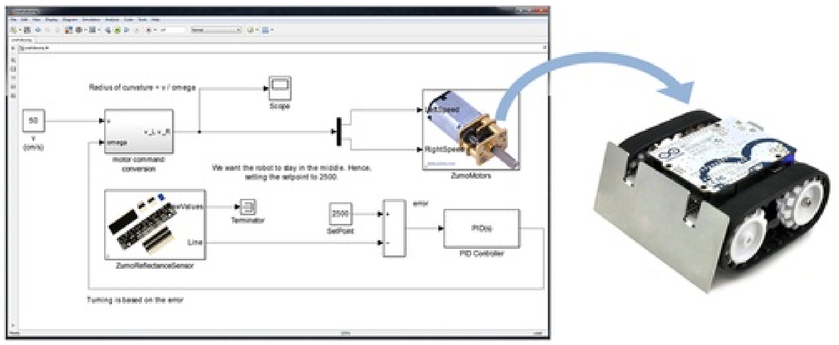
Line Following Zumo Robot Using Simulink

Created by Anuja Apte



We want the robot to stay in the middle. Hence, setting the setpoint to 2500.

Turning is based on the error

Last updated on 2014-05-19 12:30:22 PM EDT

Guide Contents

Overview

This tutorial covers how to use Simulink to program a Zumo Robot powered by an Arduino Uno to follow a line using the reflectance sensors present at the bottom of the Zumo Robot. This guide will explain how the data from the reflectance sensors can be used to control the Zumo Robot motors using a control logic known as PID control.

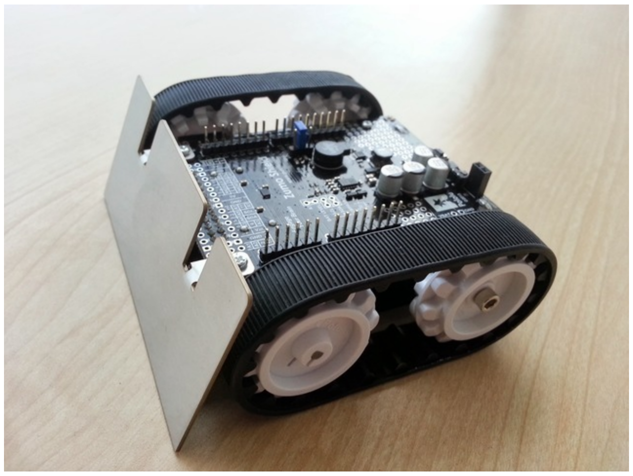
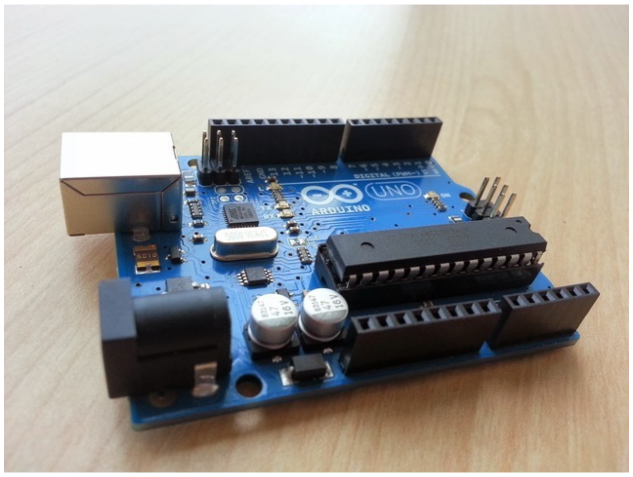
The specific steps to get there are

1. Install Simulink Support package (http://adafru.it/drt) for Arduino
2. Download the Zumo Robot Library (http://adafru.it/dru) in Simulink
3. Create a Simulink Model of a PID Controller for the Zumo Robot
4. Build and download the model to see the robot in action

This guide is the third tutorial in a series on using Arduino with Simulink. For a quick introduction to Simulink and Arduino, refer to the Set up and Blink - Simulink and Arduino Tutorial (http://adafru.it/dsO). This tutorial builds on the concepts discussed in the previous tutorial titled How to program a Zumo Robot with Simulink (http://adafru.it/dsP) and adds to the model described in that section. It is highly recommended that you check the previous tutorials before trying this one, if you are new to Simulink.

Even though this tutorial uses the Zumo robot, you can use the concepts discussed here to program any line following robot using Simulink.

Hardware

	1) Assembled Zumo Robot Kit The assembled Zumo Robot Kit serves as a shield for the Arduino Uno R3 and also includes the chassis for the Zumo Robot. To use the Zumo, you can simply add a programmed Arduino Uno and attach 4 AA batteries and you are all set!
	2) Arduino Uno R3 The Arduino Uno R3 serves as the "brains" of the Zumo Robot and is not included as a part of the assembled kit. In the tutorial, we will program the Arduino Uno using Simulink and then add it to the Zumo Robot.
	3) USB Cable The cable will be used to connect an Arduino Uno to a computer running Simulink
	4) 4 AA Batteries

Software

List of Software components:

1. **MATLAB and Simulink (R2014a)**

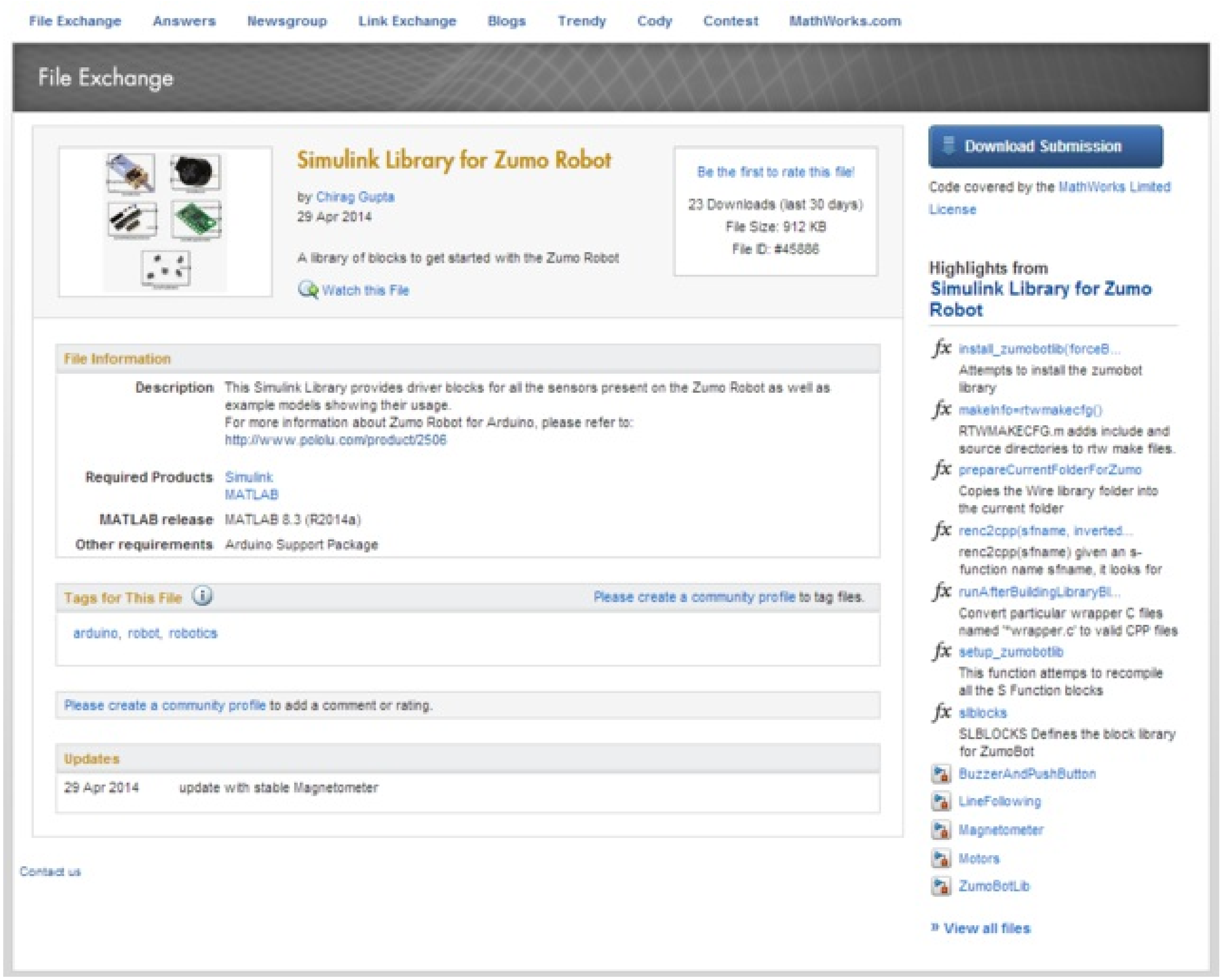
- Buy the Student version for $99 (http://adafru.it/dsQ)
- Buy a Home-use license for $194 (http://adafru.it/drx)
- See commercial and other licensing options (http://adafru.it/dry)

2. **Simulink Support Package for Arduino (http://adafru.it/drz)**

- Follow this tutorial for installation instructions: Set up and blink - Simulink with Arduino (http://adafru.it/drv)

3. **ZumoBot Simulink Library (http://adafru.it/dru)**

- The Zumobot Simulink Library is a collection of blocks used to interface specifically with different components of the Zumo Robot.
- How to program a Zumo Robot with Simulink (http://adafru.it/dsP) covered the installation of the ZumoBot Simulink Library. However, the steps to install the ZumoBot Library are repeated below:
- Download the library from MATLAB Central File Exchange



Simulink Library for Zumo Robot

by Chirag Gupta
29 Apr 2014

A library of blocks to get started with the Zumo Robot

Be the first to rate this file!
23 Downloads (last 30 days)
File Size: 912 KB
File ID: #45886

Code covered by the MathWorks Limited License

File Information

Description	This Simulink Library provides driver blocks for all the sensors present on the Zumo Robot as well as example models showing their usage. For more information about Zumo Robot for Arduino, please refer to: http://www.pololu.com/product/2506
Required Products	Simulink MATLAB
MATLAB release	MATLAB 8.3 (R2014a)
Other requirements	Arduino Support Package

Tags for This File — Please create a community profile to tag files.

arduino, robot, robotics

Please create a community profile to add a comment or rating.

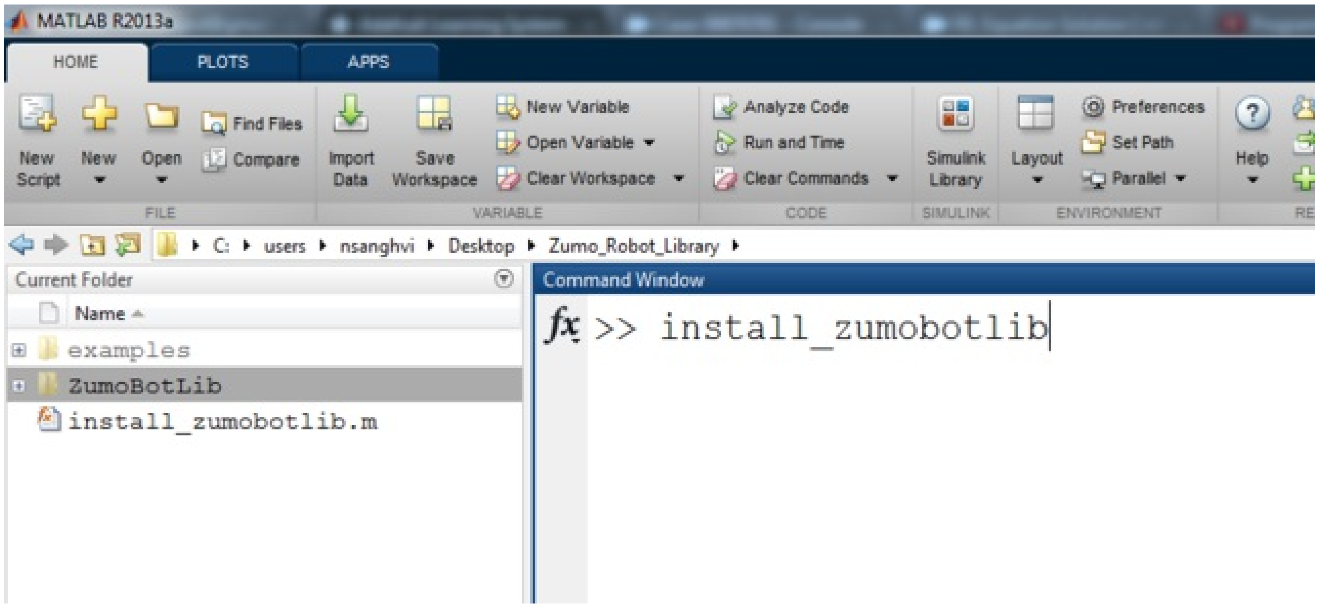
Updates

29 Apr 2014 update with stable Magnetometer

Highlights from Simulink Library for Zumo Robot

- install_zumobotlib(forceB... Attempts to install the zumobot library
- makeInfo=rtwmakecfg() RTWMAKECFG.m adds include and source directories to rtw make files.
- prepareCurrentFolderForZumo Copies the Wire library folder into the current folder
- renc2cpp(sfname, inverted... renc2cpp(sfname) given an s-function name sfname, it looks for
- runAfterBuildingLibraryBl... Convert particular wrapper C files named "wrapper.c" to valid CPP files
- setup_zumobotlib This function attemps to recompile all the S Function blocks
- slblocks SLBLOCKS Defines the block library for ZumoBot
- BuzzerAndPushButton
- LineFollowing
- Magnetometer
- Motors
- ZumoBotLib

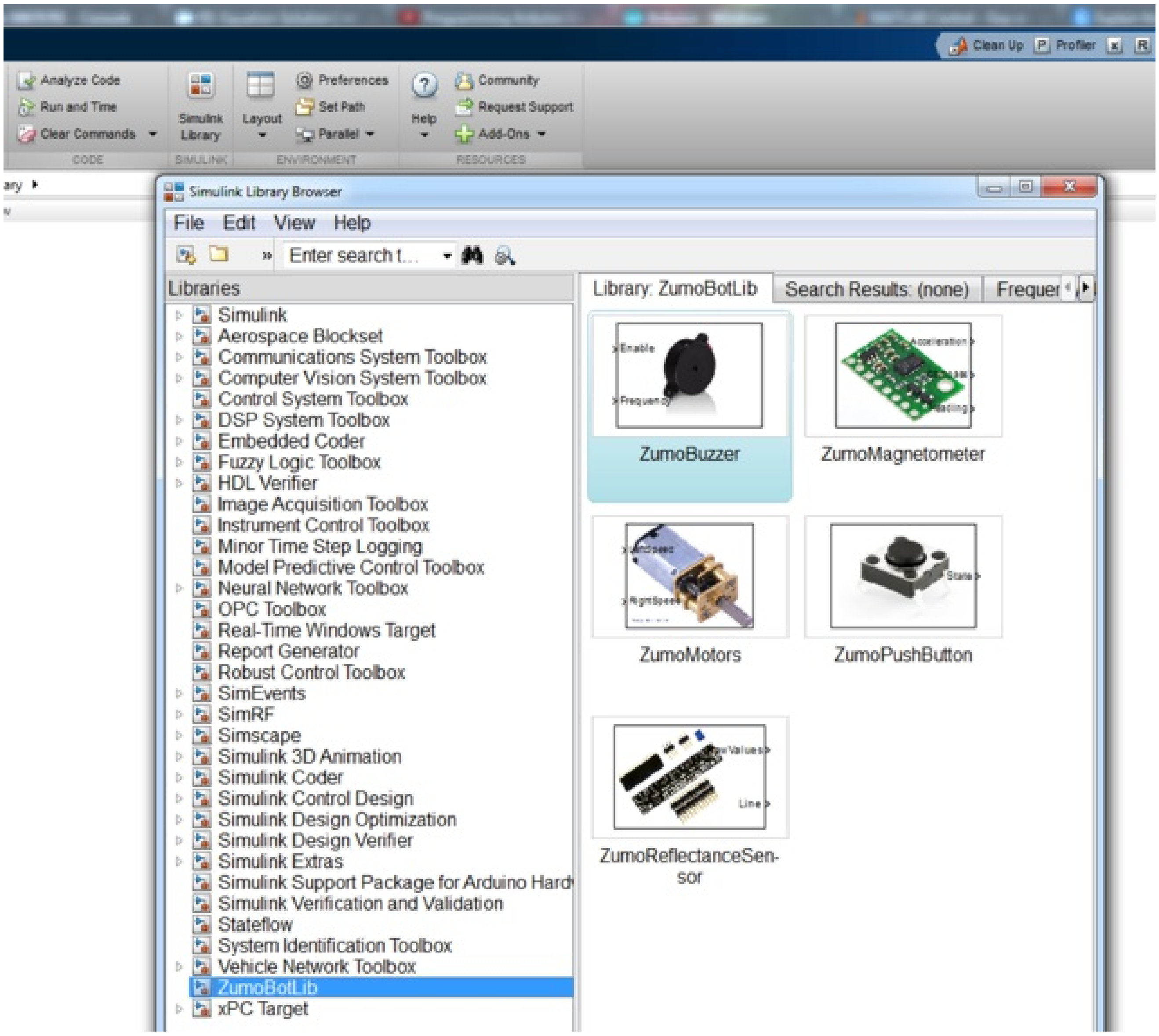
- Unzip the downloaded file to your working directory.
- Open MATLAB, and navigate to the current directory.
- Install the library by typing **install_zumobotlib** in the MATLAB command window



```
>> install_zumobotlib
```

To verify successful installation of the ZumoBot Simulink Library,

- Click on the Simulink Library Icon on the MATLAB Toolstrip to launch the Simulink Library Browser
- In the Library Browser window, check if ZumoBotLib is included in the list on the left.



Simulink
Aerospace Blockset
Communications System Toolbox
Computer Vision System Toolbox
Control System Toolbox
DSP System Toolbox
Embedded Coder
Fuzzy Logic Toolbox
HDL Verifier
Image Acquisition Toolbox
Instrument Control Toolbox
Minor Time Step Logging
Model Predictive Control Toolbox
Neural Network Toolbox
OPC Toolbox
Real-Time Windows Target
Report Generator
Robust Control Toolbox
SimEvents
SimRF
Simscape
Simulink 3D Animation
Simulink Coder
Simulink Control Design
Simulink Design Optimization
Simulink Design Verifier
Simulink Extras
Simulink Support Package for Arduino Hardw
Simulink Verification and Validation
Stateflow
System Identification Toolbox
Vehicle Network Toolbox
ZumoBotLib
xPC Target

Simulink Model

Now that the required environment has been set up, let us look at how we can create a model for the line follower.

In the previous tutorial (http://adafru.it/dsR), we had discussed how to make the Zumo Robot move along a curved trajectory using Simulink. The tutorial also discussed how we can change the amount by which the Zumo Robot turns by controlling the angular velocity and linear velocity which were inputs to the model used.

In this tutorial we will build on the model discussed in the previous guide, to make the Zumo Robot follow lines. The idea here is that the center of the Zumo Robot should always be on a black line on a track. If for some reason the Zumo Robot turns away from the line, we should make the Zumo Robot turn appropriately for it to return to the line. The steps required to perform this task are:

Acquire Sensor Input:

- Identify the relative instantaneous position of the Zumo Robot with respect to the black line.
- The amount by which the center of the Zumo Robot has deviated from the black line which serves as reference is known as the error signal
- The error signal is computed using information from the reflectance sensors.

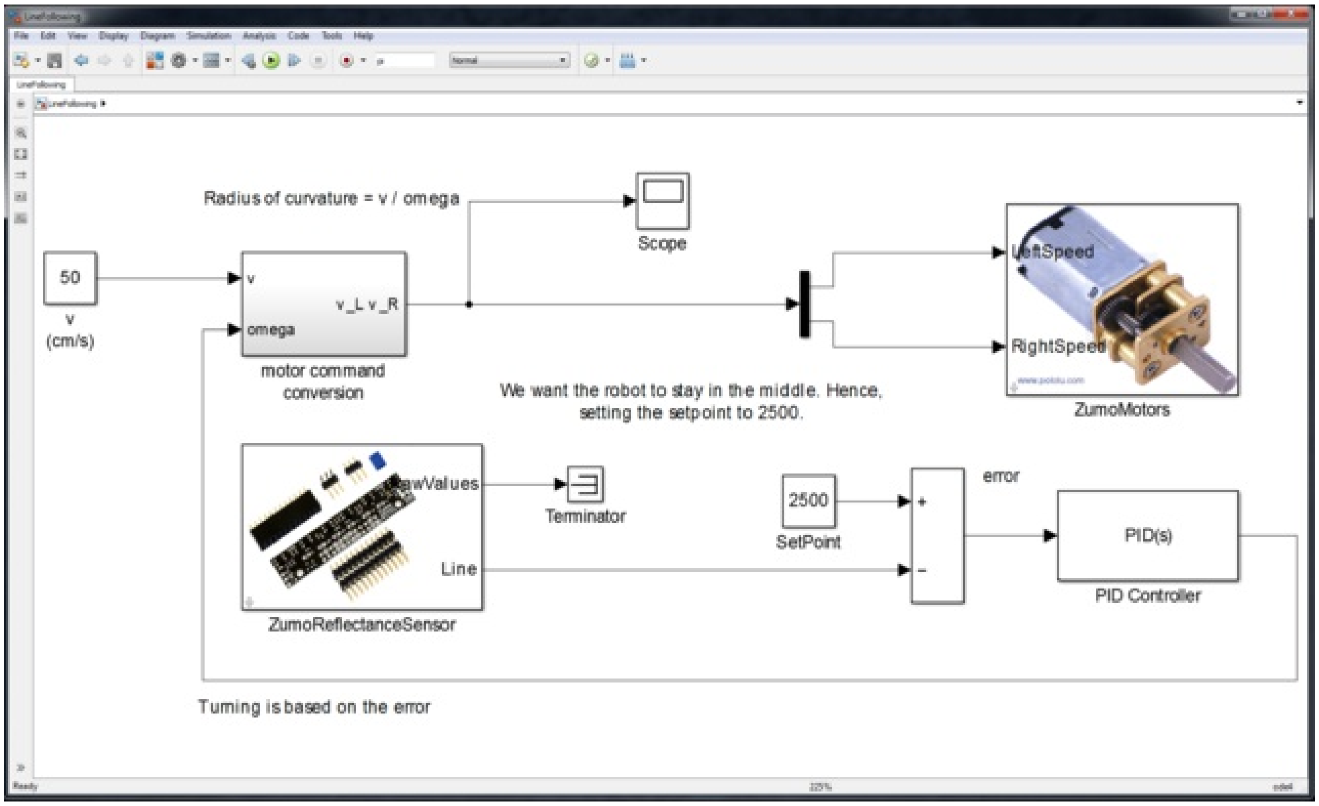
Process sensor data using Controller:

- On the basis of the error, compute amount of turn required to ensure the Zumo Robot follows the line.
- The signal sent to the Zumo Robot motors to ensure that the Zumo Robot does not deviate from the black line is called the control signal
- The control signal is computed using a control algorithm known as PID Control

Actuate Motors:

- Use the output from the PID controller to turn the Zumo Robot.
- This is done using the motor control model discussed in the previous tutorial (http://adafru.it/dsR)

The above three stages are combined in an example model included with the ZumoBot Simulink Library. To access this model, navigate to the '**examples**' directory through MATLAB, and type '**LineFollowing'** in the MATLAB command window.

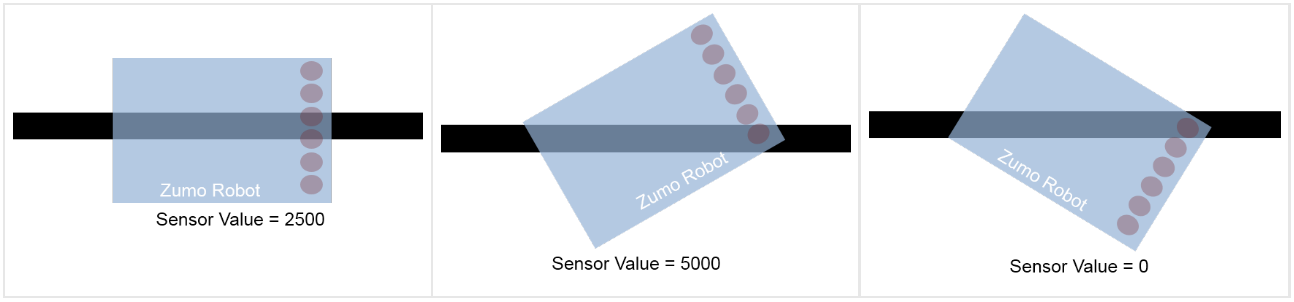


The individual stages of the model are explained in the succeeding pages

Acquire Sensor Input

The first step in programming any line follower is to get an idea about where the robot is relative to the line it has to follow. For this purpose, infrared sensors or reflectance sensors are used. These sensors detect the amount of light reflected off a surface. A black surface reflects much lesser light compared to a white surface, and thus reflectance sensors can be used to detect the presence of a black line on a white track.

The assembled Zumo Robot has an array of 6 reflectance sensors on the lower surface. Using the output from the 6 sensors we can detect the relative position of the Zumo Robot with respect to the black line. For any generic line follower, you would have to combine the output ofall the reflectance sensors in a sensible way. Fortunately, the '**ZumoReflectanceSensor**' block in the ZumoBot library does that for us and gives us one single numeric value ranging from 0 - 5000. From the numeric value, we can get an idea of where the black line is relative to the Zumo Robot. This is illustrated in the diagrams below:

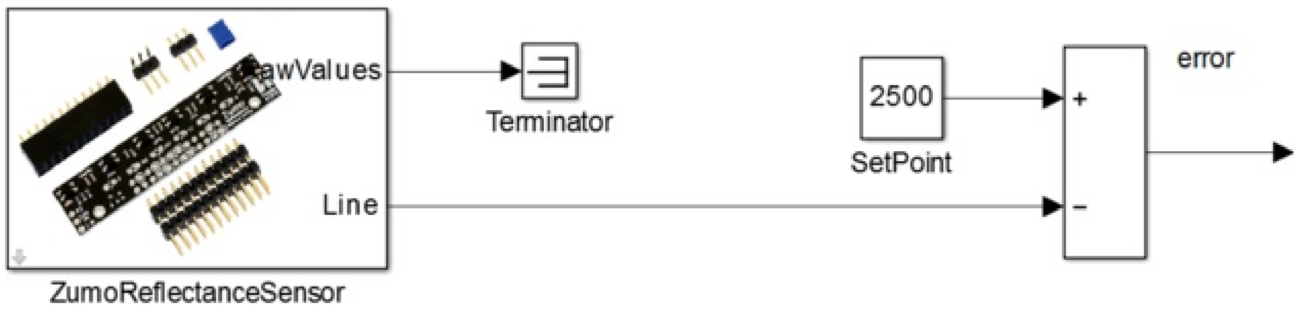


Since a numeric value of 2500 corresponds to the center of the Zumo Robot being above the black line which in turn corresponds to moving forward without turning, we can use that value as the reference.

If the numeric value exceeds 2500, then the robot is on one side of the black line, and if the numeric value is below 2500, then the robot is on the other side of the black line.

To make things easier, we can subtract 2500 from the output of the '**ZumoReflectanceSensor**' block in which case any positive difference implies that the robot is on one side of the line, and a negative difference implies that the robot is on the other side of the black line. This difference between the reference value and the actual value is also known as the 'error' signal

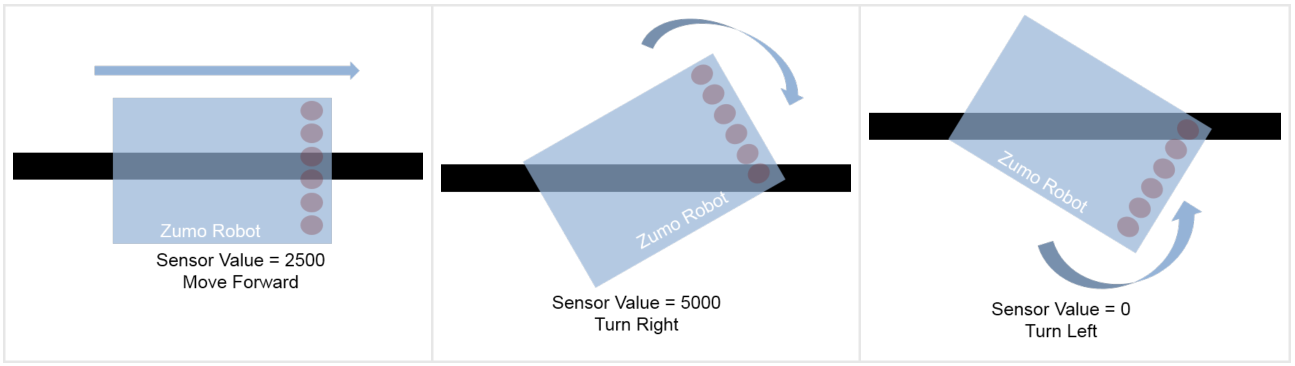
The block diagrams responsible for these steps are shown below:



In the above block diagram, the '**ZumoReflectanceSensor**' block returns a numeric value as output. We then subtract 2500 from the numeric value to obtain the error signal which is then pushed further to the controller block. We will discuss the controller block on the next page.

Process sensor data using Controller

Once we have the error signal we can decide how to move the Zumo Robot so that it stays on the black line. The signal sent to the motors to ensure that the Zumo Robot stays on the black line is called the 'control' signal. The function or component which generates the control signal is known as the controller.



The simplest controller is illustrated in the figures above. The logic being: if the Zumo Robot center is to the left of the black line, then turn right by a fixed amount so that it eventually returns to the black line. Similarly, if the Zumo Robot center is to the right of the black line, turn left so that it returns to the black line. This controller is known as the Bang Bang Controller.

For this tutorial however, we will be using a more sophisticated controller known as a PID Controller, specifically a PD controller. Let us look at what a PID Contoller is all about:

PID Controller

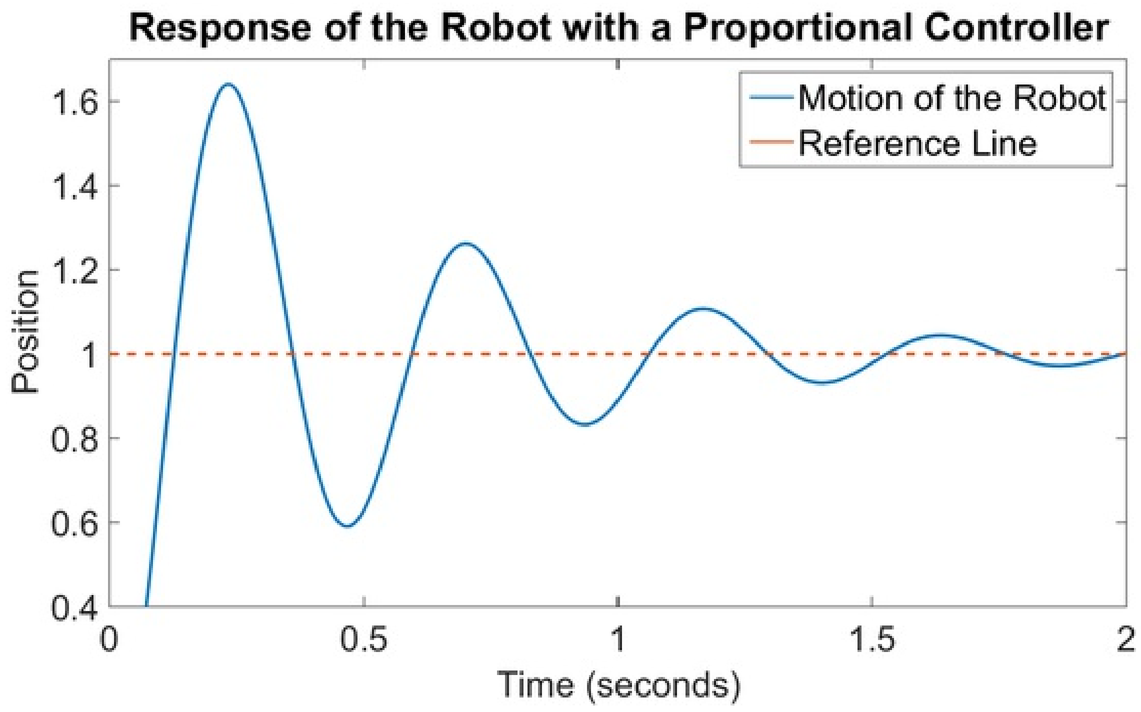
In the Bang Bang controller, we made the robot turn a fixed amount in a particular direction. However if we extrapolate the logic used, then it makes sense that the amount by which the robot turns should be proportional to the amount by which it has deviated from the black line. The further the center of the robot is from the black line, greater should be the amount by which the robot should turn to return its center on the black line.

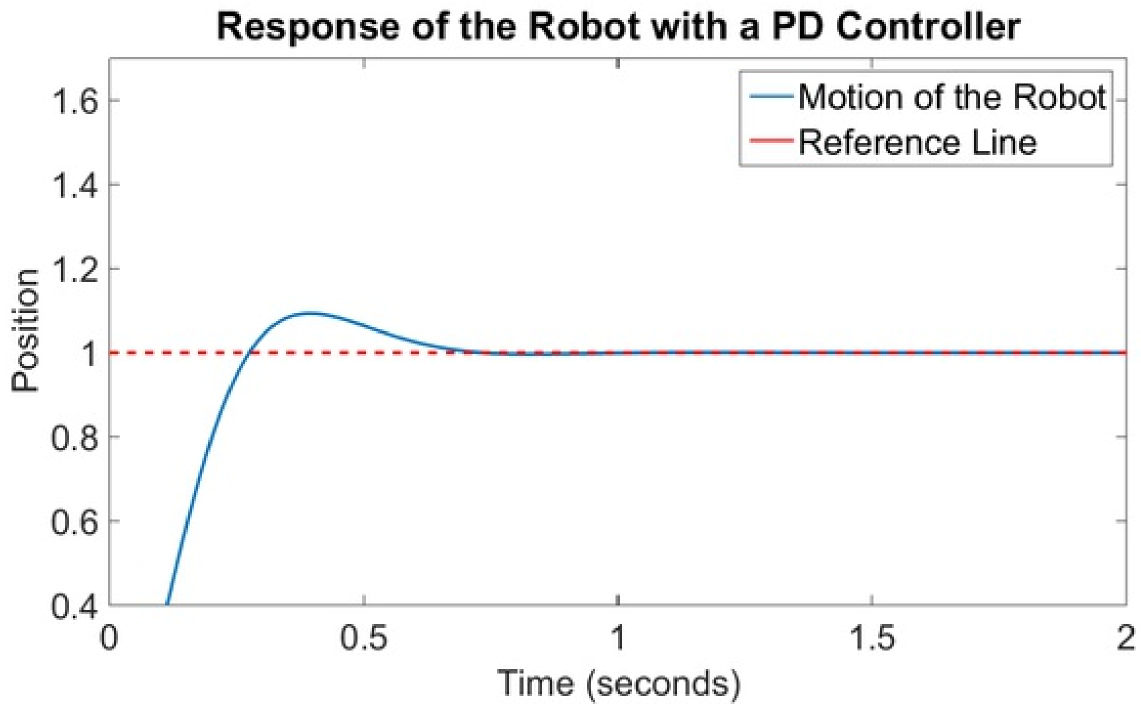
Such a control logic is employed in a Proportional Controller, in which the control signal is proportional to the error signal. The Proportional Controller corresponds to the '**P**' part of the **P**ID Controller.

Depending upon the proportionality constant, the amount by which the robot turns for a particular value of the error signal will vary. This constant of proportionality 'P' is a parameter of the model which has to be tuned as per the application.

One limitation of the Proportional Controller is that the robot can continue oscillating about the black line due to inertia. In this situation, the robot will continually oscillate about the black line while moving forward and will not be able to accurately follow the black line.

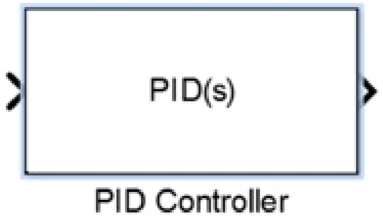
One way to dampen the oscillations so that the robot returns to the black line quickly is to add a derivative component to the control signal. In this case we compute the instantaneous derivative of the error signal which would serve to dampen the oscillations. A constant is multiplied to this derivative and the product is added to the control signal. This constant is known as the derivative gain and corresponds to the '**D**' part of the PI**D** Controller.

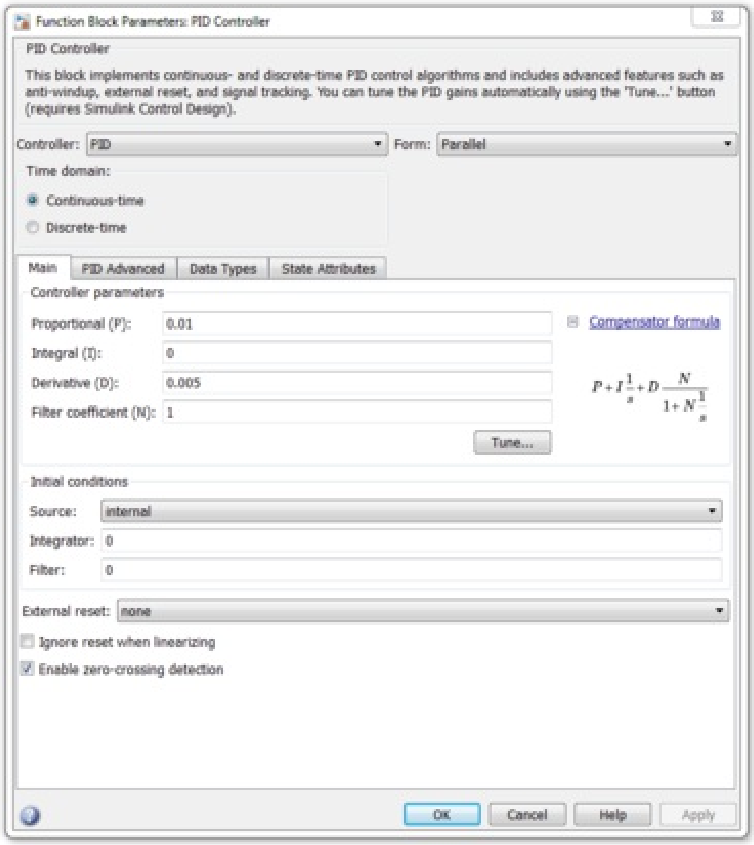




The '**I**' part of the PID controller is not used for this particular application. Thus specifically, this controller is known as a **PD** Controller.

In the model, the block labelled 'PID Controller' represents the controller described above. You can view the values for 'P','I','D' by double clicking on the block.The example model already has some default values for 'P' and 'D' while 'I' is equated to 0. For an alternate explanation for the PID Controller, check out this blog post (http://adafru.it/dsS) which provides a great analogy to explain the PID Controller using pendulums.




Function Block Parameters: PID Controller

PID Controller

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PID   Form: Parallel

Time domain:
- ● Continuous-time
- ○ Discrete-time

Main | PID Advanced | Data Types | State Attributes

Controller parameters

Proportional (P): 0.01
Integral (I): 0
Derivative (D): 0.005
Filter coefficient (N): 1

Compensator formula

$$P+I\frac{1}{s}+D\frac{N}{1+N\frac{1}{s}}$$

Initial conditions

Source: internal
Integrator: 0
Filter: 0

External reset: none

☐ Ignore reset when linearizing
☑ Enable zero-crossing detection

Tuning the PD Controller

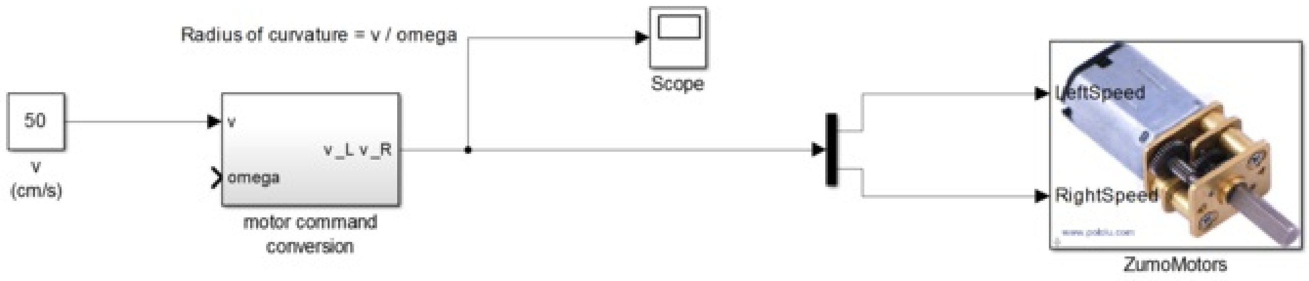
One method to tune the PD Controller without using a mathematical model of the robot is listed below:

1. Adjust the proportional gain of the controller till you obtain a suitable response.
2. As a thumb rule, set the value of the derivative gain to be equal to 1/10th of the proportional gain.
3. Now adjust the derivative gain and observe the change in the robot behavior, till a suitable behavior is obtained.
4. Increasing the proportional gain will increase the amount by which the robot turns.
5. Increase the derivative gain will dampen the magnitude of oscillations by the robot.

Try tuning the PD Controller shown in the model to see how the change in parameters affects the behavior of the robot.

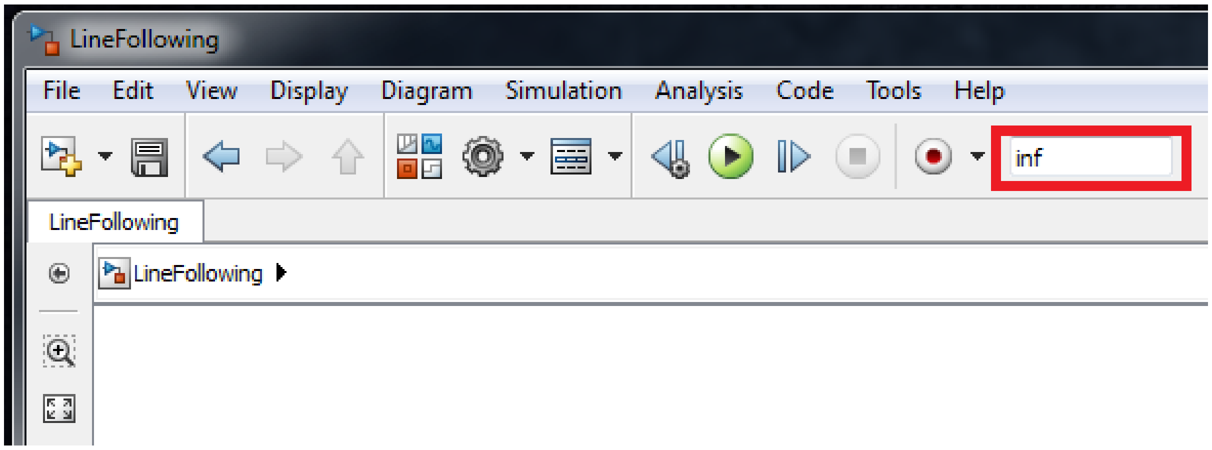
Actuate Motors

The controller uses the information obtained from the reflectance sensor to decide the amount by which the Zumo Robot has to be turned. This output is fed to the model we had discussed in the previous tutorial (http://adafru.it/dsR) as the 'omega' parameter to control the motors. For this model, we assume that the forward velocity of the Zumo Robot remains constant.



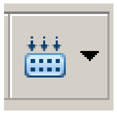
Thus with this workflow, at every time instant, we compute the position of the Zumo Robot relative to the black line. From that computation, we decide the amount by which the Zumo Robot should turn to continue staying on the line, and using that value we turn the Zumo Robot at that instant.

This process allows the Zumo Robot to follow the black line. This process can repeated an infinite number of times or till a stopping criteria is met, at which state, we can stop the computation. To vary the time duration for which the program runs, you can modify the value highlighted below to 'inf' for infinite time, or to any finite value (The unit for the number is seconds).


LineFollowing
File Edit View Display Diagram Simulation Analysis Code Tools Help
LineFollowing
LineFollowing

inf

Once you have tuned the PD Controller, to build and download this model on the ZumoBot, click on the Build or Deploy on Hardware button in the right top corner of the model:



Read the messages in the bottom status bar in the model window to confirm that the model is successfully downloaded to the target.