## **VEX Simulink Curriculum by Gary Garber is licensed under a**  [**Creative Commons Attribution-ShareAlike 4.0 International License**](http://creativecommons.org/licenses/by-sa/4.0/)**.**

# Module 2 Overview

In this module we will

* Introduce how to design a model to detect objects in the path of the robot using VEX Ultrasonic Range Finder
* Design P- and PID-Controllers for the obstacle detection and avoidance tasks using the Ultrasonic Range Finder

# Module 3.1 Ultrasonic Range Finder

In this section we will learn how to draw a Simulink model which uses the VEX Ultrasonic Range Finder. We will build a model which uses Stateflow to control the state of the robot based on sensor data from the VEX Ultrasonic Range Finder.

You will learn to

* Use the VEX Ultrasonic Range Finder
* Write a two-level controller for the robot
* Navigate to a set distance from an obstacle

The VEX Ultrasonic Ranger finder emits 40 kilohertz (KHz) sound waves and receives the waves reflected from a surface. By analyzing the time delay in the reflected waves, the Ultrasonic Range Finder can tell you the distance to an object. The Ultrasonic Range Finder works at a minimum distance of 3 cm, and a maximum distance of 3 meters. The Ultrasonic Range Finder uses two 3-wire cables, one for output (emitting the ultrasonic wave) and the other for input (measuring the reflected wave) as you can see in the following image.

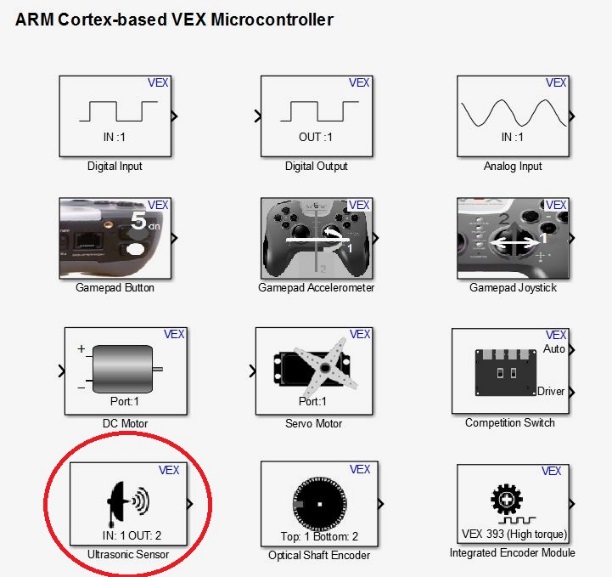
****

The VEXnet Ultrasonic Range Finder

You should attach the Ultrasonic Range Finder to your robot in such a way that the sound waves are not blocks by other parts of your robot.

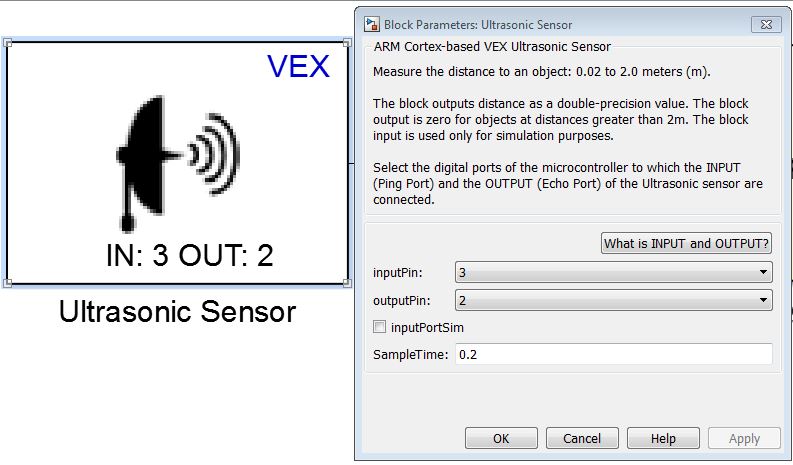
We are going to begin by creating a simple model of a robot that will move forward towards a wall and stop a set distance from the while. You can find a copy of this model in the file called MotionSensorStop. We will call the desired distance from the wall the set-point. You can think of this like setting the desired temperature on a thermostat in your home. The set-point is where we desired to be.

From the VEX Companion Library you can find the Ultrasonic Sensor block.

****

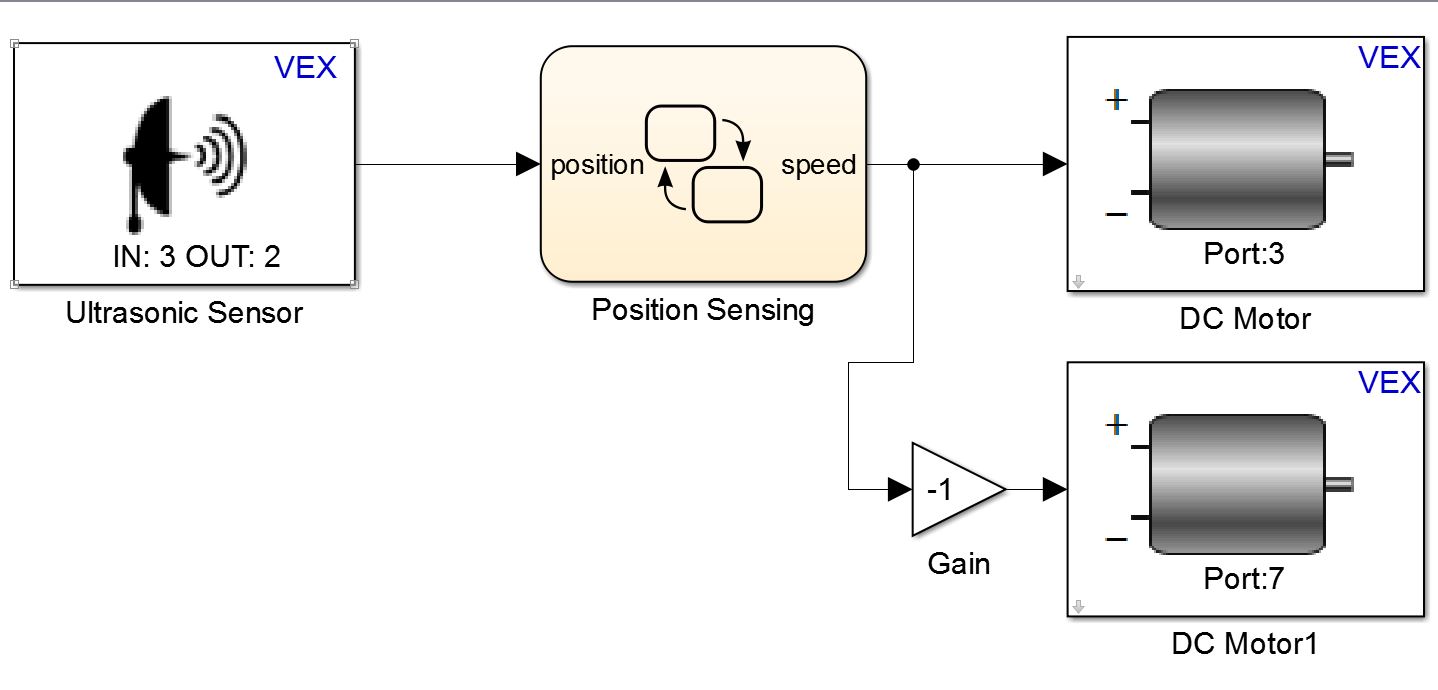
Ultrasonic Sensor Block in VEX Companion Library

Drag this block onto your Simulink diagram. Make sure the Block parameters are setup as in the following image. Remember you will have to set both the input and output ports in your block parameters to where you have physically plugged the Ultrasonic Range Finder into your VEX Cortex.

****

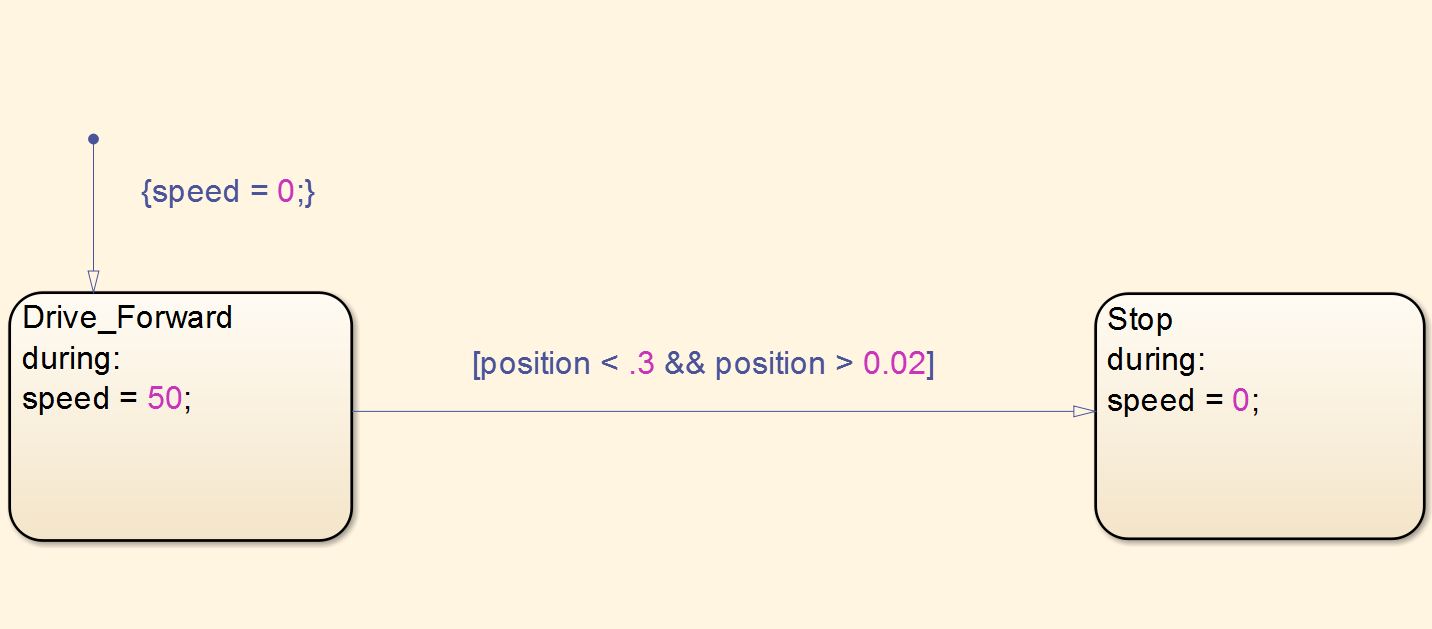
Ultrasonic Sensor Block Parameters

We will also create a Stateflow Chart and the motor blocks onto our Simulink diagram. The Stateflow Chart will shut the motor off when it reaches a certain range value. The output of the Stateflow Chart goes to the Motor Blocks.

****

Simulink Diagram using the Ultrasonic Sensor Block and Motor Blocks and a Stateflow Chart

Looking inside of the Stateflow Chart we can see the simple commands. We have created a new input variable called **position** (for how far we are from the obstacle). Note that there are a few additional steps in this Stateflow Chart.

****

Stateflow Chart using the range as the Condition

This Stateflow Chart has been written to create a model of a robot which will stop at 10 cm or 0.1 meters from an obstacle. In the Stateflow Chart, you will notice that there are two conditions on the variable position. We have used the condition [**position** <= 0.1 && **position** > 0.02]. The symbol **&&** means that both conditions must be satisfied.

When you deploy this program to your VEX Robot, you will find it will move forward until the robot is 10 cm from your wall or obstacle.

In this section we have learned how to use the Ultrasonic Range Finder.

Additional challenge: The model we created in this section was of a robot which would move forward until it was a set range from a wall, and then the model terminates. Imagine you are trying to create a model where your robot is trying to navigate to a location relative to the corner of two walls. Create a model where the robot moves forward, stop 10 cm from a wall, turns 90 degrees, and stops 10 cm from a second wall.

Additional challenge: In the above model, we created a two-level controller. The robot was moving until it went past the set-point, and then stopped. An additional challenge is to create a three-level controller. Define a range of values where you want to be, for example between 10 and 15 cm from a wall. If the robot is greater than 15 cm from the wall, it will move forward. If the robot is too close to the wall, it will move in reverse.

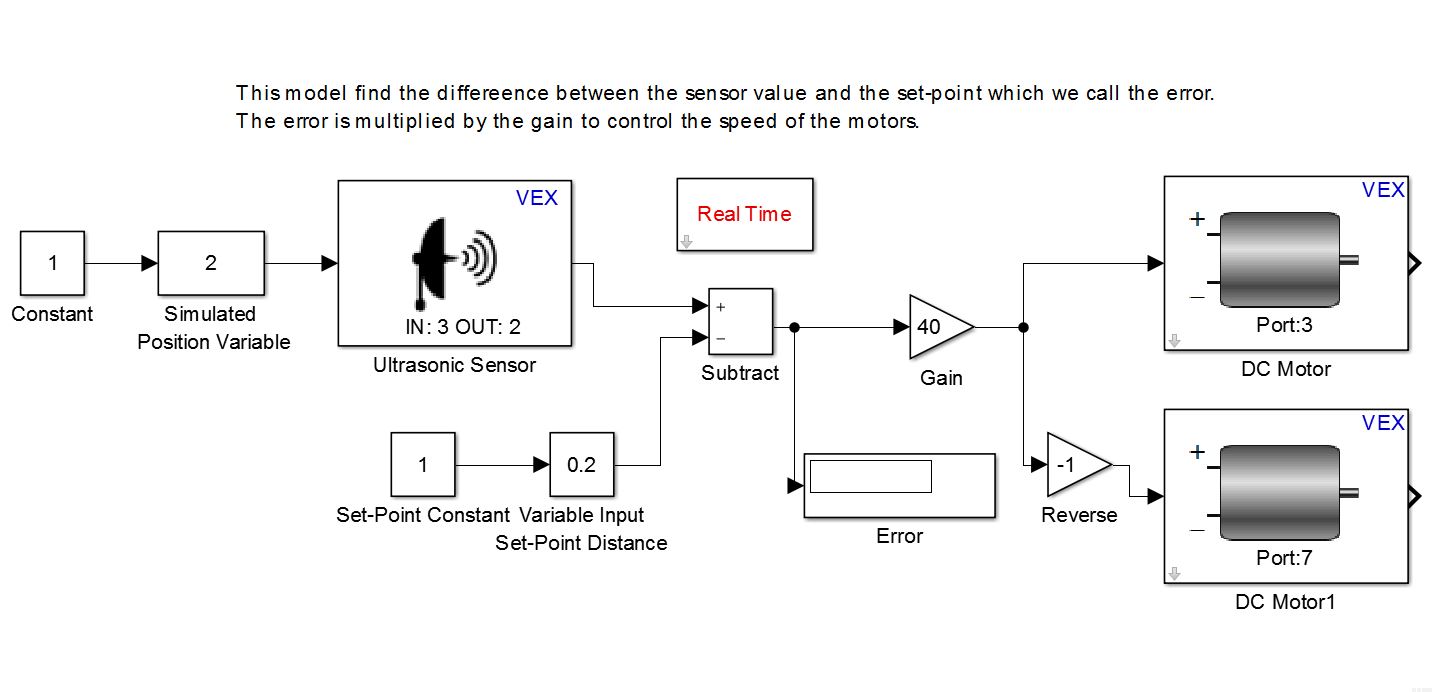
# Module 3.2 Proportional Controller

In this section we will learn how to draw a Simulink model which uses the VEX Ultrasonic Range Finder combined with a Proportional Controller. Instead of moving forward and stopping at the set-point, our robot will slow down as we approach the set-point.

You will learn to

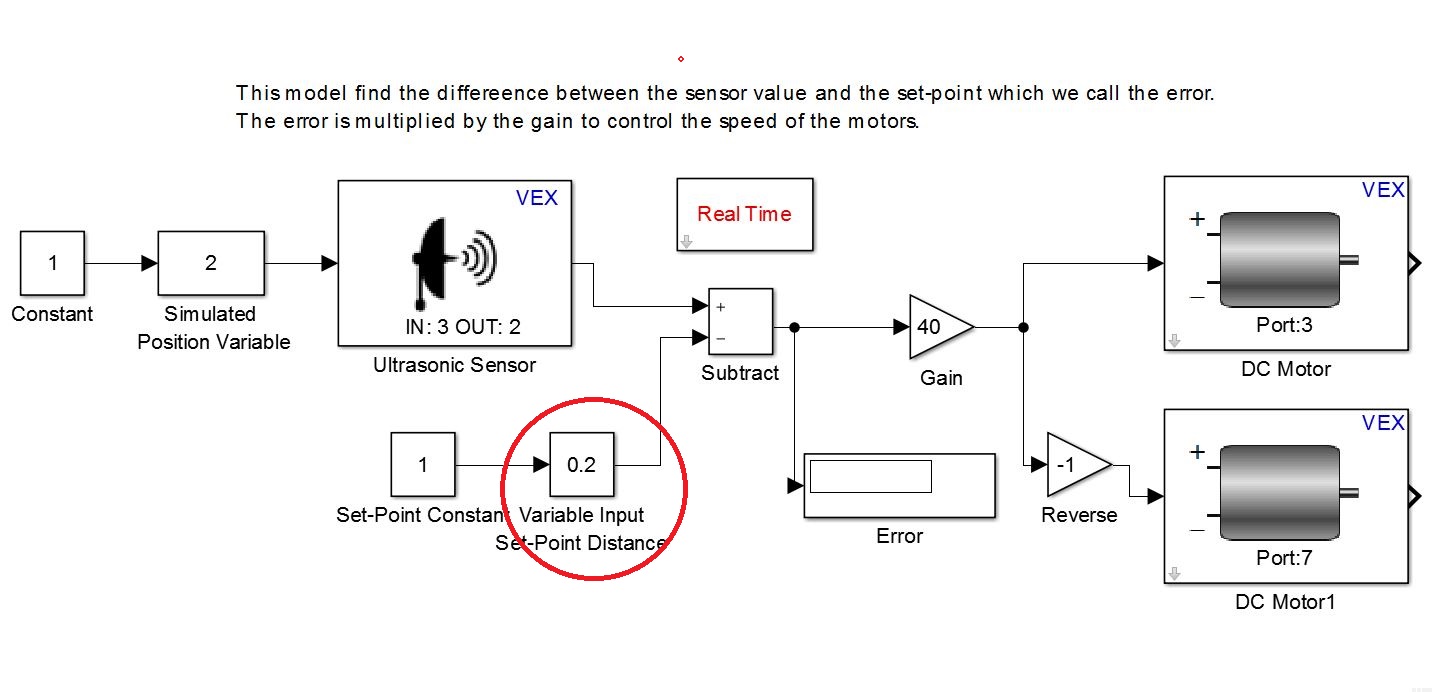
* Use the general Simulink Library
* Create a Proportional Controller using several blocks
* Run simulated varying input to our model using the Ramp block
* Graph the varying output of our model on a Scope
* Run the Simulation in External Mode

If you were driving a car, you do not slam on the brakes suddenly when you reach a stoplight. Ideally, your car will gradually slow to a stop as you reach the stoplight. If you were programming an autonomous car, you would do the same thing. The two-level controller model we created in the previous section was a model of a car where you slam on the brakes. In this section, we will create what we call a **proportional controller**. You can refer to this model in the file called Proportional.slx.



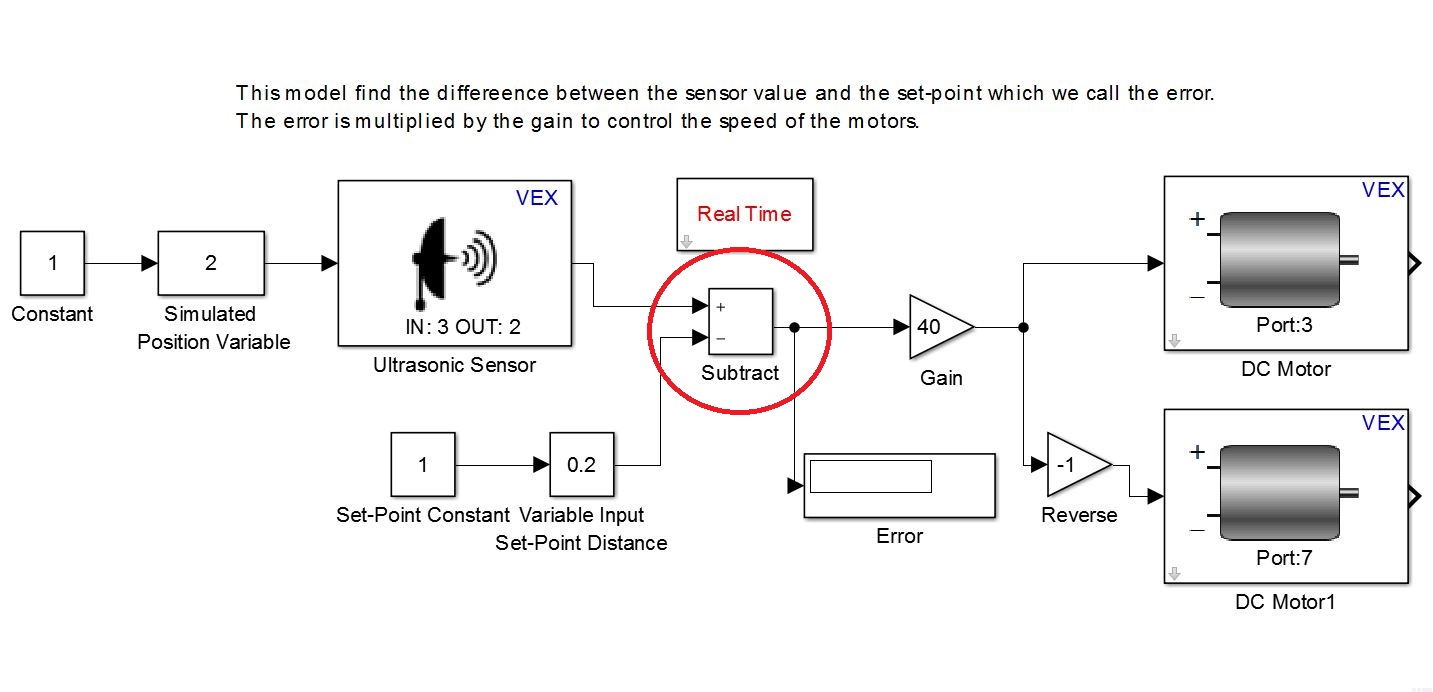
Proportional Controller

The speed of our robot will depend on how far we are from our **set-point**, the distance we want to stop from the wall.



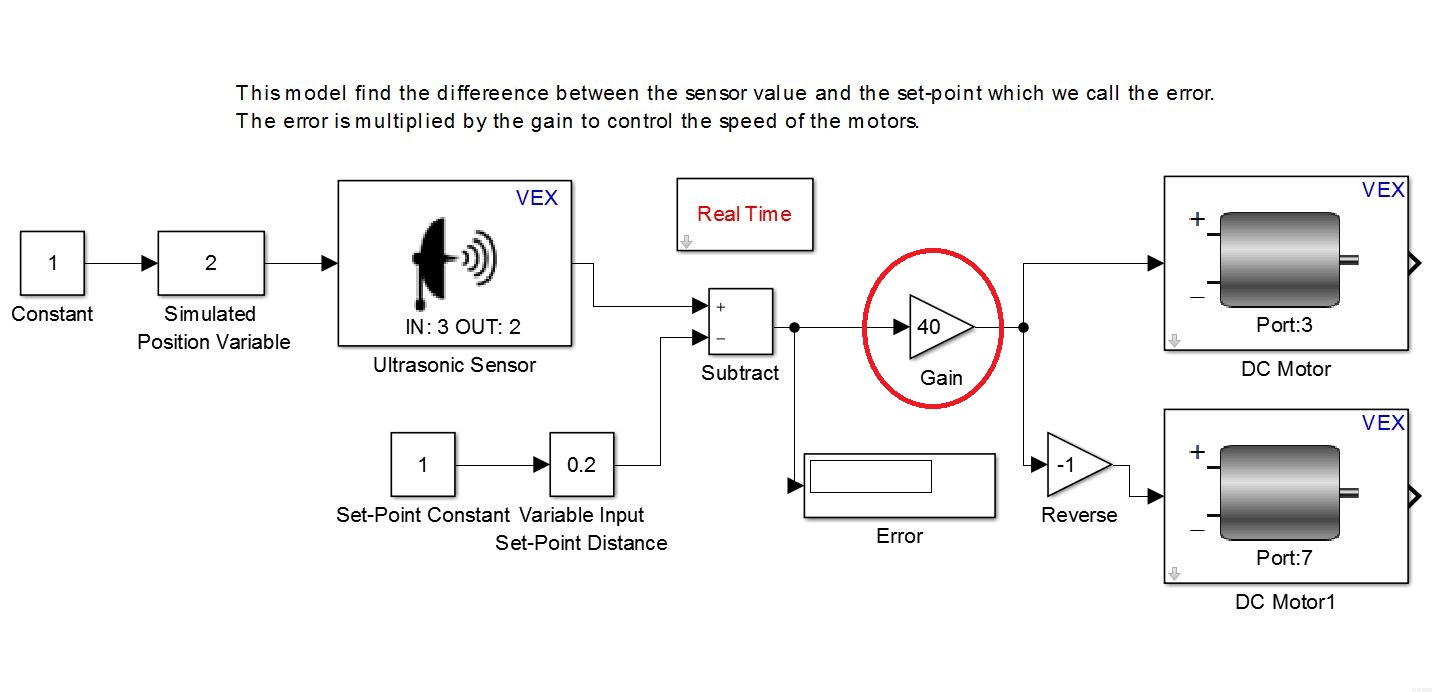
Proportional Controller - Set Point

The difference between the sensor reading and our set-point distance is called the **error**. This error is input into the proportional controller.



Proportional Controller Error

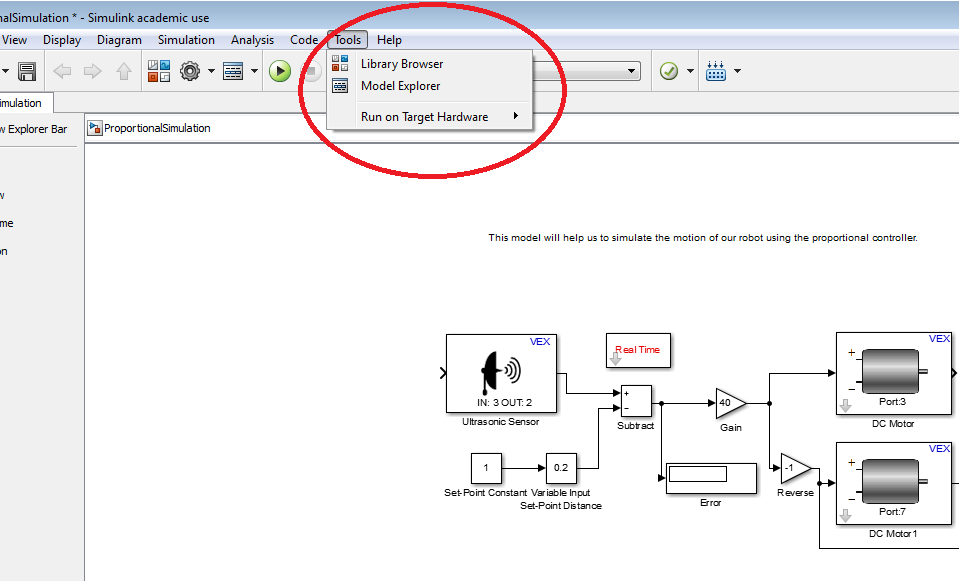
The output of the proportional controller controls the speed of our robot. The proportional controller magnifies (or diminishes) the error by multiplying it by a term we call the **gain**. The result of multiplying the gain by the error is directly input into the speed of our motor. The farther we are from our set-point, the greater the error. The greater the error, the faster our robot will move. For a small error, our robot will move slowly.



Proportional Controller Gain

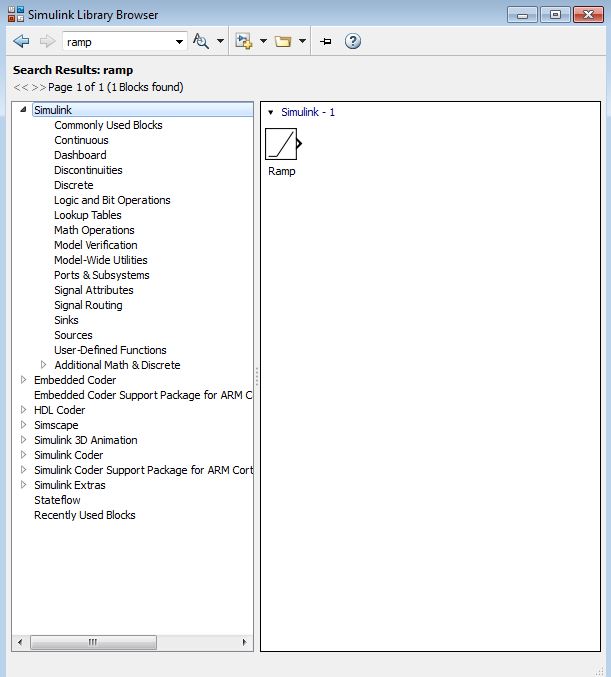
The process of determining the correct value for the gain is called **tuning** the controller. Normally, you would have to do this by trial and error with your physical robot, but Simulink allows you to create a simulation. We can add several blocks to our Simulink diagram to create a model which simulate our robot using the Proportional Controller. You can refer to this model in the file called ProportionalSimulation.slx.

We are going to use a block that is not part of the VEX Library. We must begin by opening the Simulink Library Browser. You can find this under the tools menu as you can see in the next image.



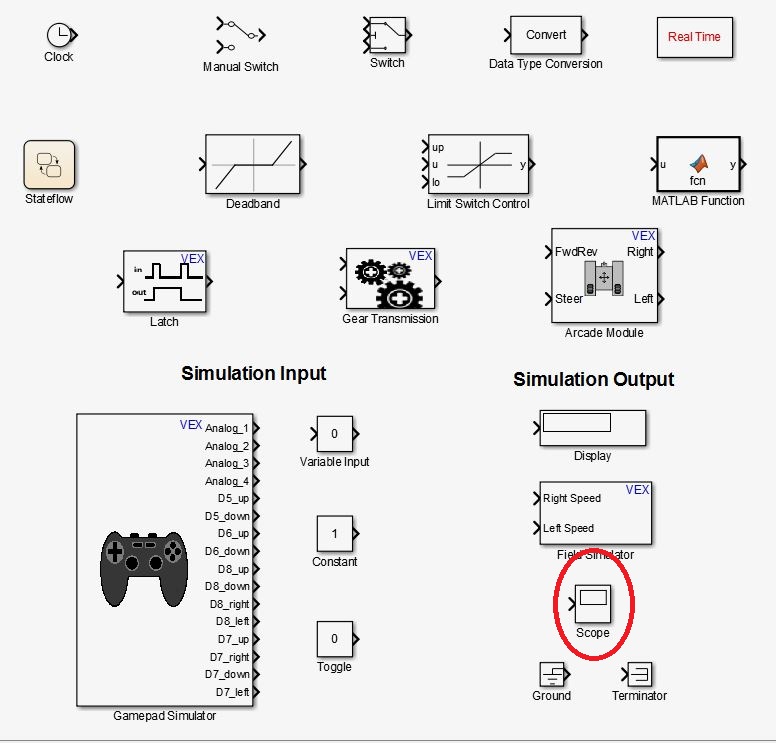
Open Library Browser

In the search menu of the Library Browser, type ramp and hit enter. You will see one block called ramp. Drag this Ramp block onto your Simulink Diagram.



Enter Ramp in the Search Menu and Choose the Ramp Block

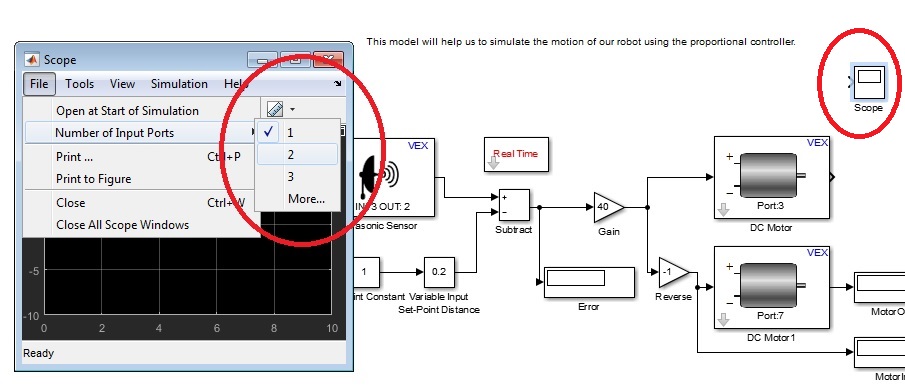
Next, drag a Scope block from the Utilities library onto your Simulink diagram in order to graph the output of our model.



Scope Block from the Utilities Library

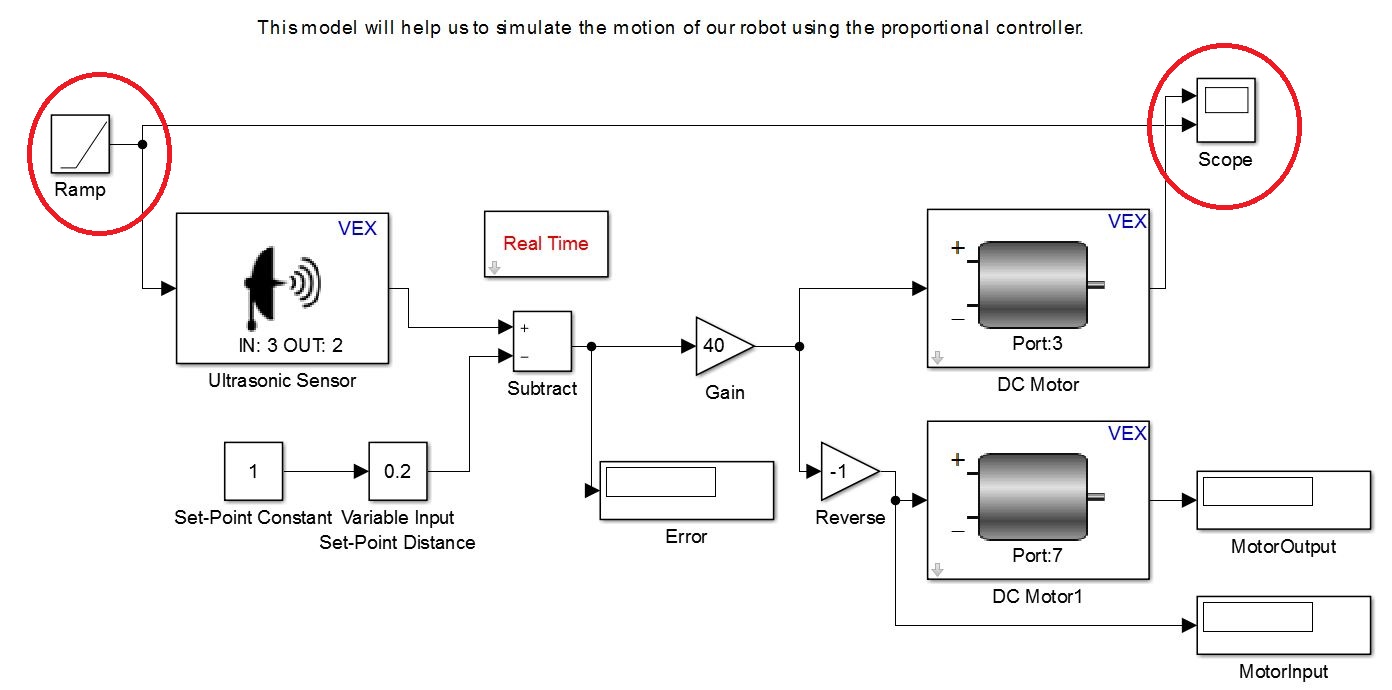
Next we will need to add additional input ports to our Scope.

1. Click on the Scope block once it is on our Simulink Diagram.
2. Choose the File dropdown Menu once the Scope open up
3. Chose Number of Input Ports
4. Select 2 Input Ports



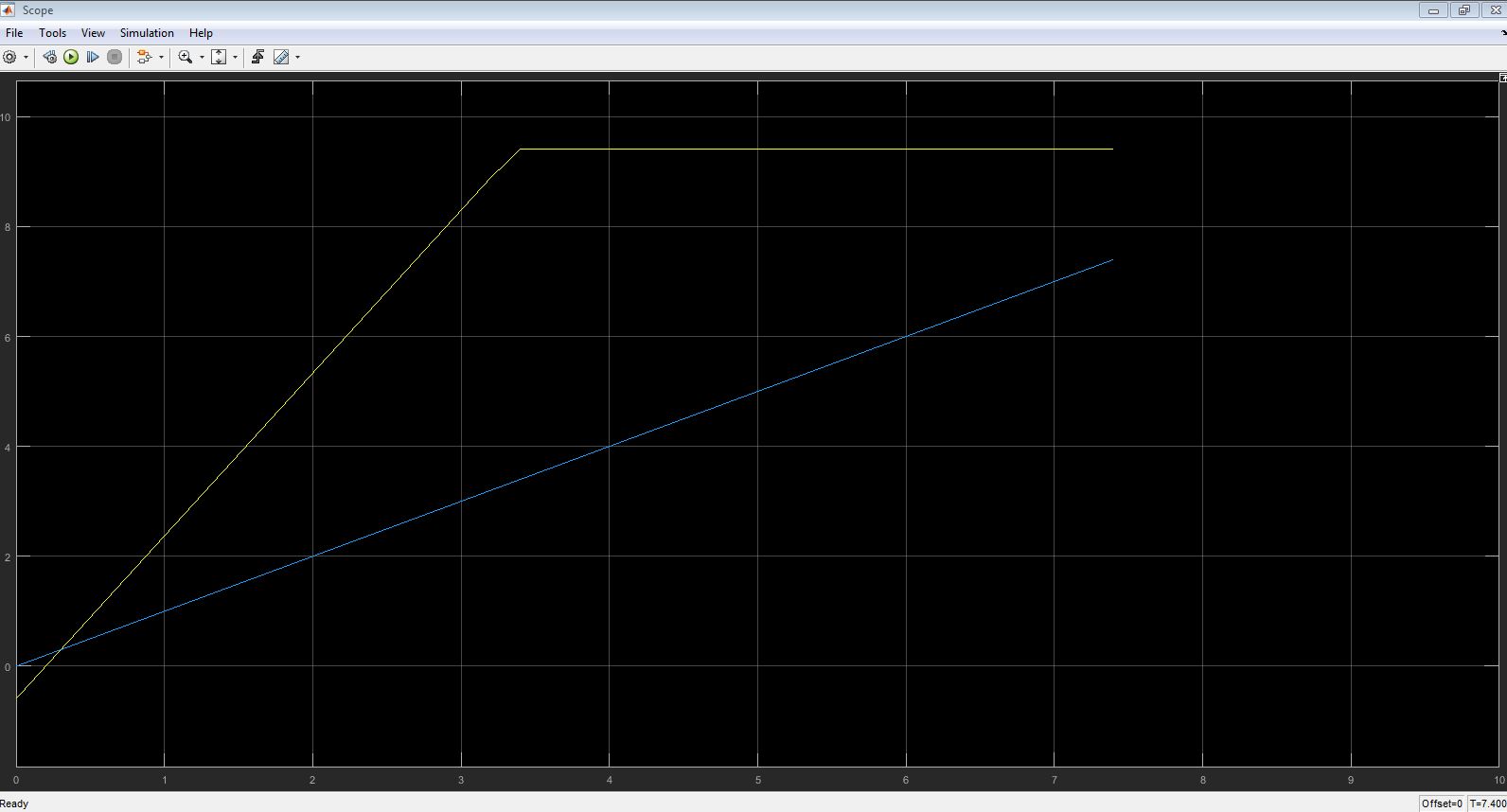
Adjust the number of Input ports for the Scope to 2

The Output port of the ramp block should be wired to the input port of the scope. Additionally the output port of one of the motor blocks should also be wired to the other input port of the scope.



Wiring the ramp and scope blocks on your Simulink diagram.

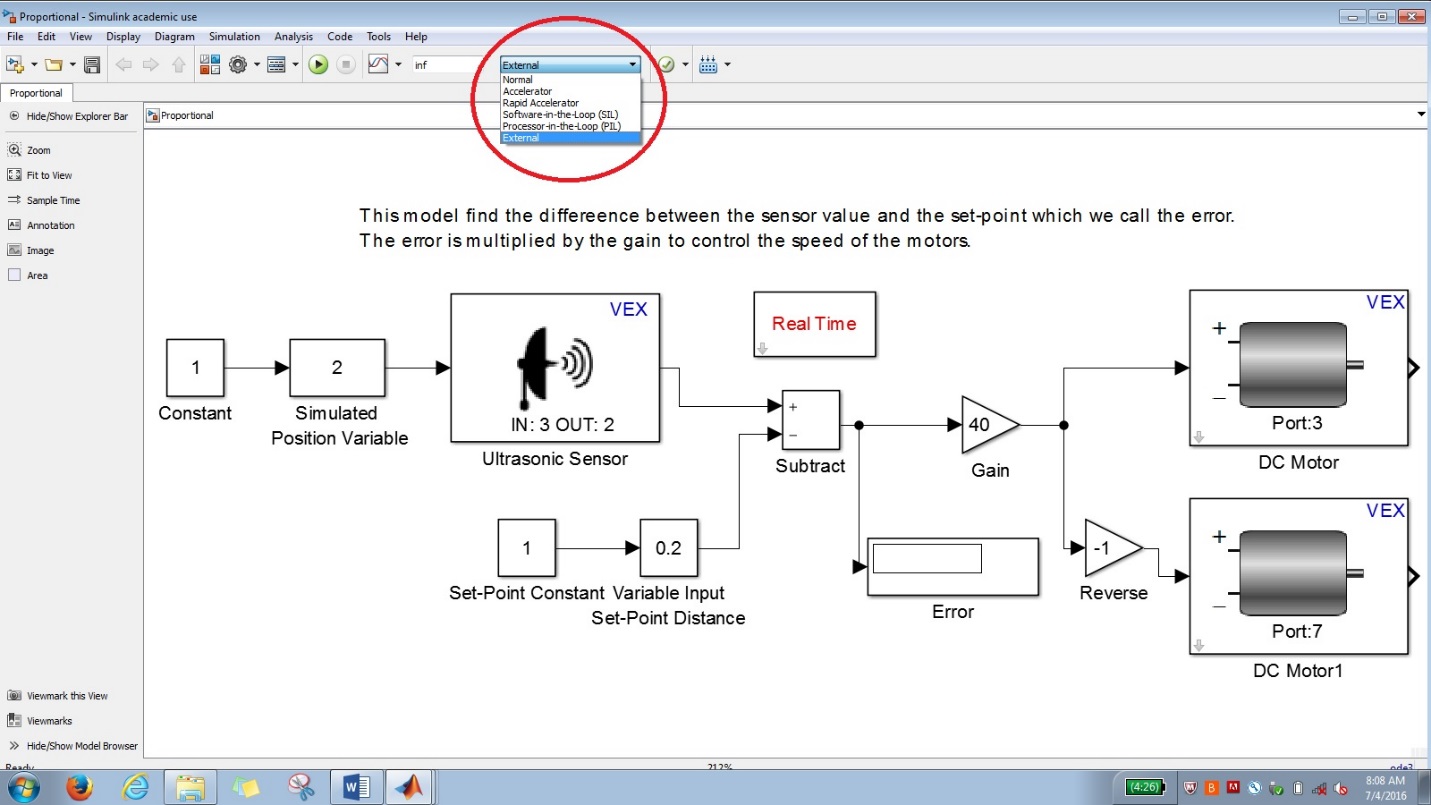
When you run this model in the simulation will generate a graph on the scope which you can view in the following image. The blue line is the input which is the position of our robot relative to the wall or obstacle. The yellow line is the output of the motor.



Graph produced by the Scope

You can see how using a combination of the Ramp block and the Scope allows us to save time and simulate the actual motion of our robot.

Previously, when we have run a model on our robot, we have deployed the model to our robot and let the robot execute the model independently of the computer. We can also run the simulation in **External** Mode while tethered to the computer to monitor the sensor feedback as you can see in the next image.



Run the Model in External Mode

In order to do this:

1. Click on the Simulation Mode drop-down menu
2. Select External from the drop-down menu
3. Then depress the **Run** button as you see in the next image.

As the simulation runs on your computer it will simultaneously run on your robot. The Display windows will display the actual data from your robot’s sensors.

In this section we saw how to create a model of a robot that uses a proportional controller. We also learned how to create a simulation of our model’s sensor input using the ramp block, a block from the general Simulink library. We ran the simulation to an external target.

In the next section we will learn how to create a controller which uses a full PID control, and not just the proportional controller.

Additional Challenge:

Instead of using the proportional Controller with feedback from the Ultrasonic Motion sensor, create a model with feedback from the Integrated Encoder Module. You will find that this will allow you to stop your robot at a precise shaft encoder value.

# Module 3.3 PID Controller

In this section we will learn how to draw a Simulink model which uses the PID Control Block. PID Control stands for Proportional-Integral-Derivative Control. The PID Controller allows us greater control than the Proportional controller.

You will learn to

* Use the general Simulink Library
* Create a Proportional Controller using the PID Control Block
* Create a full PID Controller

In the previous section, you may have found it can be challenging to tune your proportional controller to the correct value for the gain parameter. If the value of the gain parameter was too small, it would take a long time to reach your set-point. If the value of the gain parameter was too big, your model could easily overshoot your set-point. If it overshoots the set-point it make need to reverse direction for a course correction. With some models, you could even get a situation where the robot’s motion is oscillating around the set-point.

For the simple model of a car (or robot) trying to stop a certain set-point distance from a stationary wall, the proportional controller may suffice. However, if we wanted to model a true autonomous car, the car will need to maintain a safe driving distance moving target, such as another car. If the other vehicle is not moving at a constant speed, we will find that the proportional controller may not suffice.

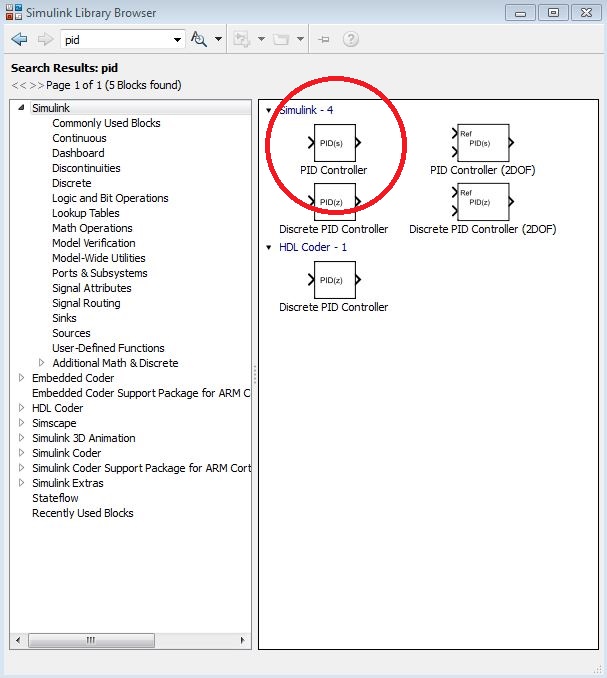
If we find that we are always lagging in trying to attain the set-point distance, it would be beneficial if our robot has a memory to allow it to compensate. The integral controller examines the past values of the error. If the error is consistently large, the integral controller can compensate for this. The integral controller works by adding up, or integrating the past values of the error.

If we find ourselves constantly overshooting the set-point distance then we need to dampen the value of our corrections. The derivative controller examines the rate that we are approaching the set-point distance. If we are approaching too fast, the derivative controller will dampen the correction.

You can refer to this model in the file called PID.slx.

Again, we will need to use the general Simulink Library to grab a PID control block. We cannot find this block in the Vex Simulink Library.

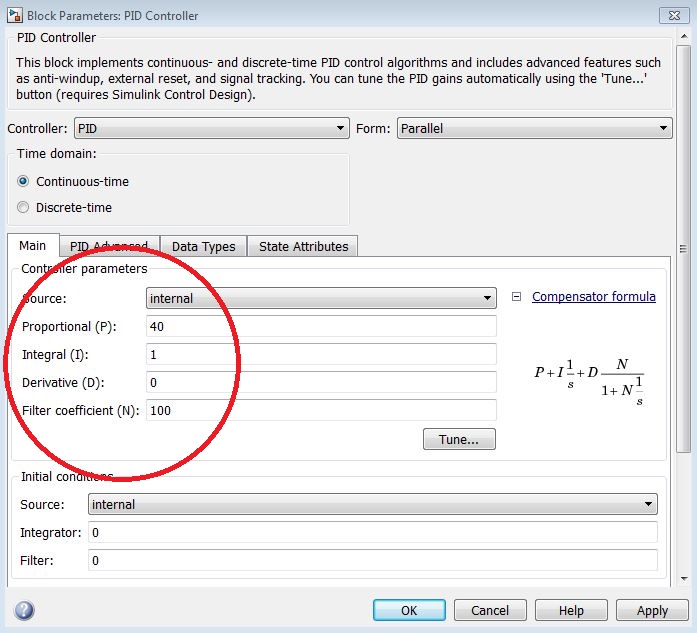
1. Open up the Simulink Library Browser as we did in the previous section.
2. Enter PID into the search menu you will come up with several PID Controller Blocks.
3. Drag the PID Controller block onto your Simulink Diagram.



Entering PID in the search menu

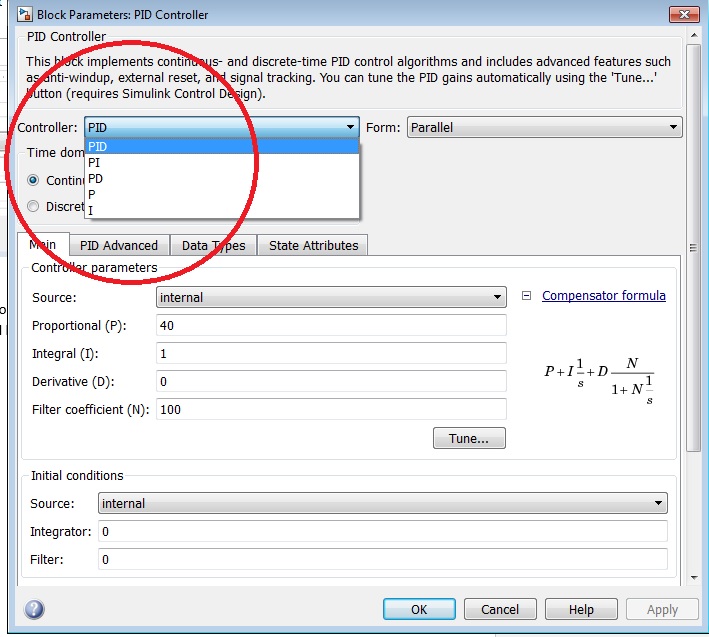
After you have placed the PID Controller on your Simulink diagram, open the block parameters for the PID Controller Block by double-clicking on the PID Block.

As you can see in the next image, you can assign values for the Proportional, Integral, and Derivative coefficients.



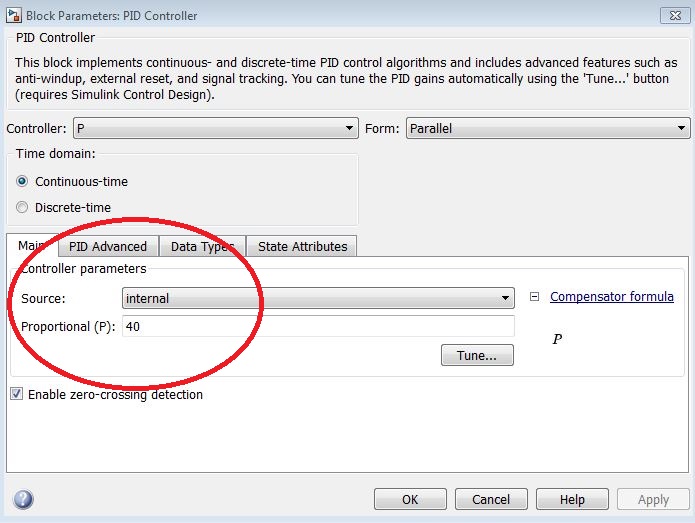
Block parameters for PID Controller

This may look confusing. Under the controller tab, we can set the controller to a P-type controller which is what we created by using individual blocks before.



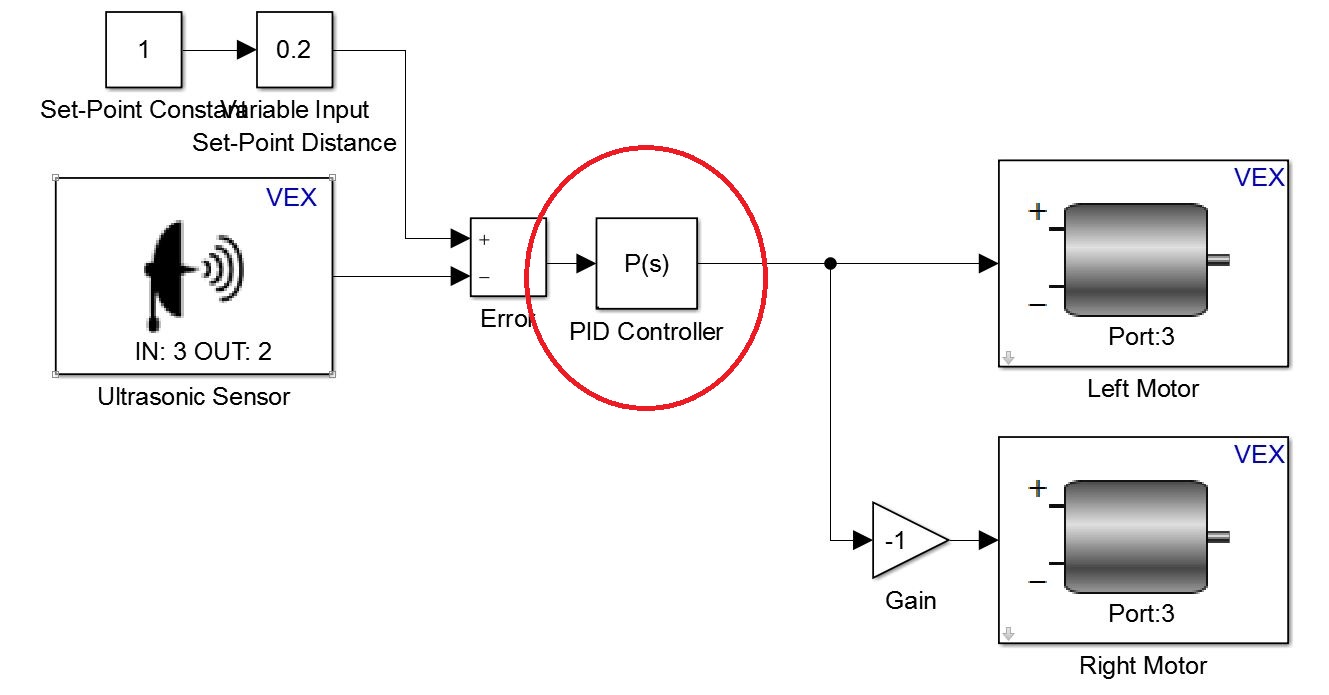
Block parameters P Controller

So if we use the simplest of the controllers, the P-controller, there is only one coefficient to determine. As we mentioned in the last section, the coefficient for the P-controller we often call the gain coefficient.



Setting the proportional constant.

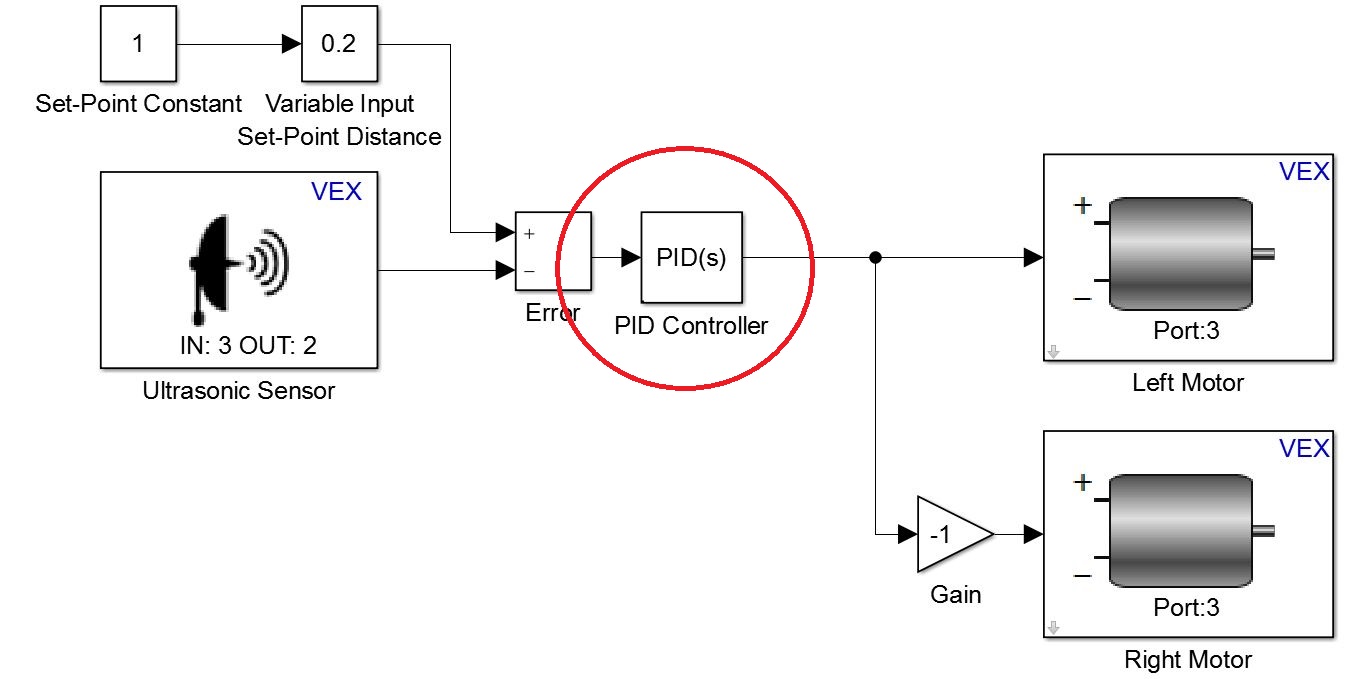
The input for our PID block is the error value. So in the case of our simple P-controller we are merely multiplying the error by the proportional gain coefficient. The output goes to our motor blocks as you can see in the following diagram. We know it is a P-Controller (as opposed to the full PID Controller) because it is labeled P(s) inside of the block.



P-Controller on Simulink Diagram

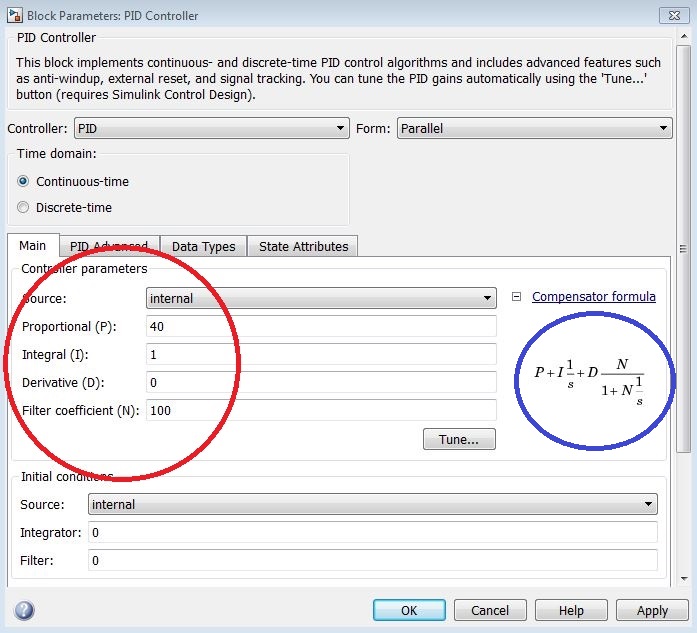
This model will behave in the same was as the Proportional Controller we designed in the previous section using individual math blocks.

If we wanted to change our P-Controller to a full PID-Controller we can do this by choosing the block parameters again.



Changing to a PID Controller

Changing to a PID Controller does not change the number of input or output ports on our controller blocks. It only changes the mathematical functions that take place inside of the block.



Changing to a PID Controller

In the previous image, you can see the coefficients circled in red, and the mathematical function that determines the output of our controller circled in blue. The process of selecting the values of the coefficients is called tuning the controller. We said in the last section that if your gain is too small, your robot will take a long time to reach the set-point. If your gain is too large, it will get there too fast, the robot will overshoot the set-point, and the robot will have to correct its position.

1. When tuning your entire PID controller, start with 0 values for the I and D controllers.
2. Select a P controller value that will just barely cause the motion of your robot to oscillate around the set-point distance.
3. Choose a value for the P controller which is about half this value
4. Tune to a small I-controller value.
5. Tune in a small amount of the D-controller value.

For most purposes, you may find that the proportional controller will solve your problems. It is only for the fine tuning that you will need the full PID Controller.

In this section we learned how to use the PID Controller block.

Additional Challenge: Use the PID controller block for the optical shaft encoder. You will find this useful if you are trying to pick up a variable load with your clawbot and want the arm to reach a certain position.