parameter	Rise Time	Overshoot	Settling Time	Steady-State Error
k_p	Decrease	Increase	Little Effect	Decrease

Rise time is typically how long it takes the system to go from 0 to 100% of the desired goal; *overshoot* is the maximum amount, usually expressed as a percentage, the response exceeds its desired target; *settling time* is how long it takes the response to reach and stay within a specified band around the desired target; and *steady-state error* is the difference between the response and the desired target in the long term.

Before Moving On. . . Write a program to implement the pseudo-code above such that the robot drives forward trying to keep the actual sensor reading around the desired mid-point Color sensor value. Test your robot on your lines or curves. Record your observations and include them in your report. Demonstrate the functionality to the TA.

4.5 PI Controller

To improve the tracking of the system to the desired state, we add a new parameter to the correction calculation called the integrative gain, k_i. This parameter is special in that it gives the system memory. By accumulating the error over all timesteps, an integrative gain will pull the response back to the target thus making the steady-state error zero.

Pseudo-Code. The following is a general approach to the implementation of proportional-integrative-gain control:

```
desired = set desired position
integral = initialize to zero
k_p = set proportional gain
k_i = set integrative gain
begin loop
   actual = measure actual position
   error = desired - actual
   integral = integral + error
   correction = (k_p × error) + (k_i × integral)
   drive + correction
end loop
```

The value of k_i can also be determined in multiple ways. Usually, the value of k_p is determined first when k_i is temporarily set to zero. Next, the integrative gain is slowly increased until the steady-state error is adequately decreased. Too much k_i and the system will become unstable. (One may also follow the Ziegler-Nichols procedure here.) The typical effect of increasing the integrative gain is qualitatively listed in Table 2.

Table 2: Effect of increasing the integrative gain

Parameter	Rise Time	Overshoot	Settling Time	Steady-State Error
k_i	Decrease	Increase	Increase	Eliminate

Before Moving On. . . Write a program to implement the pseudo-code above such that the robot drives forward trying to keep the actual sensor reading around the desired mid-point Color sensor value. Test your robot on your lines or curves. Record your observations and include them in your report. Demonstrate the functionality to the TA.

4.6 PID Controller

The final improvement to system tracking is the addition of the derivative gain, k_d. This parameter is special in that it gives the system the ability to predict the future. By comparing the difference between the current error and the error of the previous timestep, a derivative gain will expose the rate of change of the error in order to dampen the effect of the other two gains when nearing the target state.

Pseudo-Code. The following is a general approach to the implementation of proportional-integrative-derivative-gain control:

```
desired = set desired position
integral = initialize to zero
derivative = initialize to zero
lasterror = initialize to zero
k_p = set proportional gain
k_i = set integrative gain
k_d = set derivative gain
begin loop
   actual = measure actual position
   error = desired - actual
   integral = integral + error
   derivative = error - lasterror
   correction = (k_p × error) + (k_i × integral) + (k_d × derivative)
   drive + correction
   lasterror = error
end loop
```

Usually, the values of k_p and k_i are determined first as discussed previously. Next, k_d is slowly increased until there is no more improvement in tracking response. Too much k_d and the system will result in excessive response. (Again, one may appeal to the Ziegler-Nichols method.) The typical effect of increasing the derivative gain is qualitatively listed in Table 3.

Table 3: Effect of increasing the derivative gain

Parameter	Rise Time	Overshoot	Settling Time	Steady-State Error
k_d	Little Effect	Decrease	Decrease	No Effect

Before Moving On. . . Write a program to implement the pseudo-code above such that the robot drives forward trying to keep the actual sensor reading around the desired mid-point Color sensor value. Test your robot on your lines or curves. Record your observations and include them in your report. Demonstrate the functionality to the TA.

4.7 Laguna NΨ Raceway

A closed-circuit course—the "Laguna NΨ" raceway—will be provided where you are to test the maximum performance of your controller. You are to attempt to complete the circuit as quickly as possible by adjusting the motor rotation rates and controller gains to handle the turns without losing tracking performance. When you are satisfied with your robot's performance, demonstrate the functionality to the TA who will record your best time. (A schedule will be provided so be ready when your team is called to race.) This is a competition. . . of sorts; just know that there may be prizes! Discuss your controller design along with your observations and include them in your report.

5 Written Report

A short written report is required following the lab. No more than five pages, include an introduction, Color sensor calibration discussion, your comments on each of the four controller types (Bang-Bang, P, PI, and PID) discussing the ease of gain selection and their influence on performance, your best closed-circuit course controller design and the corresponding observations of performance, and end with a conclusion highlighting your findings. Neatness counts!

Due: 10h10, 26 October 2015

6 Conclusion

We have introduced some fundamental introductory control concepts and experimented with simple applications and learned that it's not really about *where* you finish, just *that* you finish.

7 Additional Resources

1. LEGO Mindstorms - `http://mindstorms.lego.com/`
2. Java Programming - `http://www.javaprogrammingforums.com/`
3. leJOS - `http://sourceforge.net/p/lejos/wiki/Home/`
4. leJOS EV3 API - `http://www.lejos.org/ev3/docs/overview-summary.html`