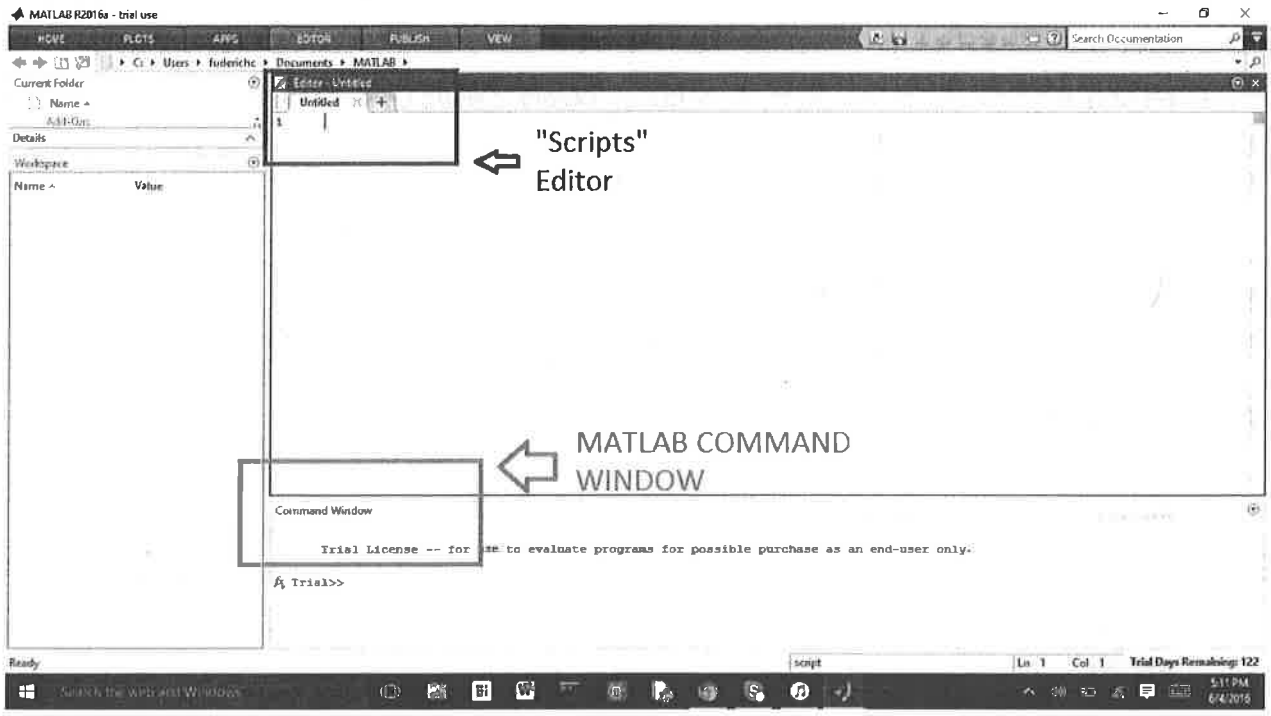
Introduction

For our exploration of programming there is a difference between the location where our code is written. There is the MATLAB command window which can execute code as it is typed thus, acting as a interpreter for what we input. There are also tabs in the top right corner, which are used for opening and writing 'scripts' in the MATLAB interface.



Let's begin by looking at the MATLAB command window:

Exploring variables:

Type into the command window, line by line:

```
A = 10
B=12
A+B= 22
```

Interfacing with the MiniQ (Arduino based robot)

Using the MATLAB command window let's make our first Arduino object using the MATLAB programming language. The miniQ is equipped with an Arduino Leonardo as it's microcontroller which is basically the "brains" for our robot.

Check your device manager to figure out what the name of the USB port the MiniQ is connected to. On my PC computer it is COM14, therefore:

HMC5883(Compass)
D2--SDA
D3--SCL

Infrared obstacle avoidance:
Transmitter: IRL--D13 IRR--D8
Receiver: IRS--D17

Button:
D4--AD6

Encoder:
D0--INT2 Right Motor
D1--INT3 Left Motor

Buzzer
D16—MOSI—Buzzer

Let's experiment with the left motor on the miniQ robot. The pins used on this motor are D5 (PWM) and D12. Calling different values to pin "D5," which supports pulse width modulation (PWM), allows us to control the direction of the motor. This is a concept we will explore further through writing code in the MATLAB command line.

Reference: Lesson 1 Unit 1 Lesson

This is the template for the code we will write:

```
writePWMDutyCycle(a, 'pin', (enter a value ranging from 0.0 to 1.0);
writeDigitalPin(a, 'pin', (1 or 0 to control direction));
```

Here is an application of the above template:

```
writePWMDutyCycle(a, 'D5', 0.5);
writeDigitalPin(a, 'D12', 0);
```

Try this one:

```
writePWMDutyCycle(a, 'D5', 1.0);
writeDigitalPin(a, 'D12', 1);
```

```
            writePWMDutyCycle(a, 'D5', .2);
            writeDigitalPin(a, 'D12', 0);

            writePWMDutyCycle(a, 'D6', .2);
            writeDigitalPin(a, 'D7', 0);

k = k+ 1;

writePWMDutyCycle(a, 'D5', 0);
writePWMDutyCycle(a, 'D6', 0);

end
end
```

Save this script under the name "pureforward" then execute the function by writing the function name "pureforward(a)" in the MATLAB command window. Check that the wheels are travelling in the direction you desire.

Task to consider:
1.Manipulate the digitalWritePin function by entering in a 1 instead of a 0. The wheels will now turn backwards.

2.Change the k value and see what happens? What effect occurs if we change the code to (k<5)? Experiment with all these situations to gain a deeper understanding of the code relative to the function of the motors.

Let's write a new function. Create another script in the MATLAB scripts editor and save this following file under then name "pureforward2." Paste the code below into this script :

"pureforward2"

```
function y = pureforward2(a, x)
%this is going forward if x is 0 and reverse if x is 1

k = 0;
while (k < 2)
            writePWMDutyCycle(a, 'D5', .2);
            writeDigitalPin(a, 'D12', x);

            writePWMDutyCycle(a, 'D6', .2);
            writeDigitalPin(a, 'D7', x);

pause(0.5)
```

```
writePWMDutyCycle(a, 'D5', 0);
writePWMDutyCycle(a, 'D6', 0);

end

end
```

"Backward"

```
function k = backward(a)

k = 0;
while (k < 2)
            writePWMDutyCycle(a, 'D5', .2);
            writeDigitalPin(a, 'D12', 1);

            writePWMDutyCycle(a, 'D6', .2);
            writeDigitalPin(a, 'D7', 1);

k = k+ 1;


writePWMDutyCycle(a, 'D5', 0);
writePWMDutyCycle(a, 'D6', 0);

end

end
```

"Right"

```
function k = right(a)

k = 0;
while (k < 2)
            writePWMDutyCycle(a, 'D5', .2);
            writeDigitalPin(a, 'D12', 1);

            writePWMDutyCycle(a, 'D6', .2);
            writeDigitalPin(a, 'D7', 0);

k = k+ 1;
```

```
x = 2;

while (x>1)
    prompt = 'which direction to go? w for forward: ';
    answer = input(prompt,'s');
        if answer == 'w'
            forward(a);
            answer = input(prompt,'s');

        end

        if answer == 's'
            backward(a);
        end

        if answer == 'a'
            right(a);
        end

         if answer == 'd'
            left(a);
         end

        if answer == 'q'
            break;
        end

end
```

Taking it Further:

Reference: Lesson 2 Unit 1

Previously we took user input to control the robot. Let's move on to figuring out how to do this with a light sensor:

The robot photoresistor (light sensor) is on pin 'A5'. Type this into the MATLAB command window:

```
readVoltage(a, 'A5')
```

```
plot(data);
xlabel('Time');
ylabel('Sensor Value');
grid on;
assignin('base', 'maxVal', max(data));
assignin('base', 'serialData', data);
disp(['Max Value is:', num2str(max(data))]);
```

From your graph, identify a baseline line follower sensor value you want to trigger the robot to go on and off. For me it was the following. For you the values may very well be different:

LightSensorBot

```
button = readDigitalPin(a,'D4');

while (button>0)
    button = readDigitalPin(a,'D4');
    reading = readVoltage(a,'A5');

if reading < 1

    forward(a);

else
     backward(a)
end
end
```

Taking It Further Tasks To Consider:

1. Create a program that controls the robot wheels to move when a button is pressed
2. A light seeking robot which follows your hand around; this will be dependent upon a shadow being cast over the photoresistor sensor.

Introductory

The MATLAB command window allows us to experiment with input and output pins on the miniQ robot. However, through the MATLAB command window we cannot deploy our code to hardware and this is important if we want our robot to function without being connected to a computer. Therefore, we must use another standalone product, Simulink, for this.

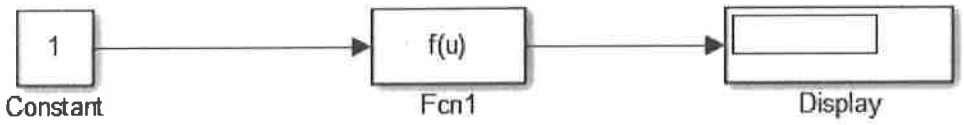
Simulink provides a graphical programming interface and is an industry standard in the fields of engineering and the sciences. Oftentime it is easier to represent code in Simulink than in the form of MATLAB scripts. This is because images tend to resonate more with people as a communicative tool. In this unit, we will explore combining the skills we have learned with regards to formatting MATLAB code but we will connect it to the graphical, Simulink environment.

Sequence:

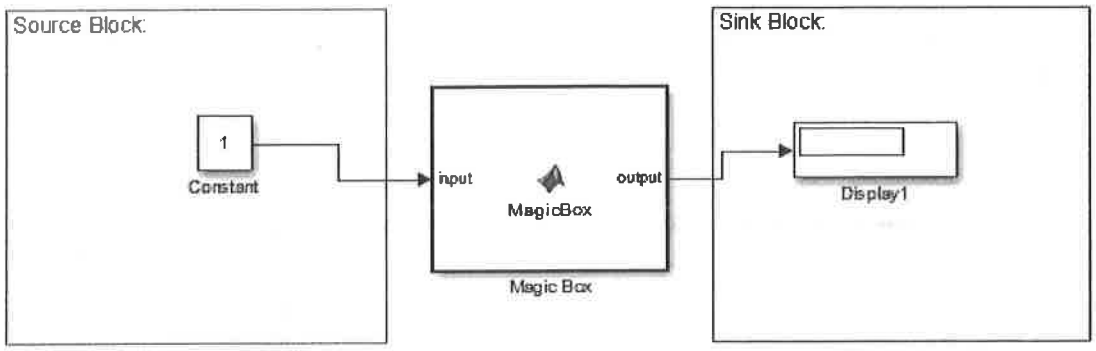
1) Open Simulink to create the model in the picture below
2) Open the Simulink Library Browser
3) The "Constant" block is in the 'Commonly Used blocks' tab
4) The "Display" block is in the Sinks tab and
5) The "Function" block in the "User Defined Functions" tab

:

6) "Addition Machine"
Example Model: AdditionMachine.slx



7) Double clicking into the function block ("Fcn1" in the picture above) and you should rewrite the mathematical equation so it is the following:



The source blocks are within the green region and the sink blocks are highlighted in red.

To build the model:

1. Begin by finding the constant blocks in the "Commonly Used Blocks" tab in the Simulink blockbuilder. Drag the following blocks into your Simulink platform to begin developing your first model.

2. Specifically, drag in one constant block, one MATLAB function block, and one Display block.

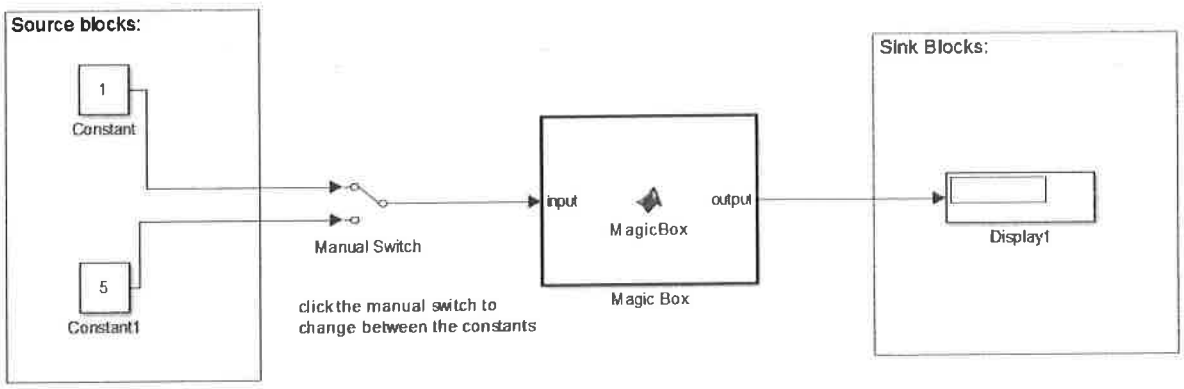
3. Double click into the constant and change the number value it holds. For this example though, set the constant block to the value 1.

4. Double click the **MATLAB function block found in the "User Defined Function"** tab and drag this into the model and change the name at the bottom of the box to "Magic Box" as in the picture above.

5. Double click into the "Magic Box" and write the script:

```
function output = MagicBox(input)
number = input;
output = number+1;
end
```

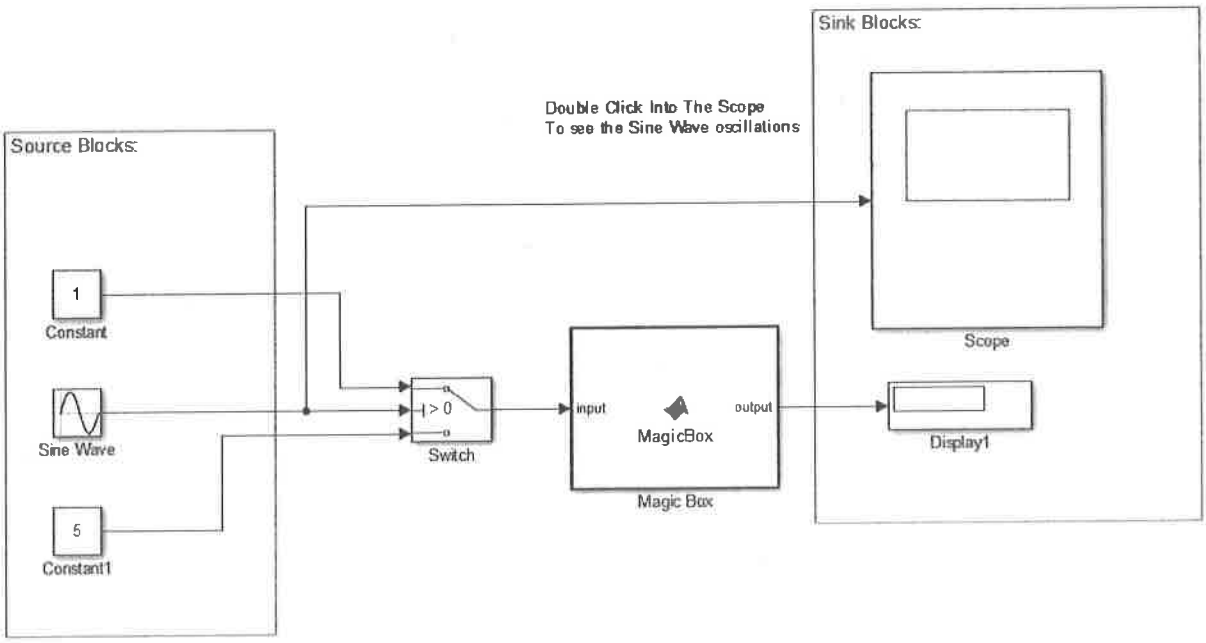
6. Go back to the Simulink block selector and in the "Sinks" tab drag in a Display Block. Press the green "play" symbol at the top of the Simulink window to see what number your Simulink display block changes to.



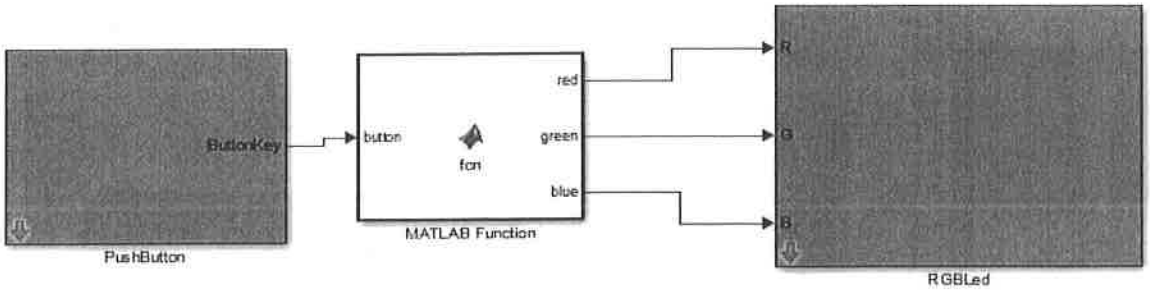
We can adapt the model above. Instead of manual control by way of mouse click. Attach a sine wave Simulink block and connect it to the middle of the switch block. Two constant values, 1 and 5, are branched to the switch block. Observe the values in the display block and why these are the particular numbers displayed. Click into the scope to further investigate.

Also, consider clicking into the switch to investigate or the "scope" box which will give you a visual of the actual sine wave itself.

Example Model: MagicBox3.slx



Intermediate



1. Double click into the MATLAB function block and enter the code below as a script.

Editor - Block: color/MATLAB Function

MATLAB Function

```
1     function [red,green,blue] = fcn(button)
2 -   if button>1
3 -       red=200;
4 -       green=0;
5 -       blue=0;
6     else
7 -       red=0;
8 -       green=0;
9 -       blue=0;
10    end
11
12
```

2. Our objective is to get a red light to turn on when the button is pressed otherwise, there should be no light shown on the MiniQ:

OPTIONAL Exercise-Button With Lights "Switch Style":
Example Model: ButtonWithLightsSwitchStyle.slx

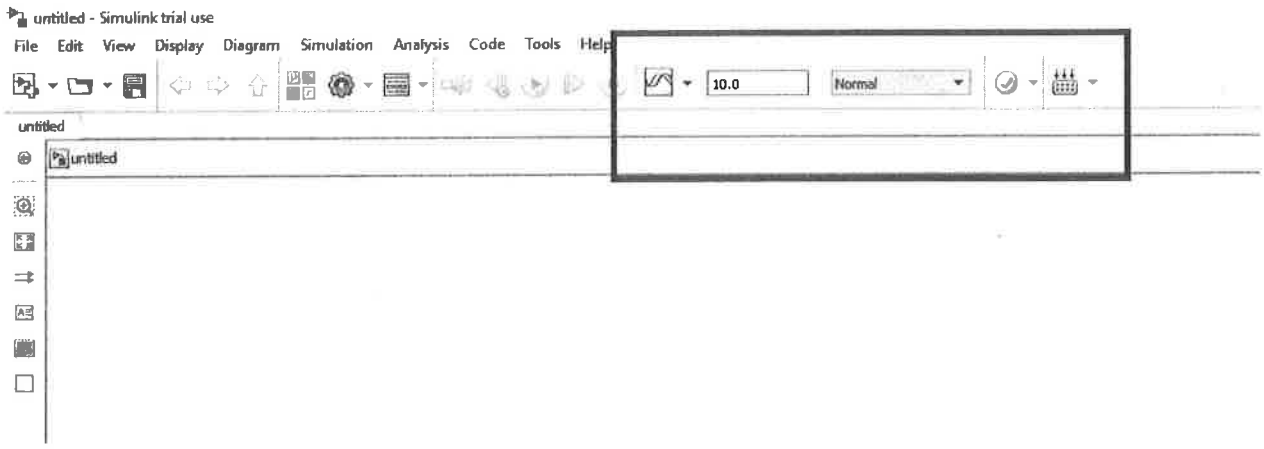
Here we have another example of a red light being light up depending on a button press. This time however, we are not relying on the construction of a function to achieve the same purpose.

Introductory

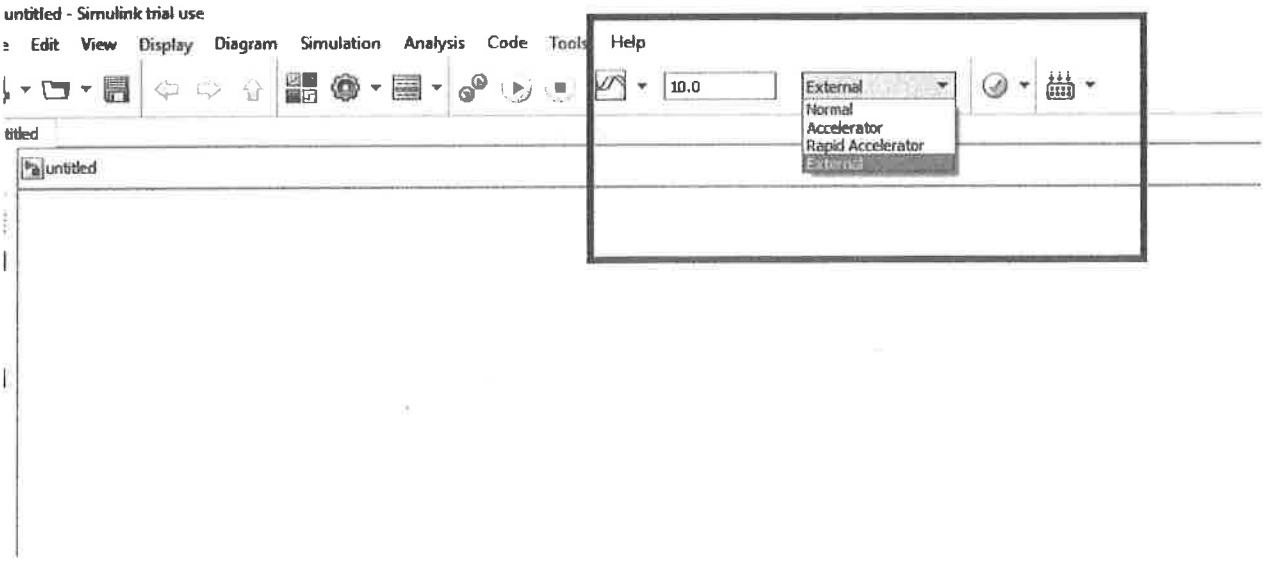
Setting Up External Mode:

External mode allows us to run simulations continuously and view features of our model, such as "display blocks" change dynamically. This can be especially useful when testing the function of sensors on our miniQ robot.

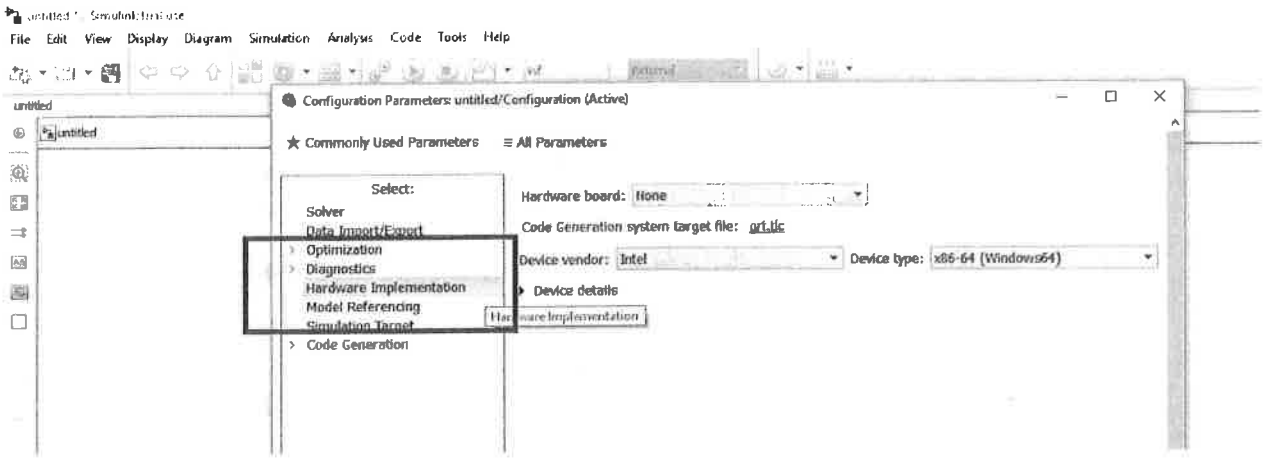
1.The red box highlights the region at the top of the Simulink interface where you can change the configuration parameters to external mode



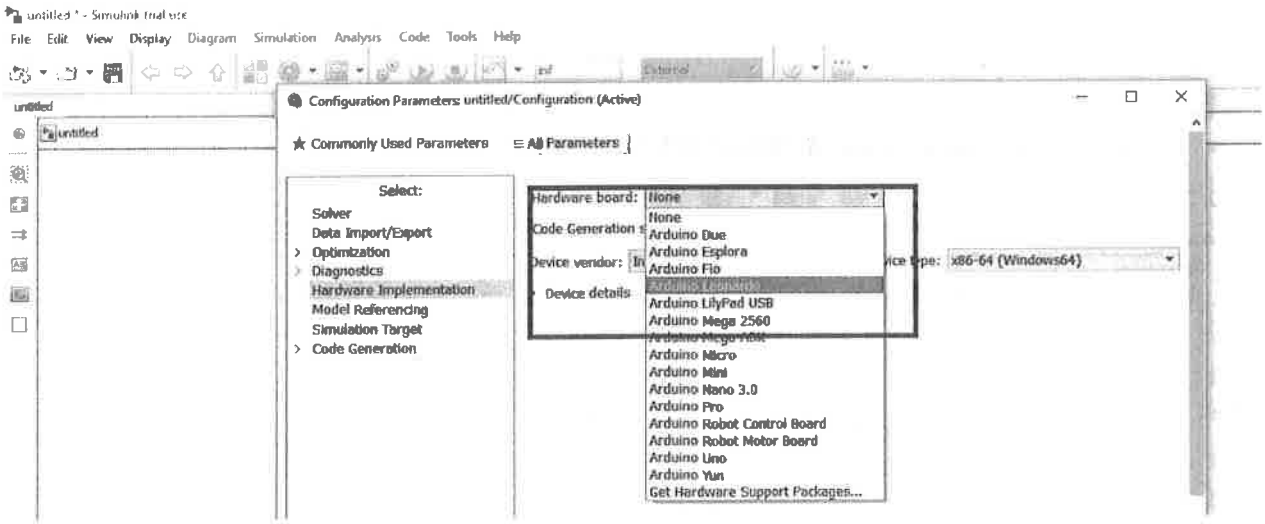
2. Select the drop down menu tab and select the option "External"



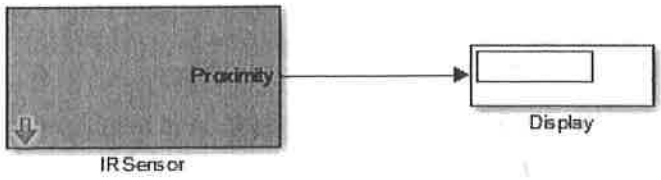
3.In the simulation time frame change the value "10.0" to "inf" as in the picture below. This will allow the simulation target time to run for an infinite period rather than just 10 seconds.



6. You will then need to select the hardware board being used. For our miniQ we are going to select the Arduino Leonardo option as seen in the picture below.

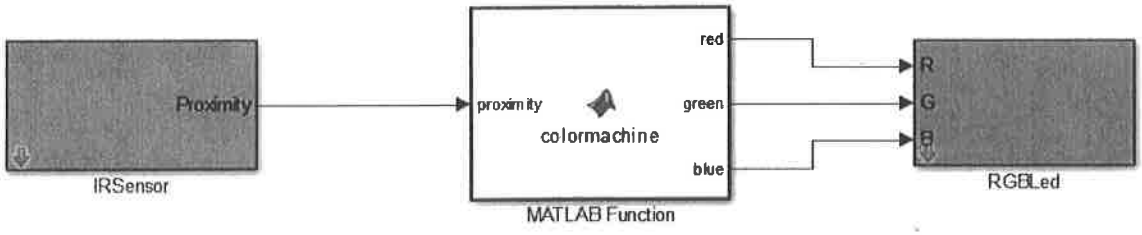


7. Make sure the right COM port is listed, you can have it set to “Automatically” but if you are having trouble deploying a model to hardware you will need to switch this tab to “Manual” and then write in the specific COM port value.



"Color Machine"
Example Model: ColorMachine.slx

The Simulink model below takes in a proximity sensor reading and converts it into a specific combination of red, green and blue output values:



The function is structured as follows and notice the use of "if","elseif" and "else" control structures:

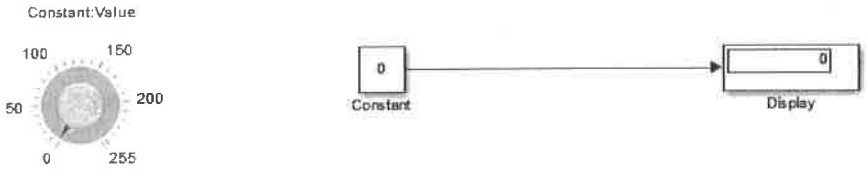
MATLAB Function*

```
1   function [red,green,blue] = colormachine(proximity)
2 -  if proximity <5
3 -      red = 0;
4 -      green=255;
5 -      blue = 0;
6 -  elseif proximity>5&&proximity<30
7 -      red = 0;
8 -      green=0;
9 -      blue = 255;
10   else
11 -     red = 255;
12 -     green=0;
13 -     blue = 0;
14
15   end
16
17
```

Optional: Dashboard Blocks and Creating User Interfaces

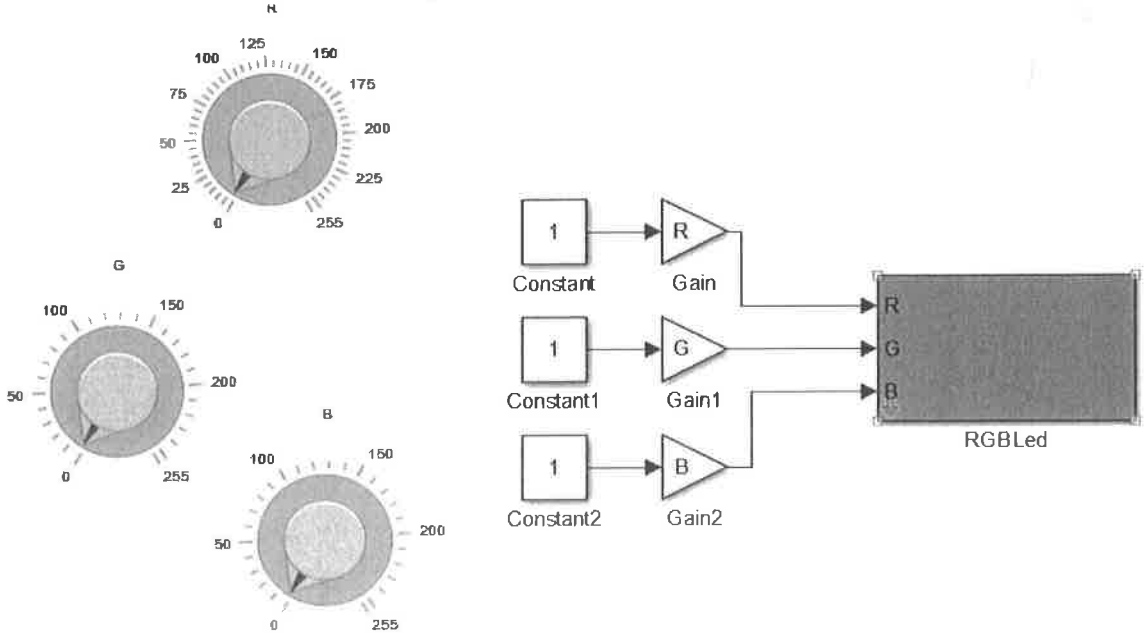
Create the following SIMULINK model. The knob block can be found in the "Dashboard" tab in the Simulink block selector. To connect the knob to the display model double click into the knob and then highlight the variable you want to control via the "dashboard" dials. In this case it is the "constant" block, so select this option when it pops up in the knob block, block parameters menu.

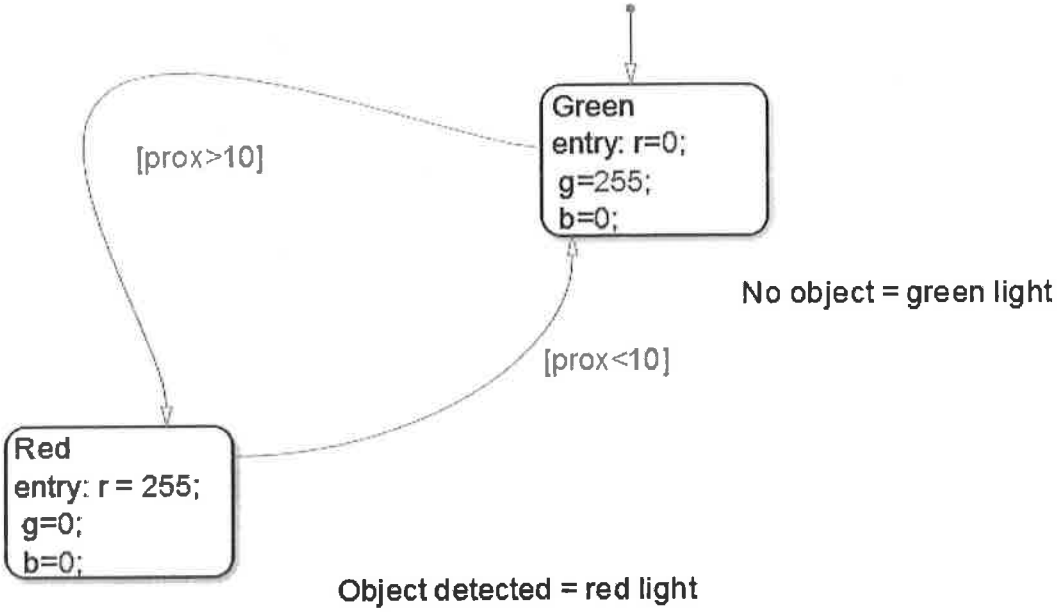
Example Model: ExternalModeDashboardSimulation.slx



Run your model in external mode and don't forget to set the simulation time frame to infinite by typing "inf" into the box next to the "external" or "normal" options. Once loaded turn the knob and you will see how the display values change, dynamically.

"RGB LED In External Mode":
Example Model: RGBLedinExternalMode.slx





Explanation: The chart flows to different combinations of red, green and blue depending upon the proximity readings from the IR sensor block. The expression of code in a flow chart rather than a function helps aid the process of representing the robot's behavior.

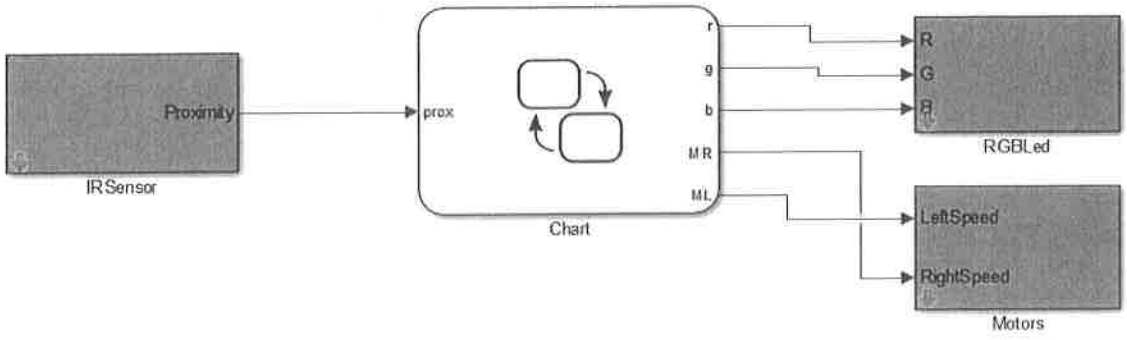
The code from unit 2, **"Color Machine With Motors"** can also be expressed in the form of a chart as seen below. Look to compare the formatting of a function with the example below:

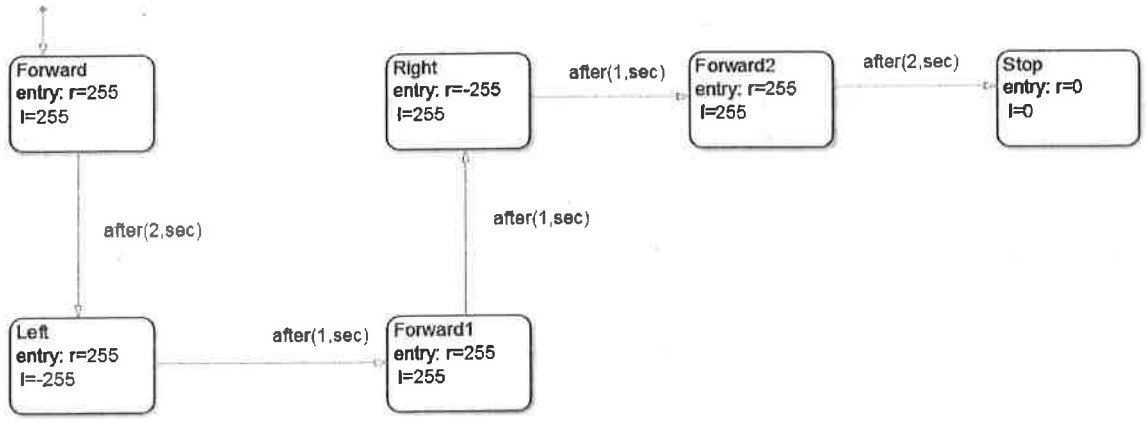
Reference Plan: Lesson 3 Unit 3

Taking This Further:

"Proximity Lights With Motors (Obstacle Avoidance Robot)"

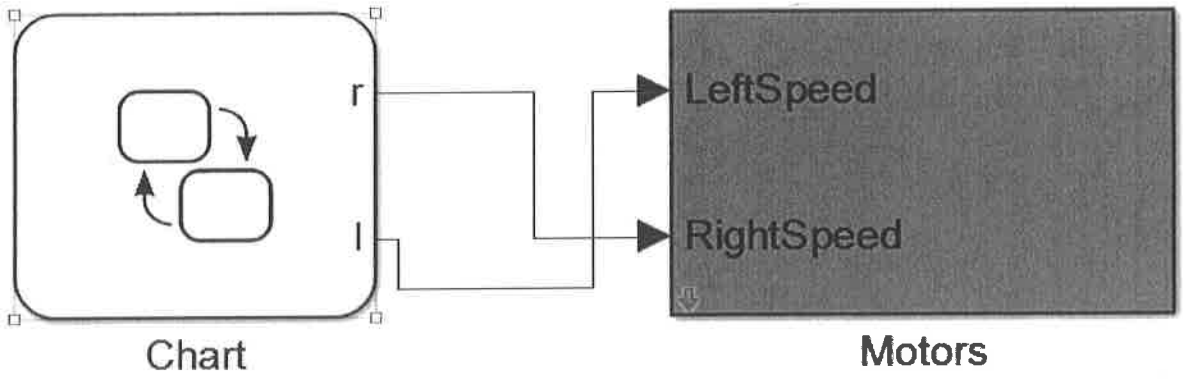
Model Index: StateFlowRGBledWithProximityMotors.slx





Open the example model **SimulinkDeadReckoningWithCharts.slx** . Pressing ctrl + d will update the inputs and outputs on the chart so you can connect arrows to the miniQ motor block.

Arrange the blocks in the pattern you expect the robot to move in order to help you organize your thoughts for this "dead reckoning" robotics exercise:



Run the model above and observe the behavior of the robot. Press the reset button on the MiniQ to run the program deployed to the hardware again. Change some of the variables in the chart to suit the pattern of movement you desire.

Intermediate:

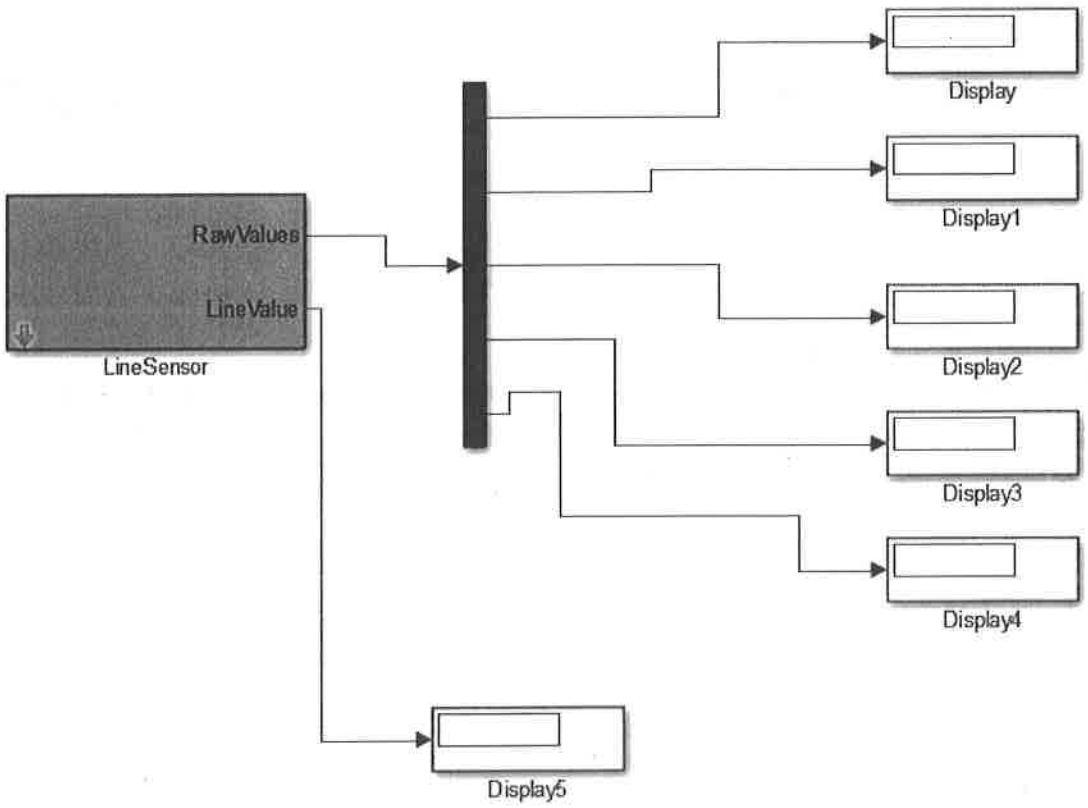
Use the dead reckoning chart to create movement for getting your robot to circle a table.
Then:

1. Create triangular movement
2. Create star movement

Unit 4: Line Following

Reference Plan: Lesson 1 Unit 4

Introductory
External Mode Line Sensor Testing:
Example Model: ExternalModeRawSensorCheck.slx



Create the following model and run on your MiniQ robot in "external mode". This will allow you to view the individual sensor readings without going into the serial monitor app. Record the values for each sensor when over black and also when over the color white for later use.

Explanation of Raw values vs. LineValue:

There are two different types of readings that the LineSensor block outputs. The RawValue readings provide a reflected light intensity value for an individual sensor ranging from an analog value of 0-1023. Using these values is helpful if we wish to utilize only one of the five sensors at the bottom of the miniQ robot. Alternatively, the LineValue gives you a reading of where the line is relative to any of the 5 sensors. For example, a line value of 3 means the line is located closest to the middle sensor, numbered "3". If the Line sensor value indicates 5 then the line is most likely positioned to the farthest outside sensor, numbered "5."

which is why it is important to take some initial readings for the sensors using external mode beforehand.

After hitting an edge the robot backs up for a certain period of time and then initiates a turn.

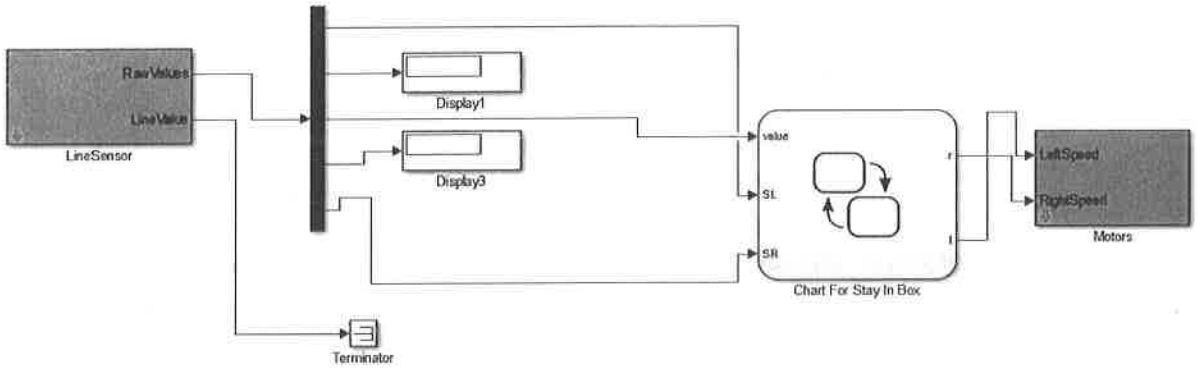
Activities to consider: create a robot which stays within the box and accelerates towards objects placed on the board. This way the robot acts as a "cleaning" robot by pushing objects outside of the squared area. This process can be timed to make the activity more game like.

Expand the model by including the proximity sensor with your mode:

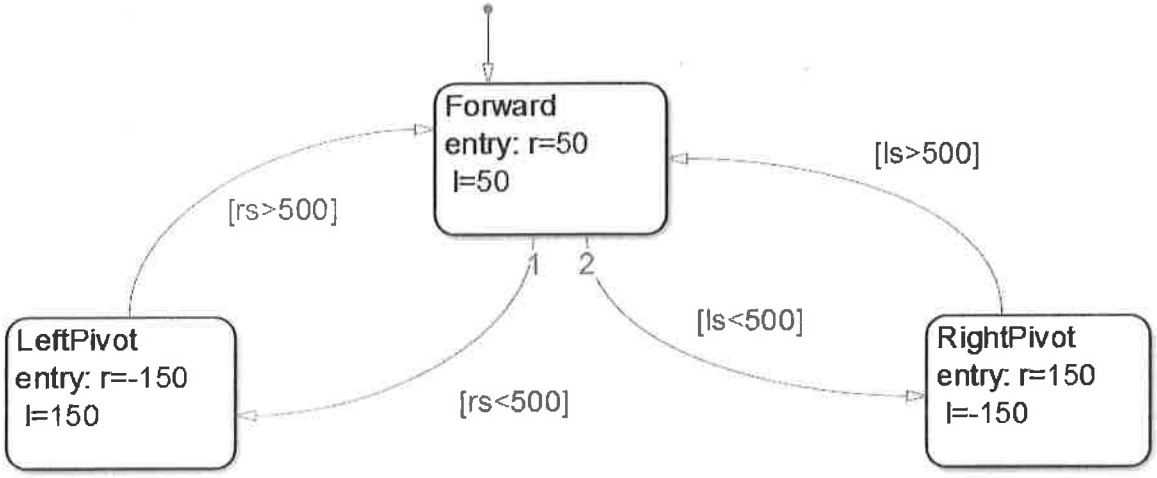
Optional: Reference- Lesson 2 Unit 4

OPTIONAL: Taking It Further: Stay In Box With Extra Sensors
Example Model: StayInBoxExtraLineSensors.slx

Using the sensors on the ends can be factored into the "Stay In Box" Simulink model as follows



In the chart, SL stands for the leftmost line sensor on the MiniQ robot while SR is the rightmost sensor.



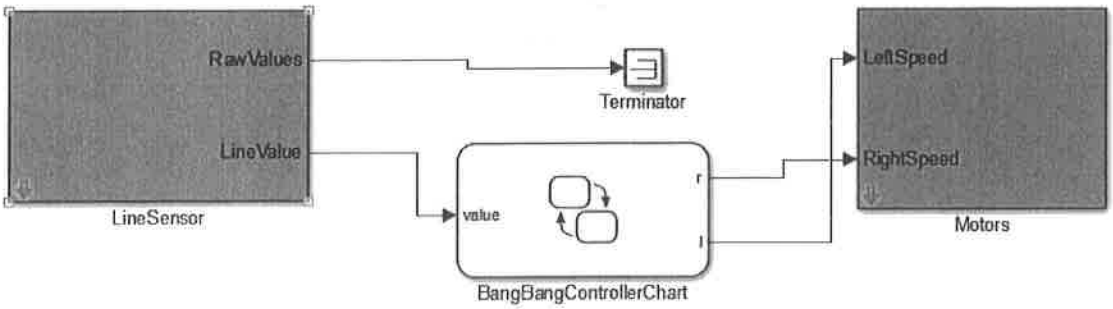
Taking it further: Experiment with different sized lines and different combinations of the inner sensors by adding those to the chart.

Reference Plan: Lesson 4 Unit 4

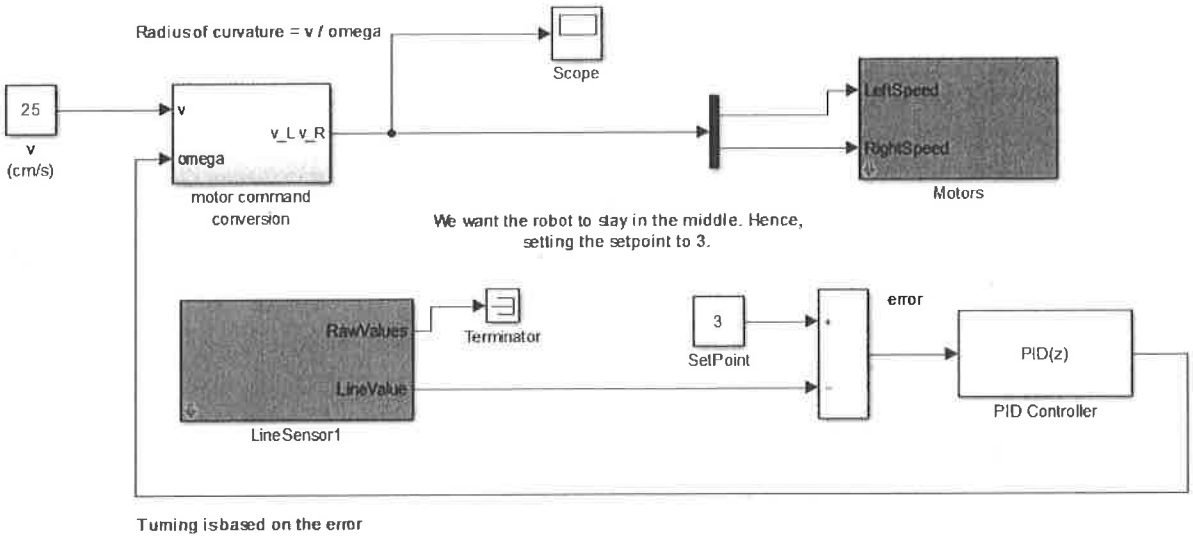
BangBang Line Following:

Example Model: BangBangLineFollow.slx

The difference with this form of line following is in how we are basing values of of the "Line Value" output on the Line Sensor Simulink block. This value factors in the position of a line relative to all 5 of the line sensors on the MiniQ:



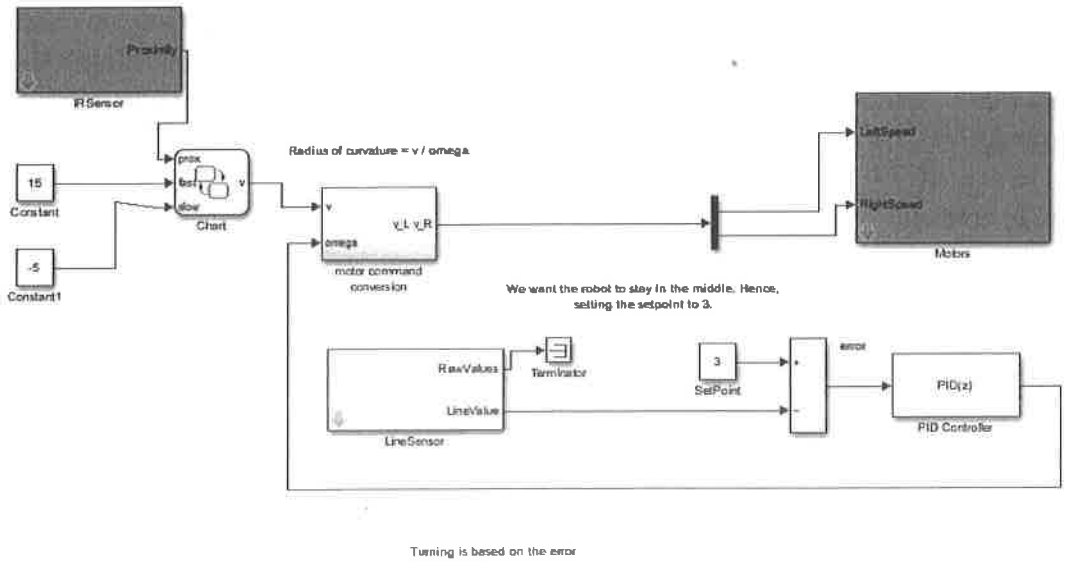
The following model was created to test a "bang-bang" version of line following behavior. The name comes from how the robot dips inwards and outwards towards a line depending upon the light readings taken from the array of five sensors at the bottom of the miniQ robot.



Run the model and notice how the robot runs smoother. A Proportional Integral and Derivative (PID) controller is used to adjust the speed relative to an optional position "set point" of a line being over sensor 3. This will ultimately result in a smoother line following behavior for the miniQ robot because there are less oscillations back and forth across the line.

This model can be extended so the proximity sensor helps the robot figure out if an object is in front of it or not.

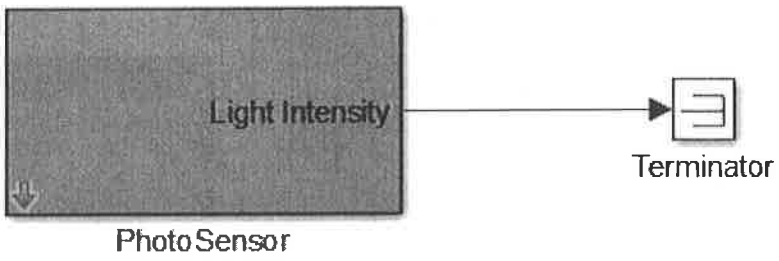
Taking It Further: Line Following With Proximity Stop:
Example Model: LineFollowingWithProxCompletV1.slx



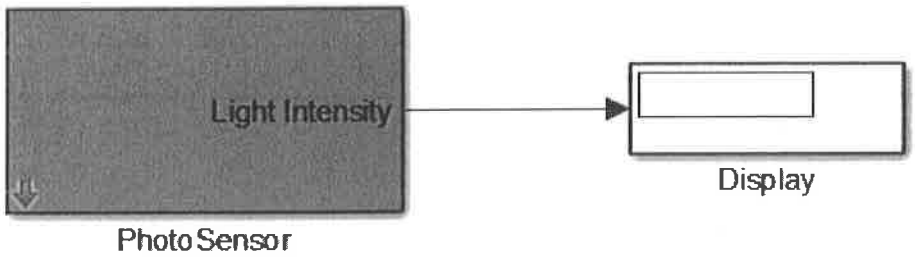
Unit 5:Light Sensing

Introductory:

Testing PhotoSensor Values:



Running the following code allows you to check the values in the serial monitor. Notice the difference in values depending upon if your hand is the the right or left from of the robot. Record these for use later on.



If you run the above Simulink model in external mode you can take readings from the light sensor in the above way. You'll find the "Display" blocks in the "Sinks" tab of the Simulink block builder.

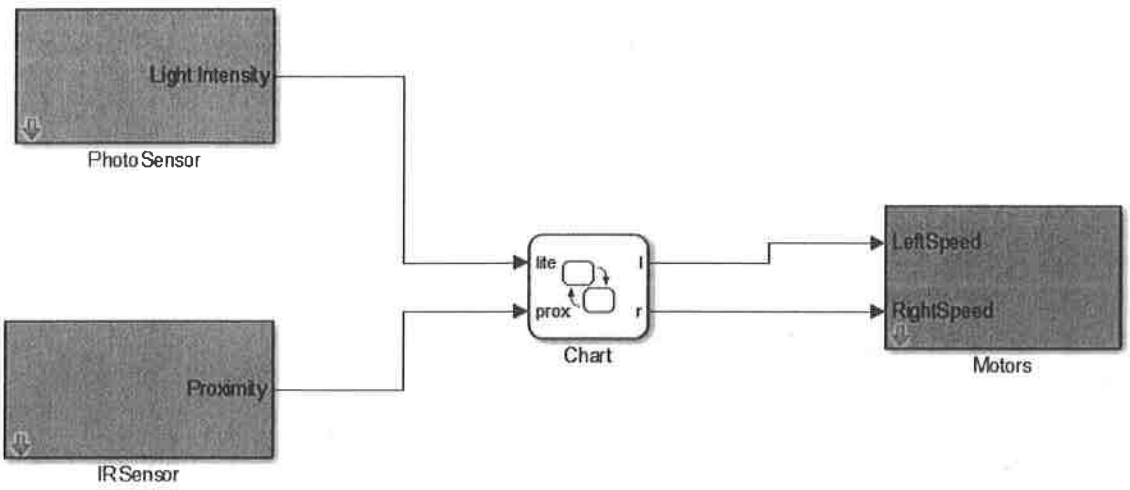
Intermediate: **"Light Turner"**
Index: LightTurner.slx

Option: Have the robot turn in the direction where there is less light and also have the RGB light turn a different color between each direction it moves. This will involve having to create more outputs in your Simulink chart to control the RBG LED blocks.

Taking It Further:

"Darkness Follow Bot"

Index: DarknessFollowBot.slx



The objective here is to create a robot which follows a hand put up close to it. The combination of close proximity to a hand casts a shadow. The light sensor can be used to figure out if a hand is left or right of the front of the robot depending on a lack of light. The proximity sensor is factored in so the robot following behavior is only present when a hand is close enough.

The Darkness Follow Bot Chart: