

Autonomous Vehicle Control, Modeling, and Design

**An introduction to the design
process used to realize the
complex systems of the future.**

Marc E Herniter

Table of Contents

I.	Introduction to Simulink and the KL25Z	13
A.	Connecting the KL25Z Board	13
B.	Blinking Light	15
C.	Flip-Flop LED Model.....	23
D.	Questions	24
E.	Exercises	24
II.	KL25Z On Board Sensors.....	26
A.	Touch Sensor Interface (TSI)	26
B.	Accelerometer.....	29
C.	Questions	35
D.	Exercises.....	35
III.	Pushbuttons, Knobs, Dip Switches, and LEDs	37
A.	Pushbuttons and LEDs	37
B.	Potentiometers and Knobs	41
C.	Dip Switches	44
D.	Questions	47
E.	Exercises	48
IV.	Servo Motor and Drive Motors	50
A.	Servo Motor	51
B.	Servo Motor Zero	57
C.	Drive Motors	58
D.	Motor Drive Example	62
E.	Questions.....	63
F.	Exercises	64
V.	Introduction to MATLAB.....	65
A.	Introduction	65
B.	MATLAB Command Window.....	68
C.	Defining Variables.	70
D.	Saving and Clearing the Workspace.....	73
E.	Questions.....	77
F.	Exercises	77
VI.	Arrays	78
A.	Row Vectors	78
B.	Column Vectors	79

C.	Functions to Create Arrays	80
1.	The Colon Operator.....	80
2.	Ones and Zeros.....	81
D.	Questions	82
E.	Exercises	82
VII.	Working with Arrays	84
A.	Built-In MATLAB Functions	84
B.	Scalar Calculations with Vectors	85
C.	Vector Calculations	86
D.	Plotting	88
E.	Questions.....	89
F.	Exercises	89
VIII.	Script Files and Control Flow	91
A.	Script Files	91
B.	The If Statement	97
C.	For Loops.....	102
D.	Questions.....	105
E.	Exercises	106
IX.	Assembling the Vehicle Steering.....	107
A.	Zeroing the Servo Motor	107
B.	Assembling the Steering	107
C.	Assembling the Tie Rods.....	114
X.	Mounting the KL25Z	118
A.	Mounting the KL25Z and TFC Shield	118
B.	Testing.....	126
1.	Servo Motor and Steering.....	126
2.	Right Drive Motor	128
3.	Left Drive Motor	128
XI.	Assembling and Mounting the Camera	130
A.	Assembling the Camera.....	130
B.	Assembling the Camera Mount	131
C.	Mounting the Camera.....	134
D.	Camera Testing.....	136
XII.	Processing Data from the Linescan Camera	143
A.	Camera Data	143
B.	Processing the Camera Data	145

C.	Plotting the Camera Data	153
D.	Characterizing the Linescan Camera Data	156
1.	Effect of Camera Focus.....	157
2.	Camera Angle	158
3.	Camera Height.....	159
4.	The Best of All Settings	160
5.	Index versus Vehicle Position.....	160
XIII.	Autonomous Steering	165
A.	Turning the Wheels Automatically.....	165
B.	Fixed Gain Steering Amplifier.....	168
C.	Adjustable Gain Steering Amplifier	170
D.	Subsystems.....	172
E.	Push Testing of the Vehicle	176
XIV.	Motoring.....	177
A.	Constant Speed Motoring and Motor Enable.....	177
B.	Controller Tuning	180
1.	Testing Procedure	181
2.	Motor Speed Based on Camera Angle	181
3.	Motor Speed versus Steering Gain.....	186
C.	Adjustable Speed Motoring.....	189
D.	Optimizing Vehicle Speed with Steering Gain and Camera Angle	190
E.	Steering Based Speed Control.....	191
F.	Optimizing Vehicle Speed – Steering Angle Based Vehicle Speed	197
XV.	Vehicle Improvements.....	199
A.	Metrics, Testing, and Evaluation	199
B.	Basic Adjustments	200
C.	Cleaning the Tires	200
D.	Torsion Bar Adjustment	200
E.	Compression Spring Adjustment.....	202
F.	Front-Rear Weight Distributions and Adjustment.....	204
G.	Dead Spot Steering	204
H.	Steering Angle Based Speed Table Adjustment	206
I.	Steering Offset.....	209
J.	Rear-Wheel Steering	209
K.	Camera Index Memory.....	211

L.	Crash Detection	215
M.	Battery Monitor	221
XVI.	Answers	223
A.	Answers to Questions	223
1.	Lesson I.....	223
2.	Lesson II.....	223
3.	Lesson III.....	225
4.	Lesson IV	226
5.	Lesson V	228
6.	Lesson VI	229
7.	Lesson VII	230
8.	Lesson VIII	230
9.	Lesson IX	233
10.	Lesson X.....	233
11.	Lesson XI.....	233
12.	Lesson XII.....	233
B.	Answers to Exercises.....	234
1.	Lesson I.....	234
2.	Lesson II.....	234
3.	Lesson III.....	234
4.	Lesson IV	235
5.	Lesson V	235
6.	Lesson VI	236
7.	Lesson VII	238
8.	Lesson VIII	240
9.	Lesson IX	242
10.	Lesson X.....	242
11.	Lesson XI.....	242
12.	Lesson XII.....	242
XVII.	Index	244

List of Demonstrations

- Demo XIII-1: With the Dip Switch 4 set to zero, verify that the front wheels do not move as the vehicle is moved from one side of the track to the other. The steering wheels should always point straight..... 167
- Demo XIII-2: With the Dip Switch 4 set to one, verify that the front wheels turn as the vehicle is moved from one side of the track to the other. 167
- Demo XIII-3: With the Dip Switch 4 set to one, verify that the front wheels turn slightly to the right when the vehicle is close to the left edge of the track. Notice that the wheels turn away from the edge to which they are closest. 167
- Demo XIII-4: With the Dip Switch 4 set to one, verify that the front wheels turn slightly to the left when the vehicle is close to the right edge of the track. Notice that the wheels turn away from the edge to which they are closest. 168
- Demo XIII-5: With the Dip Switch 4 set to zero, verify that the front wheels do not move as the vehicle is moved from one side of the track to the other. The steering wheels should always point straight..... 169
- Demo XIII-6: With the Dip Switch 4 set to one, verify that the front wheels turn as the vehicle is moved from one side of the track to the other. Note that they move more than in the previous set of tests. 169
- Demo XIII-7: With the Dip Switch 4 set to one, verify that the front wheels turn to the right when the vehicle is close to the left edge of the track. Notice that the wheels turn away from the edge to which they are closest, and they turn a lot more than in the previous set of tests. 169
- Demo XIII-8: With the Dip Switch 4 set to one, verify that the front wheels turn to the left when the vehicle is close to the right edge of the track. Notice that the wheels turn away from the edge to which they are closest. 169
- Demo XIII-9: With the Dip Switch 4 set to zero, verify that the front wheels do not move as the vehicle is moved from one side of the track to the other. The steering wheels should always point straight..... 172
- Demo XIII-10: With the Dip Switch 4 set to one and potentiometer B turned all the way counter clockwise, verify that the front wheels **do not turn** as the vehicle is moved from one side of the track to the other. 172
- Demo XIII-11: With the Dip Switch 4 set to one and potentiometer B turned 25% clockwise, verify that the front wheels turn as the vehicle is moved from one side of the track to the other. 172
- Demo XIII-12: With the Dip Switch 4 set to one and potentiometer B turned 50% clockwise, verify that the front wheels turn as the vehicle is moved from one side of the track to the other. Note that they move more than in the previous test. 172
- Demo XIII-13: With the Dip Switch 4 set to one and potentiometer B turned fully clockwise, verify that the front wheels turn as the vehicle is moved from one side of the track to the other. Note that they move more than in the previous test. 172
- Demo XIII-14: Verify that the vehicle can navigate around the track **clockwise** as you push it. When adjusted properly, the vehicle should steer by itself and you should not have to manually reposition the vehicle in order

for it to stay on the track. All you should need to do is push the vehicle. You will need to adjust the steering gain in order for the vehicle to steer properly and stay on the track. You may also find that you need to change the camera angle and camera height to make the steering perform properly..... 176

Demo XIII-15: Verify that the vehicle can navigate around the track **counter** clockwise as you push it. When adjusted properly, the vehicle should steer by itself and you should not have to manually reposition the vehicle in order for it to stay on the track. All you should need to do is push the vehicle. You will need to adjust the steering gain in order for the vehicle to steer properly and stay on the track. You may also find that you need to change the camera angle and camera height to make the steering perform properly. 176

Demo XIII-16: The track is 24 inches wide. Determine values for the steering gain, camera height, and camera angle such that as you push the vehicle around the track, the vehicle never comes closer than 6 inches to either edge of the oval track. 176

Demo XIV-1: Verify that the vehicle speed is reduced as the vehicle makes a turn to the right..... 197

Demo XIV-2: Verify that the vehicle speed is reduced as the vehicle makes a turn to the left. 197

Demo XIV-3: Verify that Potentiometer A determines the maximum speed when the vehicle is moving straight. 197

Demo XV-1: When you first test this method, hold the vehicle above a straight piece of track. Turn on and enable your vehicle, both the steering and the motors. When you move the vehicle near the line on the right side, the right tire should spin more than the left tire and the front wheels should turn to the left. When you move the vehicle near the line on the left side, the left tire should spin more than the right tire and the front wheels should turn to the right. 211

Demo XV-2: When you first test this method, start at slow speed and make sure everything works. Then slowly increase the speed and see how it does. You should be able to achieve much higher speeds around the corners. 214

Demo XV-3: Test the crash detection system. First test it by shaking the vehicle. If you shake it violently enough, LED D4 should turn on and the drive wheels should stop spinning. The vehicle will not start again until you cycle the power. 221

Demo XV-4: Demo the operation of your battery monitor. Since the battery level does not change very much and it will be hard to test at different levels of the battery state of charge, we will test it at full charge and no charge. Make sure your battery is fully charged. When you turn on the power to your vehicle, LED D4 should be illuminated constantly. Next, turn off the power to your car and plug in your KL25Z to the USB port. With the power off, the battery will appear to be fully discharged and LED D4 will flash at the fastest rate. We cannot test the other blocks in the battery level indicator until we use the vehicle for a while. However, once LED D4 starts flashing, you know that you should charge the battery soon. 221

List of Exercises

Exercise I-1: Modify the Blue_LED_Flasher so that the RED LED flashes at a rate of 2 Hz (twice per second.)	24
Exercise I-2: Modify the Blue_LED_Flasher so that the Blue LED flashes at a rate of 1 Hz but the light is only on for $\frac{1}{4}$ of a second.	24
Exercise I-3: Create a single model that flashes the Blue LED at a 1 Hz rate with 50% duty cycle, the Red LED at a 0.5 Hz rate with 50% duty cycle, and the Green LED at a 0.25 Hz rate with 50% duty cycle. Remember that Hz is frequency (f) and the period is 1 divided by the frequency, $F = 1T$. You can use more than one pulse generator to solve this problem. Also note that when more than one LED is on, the colors will be combined to form new colors.	25
Exercise II-1: What other logical operators are available with the Logical Operator block?	35
Exercise II-2: In what other Simulink library is the Logical Operator found? (In addition to the Autonomous Vehicle Competition library.)	36
Exercise II-3: List two blocks in the Simulink / Commonly Used Blocks library that are not found in the Autonomous Vehicle Competition library. Specify what these blocks do?	36
Exercise II-4: List two blocks in the Simulink / Math Operations library that are not found in the Autonomous Vehicle Competition library. Specify what these blocks do?	36
Exercise III-1: Create a model that flashes LED D3 at 1 Hz, LED D4 at 2 Hz, LED D5 at 4 Hz, and LED D6 at 8 Hz. Make sure that you set the fixed time step to auto or to 0.005.	48
Exercise III-2: Use the touch slider to create the following model:	48
Exercise III-3: Create a model that does the following with a potentiometer.....	48
Exercise III-4: Create a model that does the following with a potentiometer.....	48
Exercise IV-1: Using a Sine Wave block, create a model that rotates the Servo Motor angle back and forth between -50 and +50 degrees at a frequency of 1 Hz. Note that frequency in rad/sec is $2\pi F$, where F is the frequency in Hz. (So for a 1 HZ frequency, we need a radial frequency of 2π rad/sec.) The Sine Wave block is in the Simulink / Sources library. Set the sample time to -1 and the fixed-step time to 0.01.	64
Exercise IV-2: Using a Pulsed Generator block, create a model that rotates the Servo Motor angle back and forth between -30 and +30 degrees at a frequency of 1 Hz. Note that you can change a sum block to a difference block by changing the signs. (Double-click on the sum block and see the description.).....	64
Exercise IV-3: Using a Sine Wave block, create a model that rotates both DC Motor speeds between -0.5 and +0.5 at a frequency of 0.25 Hz. Note that frequency in rad/sec is $2\pi F$, where F is the frequency in Hz. (So for a 0.25 HZ frequency, we need a radial frequency of 0.5π rad/sec.) The Sine Wave block is in the Simulink / Sources library.	64

Exercise IV-4: Create a model that uses a single potentiometer to control the speed of both motors. When the potentiometer value is between 0 and 1, the right motor speed varies between 0 and 50% forward. (0 to 0.5 for the DC Motor block input.) At this time the left motor is at zero speed. When the Potentiometer value is between 0 and -1, the left motor speed varies between 0 and 50% forward. (0 to 0.5 for the DC Motor block input.) At this time the right motor is at zero speed. (You should be able to do the logic with product blocks and compare to constant blocks, or with Switch blocks.)	64
Exercise V-1: By hand, calculate the following values. Then check your calculations with MATLAB.....	77
Exercise V-2: What are the values of the following MATLAB commands:	77
Exercise V-3: What does NaN stand for? (Use the MATLAB help facility to answer this question.)	77
Exercise VI-1: What are the numerical values of A(1), B(3), X(7)?.....	83
Exercise VI-2: What are the numerical values of C?	83
Exercise VI-3: What are the numerical values of D?	83
Exercise VI-4: What is A+X?.....	83
Exercise VI-5: What are the values generated by 5:-1:1?	83
Exercise VI-6: What are the values of A(3:6)?	83
Exercise VI-7: What are the values of A(6:-1:3)?	83
Exercise VI-8: What are the values generated by 1:0.1:3?	83
Exercise VI-9: What are the values generated by .9:1:1.5?	83
Exercise VI-10: What are the values generated by 1:5:2?	83
Exercise VI-11: What are the values generated by 5:1:1?	83
Exercise VI-12: What is A+X(1:6)? Why does this work and not A+X?	83
Exercise VI-13: What is A(1:3)+A(4:6)?	83
Exercise VI-14: What is Q(5:-1:1)	83
Exercise VI-15: What is Q(1:5)+Q(5:-1:1)	83
Exercise VI-16: Give the command to create a column vector with the values 1, 3, 5, 7, 9.	83
Exercise VI-17: Give the command to create a row vector with the values -1, -3, -5, -7, -9.....	83
Exercise VI-18: Give the command to create a row vector with the values -1, -3, -5, -7, -9.....	83
Exercise VI-19: Give the command to find the last two values of variable A.	83

Exercise VI-20: Give the command to create a new vector G, where G has the same values as B but in reverse order.....	83
Exercise VI-21: Give the command to create the column vector with the values -0.1, 0.1, 0.3, 0.5, 0.7.	83
Exercise VII-1: What is the sum of all of the odd numbers between 1 and 100?	89
Exercise VII-2: What is the average of all of the numbers between 1 and 1000?	89
Exercise VII-3: What is the average of all of the numbers between 1 and 999?	89
Exercise VII-4: Three ways to find the sum of the numbers between 1 and 100 are shown below. Verify that the three methods yield the same answer and then explain why they are the same.	89
Exercise VII-5: If $a=[1\ 2\ 3\ 4\ 5\ 4\ 3\ 2\ 1]$, what is the numerical value of mean(a(4:6)) . Verify by hand and with MATLAB. Explain how this command works.....	89
Exercise VII-6: Continuing with the previous question, what is $3 * \text{mean}(a(4:6))/4$?	89
Exercise VII-7: Continuing with the previous question, what is $0.75 * \text{mean}(a(4:6))$? Do you think this method is more or less efficient than the method used in the previous exercise?	89
Exercise VII-8: $A1+B1$	90
Exercise VII-9: $A1-B1$	90
Exercise VII-10: $A1.*B1$	90
Exercise VII-11: $A1./B1$	90
Exercise VII-12: $(A1-B1)./B1$	90
Exercise VII-13: $(A1+B1).*B1$	90
Exercise VII-14: Generate a plot of the parabola $y=-x^2-2x+2$. Use 200 points and let x vary from -7 to +5. Label the title, x-axis, and y-axis of the plot. Display the grid on the plot.	90
Exercise VII-15: Generate a plot of the function $y=x^3-x^2-2x+2$. Use 200 points and let x vary from -7 to +7. Label the title, x-axis, and y-axis of the plot. Display the grid on the plot.....	90
Exercise VII-16: Generate a plot of the function $y= 2x+2$. Use 200 points and let x vary from -3 to +3. Label the title, x-axis, and y-axis of the plot. Display the grid on the plot.	90
Exercise VIII-1: Create a script that tests if a variable named y is equal to 17. If it is equal to 17, add 2 to it.	106
Exercise VIII-2: Create a script that tests if a variable named y is equal to 17. If it is equal to 17, add 2 to it. If it is not equal to 17, subtract 3 from it.	106

x

Exercise VIII-3: Create a script that sets x to a number between 1 and 20. If x is prime, output the text string “The number is prime.” If the number is not prime, output the message, “The number is not prime.” Solve this problem using an if and fprintf statements.	106
Exercise VIII-4: Create a script that sets x to a number between 1 and 20. If x is prime, output the text string “The number is prime.” If the number is not prime, output the message, “The number is not print.” Solve this problem using an array, for loops, if statements, and fprintf statements.....	106
Exercise VIII-5: Define x as the row vector [1, 3, 5, 8, 88, 72, 6]. Create a script that defines this array, and then displays the values of the elements in reverse order.....	106
Exercise VIII-6: Define x as the row vector [1, 3, 5, 8, 88, 72, 6]. Create a script that defines this array, then displays the values of the elements in reverse order, and then displays the values of the elements in their original order.	106
Exercise VIII-7: Define x as the row vector [1, 3, 5, 8, 88, 72, 6]. Use a for loop to display the values of 1st, 3rd, 5th, and 7th elements.	106
Exercise VIII-8: Define x as the row vector [1, 3, 5, 8, 88, 72, 6]. Use a for loop to display the values of 7th, 5th, 3rd, and 1st elements, in that order.	106
Exercise VIII-9: The MATLAB mod function finds the remainder after dividing two numbers. For example, mod(5,2) is the remainder when we divide 5 by 2. Note that mod(5,2)=1 and mod(4,2)=0. We can use this function to see if a positive integer is odd or even. Use this function to determine if a number x is odd or even. Use an IF statement to print out an appropriate text message stating whether the number is odd or even. ...	106
Exercise XV-1: Create a list of metrics that you wish to use to evaluate the Performance of your vehicle. This should be a bulleted list. Specify how you would test and measure each metric.	200
Exercise XV-2: Create a table in which to record the performance metrics listed in Exercise XV-1	200
Exercise XV-3: Measure the performance of your baseline vehicle. Record the information in your table. Keep the table in a safe location such as a notebook or file folder. You will need to compare the baseline results to all of the ideas you attempt in this section.....	200
Exercise XV-4: Measure the performance of your vehicle before you clean the tires.....	200
Exercise XV-5: Come up with a procedure for cleaning your vehicle tires. Document this procedure so that it will be repeated whenever the vehicle is used for testing or competition. Your procedure should include the materials used, any special methods that you employ, and how often the tires should be cleaned.....	200
Exercise XV-6: After cleaning the tires, measure the vehicle's performance and compare to that measured in Exercise XV-4	200
Exercise XV-7: Adjust the torsion bar to minimum stiffness. This means that you can twist it with very little force. Measure your vehicle's performance.	202
Exercise XV-8: Adjust the torsion bar to maximum stiffness. Measure your vehicle's performance and compare to Exercise XV-7	202

Exercise XV-9: Adjust the torsion for maximum performance. This may take several adjustments and several measurements. Make sure that you record a table of data for every adjustment that you make. Also make sure that you have a way of knowing the position of the adjustment screws..... 202

Exercise XV-10: Adjust the compression spring to minimum compression. (It will be completely uncompressed with no hope of actually ever being compressed. Thus, it will do nothing.) Measure your vehicle’s performance. 204

Exercise XV-11: Adjust the compression spring to maximum compression. Measure your vehicle’s performance and compare to **Exercise XV-10**. 204

Exercise XV-12: Adjust the compression spring for maximum performance. This may take several adjustments and several measurements. Make sure that you record a table of data for every adjustment that you make. Also make sure that you have a way of knowing the position of the adjustment post. 204

Exercise XV-13: Before adding weights, measure your vehicle’s performance to obtain a baseline. 204

Exercise XV-14: Add weights to specific locations on your vehicle. Document where you place the weights, why you placed them in their specific locations, and what you expect to happen. Measure your vehicle’s performance and compare to the baseline and any previous weight trials you may have made. It may take several trials to see an effect or to achieve the desired result. Make separate trials and document each trial separately so that you can compare all of the trials and have a history of all of the variations that you made. 204

Exercise XV-15: Before adding the dead spot lookup table, measure your vehicle’s performance to obtain a baseline performance. 206

Exercise XV-16: Add the dead spot lookup table to your control model. Document your changes and what you expect to happen. Measure your vehicle’s performance and compare to the baseline and any other trials that you have made. It may take several trials with different variations in the dead spot to achieve the desired effect. Make separate trials and document each trial separately so that you can compare all of the trials and have a history of all of the variations that you made. 206

Exercise XV-17: Before implementing the steering angle based vehicle speed lookup table, measure your vehicle’s performance to obtain a baseline. 208

Exercise XV-18: Modify the steering angle-vehicle speed lookup table in your control model. Document your changes and what you expect to happen. Measure your vehicle’s performance and compare to the baseline and any other trials that you have made. It may take several trials with different variations in the table to achieve the desired effect. Make separate trials and document each trial separately so that you can compare all of the trials and have a history of all of the variations that you have made. 208

Exercise XV-19: Determine the value of the steering offset constant necessary to make your wheels point straight when the steering is disabled and the vehicle is powered. 209

Exercise XV-20: Before implementing rear-wheel steering, measure your vehicle’s performance to obtain a baseline. 211

Exercise XV-21: Implement rear-wheel steering. Document your changes and what you expect to happen. Measure your vehicle’s performance and compare to the baseline and any other trials that you have made. It

may take several trials with different variations in the tables to achieve the desired effect. Make separate trials and document each trial separately so that you can compare all of the trials and have a history of all of the variations that you have made. 211

Exercise XV-22: Before implementing camera index memory, measure your vehicle’s performance to obtain a baseline. 214

Exercise XV-23: Test this method for different values of the switch threshold. (We set it to 10 in the above example.) Lowering the value means that it will throw out many more values of the index that would otherwise be valid. Making it higher means that more signals that would otherwise be invalid are used. (Lowering the threshold means that it holds more. Raising the threshold makes the system act more like it did before we added the memory.) 214

Exercise XV-24: Determine an appropriate value for the crash detection threshold. It should not disable the vehicle for bumps and high speed curves. It should stop the vehicle after it crashes into something. 221

Exercise XV-25: Discuss with your teacher how this set of blocks works. Include the signal output of the **Battery Read** block, the voltages this block reads, the thresholds of the switches, and the signal routing provided by the switches. What is the frequency of each of the pulse generators? (Remember that frequency is the reciprocal of the period.) 222

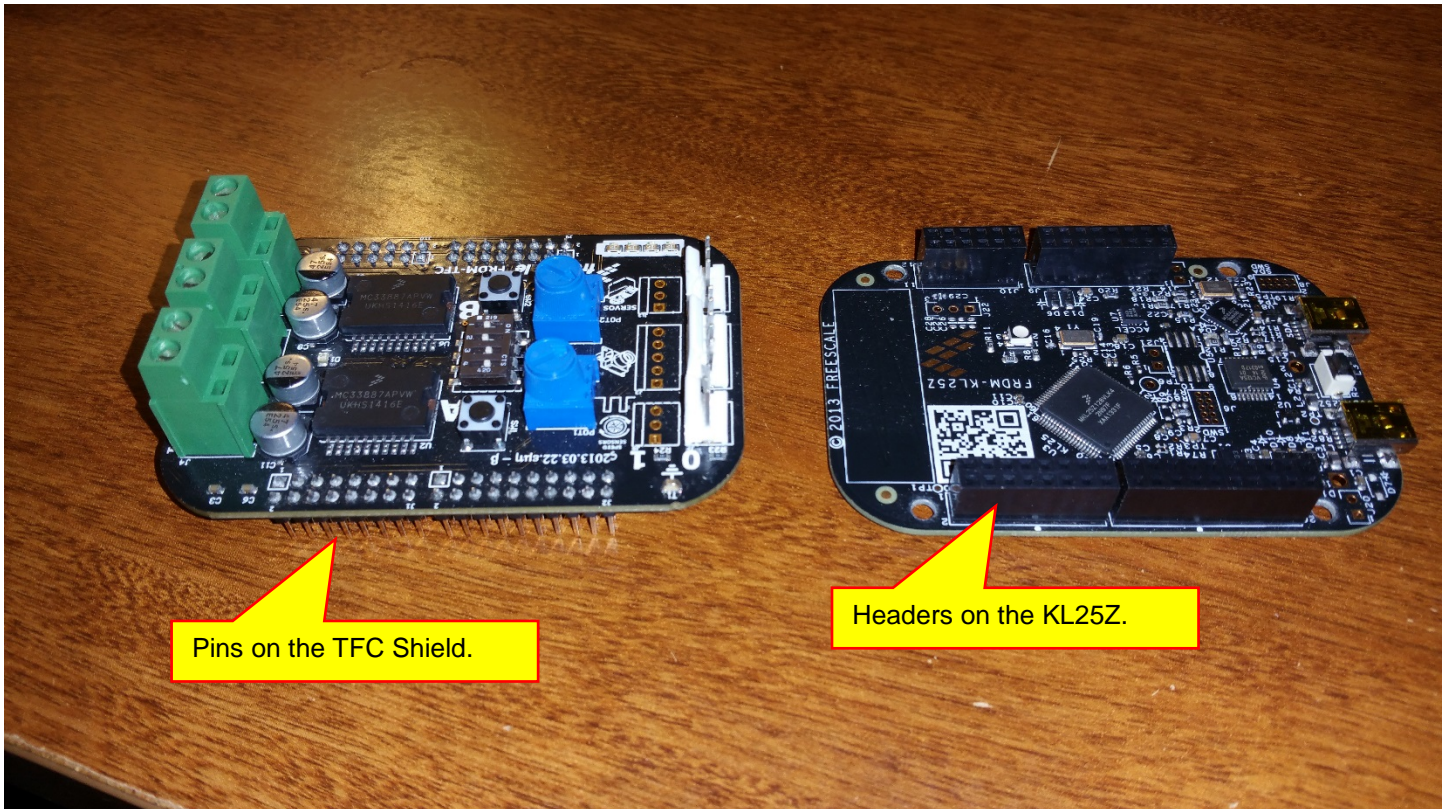
Lesson I: Introduction to Simulink and the KL25Z

A. Connecting the KL25Z Board

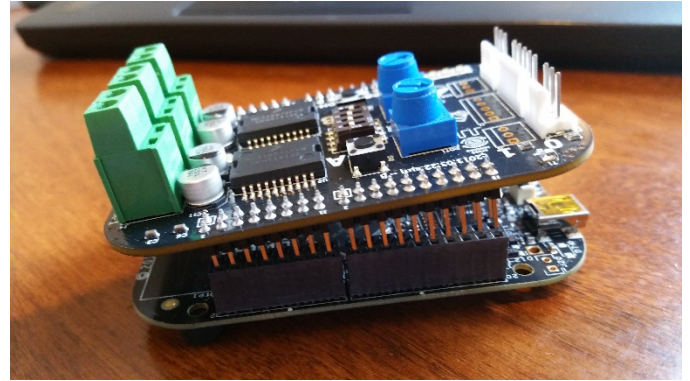
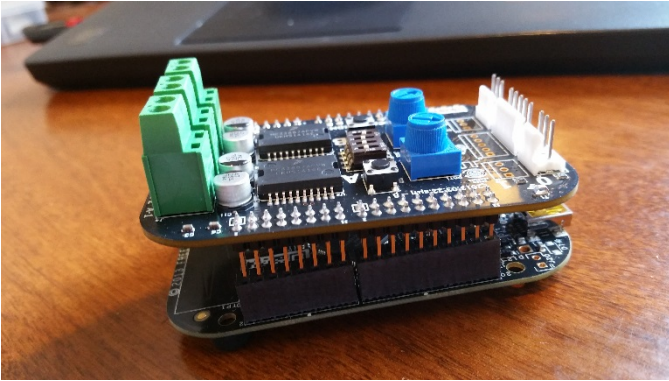
We will be using MathWorks Simulink to program the Freescale KL25Z microcontroller board. Here we will show a brief introduction to Simulink to program the tri-colored LED on the KL25Z microcontroller board. We will be using the KL25Z microcontroller by itself. The TFC Shield is mounted on the KL25Z board and both are packaged inside an antistatic bag:



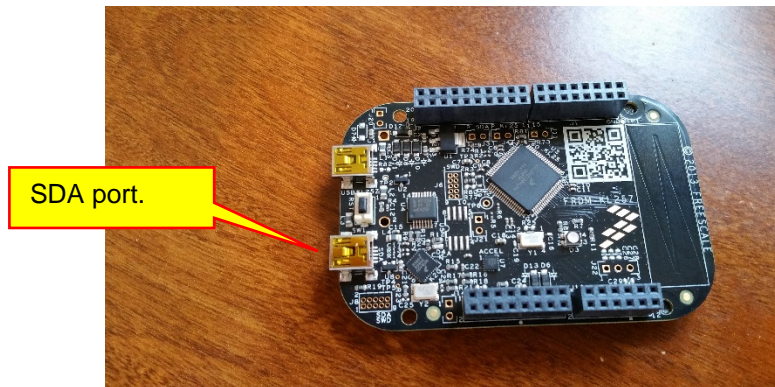
The TFC Shield is plugged into the KL25Z Board. Both boards are shown separately below:



We see that the TFC Shield has pins that plug into the KL25Z headers. Gently pull apart the two boards. Do not twist the boards. Alternately pull up one side then the other until the boards pull apart:

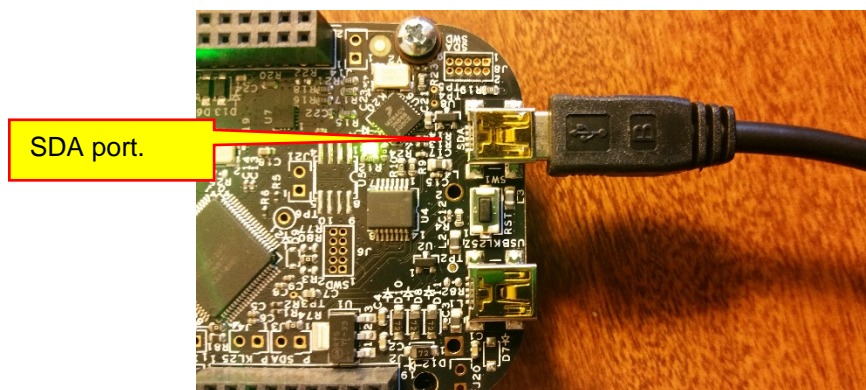


Pull up each side a small amount to avoid bending the pins. The microcontroller board is shown below:



Always store the KL25Z microcontroller board and the TFC Shield in the anti-static plastic bag when not in use.

Connect one side of the cable to the SDA port in the KL25Z board. Note that the SDA port powers the KL25Z and provides a communication link between the KL25Z and your computer.



Plug the other side of the cable into a USB port in your computer.

To check to see that your computer recognizes the KL25Z, we have created a program to detect the port. From the MATLAB command window, enter the command **Detect_KL25Z_Board** at the command prompt. If you see an error, than you board was not detected by your computer:

A KL25Z board was not found.

```
Command Window
>> Detect_KL25Z_Board
Error using Detect_KL25Z_Board>findFRDMSerialHardware (line 68)
Unable to locate any FRDM-KL25Z Boards...
    HARDWARE may not be Connected or Recognized.

Error in Detect_KL25Z_Board>getFRDM_KL25ZCOMPort (line 44)
    comPorts = findFRDMSerialHardware(sysProg, sysCmd, sysRegex);

Error in Detect_KL25Z_Board (line 8)
    com_port = getFRDM_KL25ZCOMPort;

fx >>
```

This error message means that you did not connect your board properly or that your computer does not have the proper drivers. Note that only the SDA port can be used to connect the KL25Z to your computer. The other port can power the KL25Z, but it cannot establish a communication link between the KL25Z and your computer. **If you are using the SDA port and you still receive an error, you may need to update the driver on your computer.**

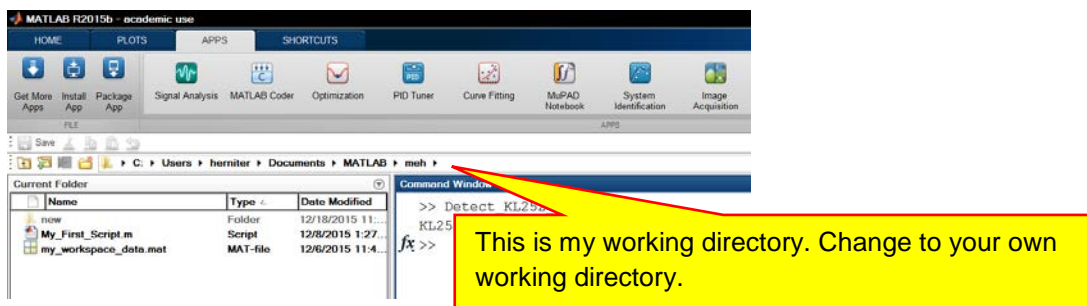
If you have proper communication between your computer and the KL25Z, you should receive a message similar to the one shown below:

```
Command Window
>> Detect_KL25Z_Board
KL25Z board detected on COM20.
fx >>
```

Your com port will be different, especially if you are using Apple OSX. Now that we have verified that the board is connected, we can create a model and download it to the KL25Z.

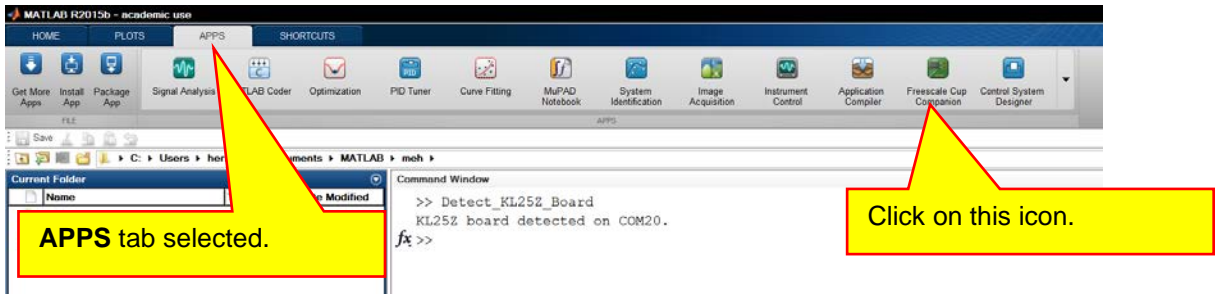
B. Blinking Light

We will now create a simple Simulink model to flash the blue on-board LED. First, make sure that you are in your own working directory. Creating these models will create a large number of support files:

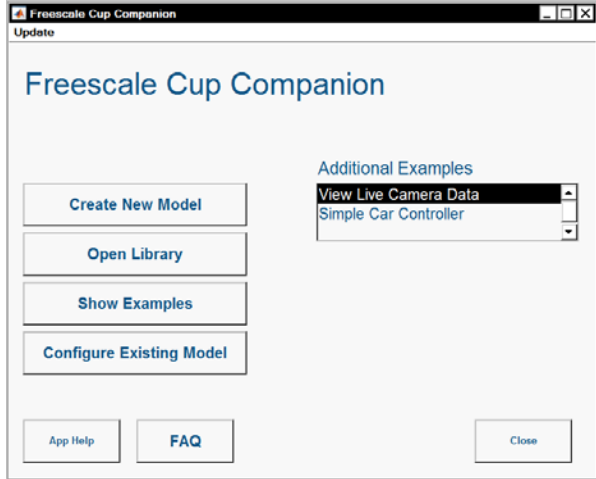


The screenshot shows the MATLAB R2015b interface with the 'Apps' tab selected. Below the MATLAB window, a File Explorer window is open, showing the current folder 'C:\Users\herniter\Documents\MATLAB\mesh'. A yellow callout box points to the 'mesh' folder with the text: "This is my working directory. Change to your own working directory."

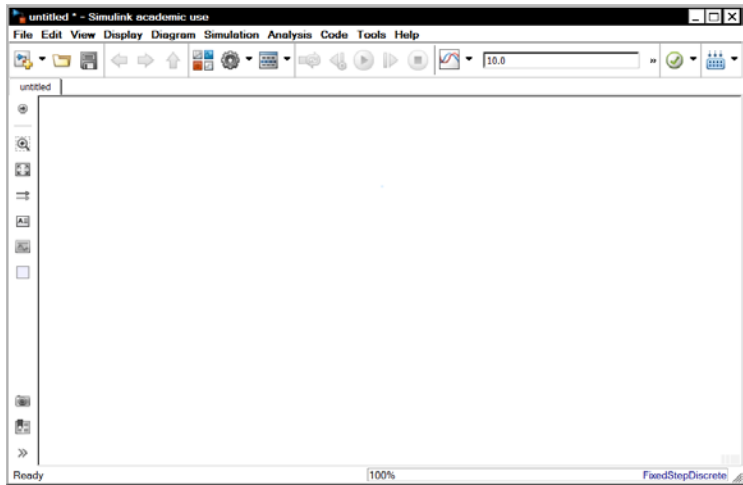
Instead of creating a new model manually and setting it up to run on the KL25Z, we will use a utility provided by MathWorks to create a new model and set up the options to have it run on the KL25Z. In MATLAB window select the **Apps** tab:



Click on the **Freescale Cup Companion** icon to run the app:

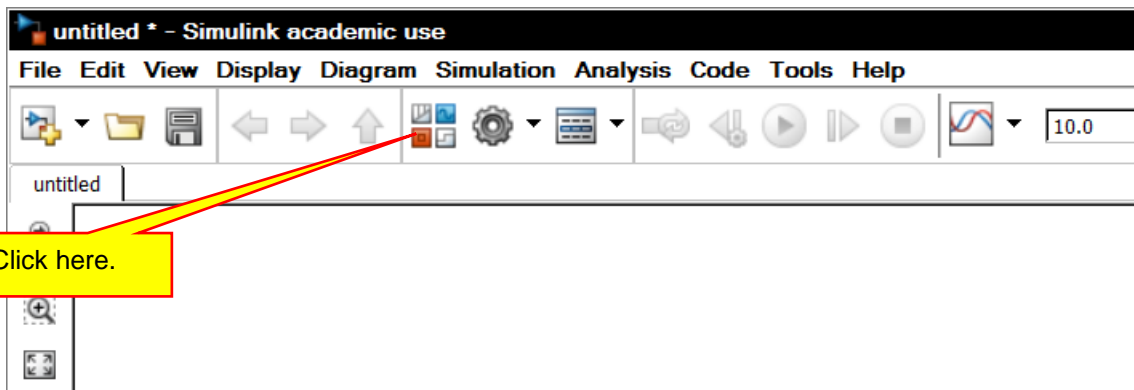


Click on the **Create New Model** button. Simulink will run and open an empty model:

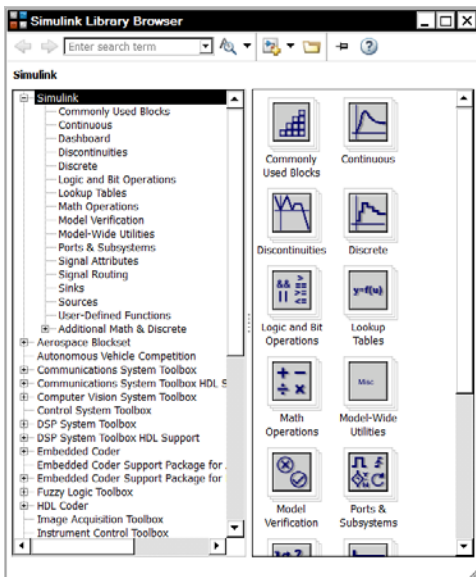


We can now place parts in this model. Open the Simulink Library Browser by clicking on the **Library Browser**

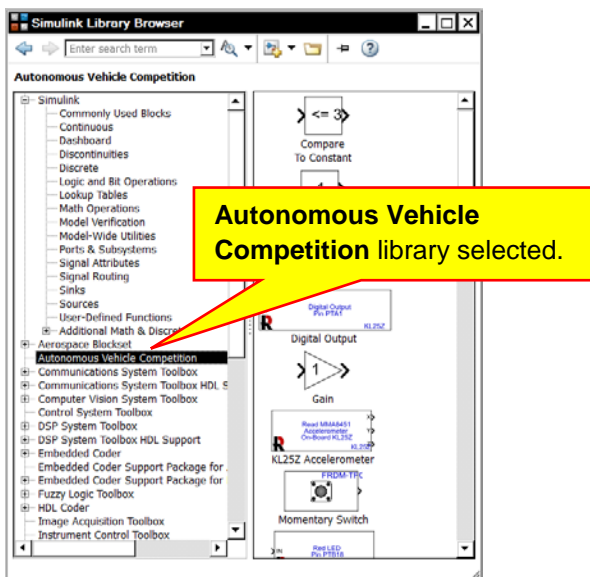
icon :



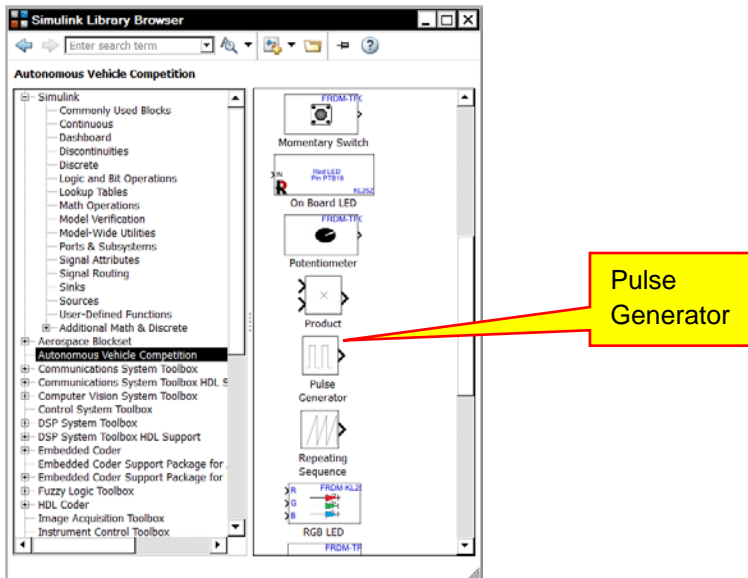
The **Simulink Library Browser** will open:



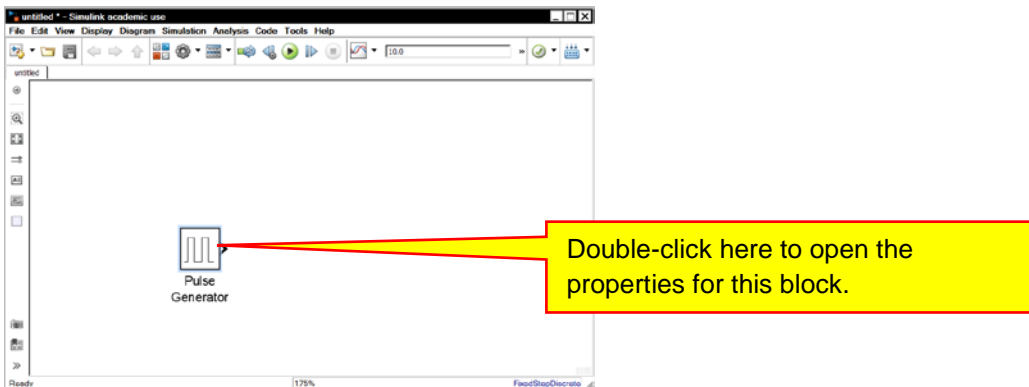
We have created a library of the most commonly used of parts for this competition in a library called **Autonomous Vehicle Competition**. Select this library as shown below:



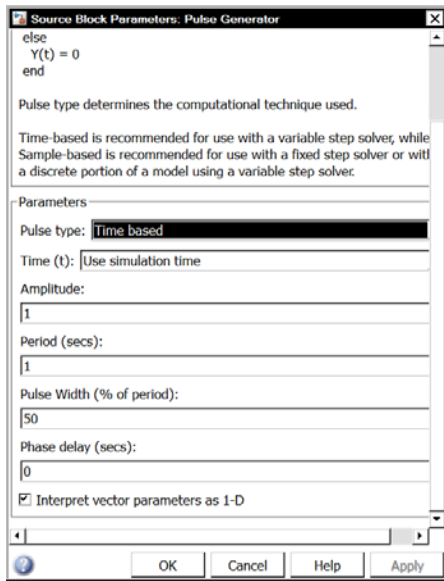
All off the parts in this library are also located in other libraries. (They were actually copied from the other libraries into this library.) You are encouraged to look through all of the other libraries to see what other blocks are available. We will use the **Autonomous Vehicle Competition** library so that parts are easy to find for the first few labs. Scroll down the right pane to locate the **Pulse Generator**:



Drag the pulse generator into your model:

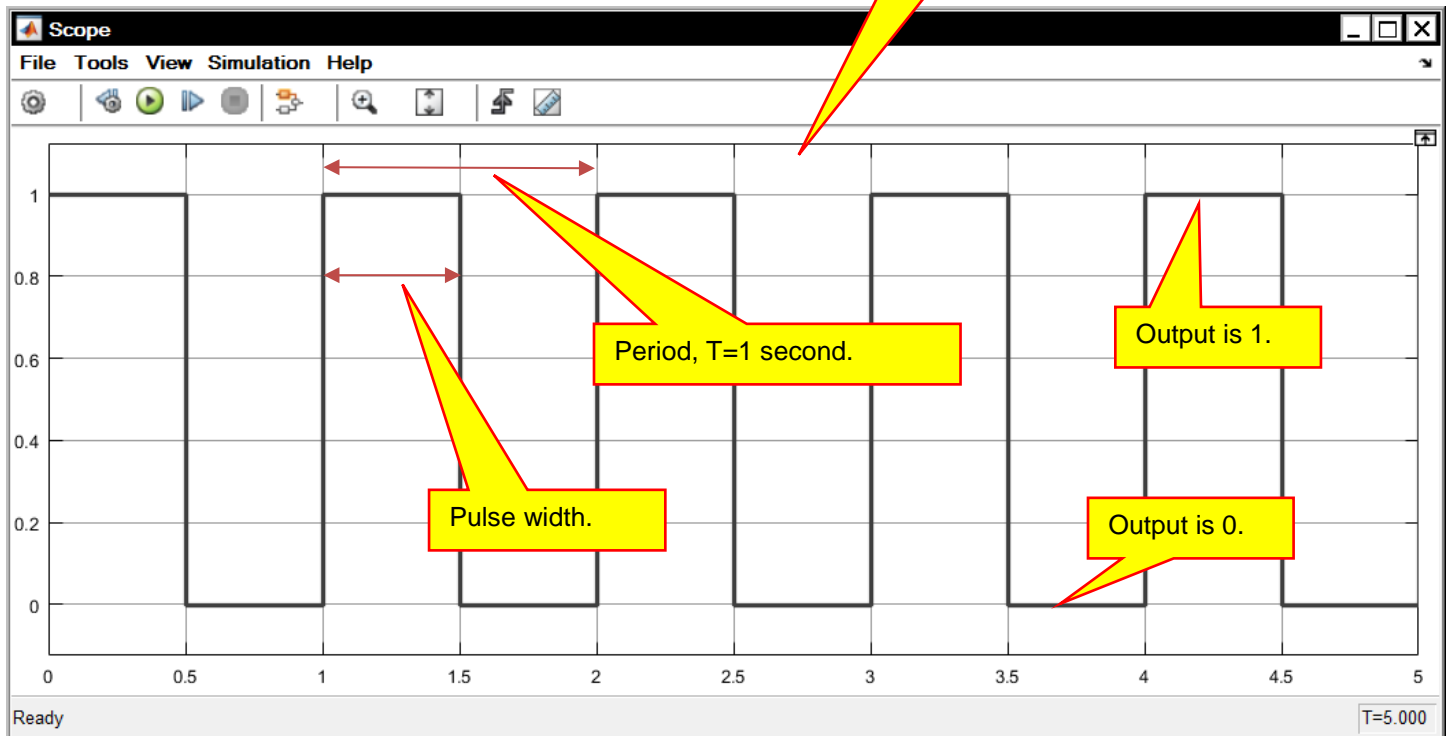


Double-click on the pulse generator part to edit its attributes. Specify a **Period** of 1 second and a **Pulse Width** of 50%.



We are not generating this plot. It is only shown for reference. (We could generate it if we wanted to.)

This block generates a square wave as shown below:

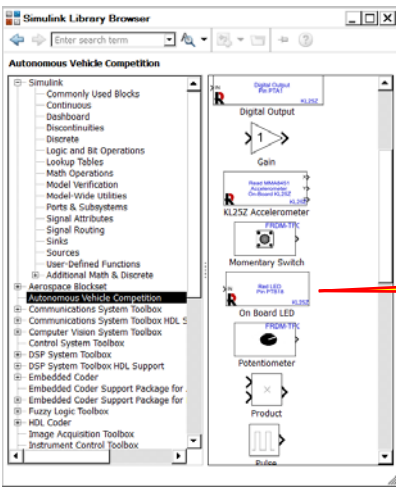


Note that the period is the amount of time the waveform takes to repeat. In this case, the waveform repeats once every second, so the period T equals 1 second. The pulse width is the amount of time the waveform spends “high.” In this case, we specified a pulse width of 50%, so the waveform is high half of the time. The frequency of a signal is one divided by the period, $F = 1/T$, for a period of 1 second, we will have a frequency of 1 Hz. Note that Hz is short for the unit Hertz, which in the old days was called cycles per second. So 1 Hz is one cycle per second. (Or, the waveform repeats once every second.)

We also see that the output of the pulse generator switches between 0 and 1. When we connect this to our LED block, a 0 will turn off the LED and a 1 will turn on the LED. For the waveform we created, when we connect this pulse source to an LED, it will cause the LED to flash on and off at a 1 Hz rate (on and

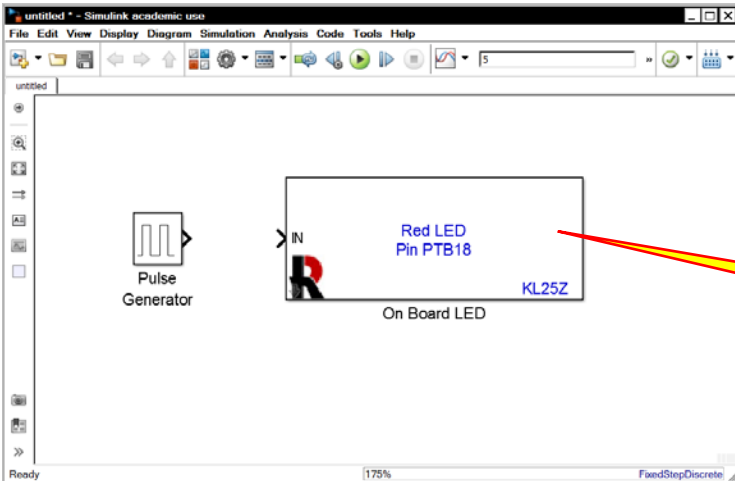
off once per second) with the LED on 50% of the time and off for 50% of the time. Click the **OK** button to accept the changes.

Locate the **On Board LED** block in the **Autonomous Vehicle Competition** library:



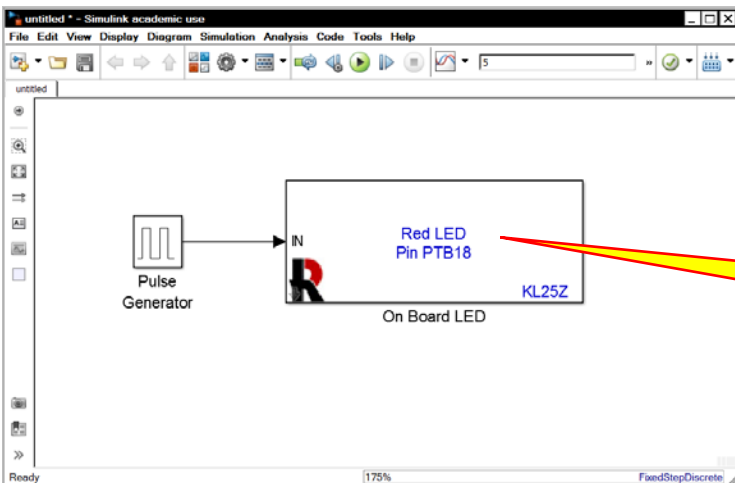
On Board LED block.

Drag one into your model:



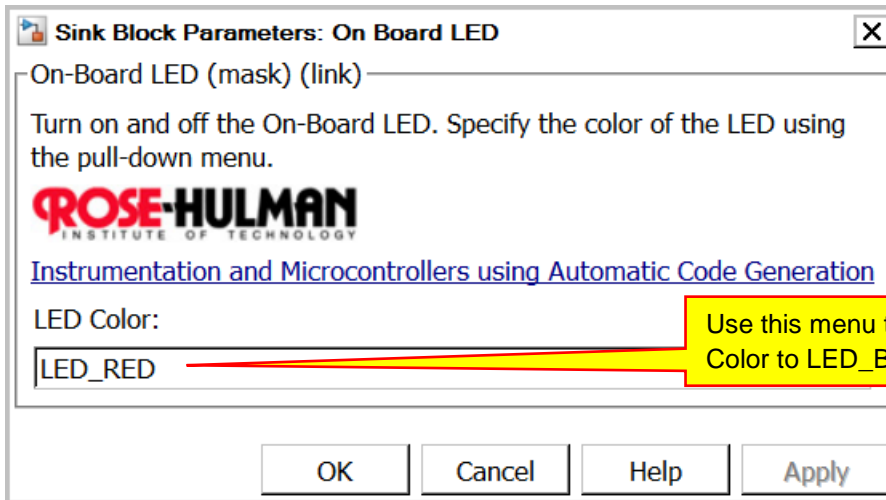
On Board LED block.

Connect the input of the **On Board LED** block to the output of the **Pulse Generator**:

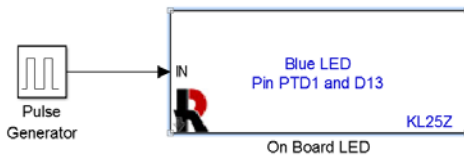


Double-click here to open the properties for this block.

Next, double-click on the **On Board LED** block to modify its properties. Change the LED Color to **LED_BLUE**:



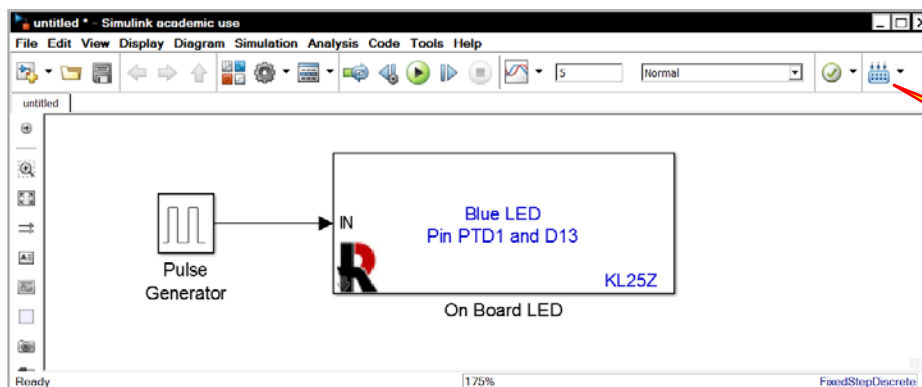
Click the **OK** button to accept the changes. Note in the model that **Blue LED** will be displayed on the block:



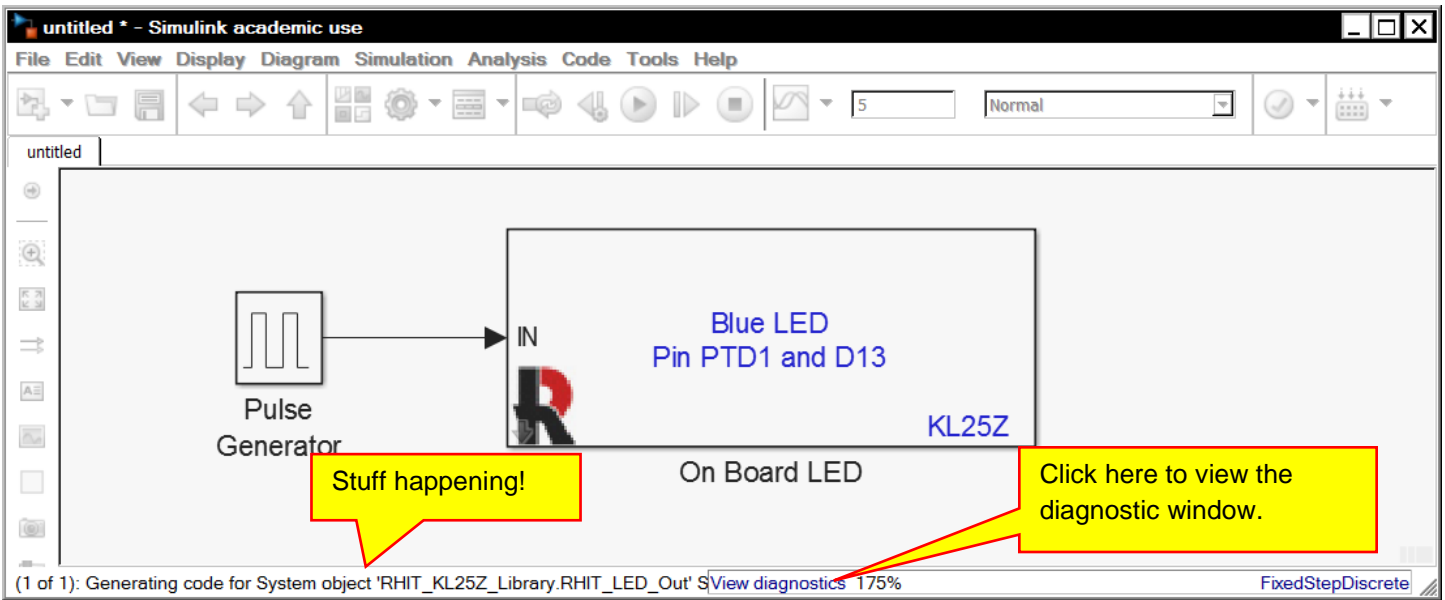
Before continuing, save your model as `Blue_LED_Flasher`. Note that there can be no spaces in your model name.

We are now ready to run our first model on the FRDM-KL25Z Microcontroller target. To build the model, download it to the FRDM-KL25Z Target, and run the model on the target, click on the **Build Model**

button  :

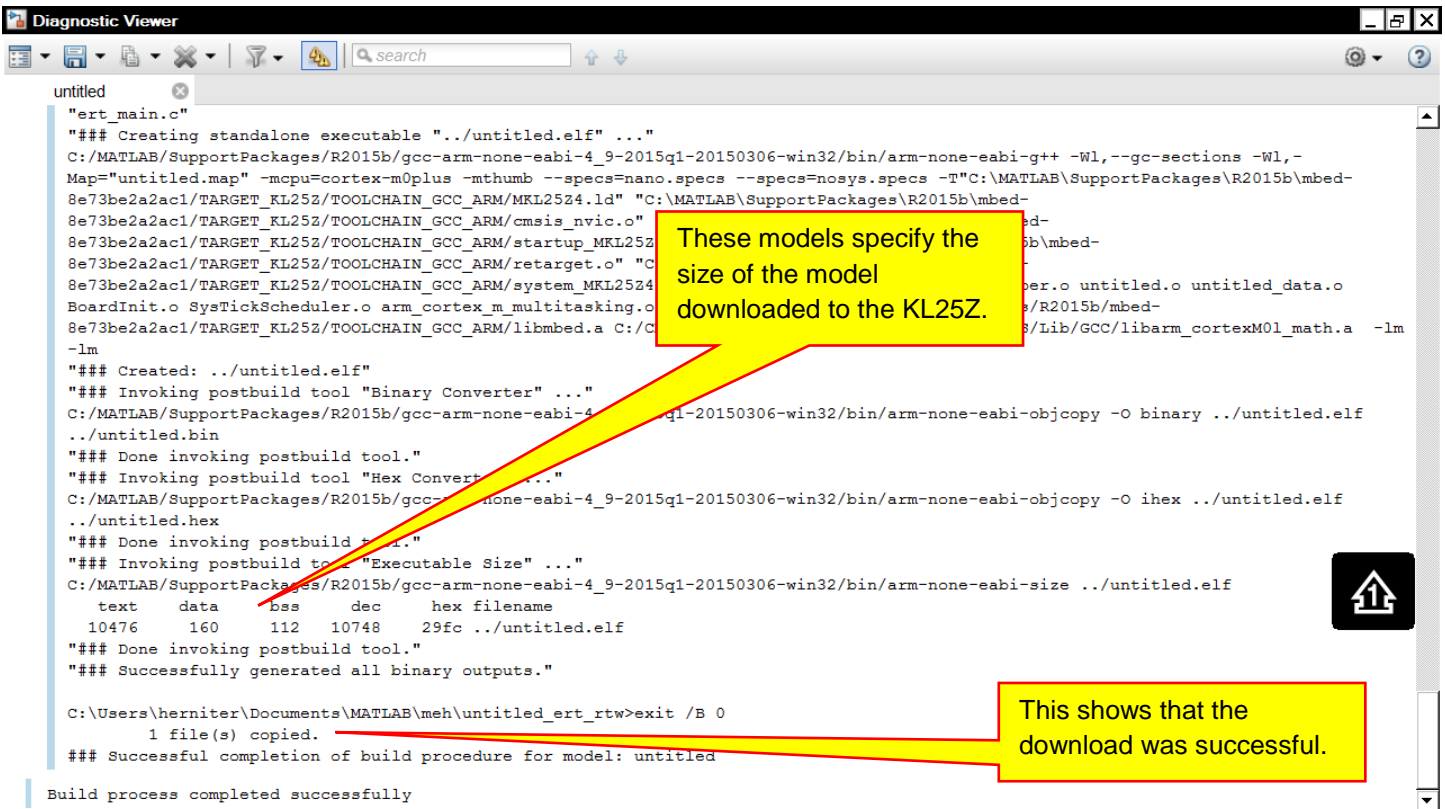


A bunch of stuff will happen:



When the download is complete, the blue LED light will flash on and off at 1 Hz.

If there was an error and your blue LED is not flashing on and off, you can view the diagnostic window by clicking on the link as shown above:



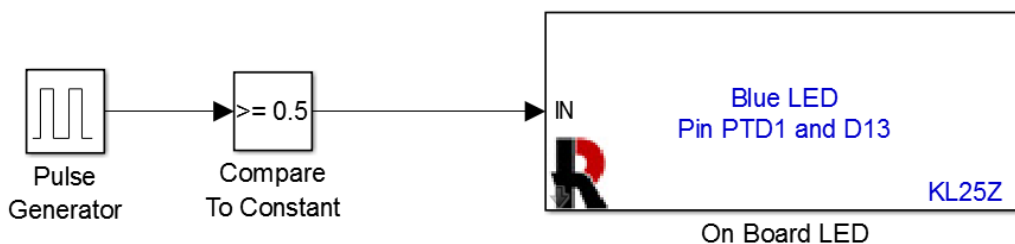
If the download was not successful, you might not have the drivers installed correctly. However, it is more likely that you forgot to plug in your KL25Z board into your computer. **You should always show the diagnostics window! It is the best way to determine that your model built correctly and was downloaded to the KL25Z.**

The numbers displayed are also important as they specify the size of the model downloaded to the KL25Z. If the model becomes too large two things may occur. If Simulink detects that the model is too large, an error message will be generated. If the model is too large and this is not detected by Simulink, the model will be downloaded to the KL25Z. The results will be a model that does not run properly or not at all. Since Simulink cannot always detect when a model is too large the general rules for size limits are that the text size should be less than 130,000, the data size below 7,900, and the BSS size below 17,000. These data sizes are not mutually exclusive, so a general rule would be to stay 10% below these limits.

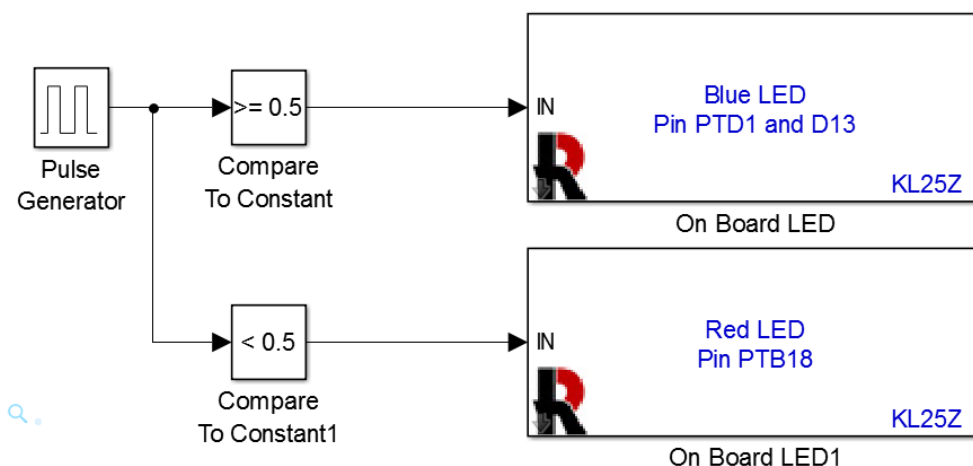
Note that the model is written to the FRDM-KL25Z flash memory so that if you cycle the power, the model will remain in memory and restart at power up or when you press the reset (RST) button on the KL25Z.

C. Flip-Flop LED Model

As a second example, we want to turn on the Blue LED half the time. When the Blue LED is not on, we want the Red LED to be on. The output of the pulsed source from the previous example is a zero or a 1. We can check this value with the compare to constant block as shown below. Note that we will start with the previous model and then use the **File / Save As** menu selections to create a new model. Save the model with the name Flip_Flop_LED:

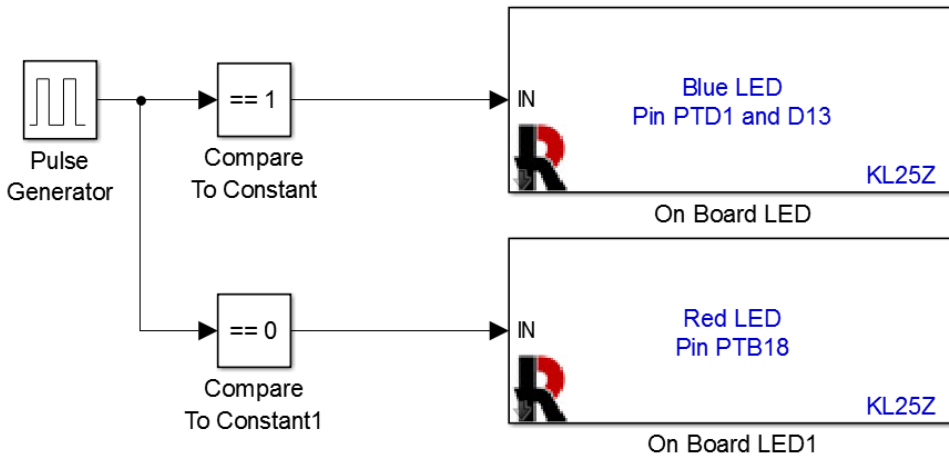


Note that we are using the **Compare To Constant** block. The output of this block is either true (1) or false (0). In this case. If the pulsed source is a 1, the output of the block is true, which is represented by the numerical value of 1. That numerical value of 1 will turn on the Blue LED. Next, we add the remainder of the model to turn on the Red LED:

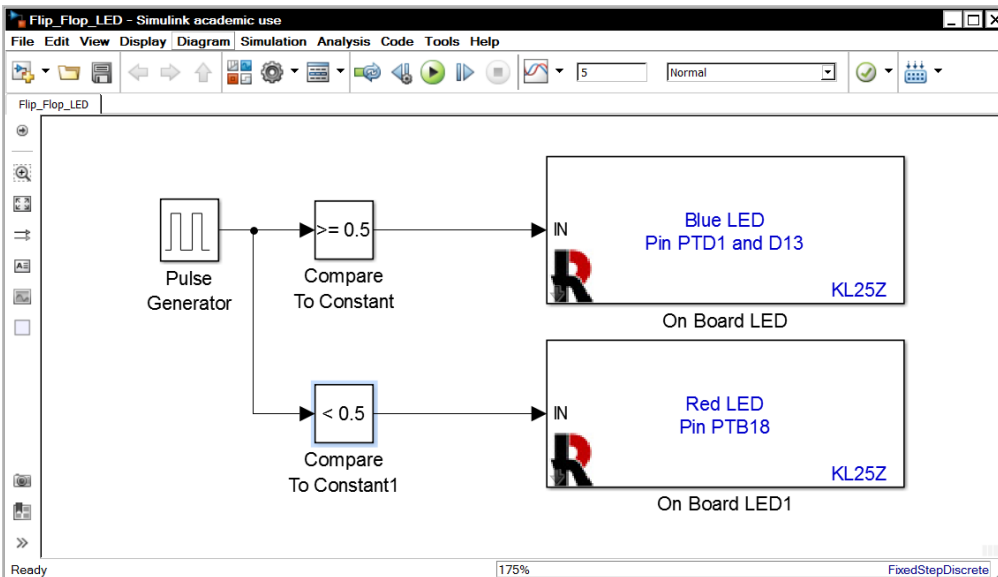


If the pulse generator output is 0, the bottom **Compare To Constant** block will be true and the red LED will turn on.

You might ask, why do we not use the implementation below:



Checking for equality is always risky when using fixed precision floating-point numbers. (These are real numbers with limited decimal places of accuracy.) Numbers in computers do not have infinite precision and calculations will always have round-off errors. Consequently, 1 may not be 1, but might be .99999999 or 1.00000001. Since our signal is either 0 or 1, it is just as easy compare it to 0.5 and not risk the equality being false when we expect it to be true. Using the second method just introduces a failure mode into your controller that can easily be avoided. Build the model below and verify that the LED flips between red and blue.



D. Questions

Question I-1: Can both ports on the KL25Z be used to connect to your computer?

Question I-2: What is the second USB port on the KL25Z used for?

E. Exercises

Exercise I-1: Modify the Blue_LED_Flasher so that the RED LED flashes at a rate of 2 Hz (twice per second.)

Exercise I-2: Modify the Blue_LED_Flasher so that the Blue LED flashes at a rate of 1 Hz but the light is only on for $\frac{1}{4}$ of a second.

Exercise I-3: Create a single model that flashes the Blue LED at a 1 Hz rate with 50% duty cycle, the Red LED at a 0.5 Hz rate with 50% duty cycle, and the Green LED at a 0.25 Hz rate with 50% duty cycle. Remember that Hz is frequency (f) and the period is 1 divided by the frequency, $F = 1/T$. You can use more than one pulse generator to solve this problem. Also note that when more than one LED is on, the colors will be combined to form new colors.

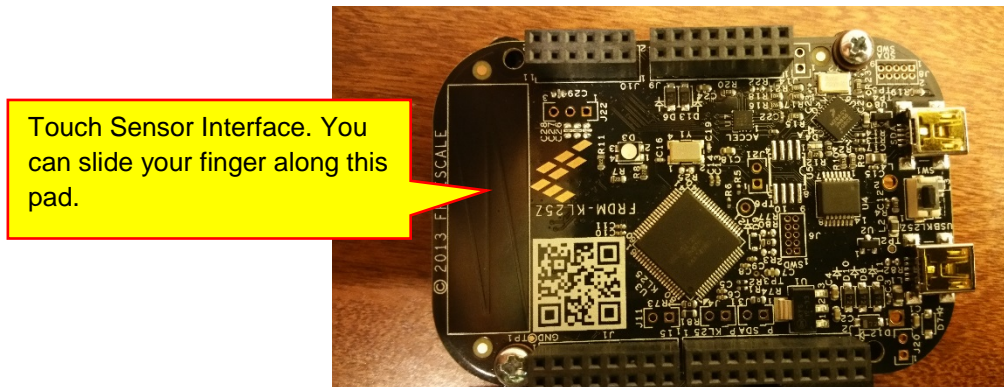
Lesson II

KL25Z On Board Sensors

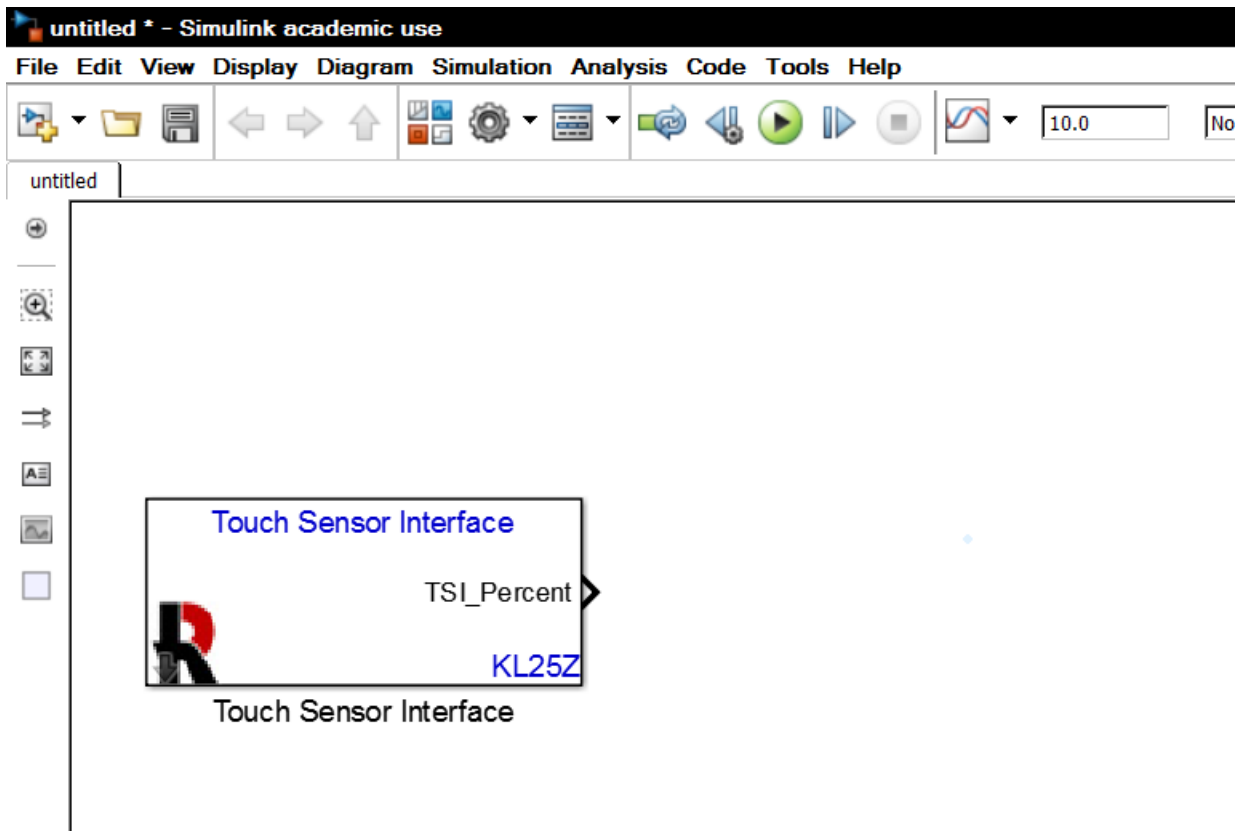
The KL25Z has an on-board accelerometer for measuring acceleration in three dimensions and a touch sensor that can be used as a slider to measure the position of your finger.

A. Touch Sensor Interface (TSI)

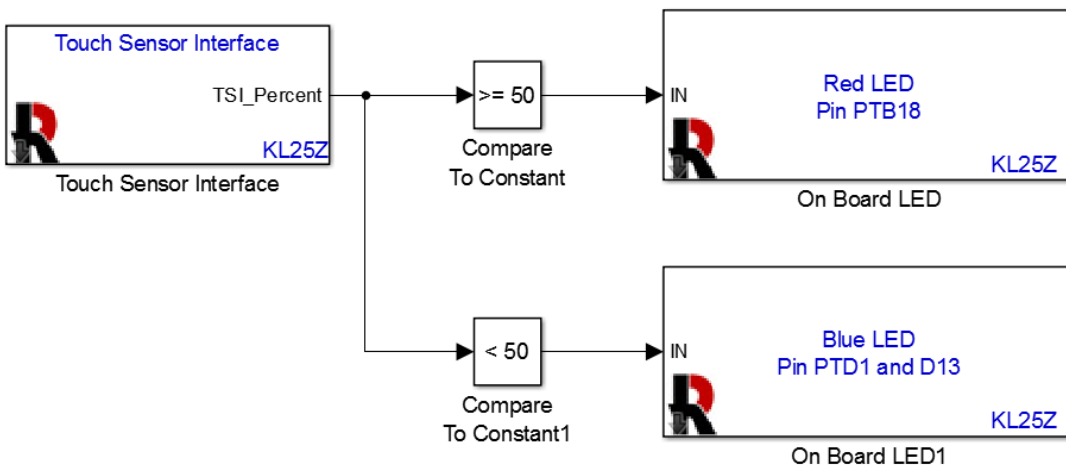
The touch sensor is that small open space on the KL25Z board enclosed in a rectangle. It kind of looks like open board space:



Create a new model (remember to use the Freescale Cup Companion App) and place the **Touch Sensor Interface** block in your model:



The output of this block is a number between 0 and 100 representing the position of your finger on the slider rectangle. As you move your finger, the output of the block will change correspondingly. (Where zero is and where 100 is, we are not sure...) We will create a simple model to test the operation of this sensor. We will turn on the red LED if the sensor output is greater than or equal to 50 and turn on the blue LED when the sensor output is less than 50. The model below accomplishes this task:



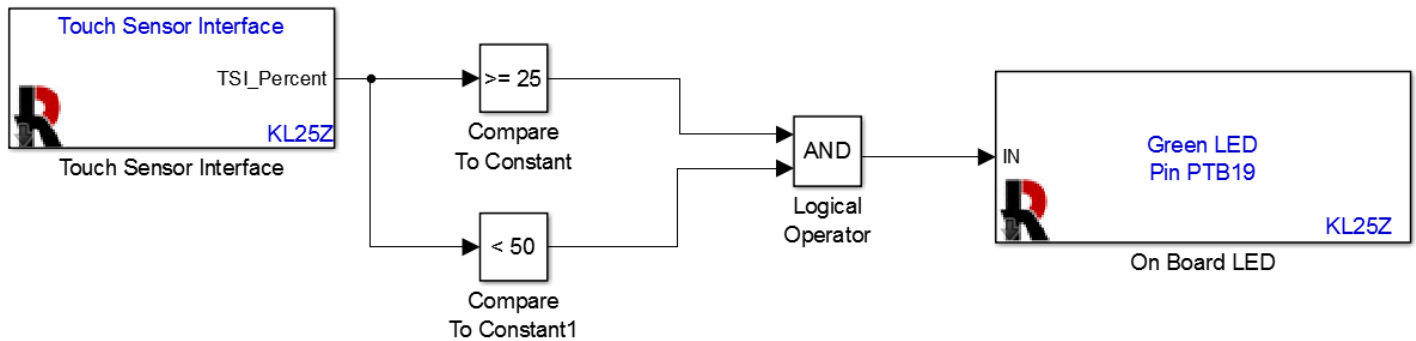
Build and download this model to your KL25Z board. Verify the following:

- 1) The red and blue LEDs turn on and off appropriately as you slide your finger.
- 2) When you are not touching the rectangle, the blue LED is illuminated.
- 3) Which side of the rectangle is zero and which side is 100.

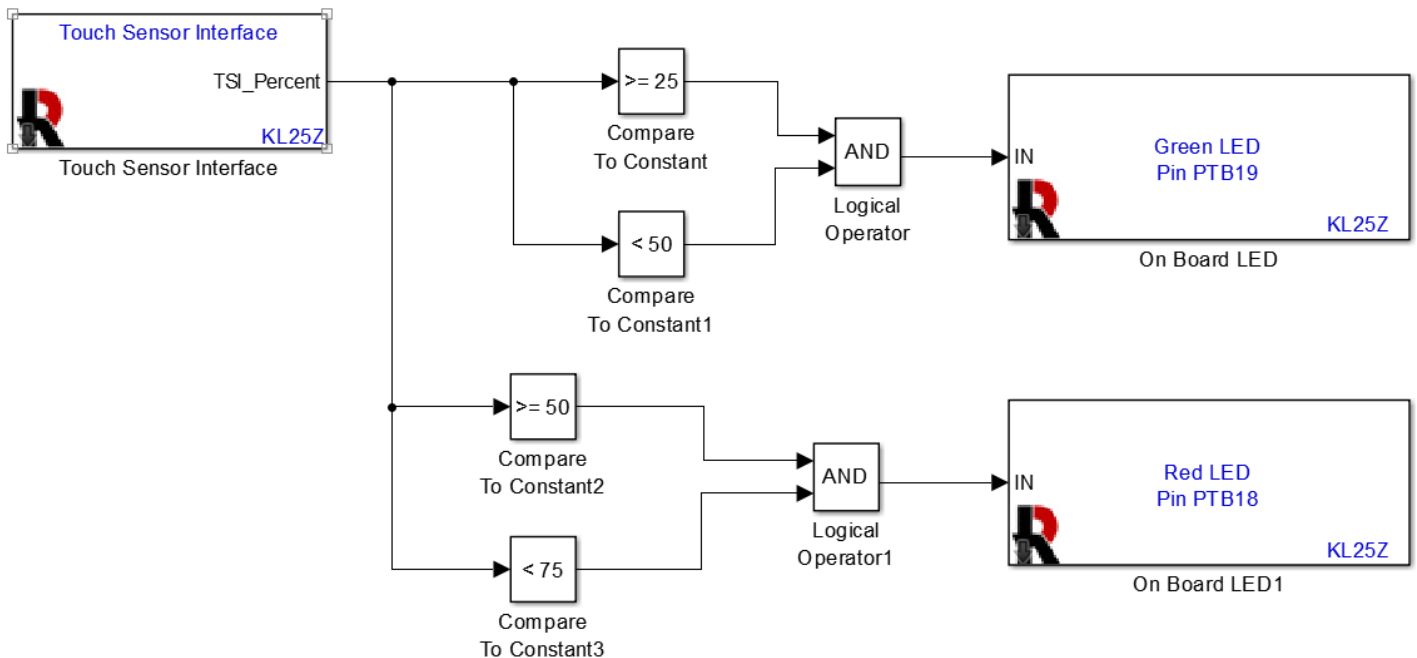
As a second example, we will use all three LED's to implement the following function:

- 1) If the TSI output is less than 25, no LED's are illuminated.
- 2) If the TSI output is greater than or equal to 25 and less than 50, the green LED is illuminated.
- 3) If the TSI output is greater than or equal to 50 and less than 75, the red LED is illuminated.
- 4) If the TSI output is greater than or equal to 75, the blue LED is illuminated.

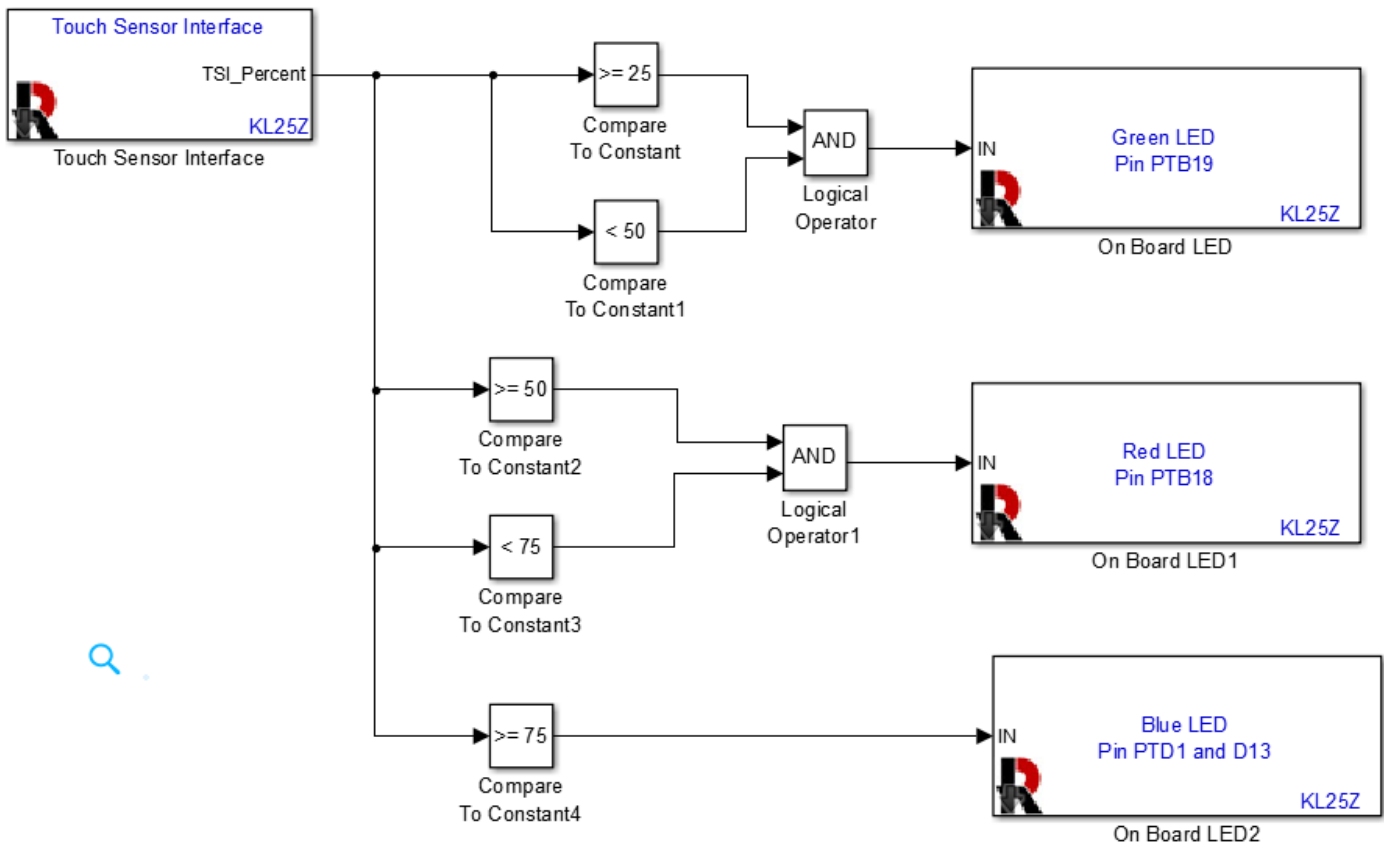
To implement this function, we will need a **Logical Operator** block to implement the AND function. The **Logical Operator** block is located in the **Autonomous Vehicle Competition** library. The logic for the green LED is shown below:



Note that the output of the logical AND block is only true (1) if both inputs are true (1). We use similar logic for the red LED:



The logic for the blue LED is a single compare to constant block:

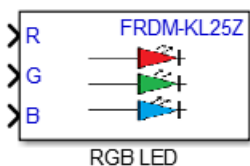


Build and download the model to verify that it works properly.

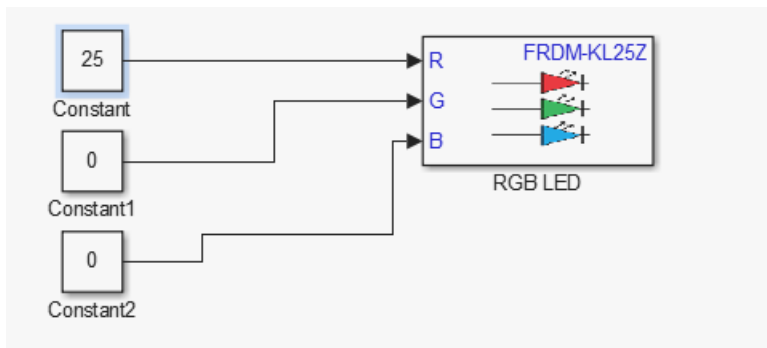
B. Accelerometer

The KL25Z has an on-board 3-axis accelerometer. It can measure acceleration in three directions up to +/- 1 g. It has three outputs, one for each axis. The output is a real number between -1 and +1, where +/- 1 represents +/- 1g of acceleration. (Remember that g is the acceleration of gravity, 9.8 m/s²). To illustrate its operation, we will display the acceleration of each axis on one of the on-board LEDs.

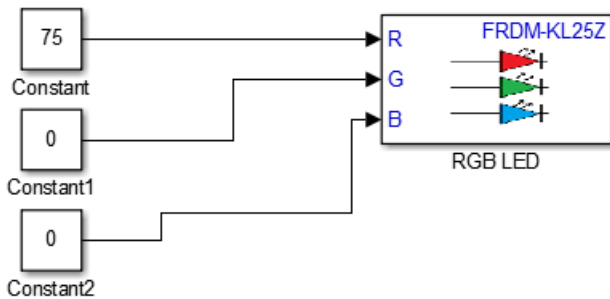
We cannot use the **On Board LED** block because it is either full on or completely off as they respond to a signal of 0 or 1. We would like to have the brightness of the LED controlled by the amount of acceleration. The higher the acceleration, the brighter the color. To do this, we will use the RGB LED block:



Each input can be a number between 0 and 255. With a zero input, the LED is off. With a 255 input, the LED is at its maximum brightness. For a number x between 0 and 255, the brightness in percent of the full brightness value is $100 \left(\frac{x}{255} \right)$. To show this, build and download the models shown below:

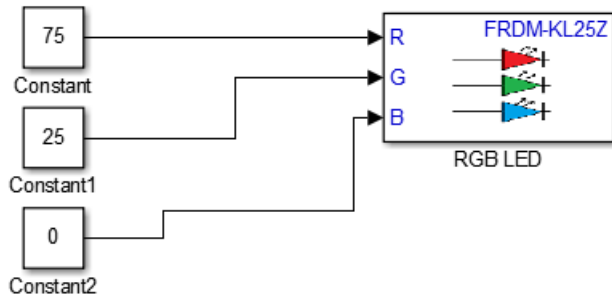


The RED LED should be illuminated dimly. Next, change the constant to 75:

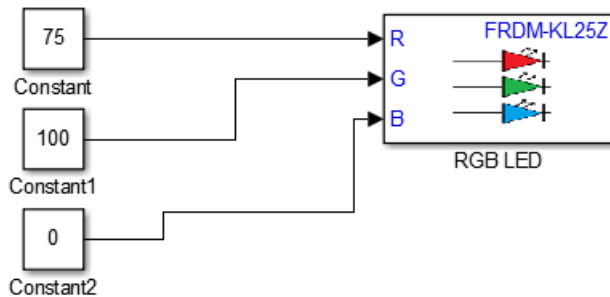


The RED LED is illuminated a little brighter.

Since we can control the brightness of each LED, we can make color combinations, such as:



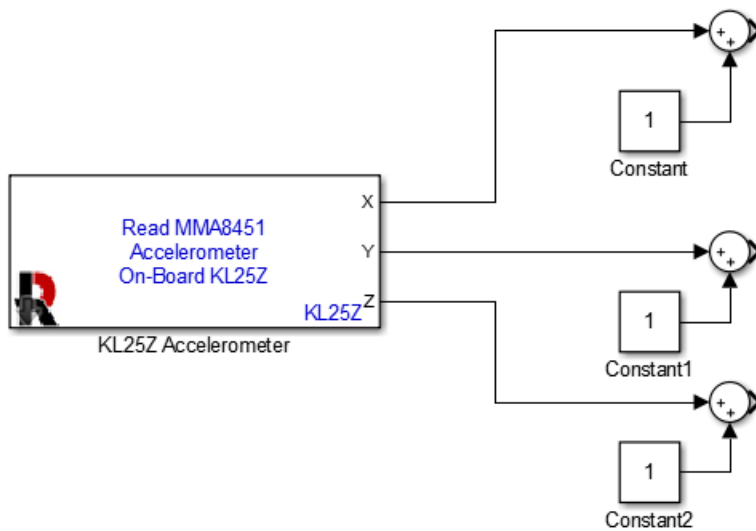
The overall color appears slightly orange. If we increase the green brightness:



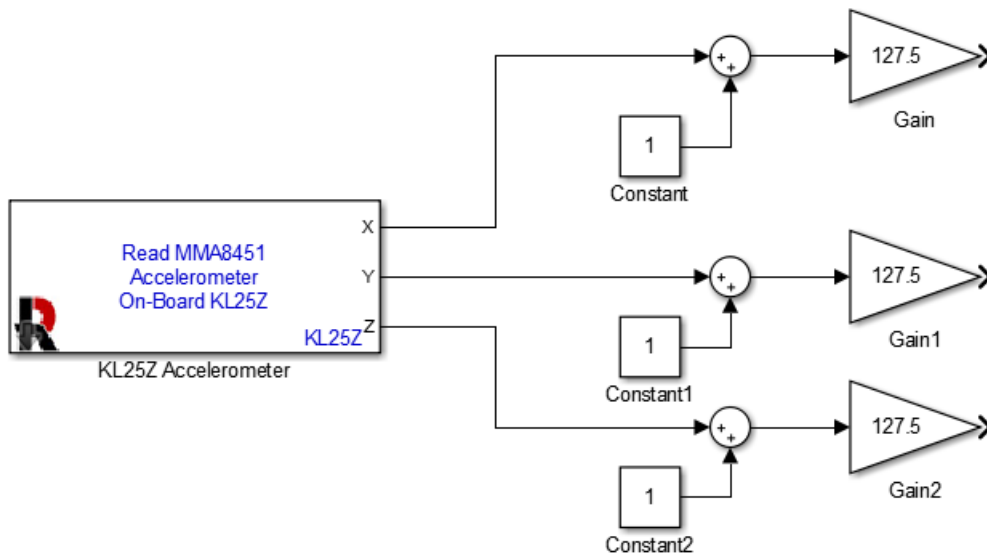
The LED will appear yellow.

Since we have 256 levels of brightness for each LED, the total number of color combinations is $256 \times 256 \times 256 = 16777216$. We will use these color possibilities so show off the 3-axis accelerometer. First,

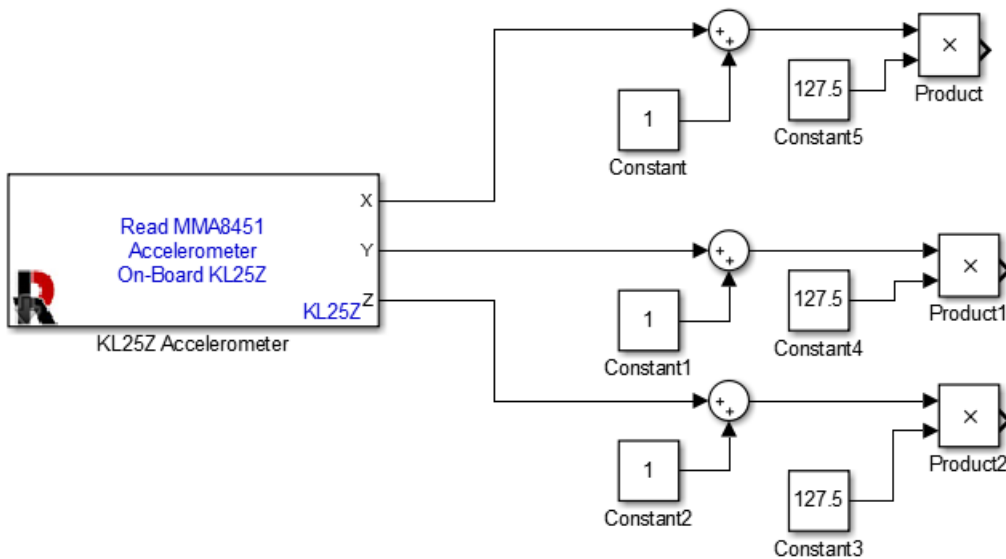
the accelerometer outputs a signal from -1 to +1. The RGB LED block only accepts positive inputs. To fix this, we will add 1 to each of the outputs:



The output of each of the sum blocks is a number between 0 and 2. The RGB block requires a number between 0 and 255. Thus we need to scale the outputs of the sum blocks by multiplying it by a constant $\frac{255}{2} = 127.5$. We can do this by multiplying by a constant or using a gain block. We will use a gain block as shown below:

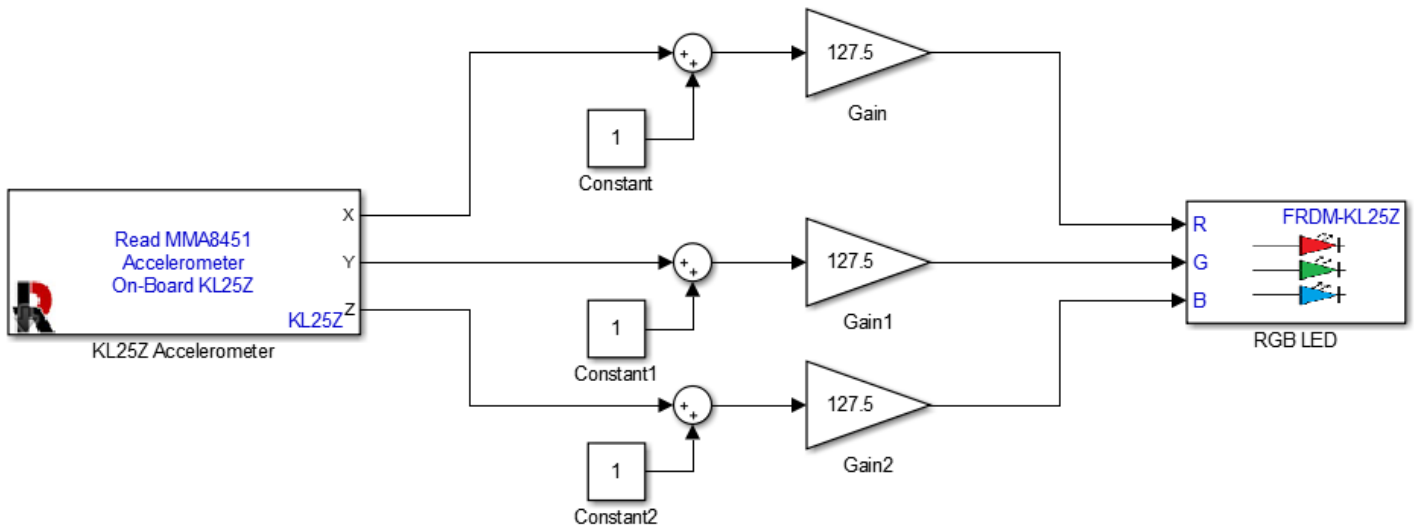


With a gain block, the output is the gain times the input. We could have done the following, but it is not as clean:




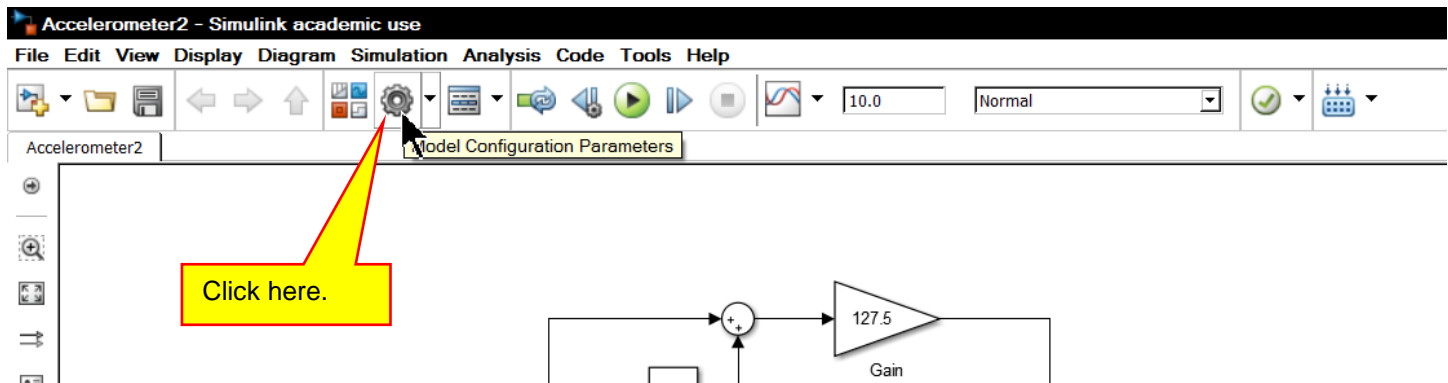
The product block is usually used to multiply two, non-constant, signals together. When multiplying by a constant, the gain block is preferred.

As a last step, connect the model to the RGB LED Block:

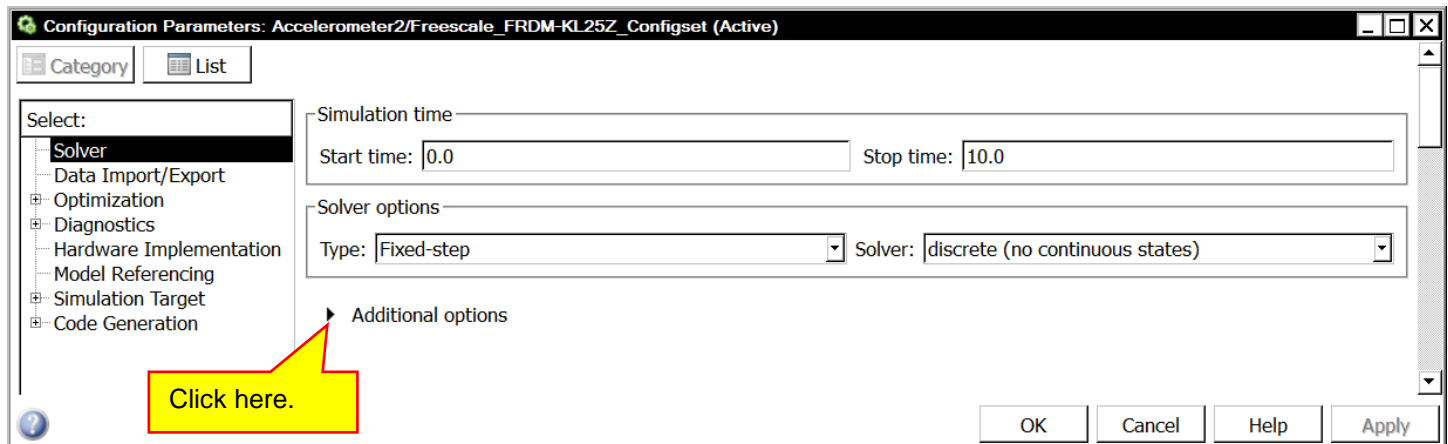


Build and download the model. Then shake your board and watch the LEDs light up!

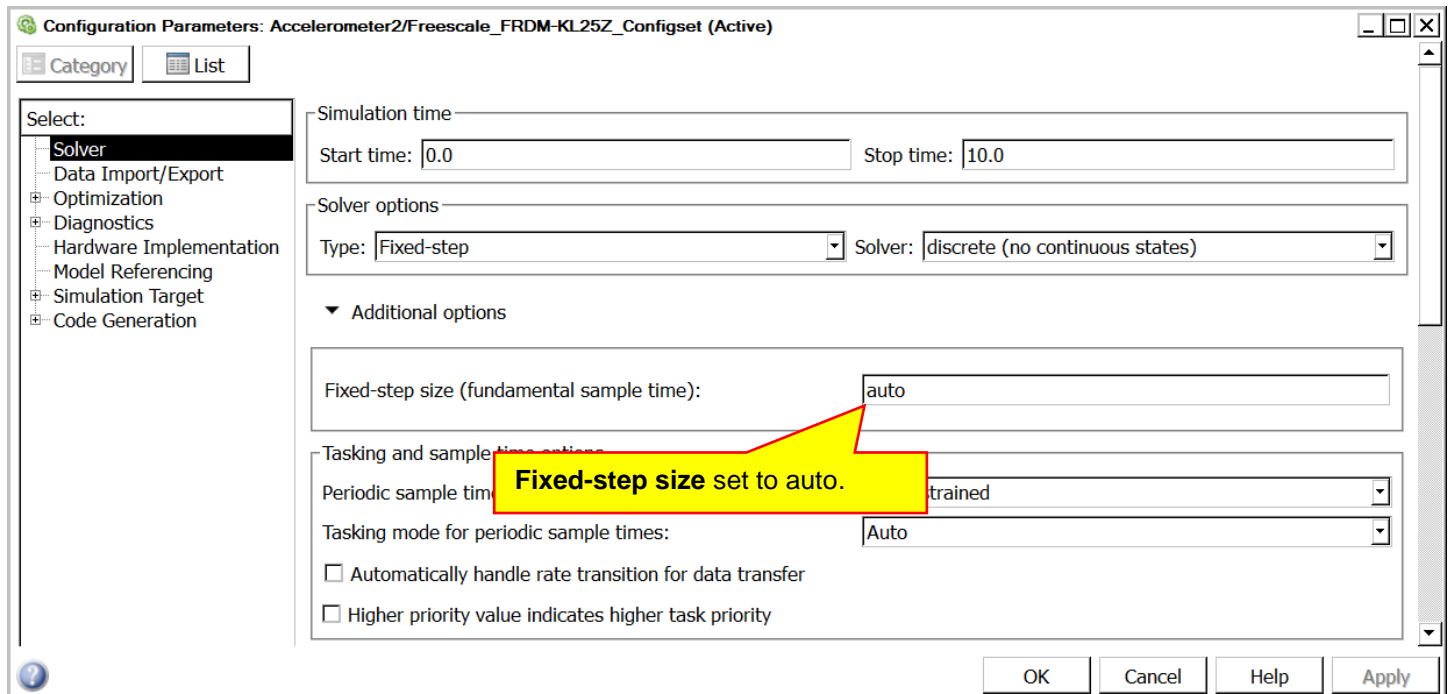
You will notice that the LEDs do not respond very quickly. This is because the model does not have a fast response time, because we did not specify the response time. To fix this problem, we will need to specify the fixed step time for the model. Click on the **Model Configuration Parameters** button , as shown below:



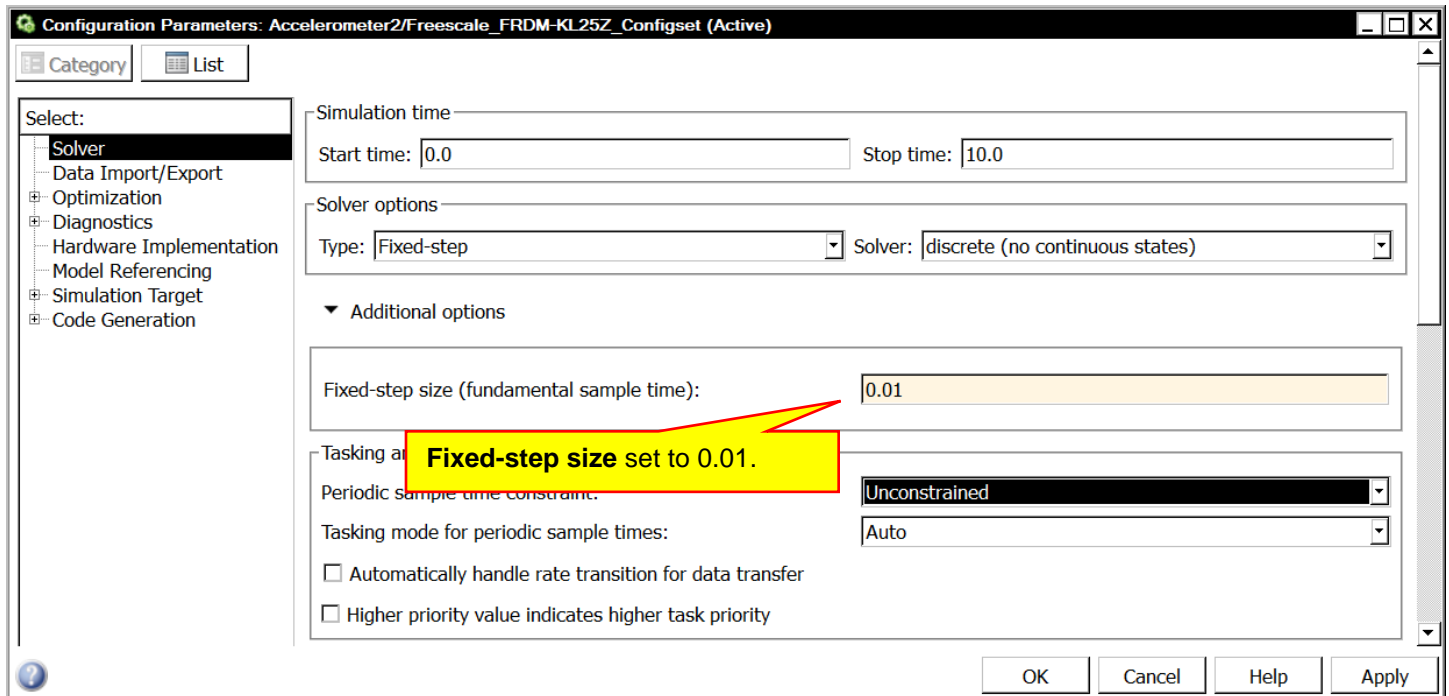
The Configuration Parameters window will open:



Click on **Additional options** as shown above:

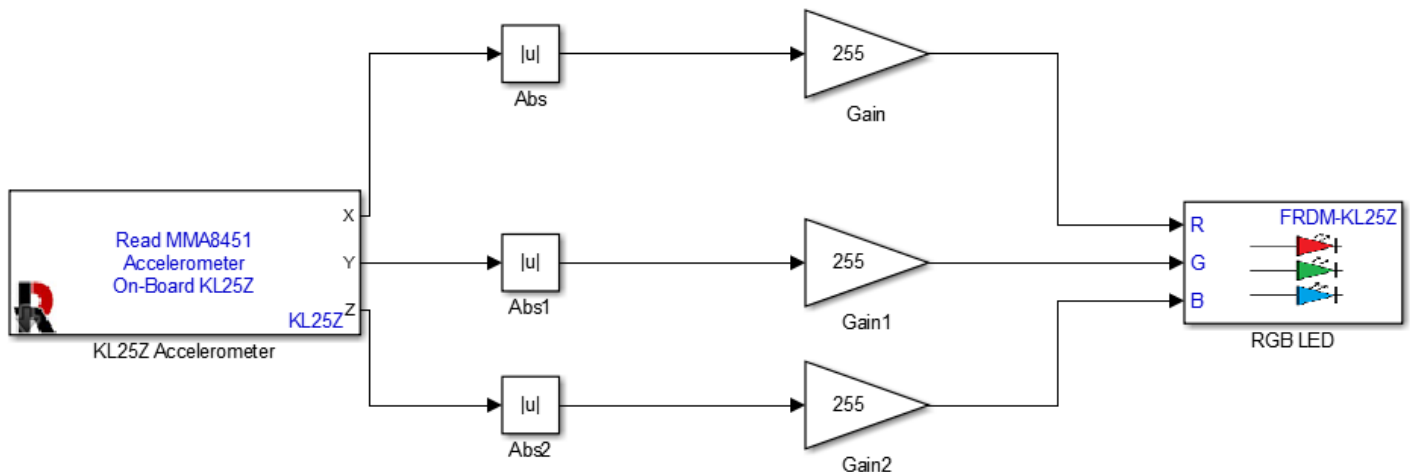


The **Fixed-step size** is how often the model executes. Since it is set to **auto**, Simulink picks the step size, and in this case, it is not small enough. Change the **Fixed-step size** to 0.01:

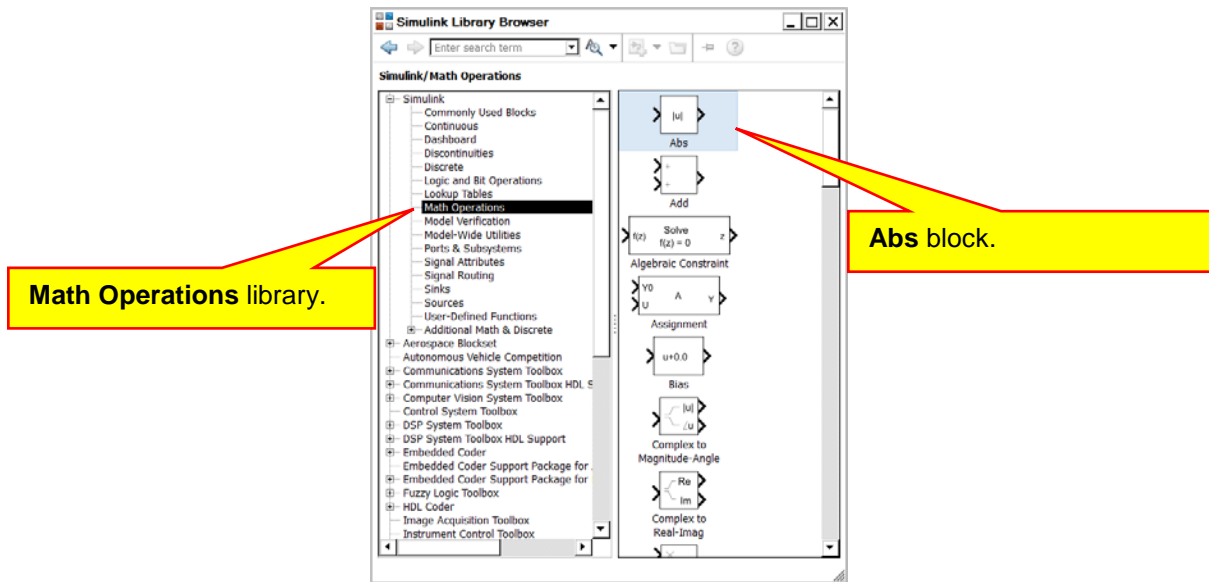


The value specified is in seconds, so 0.02 specifies a **Fixed-step size** of 0.01 seconds. A **Fixed-step size** of 0.01 seconds means that the entire model executes once every 0.01 seconds. In our case, it means that the controller reads the accelerometer, adjusts the signals to between 0 and 255, and updates the LED once every 0.01 seconds. That is pretty fast. For our model, as we move the board, the change in the LED's color should appear instantaneous. Click the **OK** button to accept the changes and then build and download the model to the KL25Z.

The model now responds well, and as you shake the board, the colors change. It is, however, difficult to determine the x, y, and z directions. This is because even though the accelerometer outputs a zero signal when there is no acceleration (when we do not move the board), our scaling makes the signals non-zero most of the time. The only time an LED will go off is when we have a -1g acceleration. To fix this problem, we will use a different method. Instead of adding 1 to the accelerometer signals, we will take the absolute value of the signals:



The absolute value block (**Abs**) is not in the **Autonomous Vehicle Competition** library. Instead, it is in the **Simulink / Math Operations** library:



From this model, you should be able to determine the x, y, and z axes for the accelerometer on the KL25Z board. Demonstrate the operation of this model. (Note that gravity never turns off. When the board is at rest, depending on how you orient the board, it will still measure 1g due to gravity on one of the axes.)

C. Questions

Question II-1: What is an accelerometer?

Question II-2: Where is the touch sensor located on the KL25Z board?

Question II-3: What is the correct way to create a new model that will run properly on the KL25Z?

Question II-4: What is the numerical output of the Touch Sensor Interface block?

Question II-5: Which side of the touch sensor rectangle is 0%? Which side is 100%?

Question II-6: What are the differences between the On Board LED block and the RGB LED block?

Question II-7: What are the valid numerical inputs to the On Board LED block?

Question II-8: What are the valid numerical inputs to the RGB LED block?

Question II-9: How many different colors are possible with the RGB LED block?

Question II-10: The accelerometer on the KL25Z measures acceleration in how many dimensions?

Question II-11: What is the maximum output of the accelerometer?

Question II-12: In what library is the Abs block located? What does the Abs block do?

Question II-13: What is the Fixed-step size in a model? How do you set it in a model?

D. Exercises

Exercise II-1: What other logical operators are available with the Logical Operator block?

Exercise II-2: In what other Simulink library is the Logical Operator found? (In addition to the Autonomous Vehicle Competition library.)

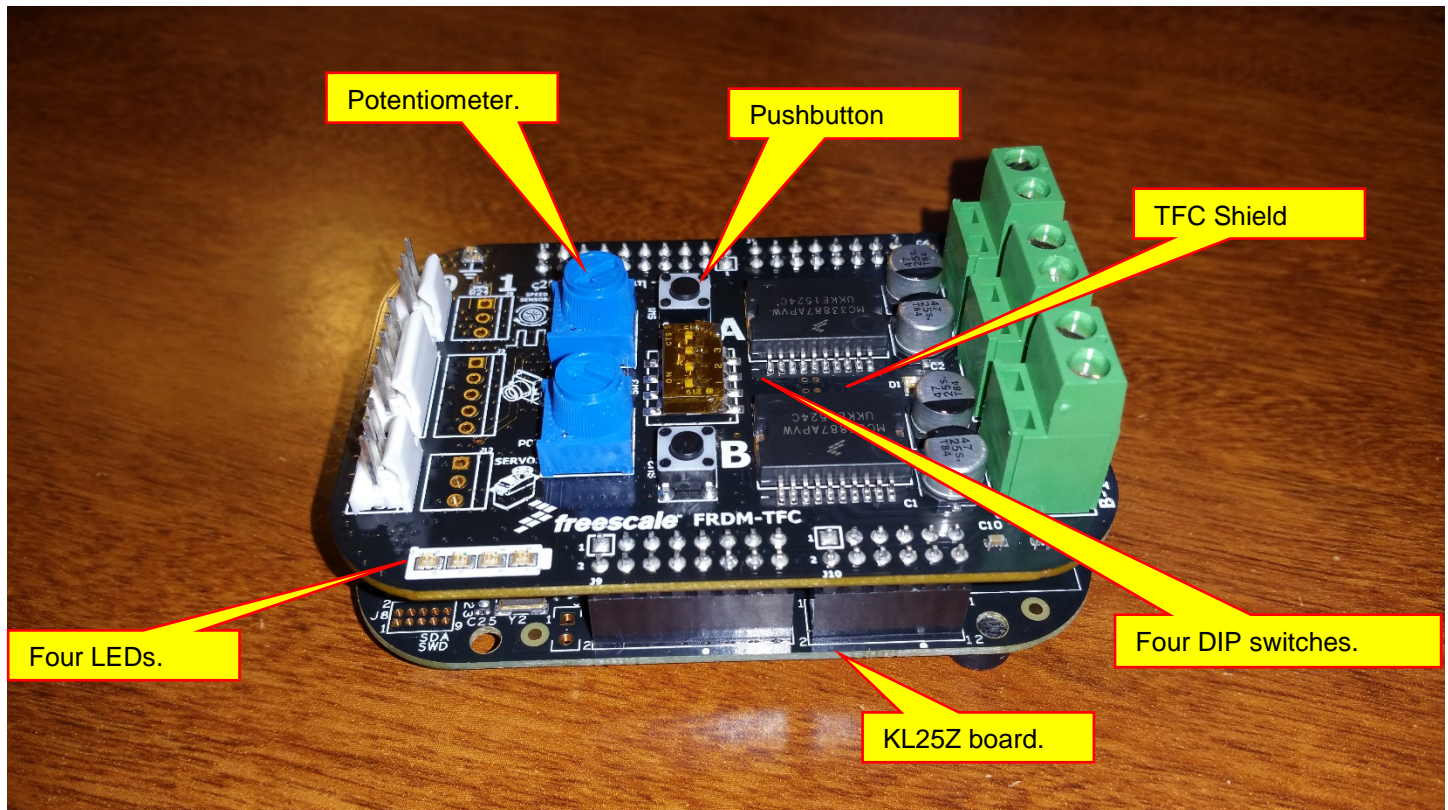
Exercise II-3: List two blocks in the Simulink / Commonly Used Blocks library that are not found in the Autonomous Vehicle Competition library. Specify what these blocks do?

Exercise II-4: List two blocks in the Simulink / Math Operations library that are not found in the Autonomous Vehicle Competition library. Specify what these blocks do?

Lesson III

Pushbuttons, Knobs, Dip Switches, and LEDs

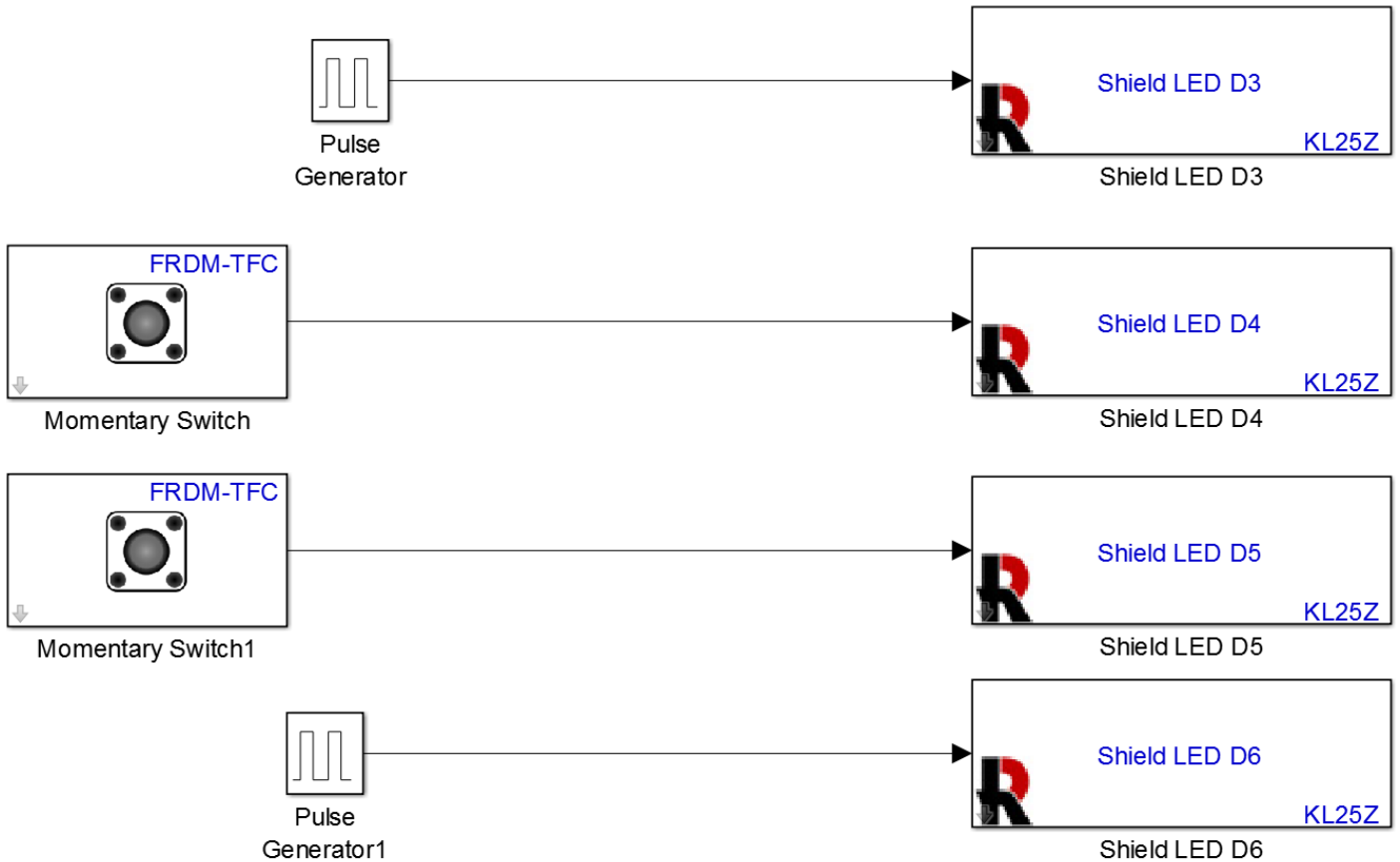
The Freescale Cup shield is a board that plugs into the KL25Z controller. We will refer to this board as the “TFC Shield.” In your kit, the two boards come packaged together:



The shield comes with two potentiometers (pots) for adjustable controls, two momentary pushbuttons, four DIP switches, and four LEDs. These devices are wired to the KL25Z board and we have Simulink blocks for using the devices. Note also that the shield has interfaces for the servo motor, drive motors, and linescan camera. The motors will be discussed in the next lab. Here, we will show how to use the switches, potentiometers, and LEDs. You will have the option of using these devices when you build the controller for your vehicle.

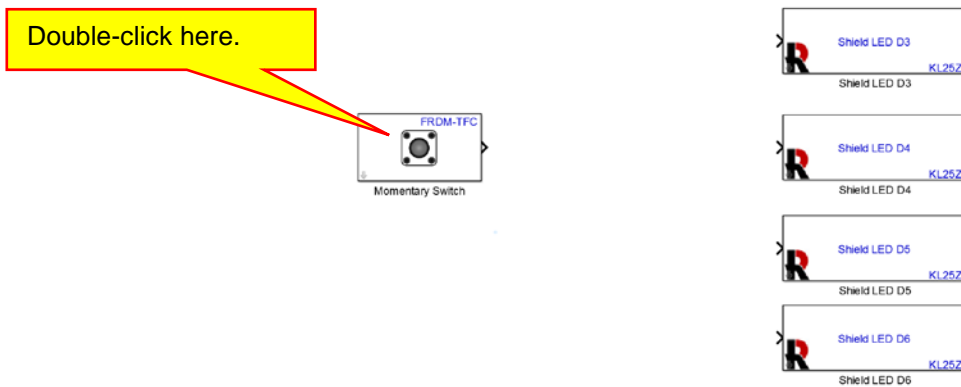
A. Pushbuttons and LEDs

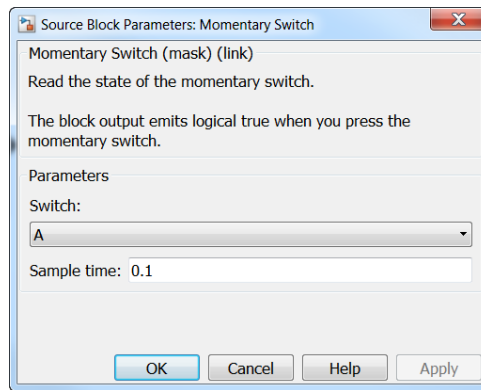
We will start with the model shown below. Remember, when creating a new model, use the Freescale Cup Companion app as shown on page 16:



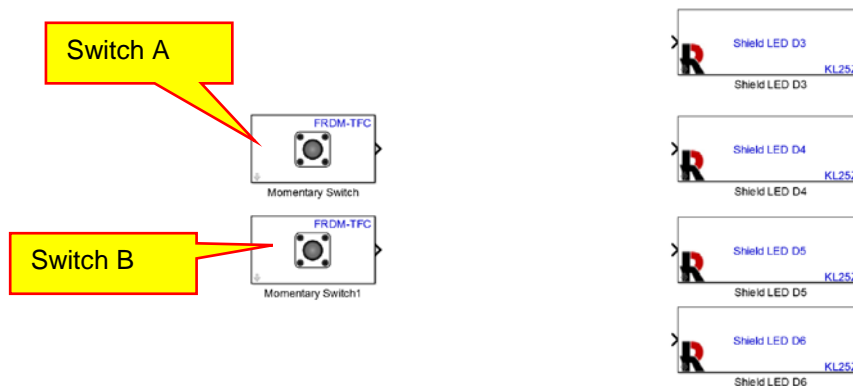
All of these blocks are located in the **Autonomous Vehicle Competition** library. There are four **Shield LED** blocks in the library, once for each LED on the shield. You will need to drag in four separate LED blocks to create this drawing. When the input to the block is a 1, the LED will turn on, when the input to the block is a zero, the LED will turn off.

There are two pushbuttons, A and B. Note that this type of switch is also referred to as a momentary switch, and the switch only changes its output while you are holding down the switch. When you release the switch, the output goes back to its original value. Drag in a **Momentary Switch** into your model. Double-click on the momentary switch block:



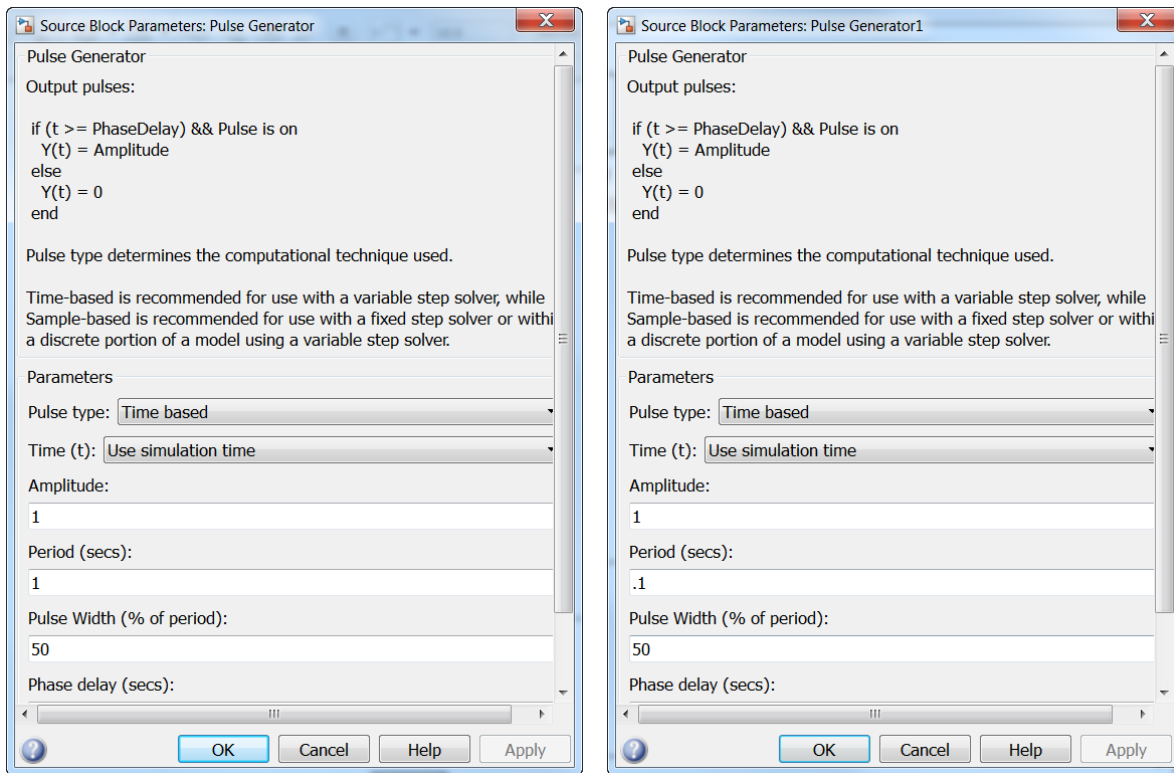


Note that we can choose between switch A and B, corresponding to the switches on the shield. Leave this block as Switch A. Note also that the sample time is set to 0.1 seconds. The sample time is how often the block is executed. In this case, executing the block reads the pushbutton switch. Thus, we will check the value of the pushbutton every 0.1 seconds. This is probably fast enough for a pushbutton. Note that if we make the sample time faster, we will not notice much of a difference, because, as humans, we probably can't press and release the button much faster than once every 0.1 seconds. If we make the sample time slower, say once a second, then we only check the pushbutton once every second. At this slow rate, the pushbutton could be pressed and released several times and the block would not notice that it was ever pressed. Thus, the sample time needs to be selected based on how fast we think the pushbutton can be pressed and released. For this application, 0.1 seconds will be fast enough. Place a second pushbutton in your model and change the switch to B.

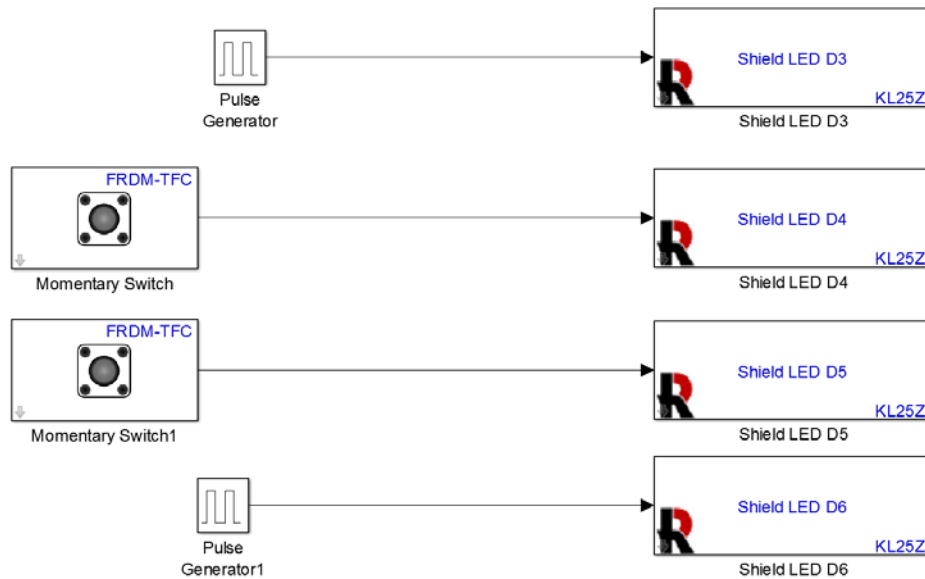


Note that the output of the momentary switch is zero when the pushbutton is not pressed. The output of the momentary switch is one when the pushbutton is pressed.

Next, place two pulsed sources in your model. Set the period of one source to 1 seconds (1 Hz frequency). Set the period of the second source to 0.1 seconds (10 Hz frequency). Set the duty cycle of both pulsed sources to 50 %. The parameters for the two sources is shown below:

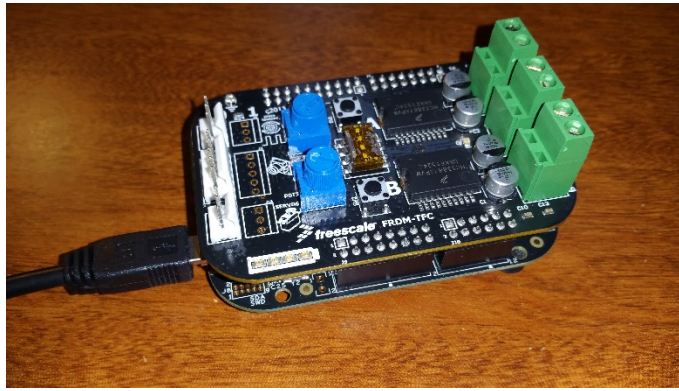


Wire the blocks as show:



With this model, LED D3 should blink at a 1 Hz rate (once per second). LED D4 should turn on when you press Momentary Switch A. LED D5 should turn on when you press Momentary Switch B. And, LED D6 should blink at a 10 Hz rate (10 times per second). **Make sure that you have one switch selected as switch A and the other switch selected as Switch B, or you will generate an error when building you model.**

Build and download your model to the KL25Z shield. Note that the USB cable still plugs into the SDA port on the KL25Z board:

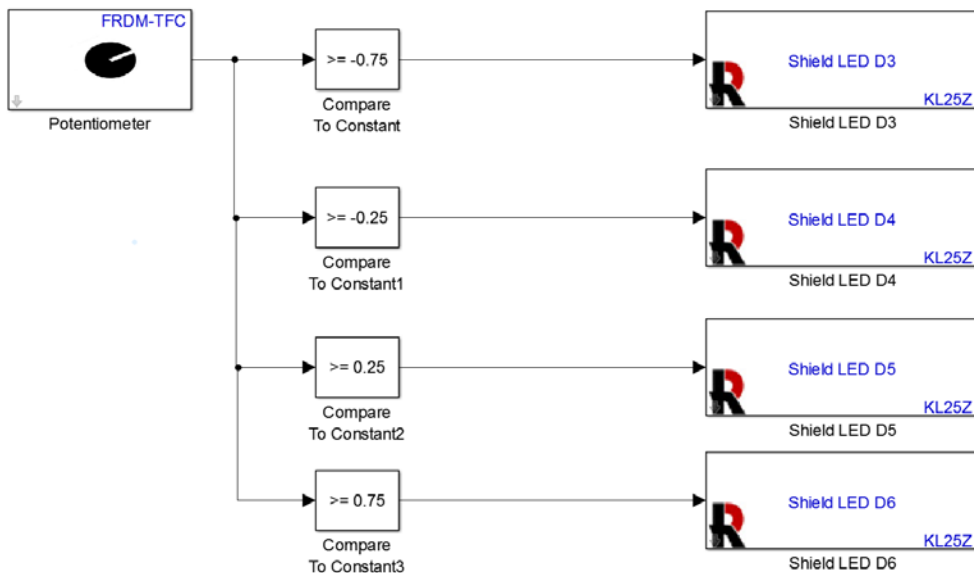


Verify the operation of your model:

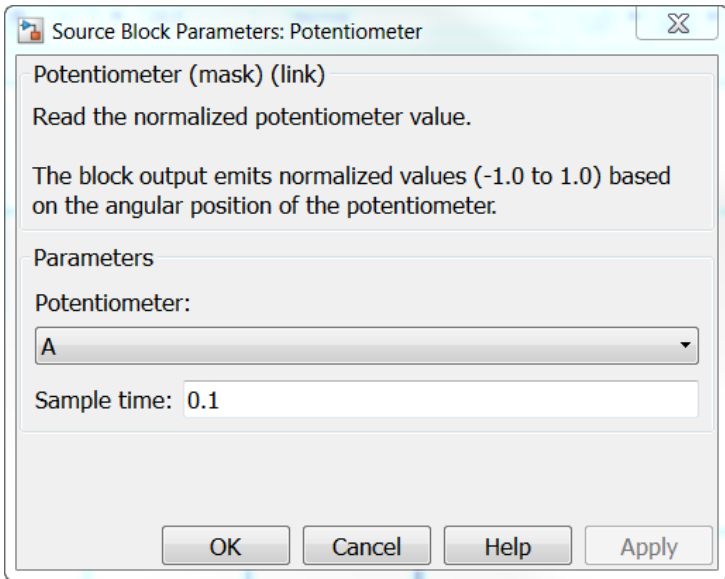
- LED D3 should blink at a 1 Hz rate (once per second).
- LED D4 should turn on when you press Momentary Switch A.
- LED D5 should turn on when you press Momentary Switch B.
- LED D6 should blink at a 10 Hz rate (10 times per second).

B. Potentiometers and Knobs

The shield has two potentiometers (abbreviated as “pot”) that output a variable signal between -1 and +1. We will use the model below:



All of the blocks have been discussed earlier except for the potentiometer. When you place the Potentiometer block in your model, you will need to select a sample time and the Potentiometer (A or B). For this example, you can use either A or B. Double-click on the potentiometer block and specify Potentiometer A and a sample time of 0.1 seconds:

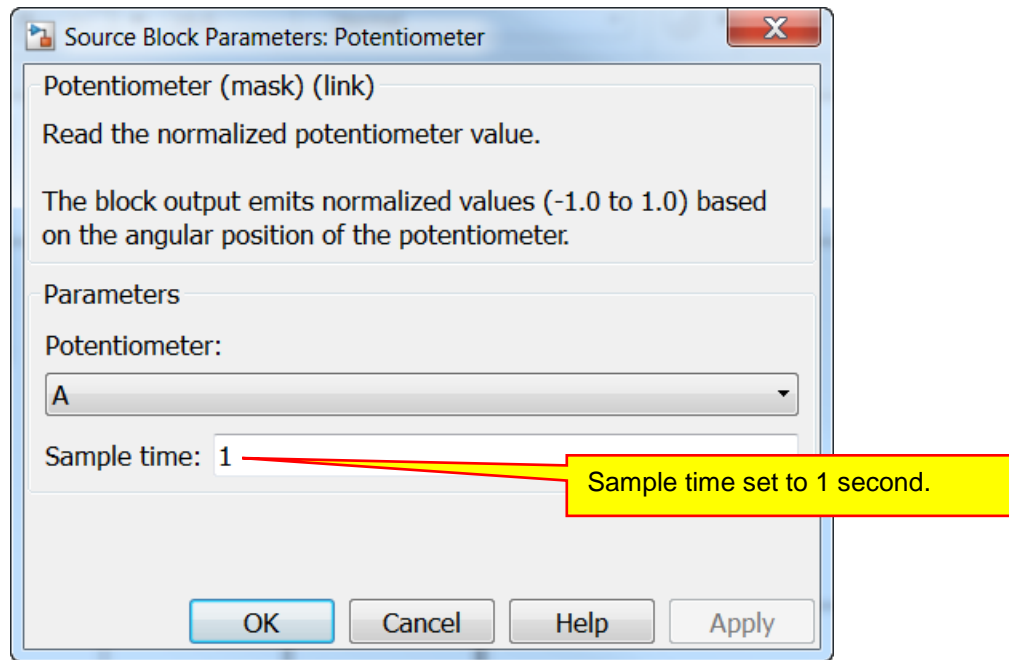


Note that the sample time is how often the value of the potentiometers is read. Since the value of this device can change as we turn the knob, a sample time of 0.1 may or may not be fast enough.

This model does the following. We will start with the knob turned all the way clockwise. In this position, the potentiometer block outputs a -1. As we turn the knob counter-clockwise, the output becomes more positive. When the output is greater than or equal to -0.75, LED D3 turns on. When the output is greater than or equal to -0.25, LED D3 and LED D4 turn on. When the output is greater than or equal to +0.25, LED D3, LED D4, and LED D5 turn on. When the output is greater than or equal to +0.75, all of the LEDs will turn on.

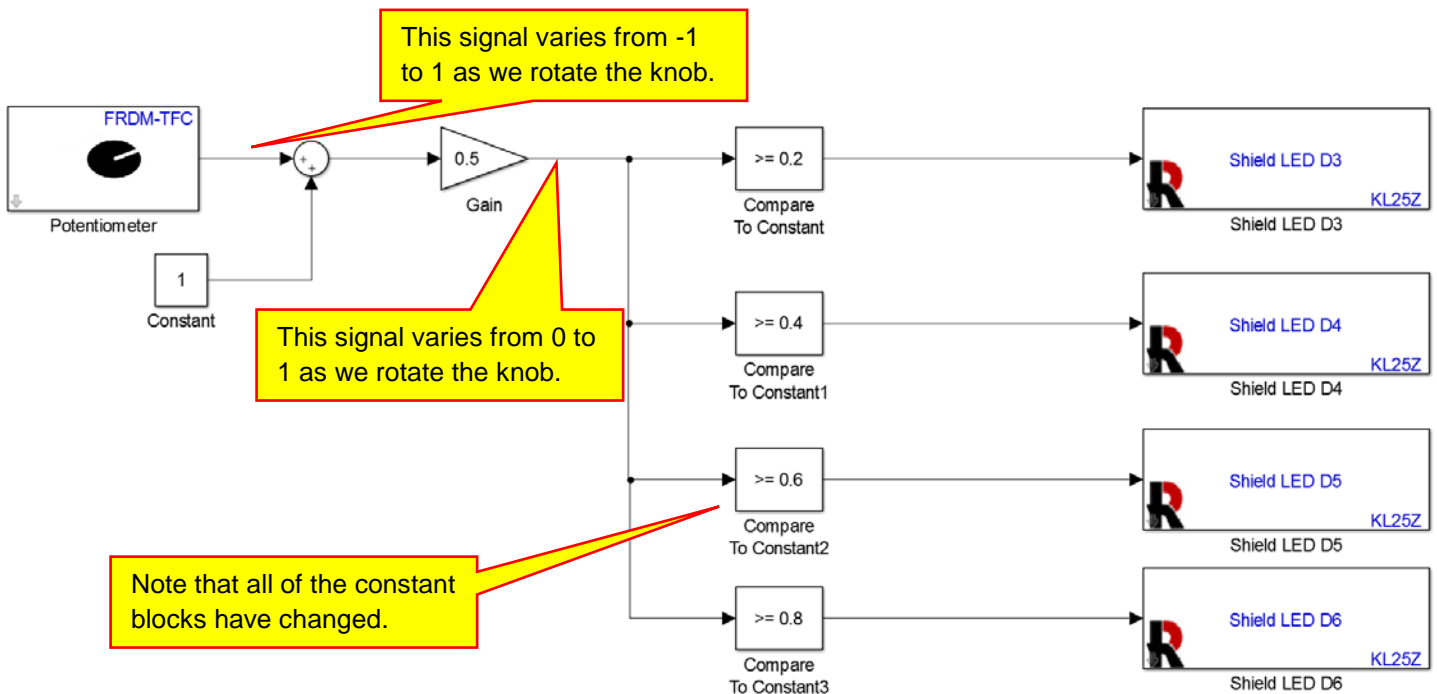
Build and download the model and verify that the LEDs turn on and off as you vary the knob of potentiometer A. Note that since the sample time is so fast (0.1 seconds), the knob and LEDs appear to turn on and off instantaneously as we vary the knob.

As a second example, double-click on the potentiometer block and change the sample time to 1 second:



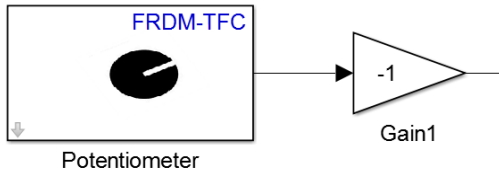
Build and download the model. With this slower sample time, you will notice that there is a large delay between when you move the knob and when the LEDs change. This is because the value of the potentiometer block is only read once a second, and we can change the position of the knob much faster than that. Thus, the choice of a sample time is based on how fast you need the microcontroller to respond. In this case, a sample time of 0.1 seconds is more appropriate for reading the potentiometer.

The potentiometer outputs a value between -1 and +1. As we rotate the knob, the value changes within this range. This range is actually a little strange for control systems and adjustments. Instead, we would like the output to vary between 0 and 1 as we fully rotate the knob. The model below accomplishes this task:

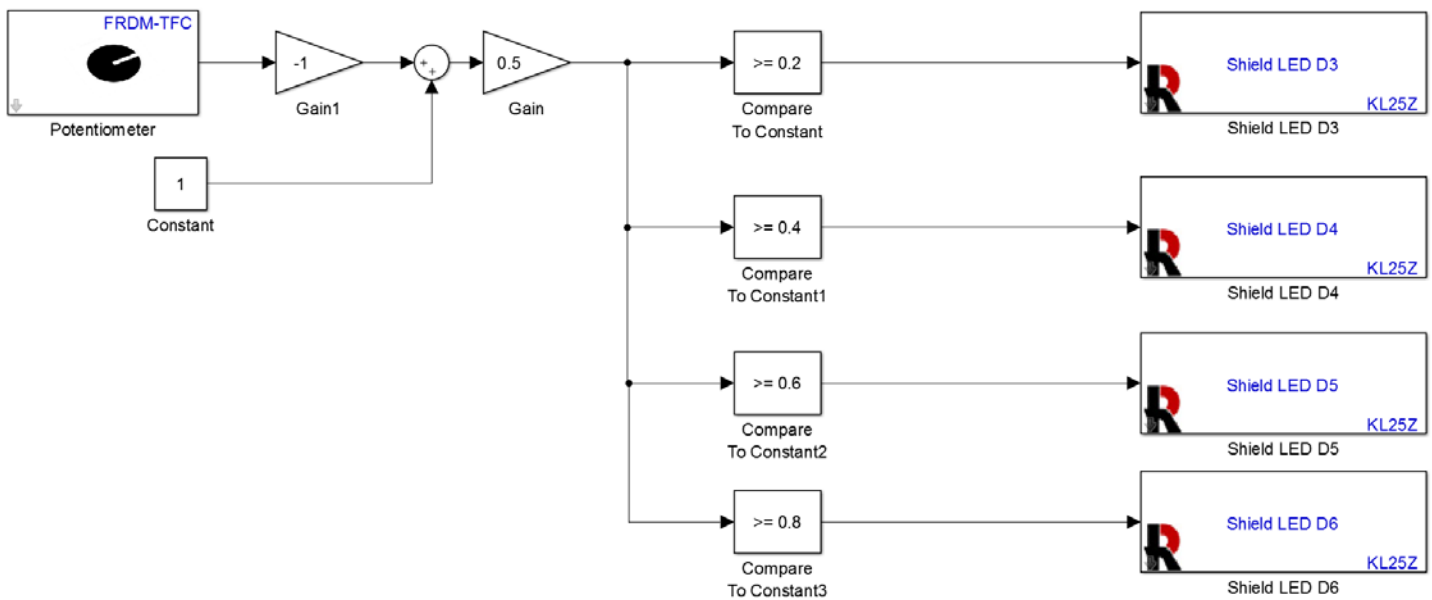


The potentiometer block outputs a signal from -1 to +1. When we add 1 to that signal, the new value varies from 0 to 2. We then divide that signal by two so that the final signal varies from 0 to 1.

We will make one last change to this model. We note that the knob behaves backwards compared to standard convention. That is, as we turn the knob clockwise, the value of the potentiometer decreases. This is the opposite of most controls which increase as we turn the knob clockwise. (I realize this is old fashioned. We no longer have knobs. We only have touchscreens with areas on the screen to mimic buttons...) To fix this problem, we will multiply the potentiometer signal by -1:



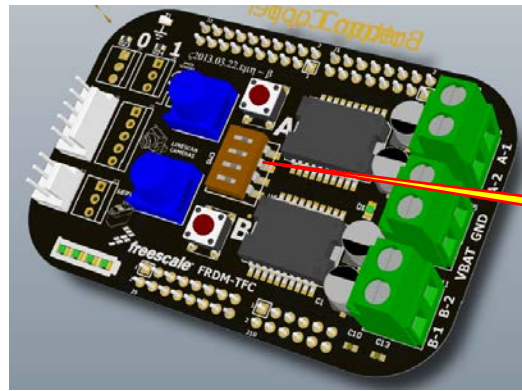
The potentiometer outputs a signal from -1 to +1. When we multiply this by -1, the result is a signal that goes from +1 to -1 (effectively changing the direction of rotation for the knob). The remainder of the model is the same as the previous model that was scaled to work from 0 to 1:



Build, download, and demonstrate this model. The more lights should turn on as the knob is turned clockwise.

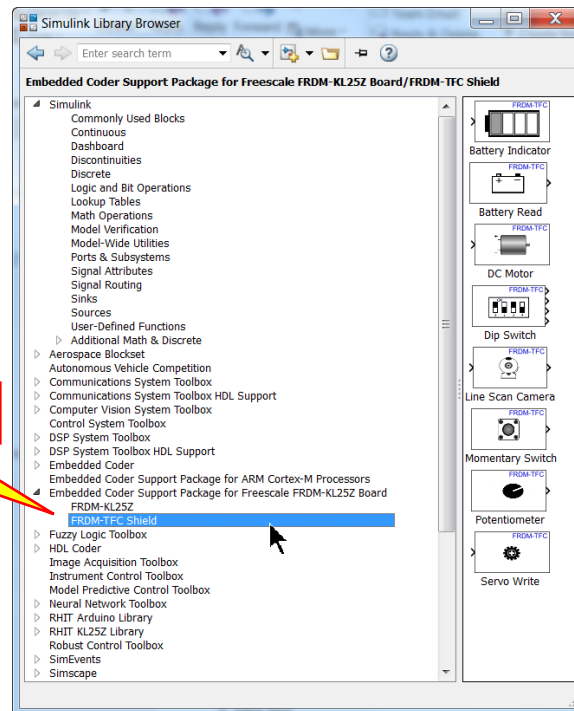
C. Dip Switches

The Freescale Cup shield has four dip switches as shown below:



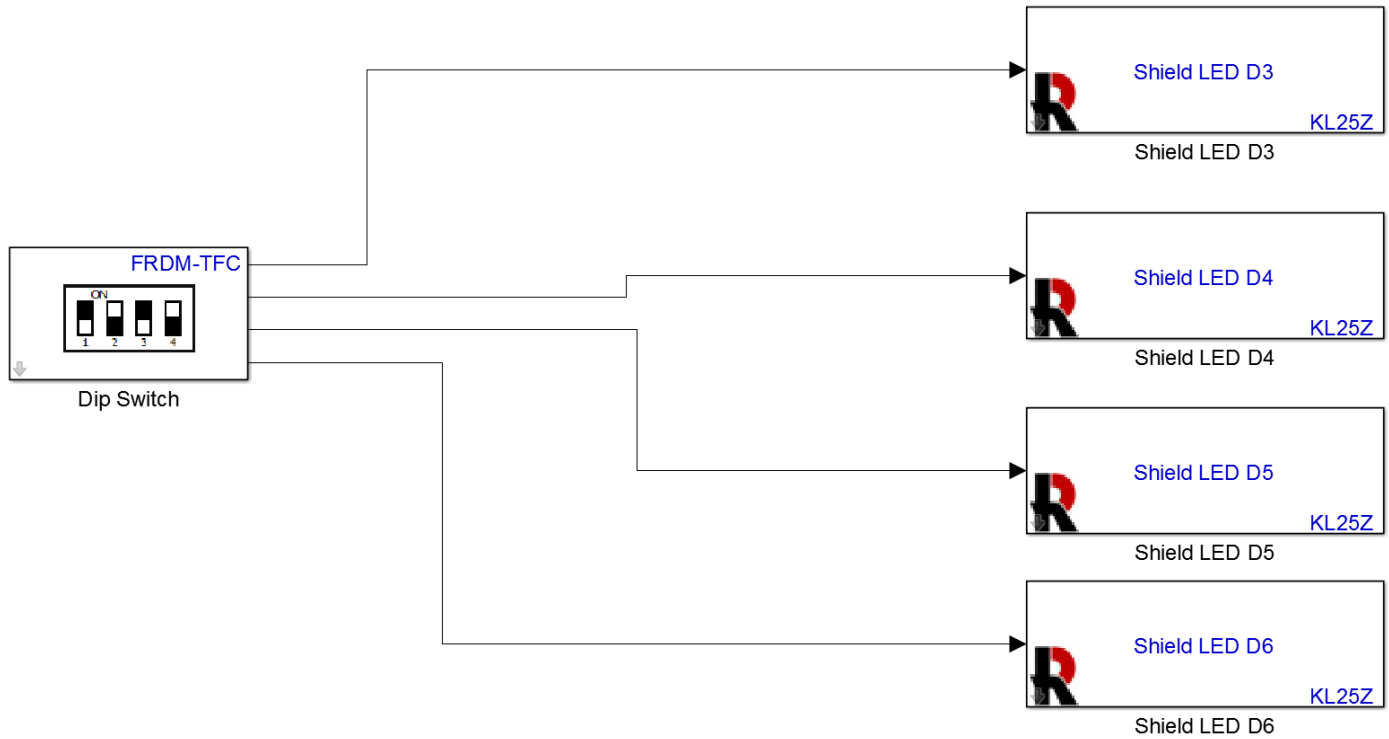
Four DIP switches.

The switches can be changed with a small screwdriver. These switches are not as easy to change their value as with the pushbuttons, so they are typically used in applications where their position is not changed frequently. However, they do provide four switches for our use, if needed. The DIP Switch block is located in the **Embedded Coder Support Package for Freescale FRDM-KL25Z Board / FRDM-TFC Shield Library**.



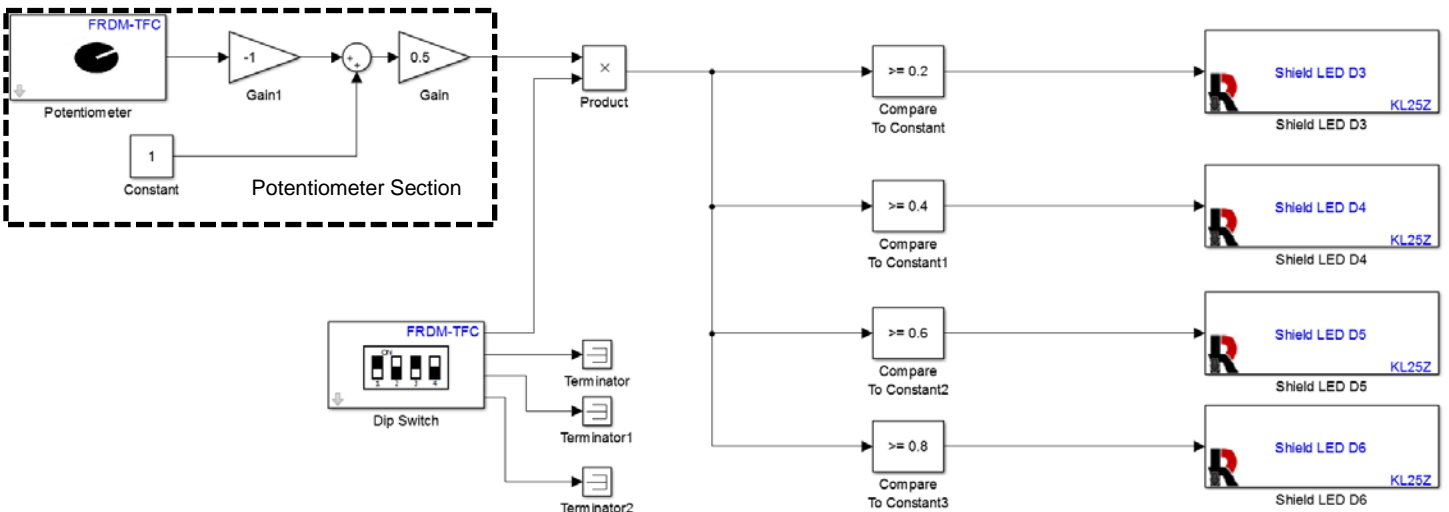
DIP Switch is in this library.

Create the model shown below:



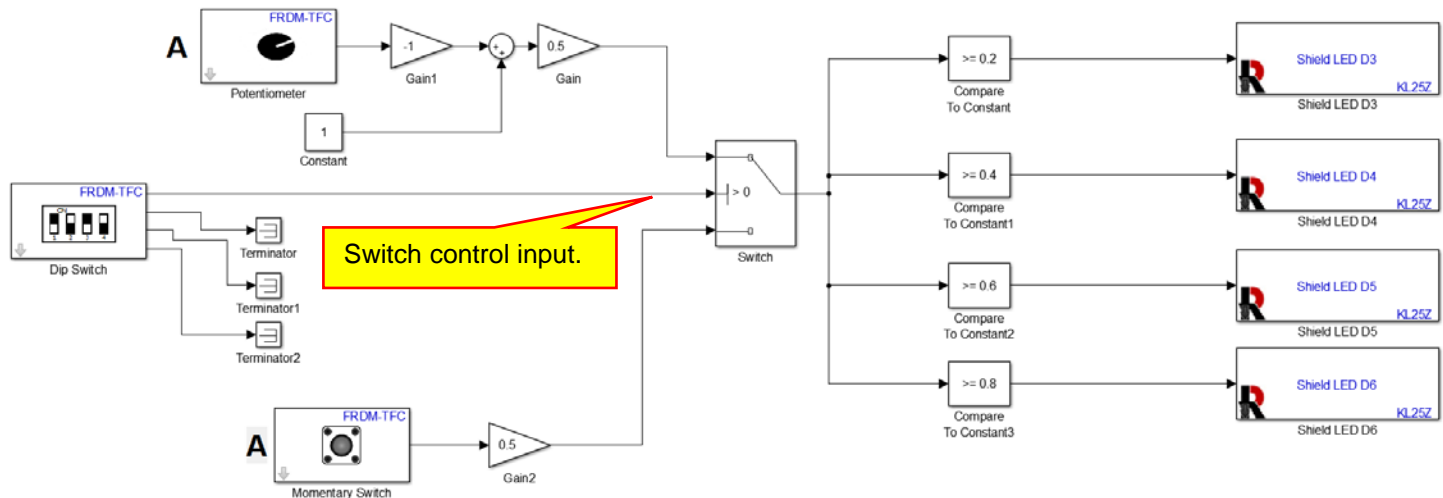
Note that the block has four outputs, one for each switch. The value of each switch is either a 0 or a 1, depending on the position of the switch. Typically, we will use these switches to enable or disable a part of our vehicle controller. Here, all we will do is turn on or off an LED based on the switch position. Build and download the model. Verify that each individual switch turns on and off a specific LED. Note that the DIP switch is covered with a small piece of plastic to keep it clean. Do not remove the plastic cover. You will need to poke through the cover to change the positions of the switches.

We will show a few examples of using a DIP switch for enabling or selecting functions in a controller. In the example below, position 1 of the DIP switch enables the potentiometer section of the model. The output of the DIP switch is a 0 or a 1. When the output is a zero, the output signal of the product block is a zero, and thus the potentiometer section has no effect on the remainder of the system. When the DIP switch output is a 1, the output of the product block is the same as the output from the potentiometer section, and acts as a “pass-through.” Thus, the DIP switch “enables” or “disables” the potentiometer section of the model in the sense that the function is used or not used by the model.



Build and download this model. Verify that when the switch output is a 1, adjusting the potentiometer turns on and off the LEDs based on the position of the potentiometer. Verify that when the switch output is a 0, adjusting the potentiometer has no effect on the LEDs and that all of the LED's will always be off.

As a second example, we will use the model shown below. This model used a **Switch** block which is located in library **Simulink / Commonly Used Blocks** library:



In the **Switch** block, the threshold is specified as zero. If the switch control input (shown above) is greater than 0, the output of the switch is the top input. (Or, the output of the switch block is the potentiometer control.) If the switch control input is **not** greater than 0, the output of the switch is the bottom input. (Or, the output of the switch block is the **Momentary Switch** control.) Thus, the DIP switch in this example allows us to choose between two different methods of controlling the LEDs. Build and download this model to you KL25Z. Verify that when the switch output is a 1, adjusting the potentiometer turns on and off the LEDs based on the position of the potentiometer, and that the momentary switch has no effect on the LEDs. Verify that when the switch output is a 0, adjusting the potentiometer has no effect on the LEDs and that two of the LED's will turn on and off as you press the Momentary Switch.

These two examples show how we would typically use the DIP switches. The switches are not made to be changed frequently as they will wear out quickly. However, we can use them to enable or disable functions, or switch between control methods as we would typically not change these functions very often. If you need to frequently change a switches' position, use the momentary switch.

D. Questions

Question III-1: What is a shield?

Question III-2: "Pot" is an abbreviation for what device on the TFC Shield?

Question III-3: How do you make a 10 Hz square wave with a Pulse Generator block?

Question III-4: What are the numerical values that turn on and off the Shield LEDs?

Question III-5: What is a synonym for "momentary switch?"

Question III-6: What affect does the Sample Time in the Momentary Switch block have on the operation of the momentary switch?

Question III-7: What is the output signal range of a potentiometer block on the TFC Shield?

Question III-8: What are DIP switches typically used for in a control system?

Question III-9: In what library is the DIP Switch block located?

Question III-10: What are the numerical output values of the DIP Switch block?

E. Exercises

Exercise III-1: Create a model that flashes LED D3 at 1 Hz, LED D4 at 2 Hz, LED D5 at 4 Hz, and LED D6 at 8 Hz. Make sure that you set the fixed time step to auto or to 0.005.

Exercise III-2: Use the touch slider to create the following model:

- When the slider output is greater than or equal to 25, LED D3 turns on.
- When the output is greater than or equal to 45, LED D3 and LED D4 turn on.
- When the output is greater than or equal to 65, LED D3, LED D4, and LED D5 turn on.
- When the output is greater than or equal to 85, all of the LEDs will turn on.
- You may have to adjust some of the thresholds slightly based on your Touch Slider Interface.

Exercise III-3: Create a model that does the following with a potentiometer.

- As the potentiometer is turned clockwise, a scaled signal goes from 0 to 3.5. For no rotation, the signal is zero. For full rotation, the signal is 3.5. (Note that we made a similar system that goes from 0 to 1.)
- When the scaled signal is less than 1, all of the LEDs in the RGB LED on the KL25Z board are off.
- When the scaled signal is greater than or equal to 1 and less than 2, the blue LED in the RGB LED on the KL25Z board is on. All other LEDs are off.
- When the scaled signal is greater than or equal to 2 and less than 3, the red LED in the RGB LED on the KL25Z board is on. All other LEDs are off.
- When the scaled signal is greater than or equal to 3, the green LED in the RGB LED on the KL25Z board is on. All other LEDs are off.

Exercise III-4: Create a model that does the following with a potentiometer.

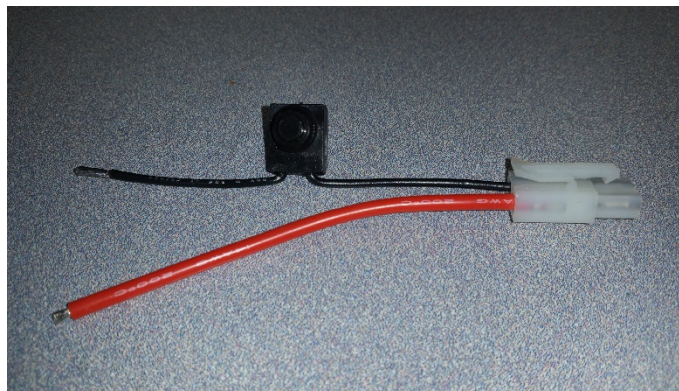
- As the potentiometer is turned clockwise, a scaled signal goes from 0 to 3.5. For no rotation, the signal is zero. For full rotation, the signal is 3.5. (Note that we made a similar system that goes from 0 to 1.)
- When the scaled signal is less than 1, all of the LEDs in the RGB LED on the KL25Z board are off.
- When the scaled signal is greater than or equal to 1 and less than 2, the blue LED in the RGB LED on the KL25Z board is on. All other LEDs are off.
- When the scaled signal is greater than or equal to 2 and less than 3, the red and blue LEDs in the RGB LED on the KL25Z board are on. The green LED is off.
- When the scaled signal is greater than or equal to 3, all LEDs in the RGB LED on the KL25Z board are on.

Lesson IV

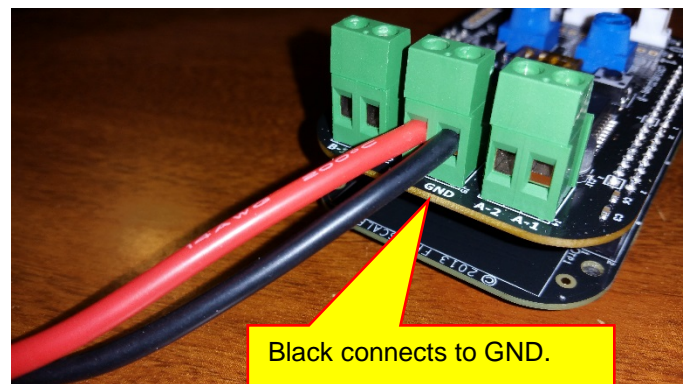
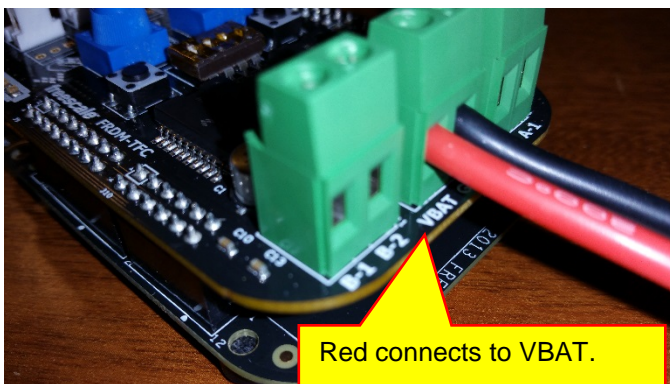
Servo Motor and Drive Motors

In this lesson we will learn how to control the servo motor and drive motors. Servo motors are position control motors. With these motors the signal we pass to it specifies the angle to which the motor turns. (The angular position.) With the drive motors, the signal we pass determines the direction (forward or backwards) and speed at which the motor turns. Both of these motors require that we connect the battery, so we will wire up the battery first. The battery powers the motors on the shield as they would draw too much power from the USB port on your computer. **You will probably need to charge your battery before doing this lesson.**

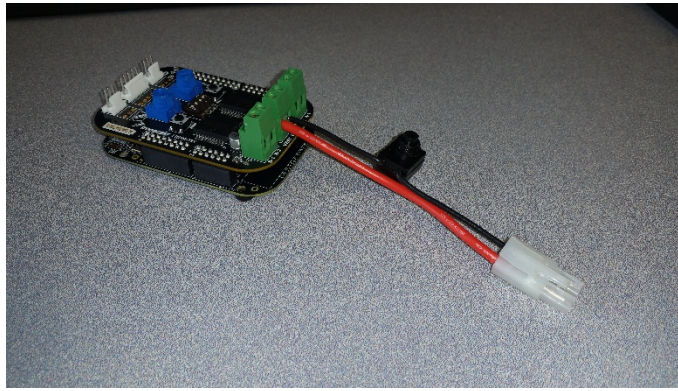
Locate the battery connector. It is shown below:



For this project, we will use the color code that red is always positive and black is ground (GND). Connect the battery as shown to your TFC Shield board. You will need to use a screw driver:



Note that RED connects to the VBAT terminal and black connects to the GND terminal. **(Do not reverse the two. Connecting these incorrectly will damage both boards.)** The connector properly wired is shown below:



Connect your battery to the connector. The connector is keyed so that it will only connect one way. Thus, if you wired your connector properly to the TFC shield, you cannot connect the battery incorrectly. The complete setup is shown below:



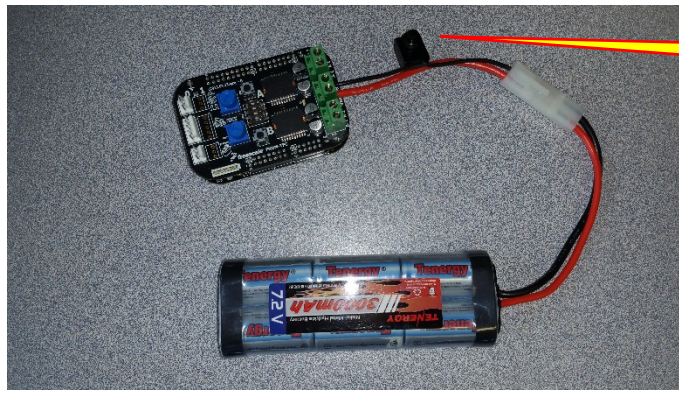
Note that the battery will power the KL25Z so your last model will be running on the microcontroller. Also note that with the battery connected, you can still connect your KL25Z to your computer through the USB port:



The battery will power the KL25Z and the TFC shield. However, you can still program the KL25Z from your computer.

A. Servo Motor

We will now connect the servo motor to the FTC shield and then control its position with a model on the KL25Z. First, unplug the battery:

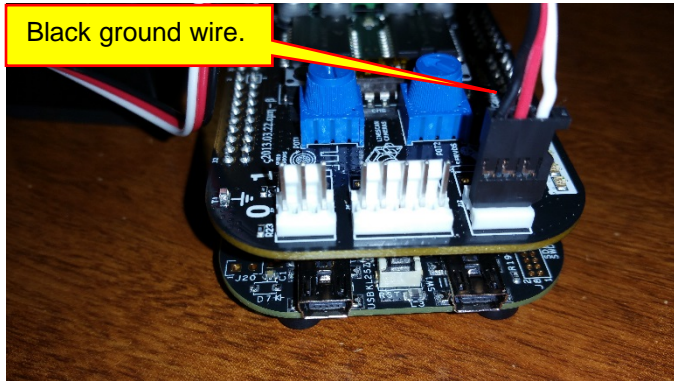


Switch off.

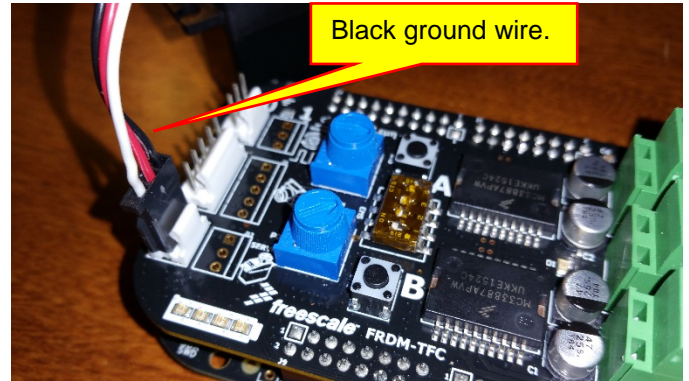
Next, locate the servo motor. Note that it has a 3-wire connector and locate the black wire on this connector:



Plug the 3-wire connector into the TFC shield as shown:

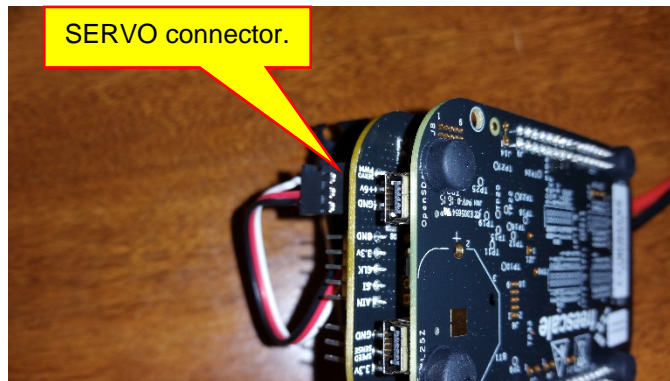


Black ground wire.



Black ground wire.

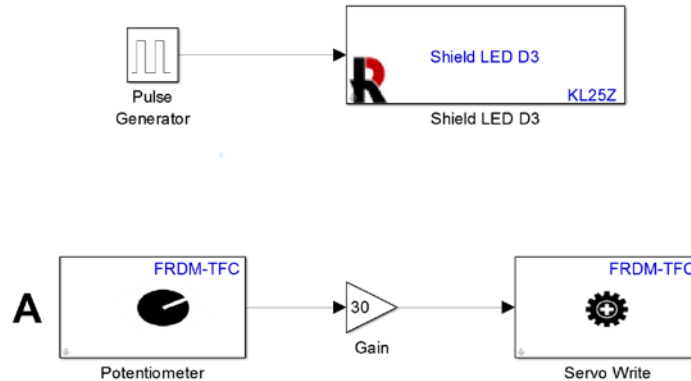
If you look on the underside of the TFC Shield, you will notice that the connector we are using is labeled as SERVO PWM and that the black wire of the plug is connected to the pin labeled GND:



SERVO connector.

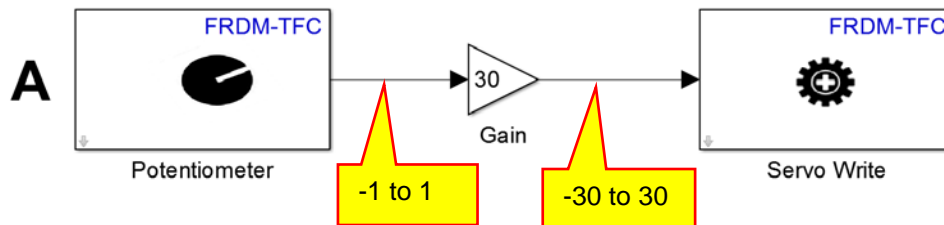
We now have the servo connector connected to the TFC shield. We can now control the servo motor with the KL25Z microcontroller. We will create a simple controller to vary the position of the servo motor with one of the potentiometers.

Create a new model using the Freescale Cup Companion. (See page 16 for the procedure.)
Create the mode shown below:

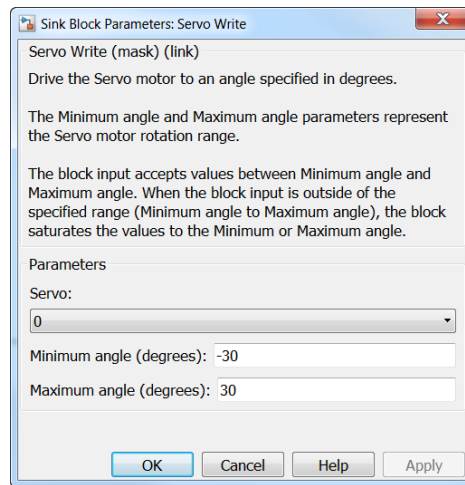


The pulse generator has a period of one second and a pulse width of 50%. (This is a 1 Hz square wave.) This waveform pulses LED D3 on and off. We will use this as a “Board Alive” light. Whenever we see LED D3 pulsing on and off at 1 Hz, we know that the board is running. If the LED does not pulse on and off, we know that the model we created caused the microcontroller to stop running. (Referred to as a crash or the microcontroller is hanging.)

The part of the model below controls the servo motor.



Potentiometer A outputs a signal from -1 to 1 as the knob is rotated. The **Gain** block multiplies this signal by 30, so the output of the Gain block is -30 to 30 as the potentiometer knob is rotated. This is the signal input to the **Servo Write** block. The input to this block is the angle that the servo motor will turn to. Thus, as the knob is rotated, the servo motor will turn from -30 degrees to +30 degrees. Double-click on the **Servo Write** block and fill in the parameters as shown:



Note that we are using Servo 0 because we plugged our motor in the receptacle for Servo 0. Note also that there are limits on the angle for the motor. These are hard limits to protect the motor. For example, if my control system had an error and told the servo motor to turn to 381 degrees. Without any type of protection, the motor would attempt to turn this rotation. It could never reach this value, so the motor would be on all of the time and could possibly damage the motor. To prevent damage, the servo motor has limits that you can specify. In this case, the motor is limited to turning ± 30 degrees. If the input to the block is between ± 30 , the motor will turn to that position. If the input is outside that range, the block will tell the motor to turn to the specified limit. In this way, if our model specifies a bad signal, the block protects the motor by limiting the signal to a limit that we know is safe.

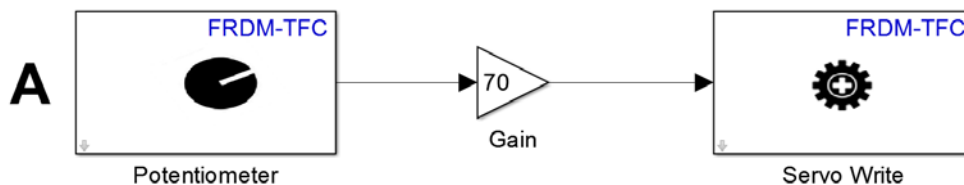
Click the **OK** button. Connect your KL25Z board to your computer using the USB cable. Make sure that you use the SDA port on the KL25Z. The battery does not have to be connected to program the KL25Z. Build and download the model to the KL25Z. Once the model downloads, connect your battery. You should verify the following:

- 1) LED D3 should flash at 1 HZ.
- 2) The servo motor should rotate as you rotate the knob for potentiometer A.

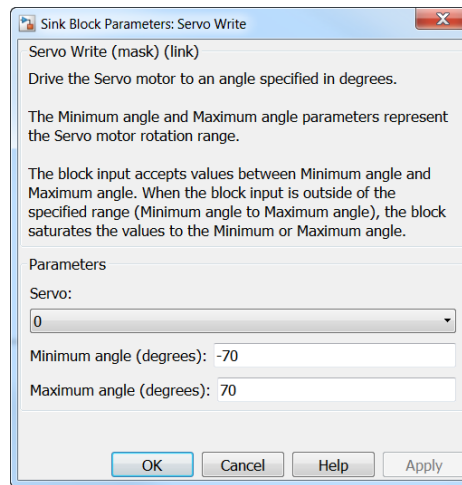
You will note a few things about the operation:

- 1) ± 30 degrees of rotation is not that much.
- 2) As you rotate the knob, the motor rotates in a very jerky fashion.

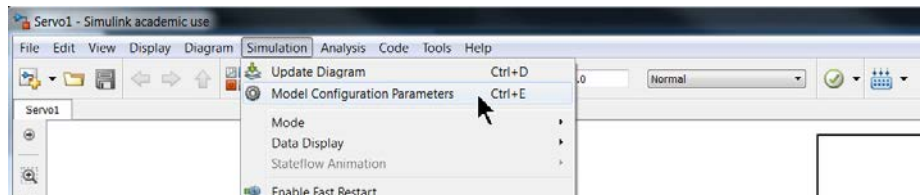
To get more rotation, we need to change the value of the **Gain** block and the angle limitations in the **Servo Write** block. Change the gain to 70:



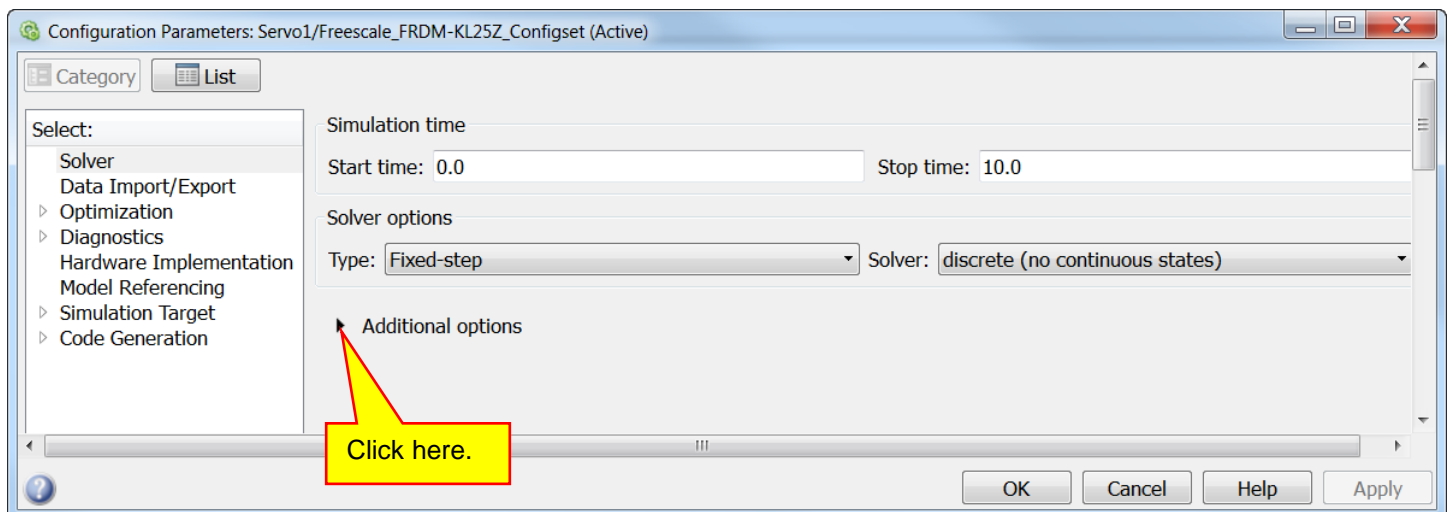
Change the limits in the **Servo Write** block to ± 70 :



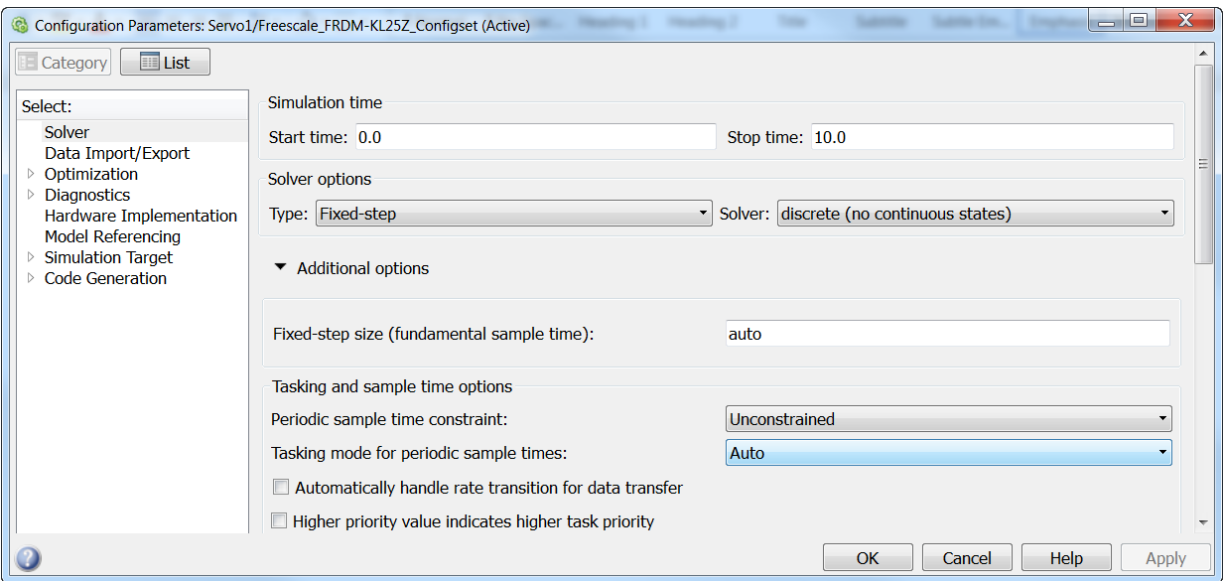
The jerky rotation is because the model has a large fixed time step. We did not set the fixed time step. When we create a model using the Freescale Cup Companion app, the time step is set to Auto, which means that Simulink picks the time step. In this case the time step is very large. To specify a time step, select **Simulation** and then **Model Configuration Parameters** from the Simulink menus:



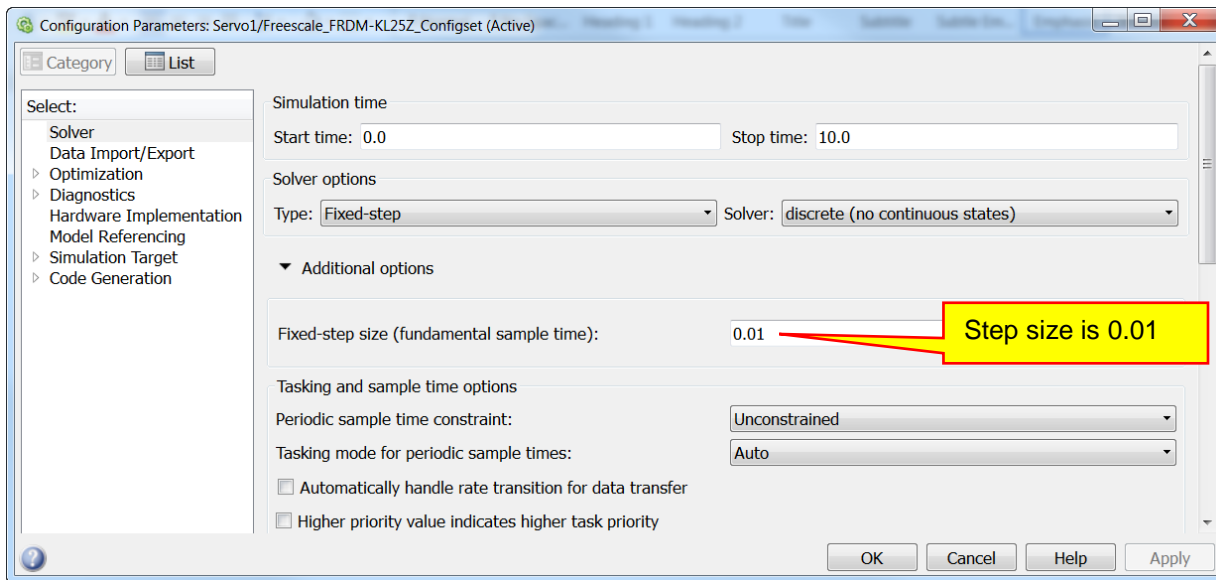
The dialog box below will open:



Click on **Additional Options** as shown:

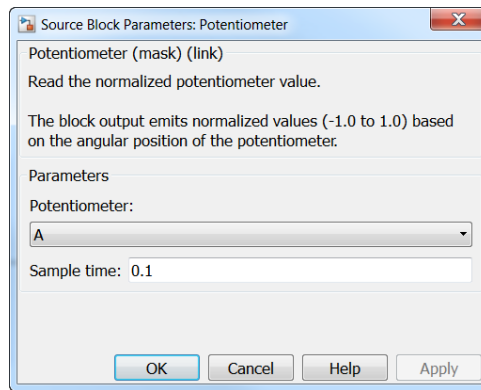


We see that the **Fixed-step size** is set to auto. We always want to specify a step size for our models. The step size is how fast the model runs. In our case, set the step size to 0.01 seconds:

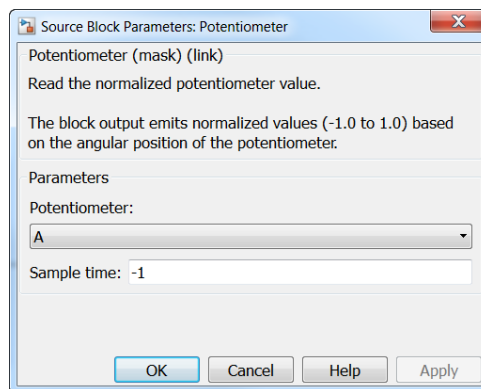


With this step size, every block in the model will execute once every 0.01 seconds. This means that the KL25Z will read the potentiometer and send a new signal to the servo motor every 0.01 seconds. Thus, an adjustment to the servo motor will be made every 0.01 seconds. For humans, this will appear instantaneous. That is, it will appear that the motor adjusts as soon as we turn the knob. In reality, it takes 0.01 seconds to make an adjustment. Click the **OK** button to accept the parameters.

We need to make one more change to the model. Double-click on the **Potentiometer** block:



Even though the model has a 0.01 second step size, this block only reads the potentiometer once every 0.1 seconds, which is too slow. To fix this, change the sample time to -1:



A sample time of -1 means that the sample time is “inherited.” In this case, the sample time will be inherited from the model which has a fixed-step time of 0.01 seconds. Thus, specifying that the sample time is inherited means that this block will run at the model step size of 0.01 seconds. (Note that we could have also specified a step time of 0.01 seconds in the block.)

Build and download the model to the KL25Z. You should notice the following:

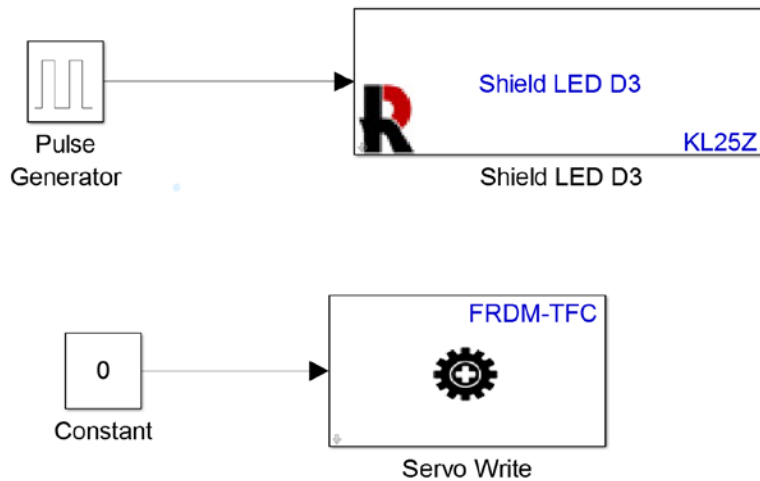
- 1) The motor turns through many more degrees of rotation (+/- 70 degrees).
- 2) It turns much more smoothly as you adjust the potentiometer knob.
- 3) You will also notice that if you make large changes in the potentiometer knob, the motor takes time to rotate to the specified position. This is the “slew rate” limitation of the motor and is as fast as it can turn. We will have to live with this limitation.

B. Servo Motor Zero

Before we continue to the next part, we want to zero the servo motor. This is necessary for when we build the vehicle. When we assemble the steering, we will adjust the tie rods assuming that the servo motor is at an angle of zero. If this is not true, the steering will have a large offset and it will be difficult to have the vehicle drive straight. To avoid this problem, we will set the servo motor angle to zero now.

Before continuing, run the previous model on the KL25Z and set the potentiometer fully clockwise or counterclockwise. (Rotate the knob to one of the limits.) The servo motor should turn all the way in one direction as you rotate the knob.

Save the previous model as Servo_Zero and change the model as shown below:

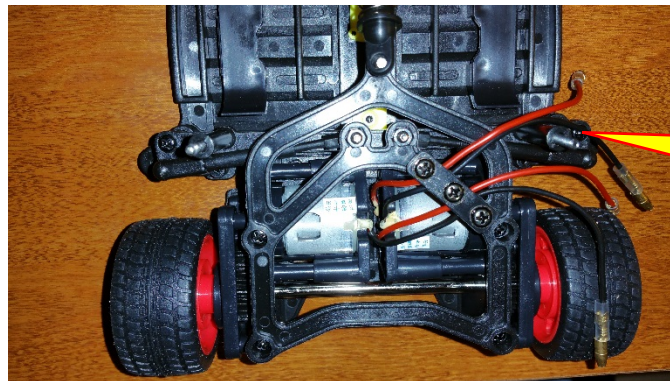


All this model does is flash LED D3 and set the servo motor angle to zero. Build and download the model. When the model runs, you should hear and see the motor turn to zero angle and then stay there. Hearing and seeing it move verifies that you have zeroed the motor.

After you hear the motor zero, unplug the battery and then unplug the servo motor from the TFC Shield. We will not need the servo motor again until we build the vehicle.

C. Drive Motors

We will now identify the right and left motors, verify which direction is forward and reverse, and learn how to use the motor drive blocks. We will work with the right motor first and then repeat the procedure for the left motor. First attempt to identify the wires for the right motor. Each motor should have one black wire and one red wire:



These two wires are for the right motor on my vehicle.

You should probably mark the wires with tape to identify right and left.

Next, find the connection wires for the motor. They should be in a small plastic bag:

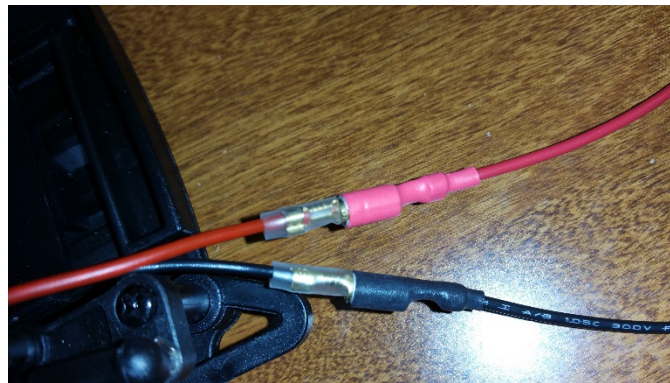


Take one red wire and one black wire and strip off about ¼ inch on insulation:



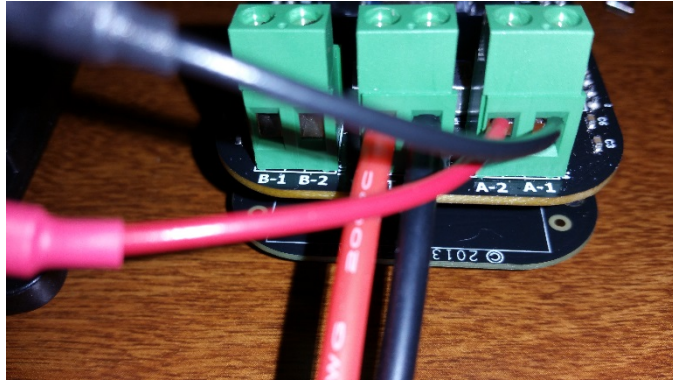
You might want to tin up these leads with solder if you have a soldering iron.

Connect the wires to the right motor as shown. The wires should snap on and should not be able to be removed easily:

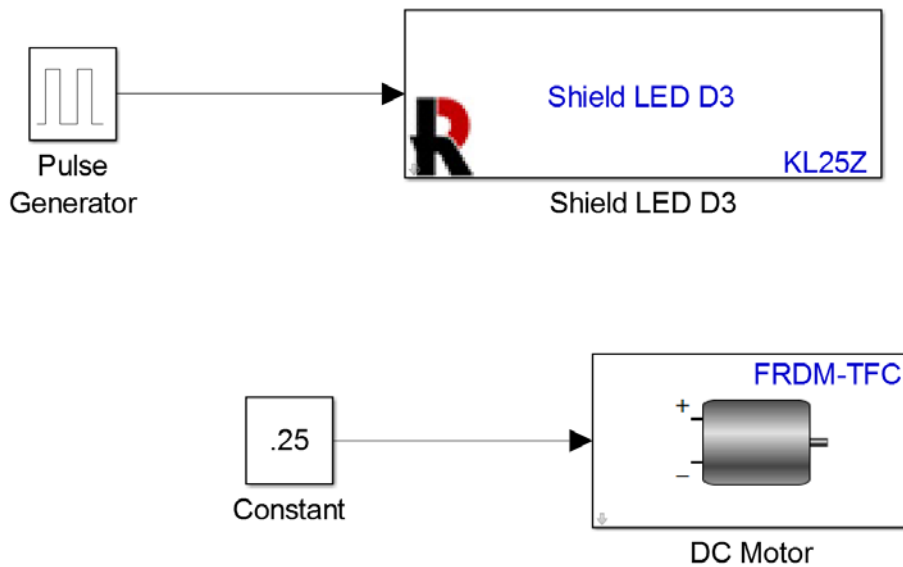


Next, make sure that the battery is disconnected. Connect the two wires to the A-2 and A-1 connectors on the TFC Shield. This may or may not be correct as we might need to flip which wire is connected to A-2 and A-1.

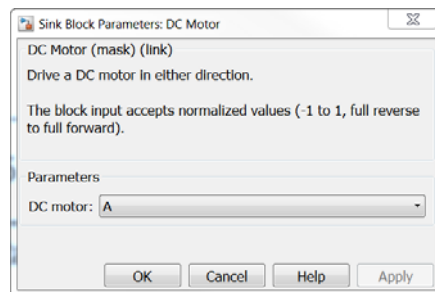
As a first attempt, we will connect red to A-2 and black to A-1:



We will now create a model to send a constant signal to motor A:

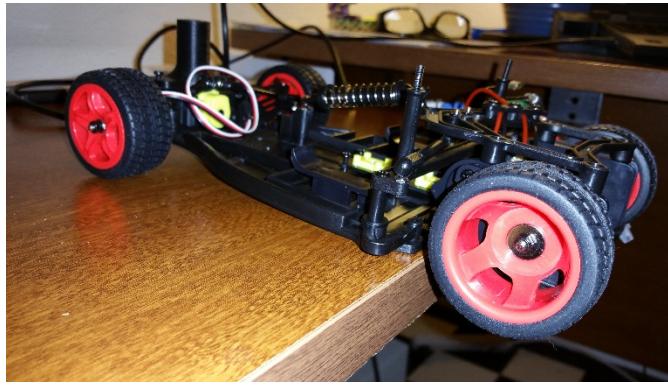


Double-click on the **DC Motor** block and make sure that it specifies DC motor A:

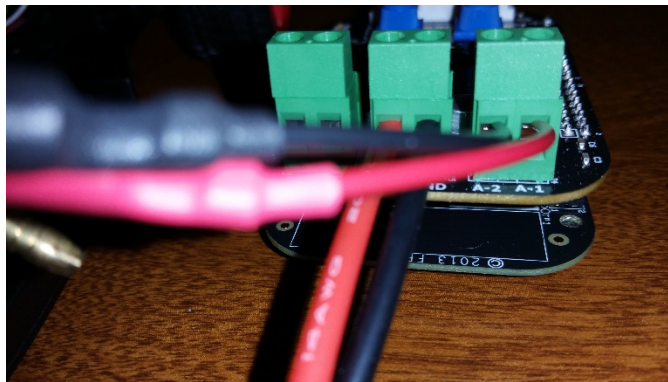


Connect your computer to the SDA port on the KL25Z board. Make sure that the battery is disconnected. Build and download the model.

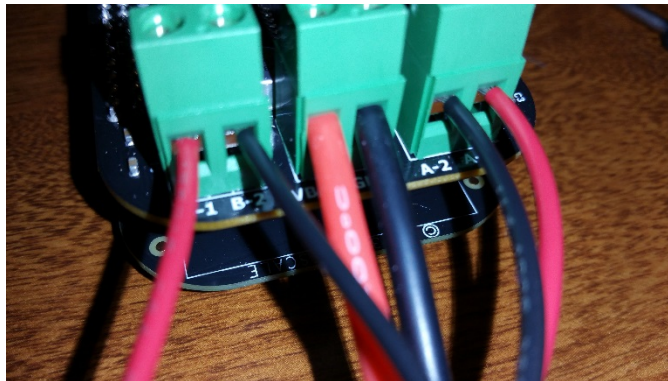
Next, let the rear wheels of the vehicle hang off the edge of the table so the wheels do not contact the table:



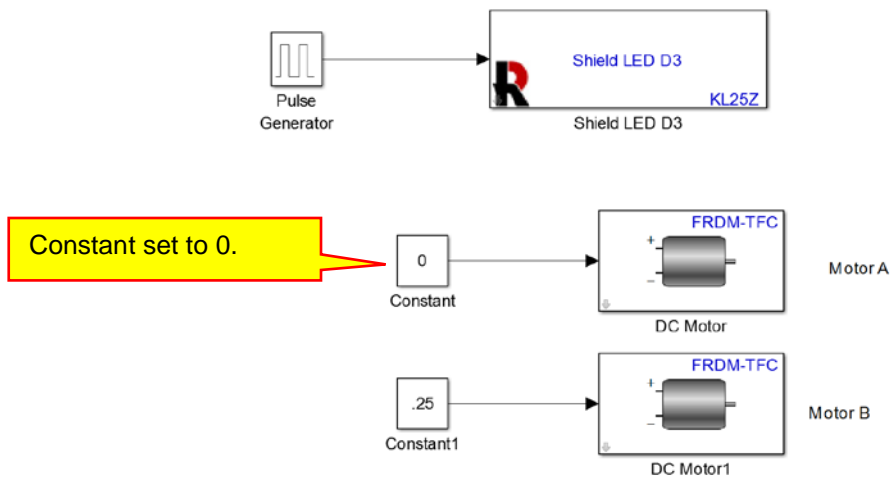
Plug in the battery. Determine if the wheels are spinning forwards or backwards. It appears that my wheels are spinning backwards, so I need to reverse the red and black wires. So, it appears that the red wire should connect to A-1 and the black wire to A-2:



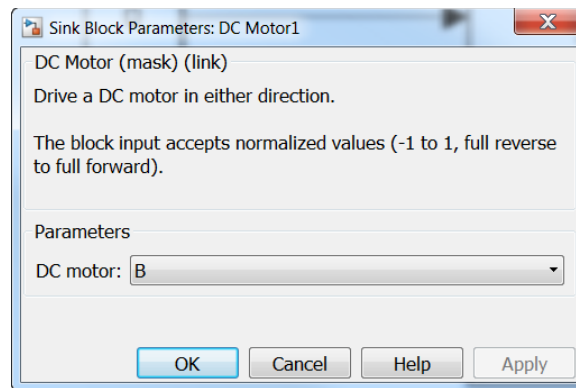
Repeat the same procedure for the left drive motor. Given what we learned from motor A, we will hook the black wire to B-2 and the red wire to B-1:



Next, add motor B to the model. Set the constant speed for motor A to 0:



Make sure that you double-click on the second DC motor and specify the motor as Motor B:

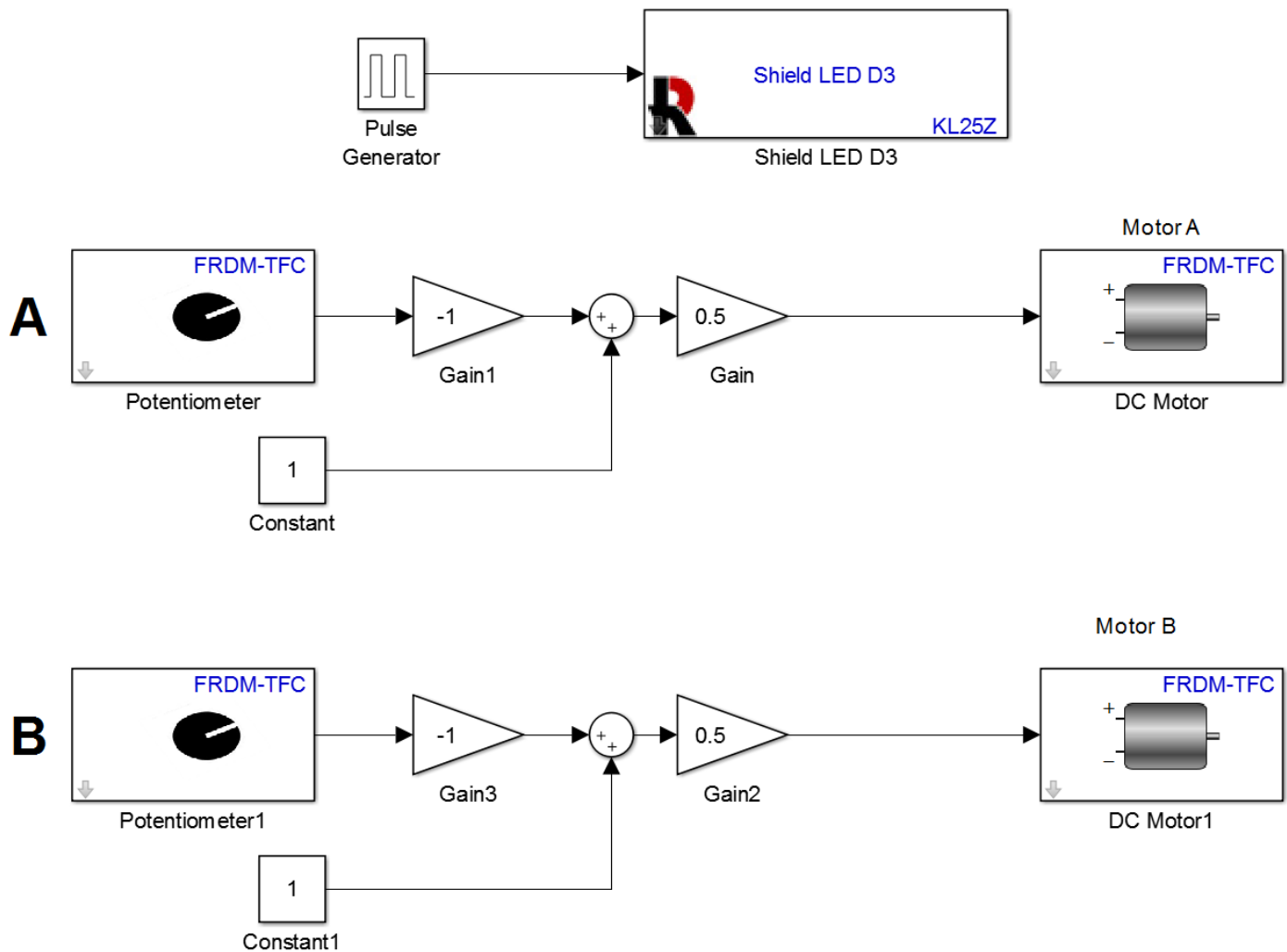


Build and download the model. Only motor B should spin. Make sure that it spins forward. If not, flip the red and black wires for Motor B. The wiring appears to be correct on my vehicle as the motor spins forward.

We have now determined the wiring for the right and left motors and forward direction. Record the wiring you have just figured out. We will need to take off the wiring to build the vehicle. If you do not record the wiring, you will need to figure it out all over again.

D. Motor Drive Example

As a last example, build the model below:



Note that the sample time for the potentiometer blocks should be set to -1. In this model, Potentiometer A controls the speed of the Motor A (the right motor), and Potentiometer B controls the speed of the Motor B (the left motor). Note that the motors will only go in the forward direction as the input signal to the DC Motor blocks is between 0 and 1. Build and demonstrate the operation of this model. **Make sure that the wheels of the car are not touching anything or the car chassis will fly off the table when you build the model!**

E. Questions

Question IV-1: What is the difference between a servo motor and a drive motor?

Question IV-2: Why is the battery required to run the servo motor and drive motor?

Question IV-3: On the battery connector, what color wire connects to the VBAT terminal? What color wire connects to the GND terminal?

Question IV-4: What does it mean when a connector is keyed?

Question IV-5: The black wire on the servo motor connects to which pin on the SERVO PWM connector?

Question IV-6: What is the purpose of having an LED on the board always flashing on and off at a 1 HZ rate?

Question IV-7: What does a Gain block do?

Question IV-8: Why are there maximum and minimum angle limits on the Servo Write block?

Question IV-9: What is the function of the Fixed-step size in a model?

Question IV-10: What is the effect of the Step Size in the Potentiometer block?

Question IV-11: Why do we need to zero the Servo Motor?

Question IV-12: When using a drive motor, why do we turn off power from the battery when we program the KL25Z board?

F. Exercises

Exercise IV-1: Using a Sine Wave block, create a model that rotates the Servo Motor angle back and forth between -50 and +50 degrees at a frequency of 1 Hz. Note that frequency in rad/sec is $2\pi F$, where F is the frequency in Hz. (So for a 1 HZ frequency, we need a radial frequency of 2π rad/sec.) The Sine Wave block is in the Simulink / Sources library. Set the sample time to -1 and the fixed-step time to 0.01.

Exercise IV-2: Using a Pulsed Generator block, create a model that rotates the Servo Motor angle back and forth between -30 and +30 degrees at a frequency of 1 Hz. Note that you can change a sum block to a difference block by changing the signs. (Double-click on the sum block and see the description.)


Exercise IV-3: Using a Sine Wave block, create a model that rotates both DC Motor speeds between -0.5 and +0.5 at a frequency of 0.25 Hz. Note that frequency in rad/sec is $2\pi F$, where F is the frequency in Hz. (So for a 0.25 HZ frequency, we need a radial frequency of 0.5π rad/sec.) The Sine Wave block is in the Simulink / Sources library.

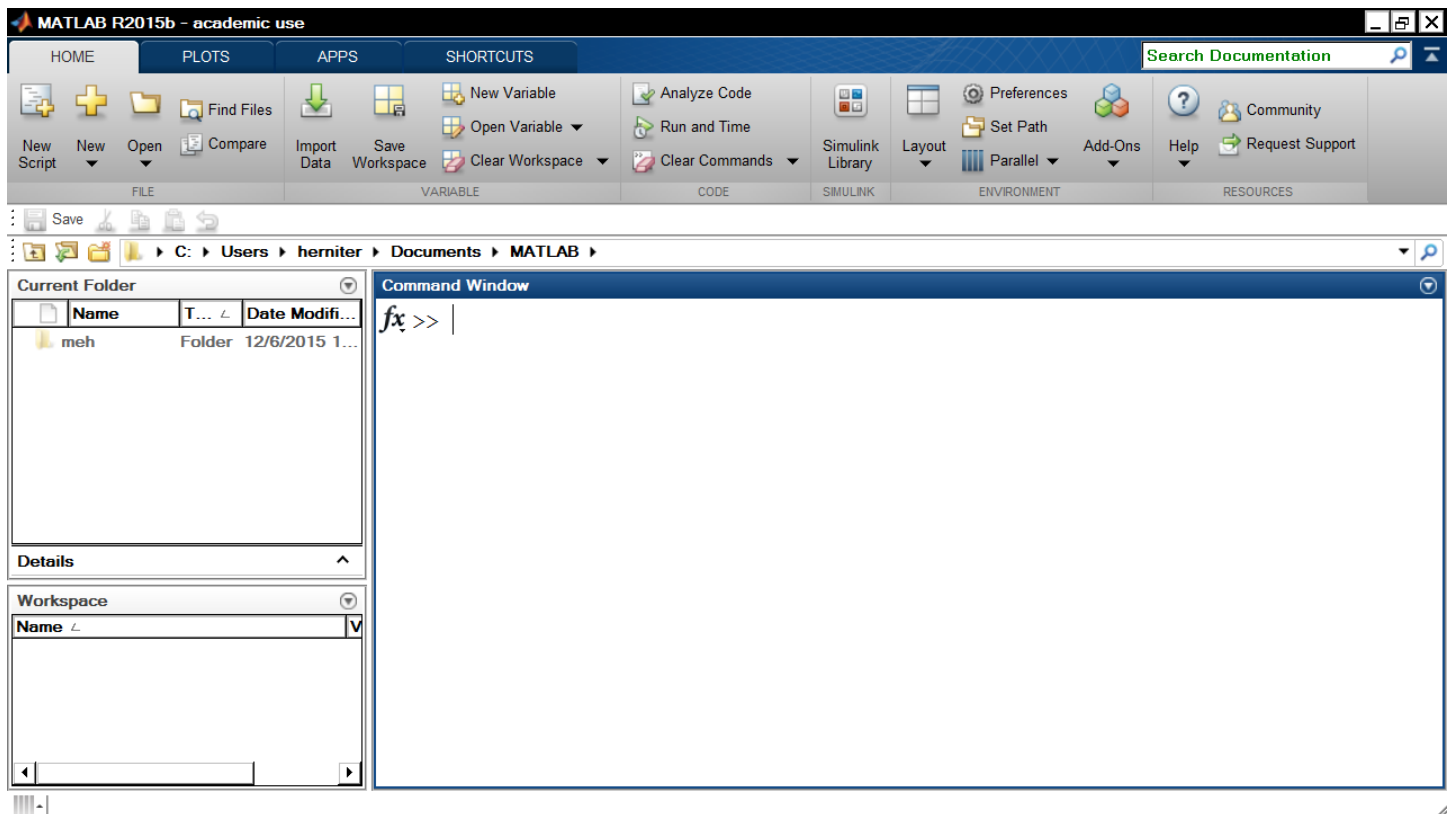
Exercise IV-4: Create a model that uses a single potentiometer to control the speed of both motors. When the potentiometer value is between 0 and 1, the right motor speed varies between 0 and 50% forward. (0 to 0.5 for the DC Motor block input.) At this time the left motor is at zero speed. When the Potentiometer value is between 0 and -1, the left motor speed varies between 0 and 50% forward. (0 to 0.5 for the DC Motor block input.) At this time the right motor is at zero speed. (You should be able to do the logic with product blocks and compare to constant blocks, or with Switch blocks.)

Lesson V: Introduction to MATLAB

A. Introduction

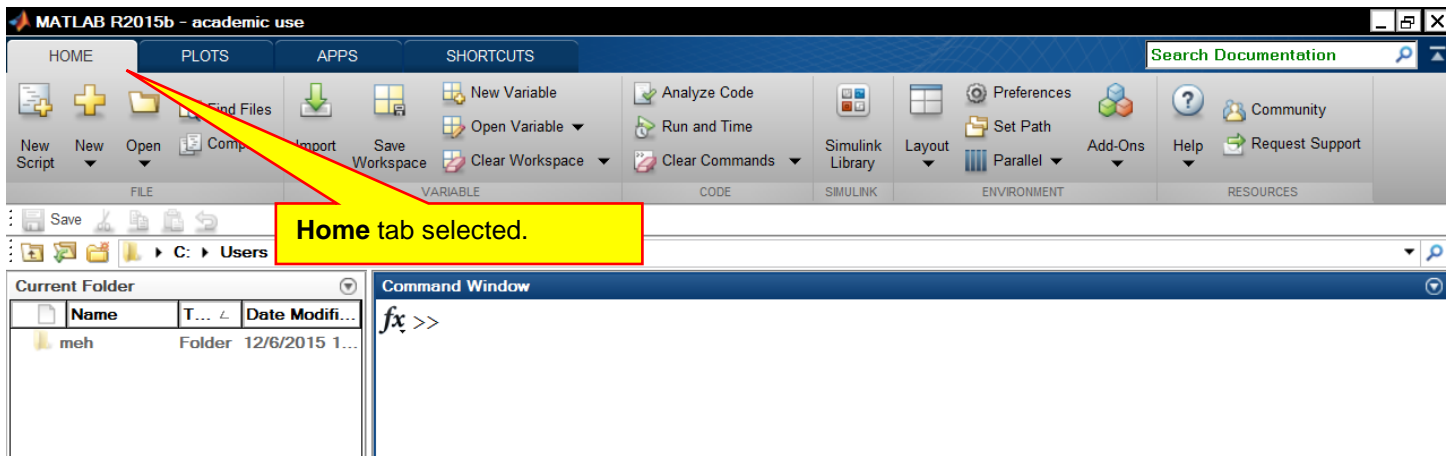
MATLAB and Simulink are industry standard tools used across multiple disciplines including electrical engineering, mechanical engineering, aerospace engineering, biological systems, medical systems, weather systems, economic system, mathematics, and much more. We will be using this tool because, no matter what STEM field (science, technology, engineering, and math) you pursue, you are likely to use MATLAB and Simulink for analysis and design. In fact, these tool are used in many other fields outside of the STEM fields. These tools are taught extensively in secondary education and used throughout industry. For us, we will focus on using the tool for control design and data visualization as related to designing an autonomous vehicle.

Start MATLAB by clicking on the MATLAB icon (). Note that this manual is designed for MATLAB R2015b¹. In Windows, this is located in the start menu and maybe on your desktop. In MAC OSX, the same icon should be located in the Launchpad. MATLAB will open with a window similar to the one below:

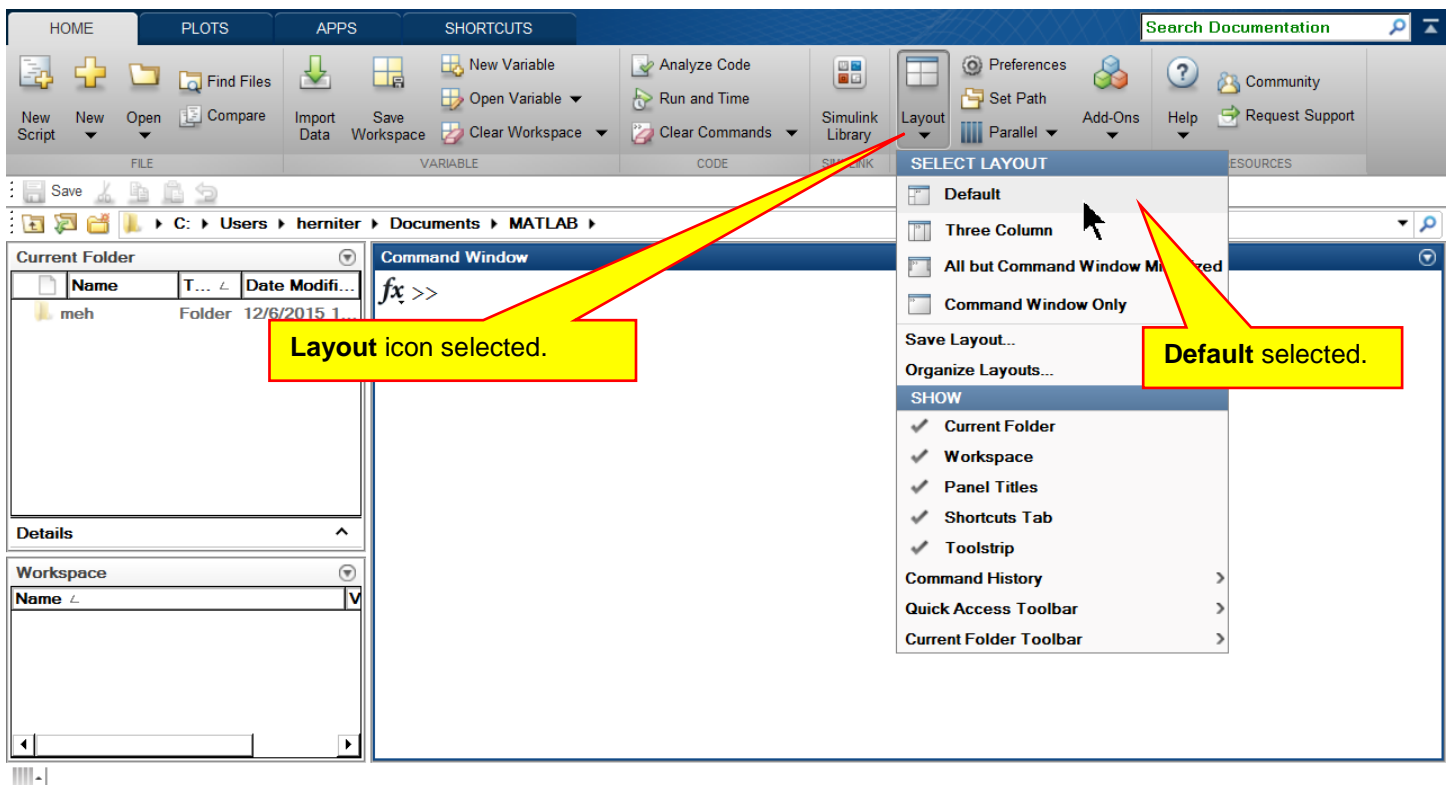


We will go through a few steps to make sure that all your windows match those shown in this manual. First, make sure that the **Home** tab is selected:

¹ If you are using a different version, your screen captures may not match those show here, however, they should be similar.

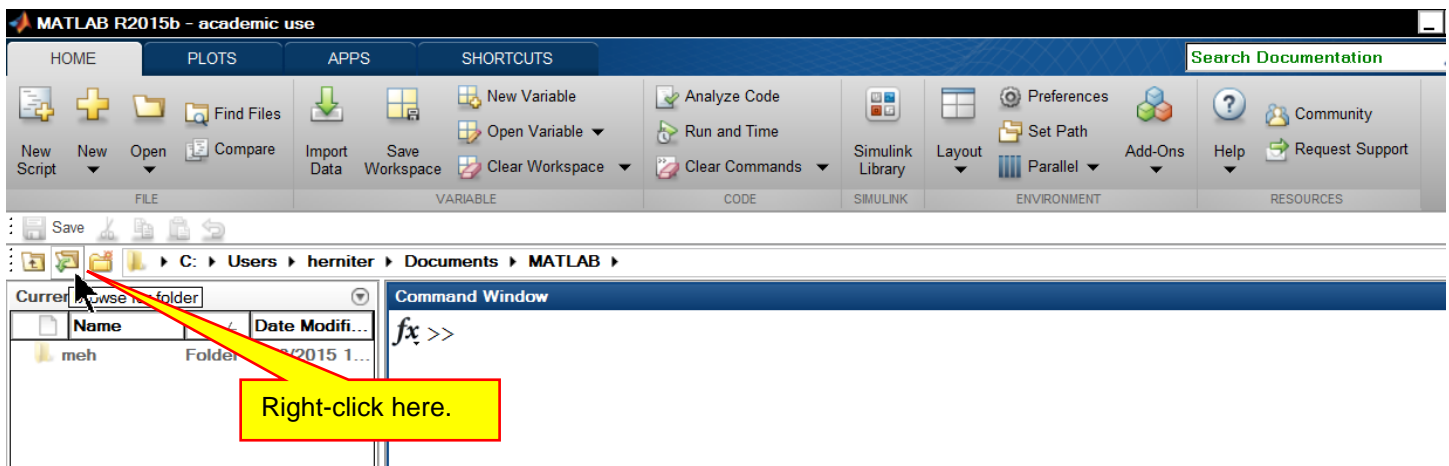


Next, click on the **Layout** icon and then select **Default**

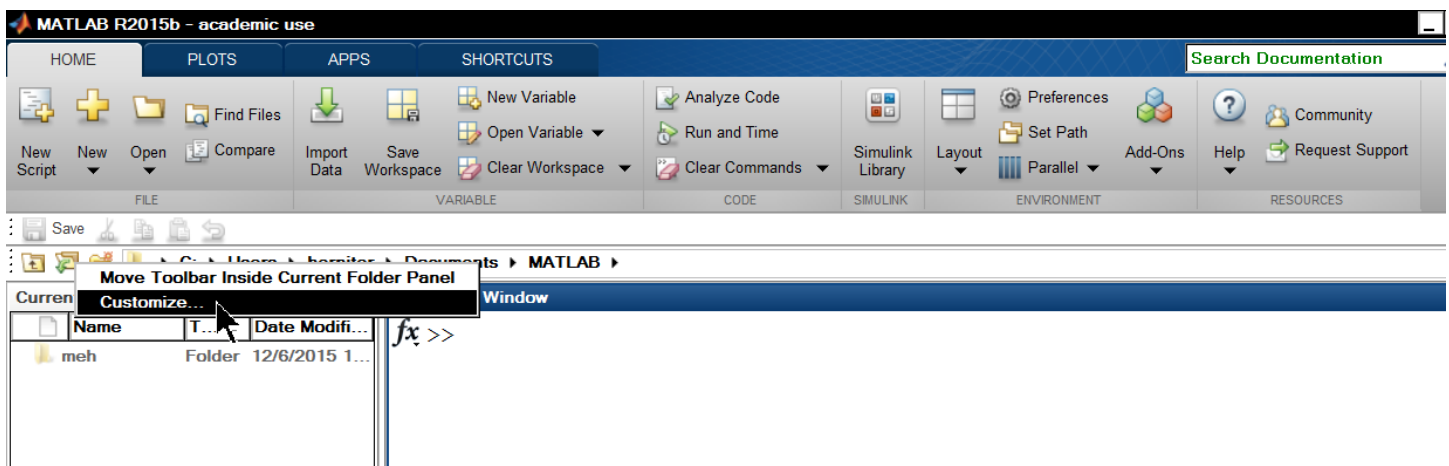


Selecting this option should cause your windows to match those show here.

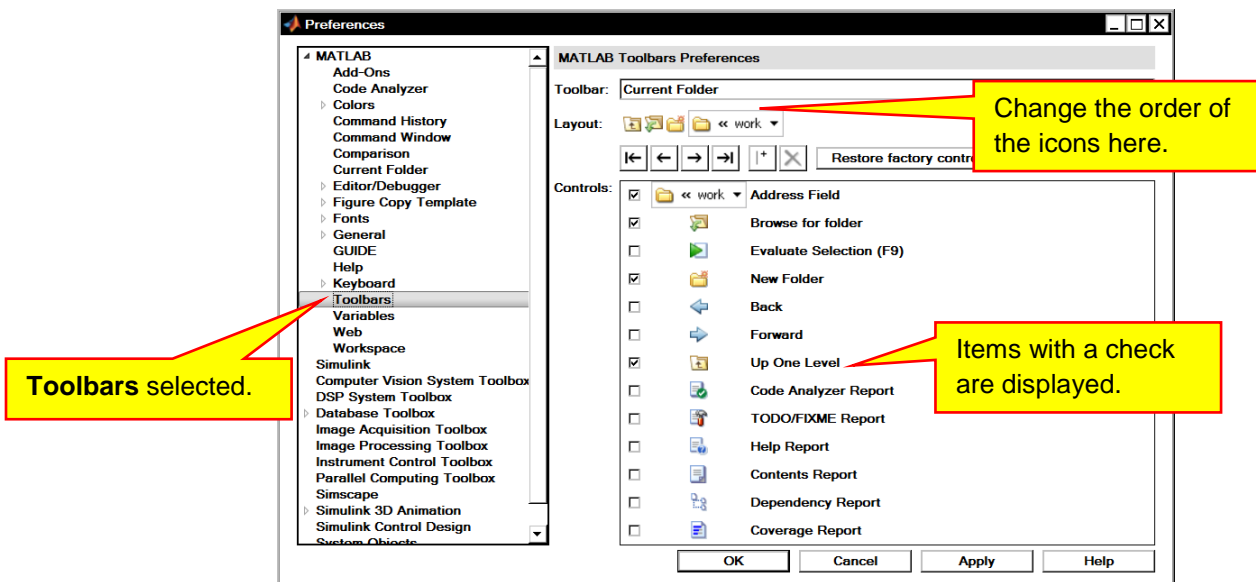
Next, we will customize the toolbar so that it has some useful functions. Right-click as shown below (OSX: control click):



Select customize from the menu that appears:



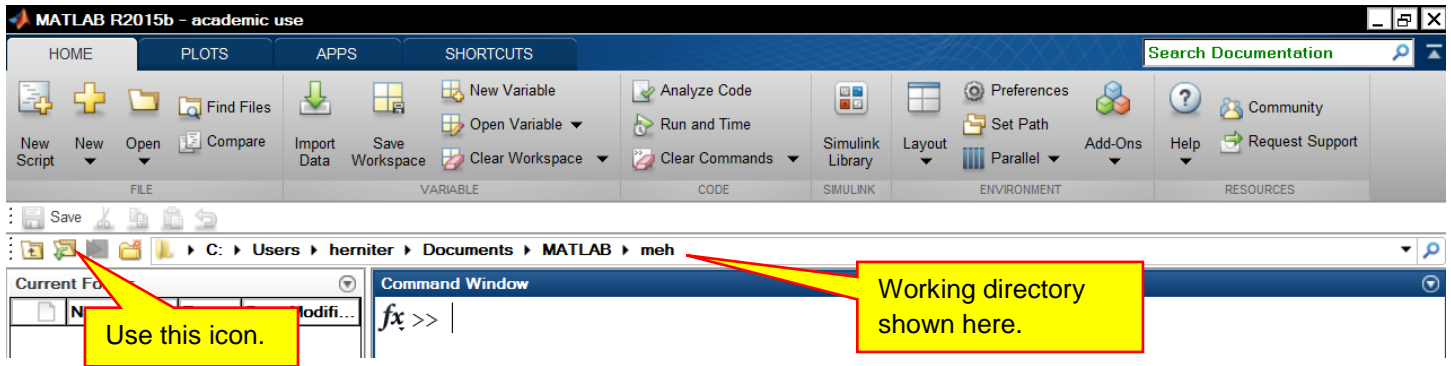
The window below will appear. Choose the selections as shown:



If you choose the same options, your toolbar should match mine, though the icons might be in a different order. If the icons are in a different order, you can rearrange the order in the **Layout** section of the window, as shown

above. Click the **OK** button when done. Your MATLAB window layout should now match those shown in this manual.

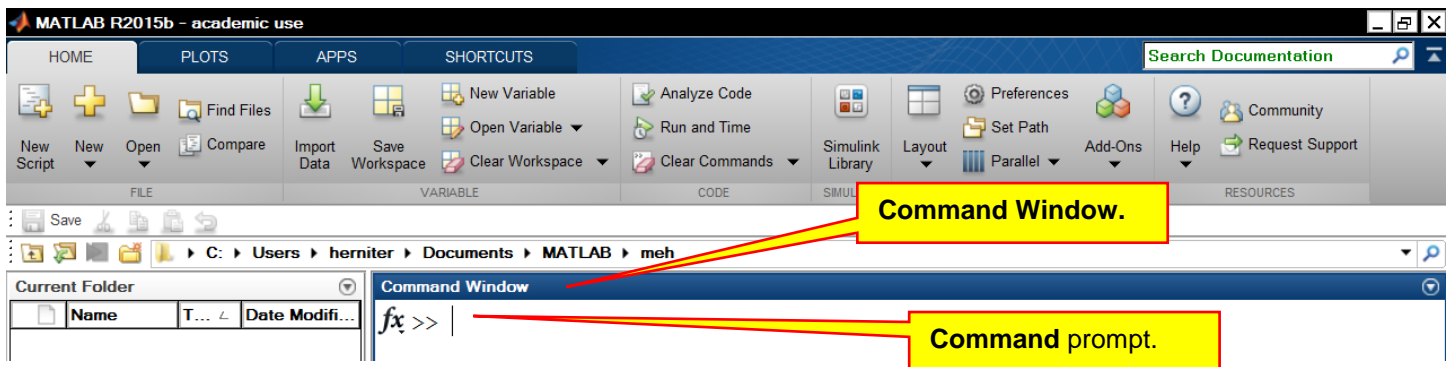
Next, we need to choose a directory where your files will be stored. This will be referred to as the working directory. Whenever you start MATLAB, you should navigate to this directory. Use the icons we just placed in the toolbar to create and or navigate to your working directory. On my system, I will be using the directory named C:\Users\Herniter\Documents\MATLAB\meh:



Each student should use a unique working directory in which to store files. Note that every time you start MATLAB, you will have to change to your working directory before you start doing any work. All of your files will be stored in your working directory or subdirectories within your working directory. **If you do not change to your own working directory, you may lose files or overwrite someone else's files.** We are now ready to start using MATLAB.

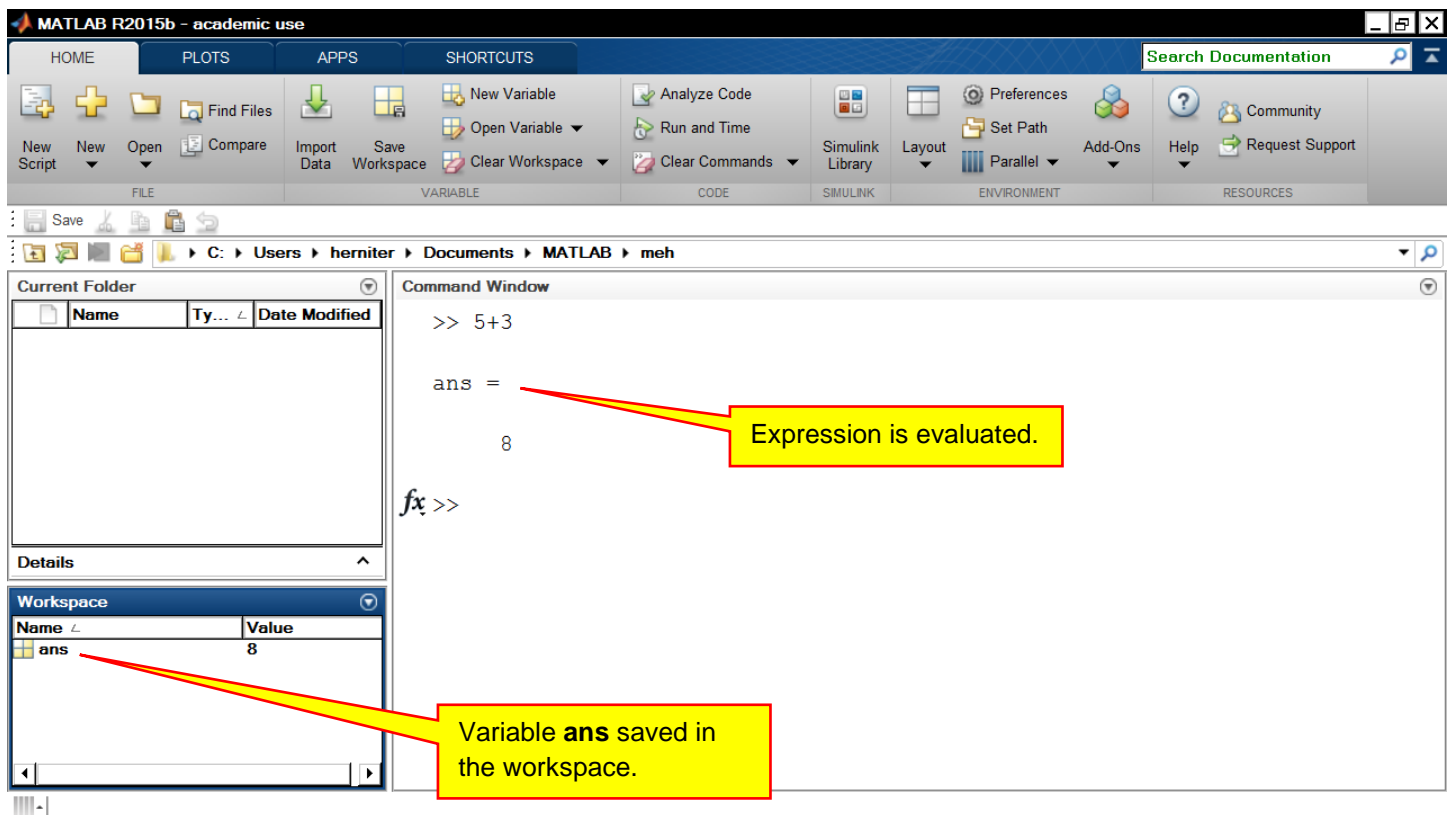
B. MATLAB Command Window

The **Command Window** is shown below. In this window we can enter commands and they will be evaluated by MATLAB. Note that the command prompt is indicated by the double right caret `>>`:



MATLAB functions and commands can be entered at the command prompt.

We can enter basic arithmetic expressions at the command prompt. When you press the **ENTER** key, the expressions will be evaluated. For example, type `5+3` and press the **ENTER** key:



The screenshot shows the MATLAB R2015b interface. The Command Window displays the command `>> 5+3` and the output `ans = 8`. A yellow callout box points to the output with the text "Expression is evaluated." The Workspace window shows a table with the variable `ans` and the value `8`. A yellow callout box points to the `ans` variable with the text "Variable `ans` saved in the workspace."

The expression is evaluated and the value is stored in a variable named **ans**. All variables are saved into what is referred to as the MATLAB workspace. The MATLAB workspace consists of the variables you create and store in memory during a MATLAB session². Note that values are saved in the MATLAB workspace while MATLAB is running. If you shut down MATLAB, all variables in the workspace will be lost unless you save the workspace.

MATLAB follows the standard conventions for the order of evaluating mathematical operations. The highest precedence operators are evaluated first. Raising a number to a power has the highest precedence, followed by multiplication and division which have the same precedence, followed by addition and subtraction which also have the same precedence. When two operations have the same precedence, they are evaluated from left to right. Some examples are shown below:

```
>> 5+3-4*6+7/2
```

4*6 evaluated first, then 7/2, then the rest.

```
ans =
```

```
-12.5000
```

```
>> 5+6/2
```

6/2 evaluated first.

```
ans =
```

```
8
```

² What is the MATLAB workspace? From the MATLAB help system.

Use parentheses to change the order of operations:

```
>> (5+6)/2
```

(5+6) evaluated first.

```
ans =
```

```
5.5000
```

```
>> 5+6/7+4
```

6/7 evaluated first.

```
ans =
```

```
9.8571
```

```
>> (5+6)/(7+4)
```

(5+6) evaluated first, then (7+4) is evaluated. Division is performed last.

```
ans =
```

```
1
```

The ^ key is used to indicate the power operator. Note that the power operator has the highest precedence and is executed first unless parenthesis are used to change the order:

```
>> 2^2/4
```

2^2 evaluated first.

```
ans =
```

```
1
```

```
>> 2^(2/4)
```

2/4 evaluated first.

```
ans =
```

```
1.4142
```

Note that 2^2 means 2 squared, or 2^2 , which is equal to 4. Also note that $2^{0.5}$ is $2^{0.5}$, which is the $\sqrt{2}$.

C. Defining Variables.

We can define variables in MATLAB by typing the variable name followed by an equal sign and then a value or mathematical expression. The variables will be saved in the MATLAB workspace.

Some examples are shown below:

```
>> A=5
```

```
A =
```

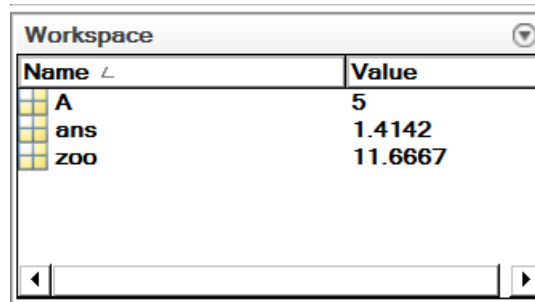
```
5
```

```
>> zoo=35/3
```

```
zoo =
```

```
11.6667
```

We created two new variables named A and zoo. If you look in the MATLAB Workspace window, you will see the two new variables displayed



A valid variable name starts with a letter, followed by letters, digits, or underscores. MATLAB® is case sensitive, so **A** and **a** are not the same variable³. The maximum length of a variable name is 63 characters.

Some examples of variable names are

```
>> A5=6
```

```
A5 =
```

```
6
```

```
>> b17=789.44
```

```
b17 =
```

```
789.4400
```

```
>> Cat_Dog = 7
```

```
Cat_Dog =
```

```
7
```

Note that variable names are case sensitive, so the three variables listed below are different:

```
>> This_is_a_long_variable_name=3
```

```
This_is_a_long_variable_name =
```

```
3
```

```
>> This_is_a_Long_variable_name=4
```

```
This_is_a_Long_variable_name =
```

³ From MATLAB help documentation, "Variable Names."

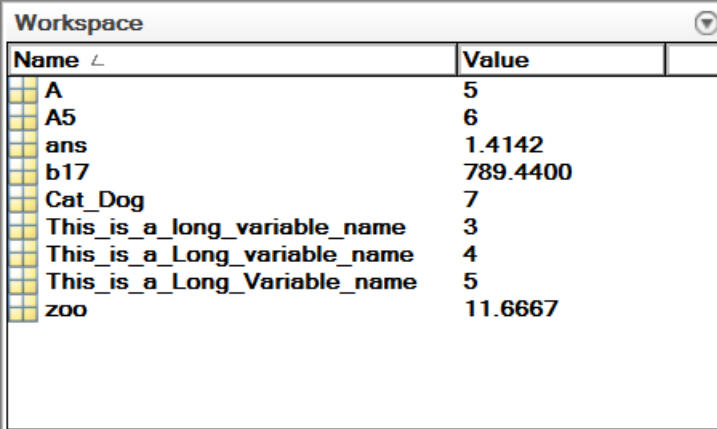
4

```
>> This_is_a_Long_Variable_name=5
```

```
This_is_a_Long_Variable_name =
```

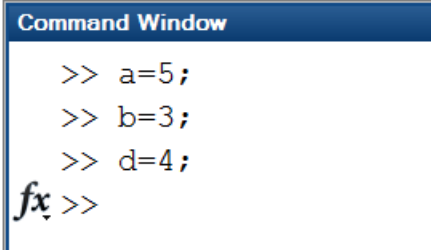
5

When we look at the Workspace window, we see that the three variables are listed separately, indicating that they are different variables:



Name	Value
A	5
A5	6
ans	1.4142
b17	789.4400
Cat_Dog	7
This_is_a_long_variable_name	3
This_is_a_Long_variable_name	4
This_is_a_Long_Variable_name	5
zoo	11.6667

Note in MATLAB that when the semicolon is used at the end of a statement, it causes MATLAB to not display the result of a calculation:



```
Command Window
>> a=5;
>> b=3;
>> d=4;
fx >>
```

The expressions were evaluated and values were saved. The result was just not displayed. Note that to display the value of a variable, all you need to do is type the name followed by the **ENTER** key:

```
Command Window
>> a=5;
>> b=3;
>> d=4;
>> d

d =

    4

fx >> |
```

Since variables are stored in the workspace, their values can be used at a later time. Variables can also be used to make calculations:

```
Command Window
>> a5=7;
>> boo=2;
>> D=3;
>> x=a5+boo/D

x =

    7.6667

fx >> |
```

Note that you can change the value of variables:

```
>> x=(a5+boo)/D

x =

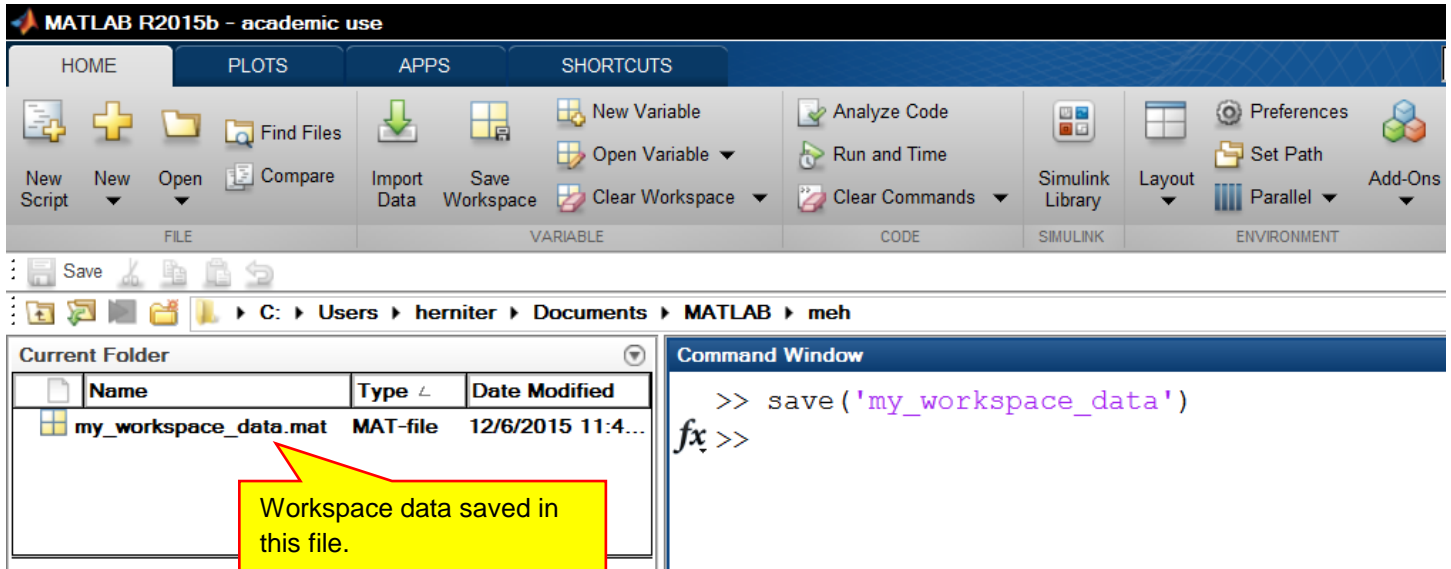
    3
```

D. Saving and Clearing the Workspace

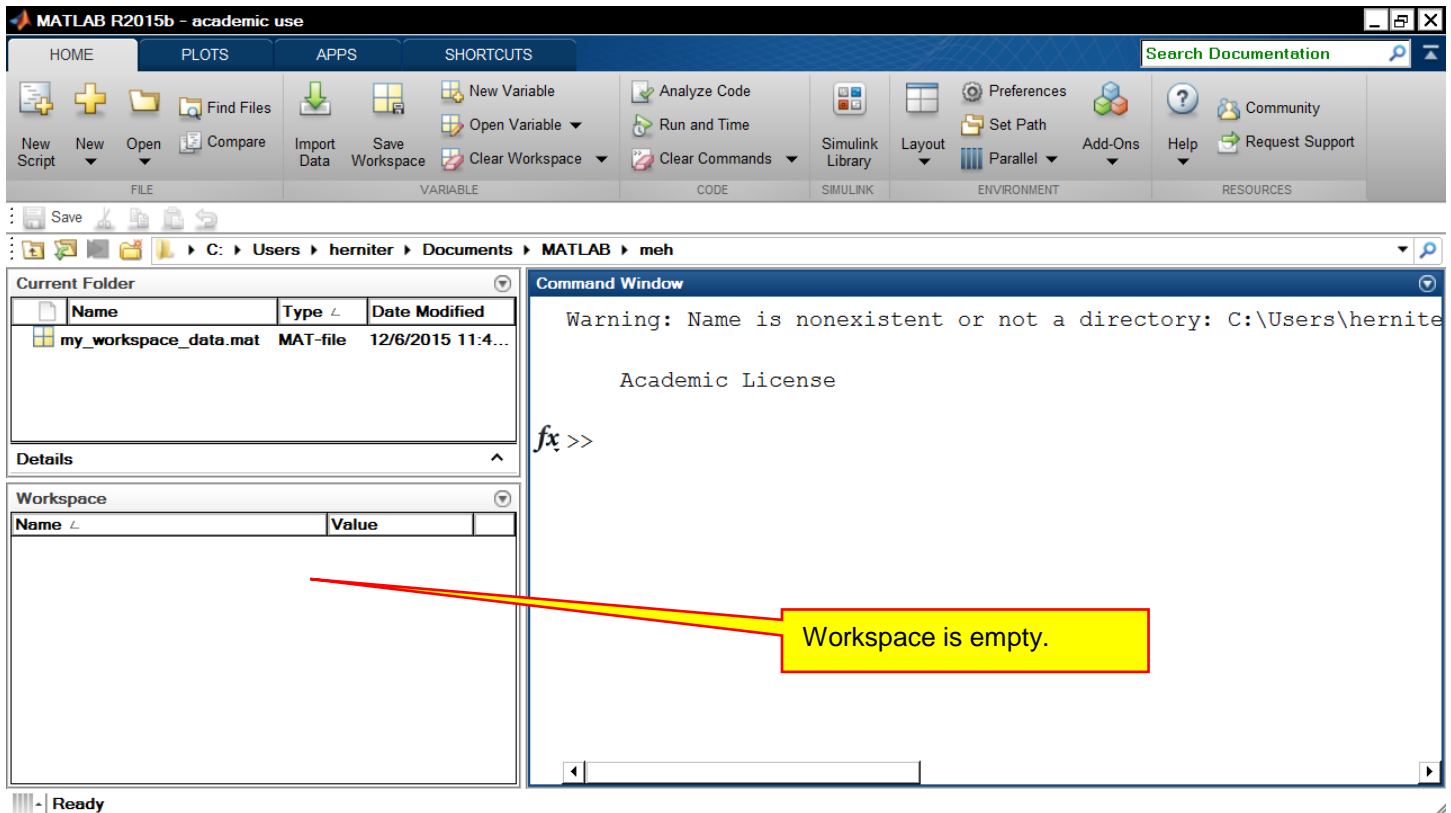
The variables in the workspace will remain in the workspace until you shut down MATLAB, at which time they will be lost unless you save them. To save the data in the workspace for use in a later MATLAB session, use the MATLAB **save** command. Specify a file name when you use this command:

```
Command Window
>> save('my_workspace_data')
fx >> |
```

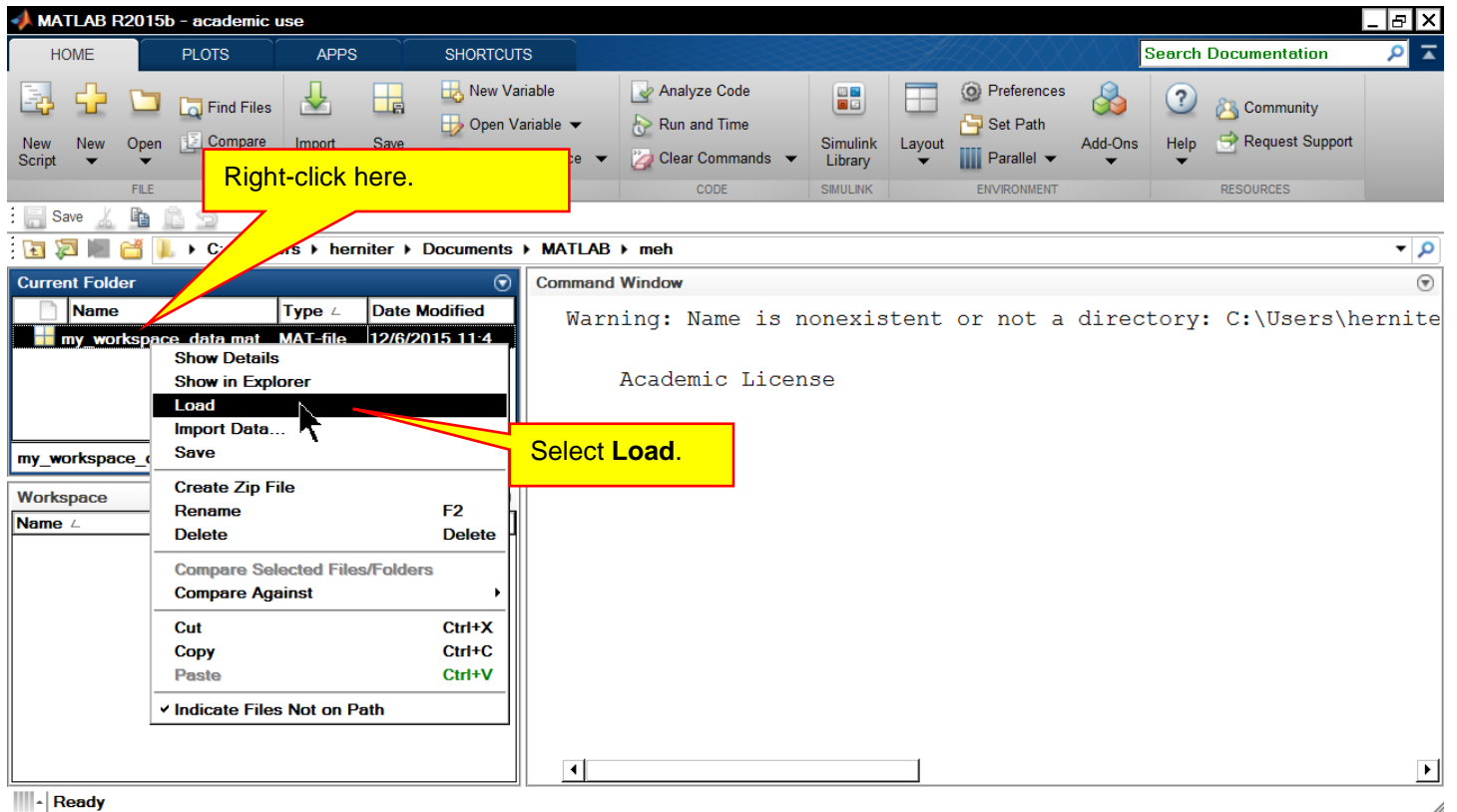
You will notice that a file now appears in your current folder with the specified name and a .mat extension:



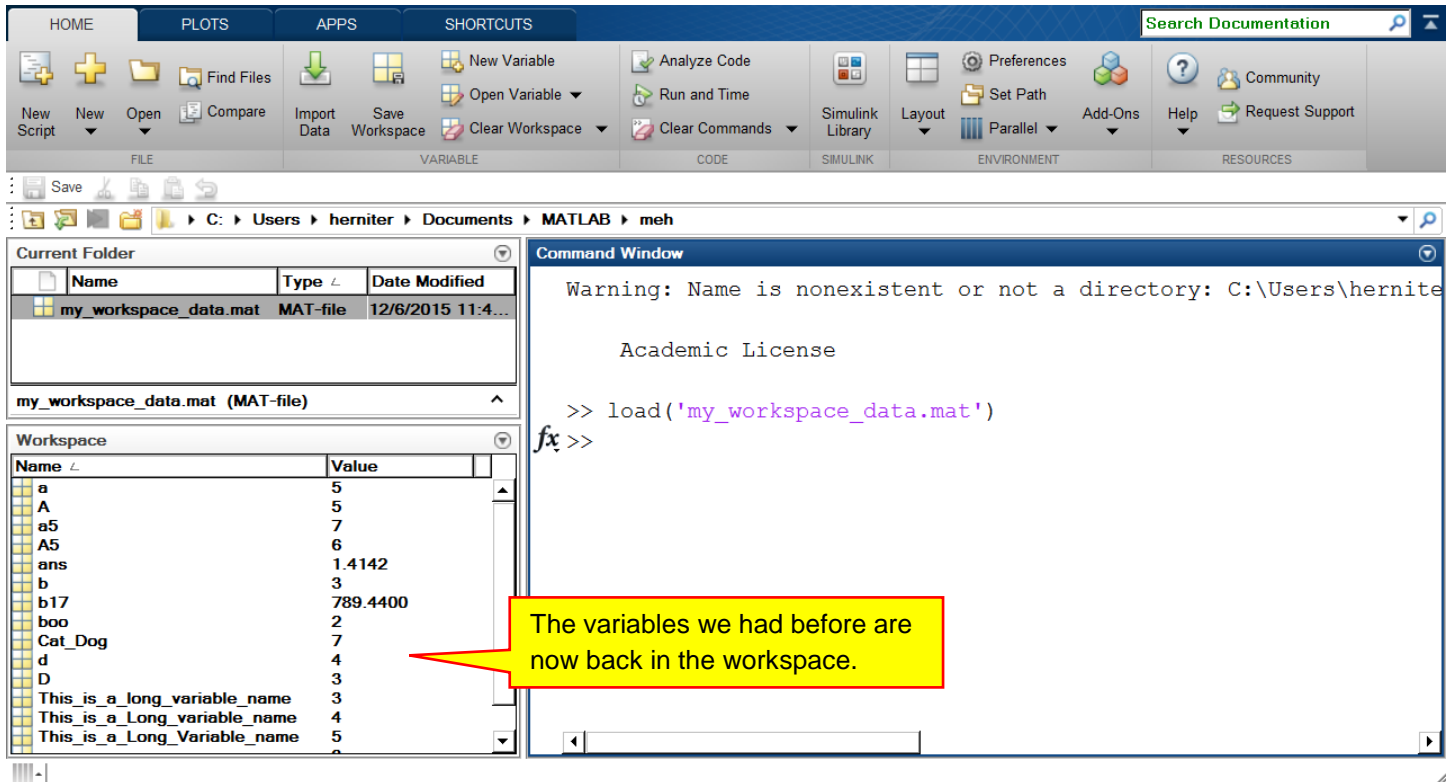
Exit out of MATLAB and then restart MATLAB. You will see that the workspace is empty:



Navigate to your directory. The file we saved is located in that directory. Right-click on the .mat file (MAC: control-click) and then select **Load**:



After you select **Load**, you will see that the variables are back in the MATLAB workspace:



Warning: Name is nonexistent or not a directory: C:\Users\hernite

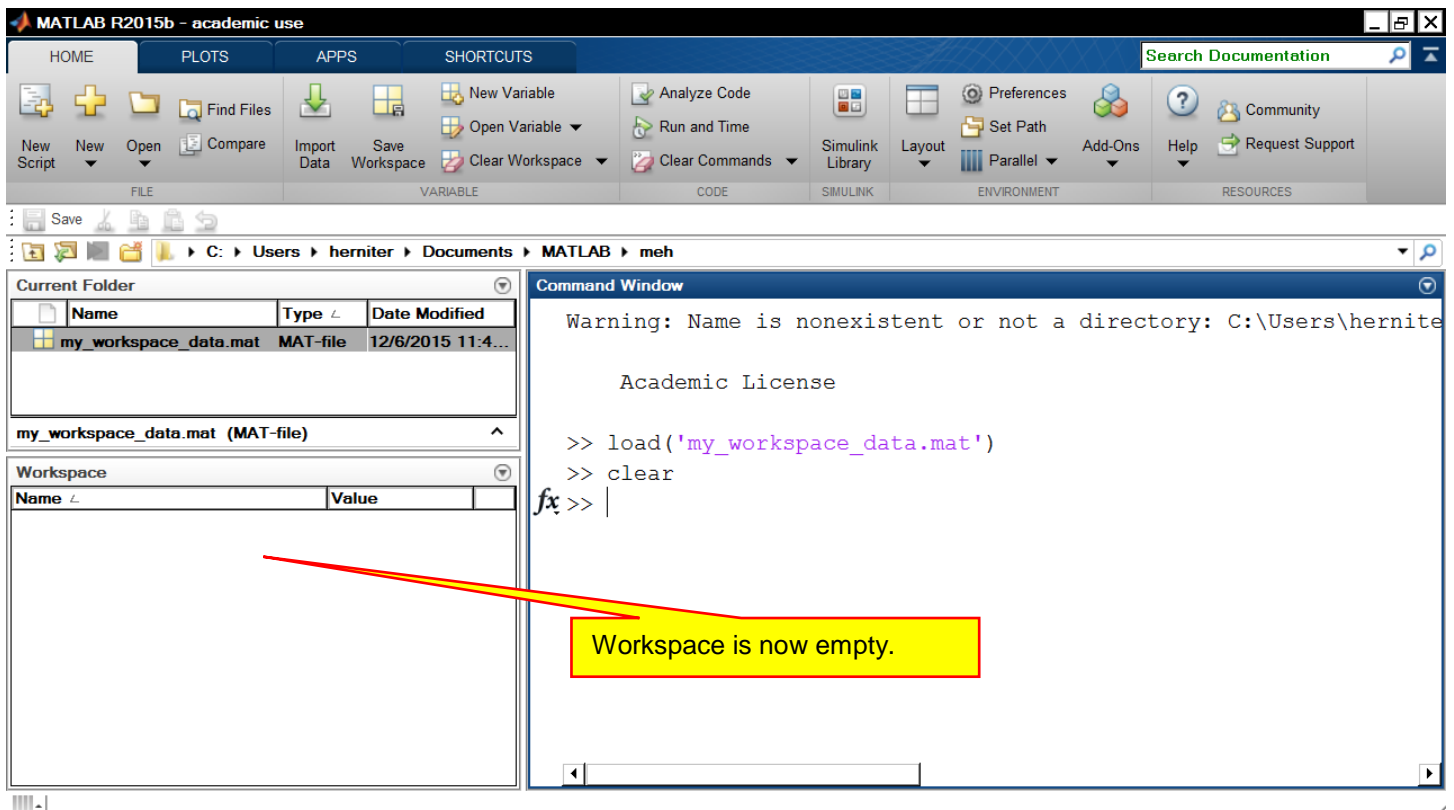
Academic License

```
>> load('my_workspace_data.mat')
fx >>
```

Name	Value
a	5
A	5
a5	7
A5	6
ans	1.4142
b	3
b17	789.4400
boo	2
Cat_Dog	7
d	4
D	3
This_is_a_long_variable_name	3
This_is_a_Long_variable_name	4
This_is_a_Long_Variable_name	5

The variables we had before are now back in the workspace.

To remove all data from the MATLAB workspace, use the **clear** command:



Warning: Name is nonexistent or not a directory: C:\Users\hernite

Academic License

```
>> load('my_workspace_data.mat')
>> clear
fx >> |
```

Name	Value
------	-------

Workspace is now empty.

E. Questions

Question V-1: What is the purpose of the working directory in MATLAB?

Question V-2: How long are values stored in the MATLAB workspace?

Question V-3: Right-click in Windows is the same as what key sequence on the MAC with OSX?

Question V-4: What is the MATLAB workspace and what is it used for?

Question V-5: What does precedence mean in terms of mathematical operations?

Question V-6: Are variables `Wax` and `WaX` the same or different in MATLAB?

Question V-7: What is the function of the semicolon in MATLAB assignments?

Question V-8: Specify the characters that are legal to use in MATLAB variable names.

Question V-9: What is the maximum size of a variable name?

F. Exercises

Exercise V-1: By hand, calculate the following values. Then check your calculations with MATLAB.

- a. $5+4-3$
- b. $5+4*3$
- c. $3*4+4$
- d. $3*(4+4)$

Exercise V-2: What are the values of the following MATLAB commands:

- e. `pi`
- f. `Pi`
- g. `1/0`
- h. `-1/0`
- i. `0/0`

Exercise V-3: What does NaN stand for? (Use the MATLAB help facility to answer this question.)

Lesson VI: Arrays

A. Row Vectors

Arrays are variables that contain more than one value. MATLAB is designed to handle arrays as easily as it handles scalar variables. A scalar variable is a variable with a single value. An example of an array will make things easier:

```
>> a=[1, 4, 88, 3, 2]
a =
     1     4    88     3     2
```

We have defined variable **a** as an array of 5 values. This is a one dimensional array referred to as a row vector because all of the values are on a single line, referred to as a row. Note that when we define an array, we enclose the values inside square brackets, [and], and the values are separated by commas or spaces. A second example is shown below:

```
>> b = [32 88 9 33 2]
b =
    32    88     9    33     2
```

We can access individual elements of an array with an index. For example the 3rd element of **b** is :

```
>> b(3)
ans =
     9
```

In the statement **b(3)**, the number 3 is referred to as the index. Accessing an element is referred to as **addressing** the element. The first element of **a** is:

```
>> a(1)
ans =
     1
```

Again, in the statement **a(1)**, the number 1 is referred to as the index. If we use an index outside of the range of where the variable is defined, we will get an error:

```
>> a(6)
Index exceeds matrix dimensions.
```

Note that an index is always a positive integer greater than or equal to 1. An individual value in an array is referred to as an element. When you want to work with a specific element in an array, it is referred to as addressing the array or element. The properties of an index are that it is an integer between 1 and the maximum size of the array that you are addressing.

MATLAB is designed to handle arrays naturally. So, we can add arrays **a** and **b** because they have the same number of elements and because both are both row vectors:

```
>> c=a+b
c =
```

```
33    92    97    36    4
```

Notice that we created a new row vector called **c**. Since **a** and **b** were both row vectors with 5 elements, **c** is a row vector of 5 elements. Note that the first element of **c** is equal to the first element of **a** plus the first element of **b**. The second element of **c** is equal to the second element of **a** plus the second element of **b**, and so on.

The statement $c=a+b$ was equivalent to the following:

```
>> c(1) = a(1)+b(1);
>> c(2) = a(2)+b(2);
>> c(3) = a(3)+b(3);
>> c(4) = a(4)+b(4);
>> c(5) = a(5)+b(5);
>> c
c =
```

```
33    92    97    36    4
```

Obviously $c=a+b$ is much more convenient and will actually be executed faster on the computer. (meaning that it is more efficient.)

B. Column Vectors

A row vector is a one-dimensional array with all of the values in a single row. We will also encounter column vectors. This also a one dimensional array. However, all of the values are in a single column. For column vectors, the values are separated by a semicolon when you create the array. Examples of column vectors with 7 values are:

```
>> x=[1; 4; 55; 33; 5; 99; -4]
x =
     1
     4
    55
    33
     5
    99
    -4
>> y=[-1; -3; 4; 6; -1; -5; 0]
y =
    -1
    -3
     4
     6
    -1
    -5
     0
```

We address column vectors the same way we address row vectors:

```
>> y(5)
ans =
    -1
>> x(2)
ans =
     4
```

Since **x** any **y** are both column vectors with the same number of elements, we can add them together:

```
>> x+y
ans =
     0
```

```

1
59
39
4
94
-4

```

Again, the first element of \mathbf{x} is added to the first element of \mathbf{y} . The second element of \mathbf{x} is added to the second element of \mathbf{y} , and so on.

Note that we cannot add \mathbf{x} to \mathbf{a} for two reasons. First, they have a different number of elements. Second, you cannot add a row vector to a column vector:

```

>> a+x
Error using +
Matrix dimensions must agree.

```

C. Functions to Create Arrays

MATLAB has a number of functions for creating arrays. Here we will only look at a few simple methods.

1. The Colon Operator

The colon operator (*first: step: last*) generates a row vector from the first value to the last value with the specified step. For example, to generate values 1, 3, 5, 7, 9, we would use 1:2:9

```

>> a=1:2:9
a =
    1     3     5     7     9

```

The step can be negative:

```

>> b=10:-1:5
b =
   10     9     8     7     6     5

```

And, the step does not have to be an integer:

```

>> c=-5:.2:-4
c =
 -5.0000  -4.8000  -4.6000  -4.4000  -4.2000  -4.0000

```

If we only use two numbers with the colon operator (*first: last*), a step of 1 is assumed:

```

>> 55:63
ans =
    55    56    57    58    59    60    61    62    63

```

The colon operator with a unit step is convenient for addressing a range of elements in an array. For example, suppose \mathbf{a} is the following array:

```

>> a=[1, 44, 77732, 33, -5, 44, 8]
a =
    1    44  77732    33    -5    44     8

```

Suppose that we want a new array that contains the 3rd through 6th elements of this array. One way is to use the method below:

```

>> c=[a(3), a(4), a(5), a(6)]

```

```
c =
    77732         33        -5         44
```

Basically, what we did is create a new row vector **c** that contains the specific elements that we wanted from row vector **a**. This method works fine, but it is a bit cumbersome, especially if we had a few hundred elements rather than four elements.

A better method is the following:

```
>> c=a(3:6)
c =
    77732         33        -5         44
```

3:6 generates indices 3, 4, 5, 6, so `a(3:6)` takes the 3rd through 6th elements of **a** and places them in a new row vector.

As another example, suppose we wanted every other element of **a**:

```
>> a(1:2:7)
ans =
     1    77732        -5         8
```

We can also use this method to reverse the order of **a**:

```
a =
     1         44    77732         33        -5         44
     8
>> a(7:-1:1)
ans =
     8         44        -5         33    77732         44
     1
```

This method of addressing arrays will be used when we obtain data from the linescan camera on the vehicle.

2. Ones and Zeros

MATLAB has several functions for creating arrays. The **ones** and **zeros** functions create arrays and fill them (not surprisingly) with ones and zeros. We will use these functions to initialize arrays. These functions are designed for multidimensional arrays. However, for this project we are only using one dimensional arrays that we refer to as row vectors and column vectors. The syntax of the functions is **ones(r,c)** and **zeros(r,c)**, where **r** is the number of rows, and **c** is the number of columns. For us, one of those numbers will always be 1. To create a row vector with five elements, all of which are 1, we would use the following:

```
>> ones(1,5)
ans =
     1     1     1     1     1
```

The (1,5) means 1 row and 5 columns. Or for us, a row vector of five elements. To create a row vector with ten elements, all of which are zero, we can use:

```
>> zeros(1,10)
ans =
     0     0     0     0     0     0     0     0     0     0
```

To create a column vector, the number of columns will be 1. So, to create a column vector with five elements, all of which are 1, we would use the following:

```
>> ones(5,1)
ans =
     1
     1
     1
     1
     1
```

```
1  
1  
1
```

The (5,1) means 5 rows and 1 column. Or for us, a column vector of five elements. To create a column vector with ten elements, all of which are zero, we can use:

```
>> zeros(10,1)  
ans =  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0
```

D. Questions

Question VI-1: What is a scalar variable;

Question VI-2: What is a row vector.

Question VI-3: What is an index.

Question VI-4: If a and b are arrays. What must be true about a and b so that we can add a and b together?

Question VI-5: How do you address a single value in a vector?

Question VI-6: What is the difference between a row vector and a column vector?

Question VI-7: What are the requirements of an index?

Question VI-8: What is the colon operator?

Question VI-9: In the statement ones(1,5), what does the 1 do? How many elements are in the array? Is this a row or column vector?

Question VI-10: In the statement zeros(5,1), what does the 1 do? How many elements are in the array? Is this a row or column vector?

E. Exercises

For the exercises below use the following variables:

```
A=[1, 7, 2, 8, 3, 9]
```

```
B=[-3, 4, 1, 8, 9, 2]
```

```
Q=1:5
```

```
X=[1, 4, 6, 9, 8, 6, 8]
```

```
C=A+B
```


$D=A-B$

These exercises should be done by hand and then checked with MATLAB.

Exercise VI-1: What are the numerical values of $A(1)$, $B(3)$, $X(7)$?

Exercise VI-2: What are the numerical values of C ?

Exercise VI-3: What are the numerical values of D ?

Exercise VI-4: What is $A+X$?

Exercise VI-5: What are the values generated by $5:-1:1$?

Exercise VI-6: What are the values of $A(3:6)$?

Exercise VI-7: What are the values of $A(6:-1:3)$?

Exercise VI-8: What are the values generated by $1:0.1:3$?

Exercise VI-9: What are the values generated by $.9:.1:1.5$?

Exercise VI-10: What are the values generated by $1:5:2$?

Exercise VI-11: What are the values generated by $5:1:1$?

Exercise VI-12: What is $A+X(1:6)$? Why does this work and not $A+X$?

Exercise VI-13: What is $A(1:3)+A(4:6)$?

Exercise VI-14: What is $Q(5:-1:1)$?

Exercise VI-15: What is $Q(1:5)+Q(5:-1:1)$?

Exercise VI-16: Give the command to create a column vector with the values 1, 3, 5, 7, 9.

Exercise VI-17: Give the command to create a row vector with the values -1, -3, -5, -7, -9.

Exercise VI-18: Give the command to create a row vector with the values -1, -3, -5, -7, -9.

Exercise VI-19: Give the command to find the last two values of variable A .

Exercise VI-20: Give the command to create a new vector G , where G has the same values as B but in reverse order.

Exercise VI-21: Give the command to create the column vector with the values -0.1, 0.1, 0.3, 0.5, 0.7.

Lesson VII

Working with Arrays

MATLAB was designed to work with arrays, so array handling is built in to just about everything MATLAB does. MATLAB was originally designed to make matrix calculations easier, as its name stands for **MAT**rix **LAB**oratory. A matrix is a 2-dimensional array like

```
>> a=[1 2 3; 3 9 7; 3 8 4]
a =
     1     2     3
     3     9     7
     3     8     4
```

Thus, if you will look deeper, you will find a huge amount of facilities available for array manipulation. Here we will show a few operations and functions that are useful for the autonomous vehicle project. Note that the functions given here will work multidimensional arrays. However, for our discussion, we will only address use for vectors, which are one dimensional arrays.

For this discussion, we will define a few vectors to use in our calculations:

```
>> a=[1 2 3 4 5 4 3 2 1];
>> b=[10 9 8 7 6 7 8 9 10];
>> c=[1 2 3 4 5 6 7 8 9 ];
>> d=[ 9 8 7 6 5 4 3 2 1];
```

A. Built-In MATLAB Functions

MATLAB has a number of built-in functions that are useful for our project. The **length** function tells us how many elements are in a vector:

```
>> length(a)
ans =
     9
>> length([1 3 5 7 9])
ans =
     5
```

We see that vector **a** has 9 elements and that the second array has 5 elements.

The MATLAB **sum** command adds up all of the elements in an array:

```
>> sum([1 2 3 4 5])
ans =
    15
>> sum(b)
ans =
    74
>> sum(1:100)
ans =
   5050
```

In the last example, 1:100 generates the numbers 1, 2, 3, 4, 5, ..., 99, 100. Thus the statement `sum(1:100)` finds the sum of the integers from 1 to 100.

The MATLAB **mean** function calculates the average of the numbers in a vector:

```
>> c
c =
     1     2     3     4     5     6     7     8     9
>> mean(c)
ans =
     5
```

Note that we can also calculate the mean of a vector using the **sum** and **length** functions:

```
>> sum(c)/length(c)

ans =

     5
```

The **max** and **min** functions find the max and min of a vector:

```
a =
     1     2     3     4     5     4     3     2     1
>> max(a)
ans =
     5
>> b
b =
    10     9     8     7     6     7     8     9    10
>> min(b)
ans =
     6
```

B. Scalar Calculations with Vectors

A scalar is a number with a single value. (What you are used to calling numbers.) When you use a scalar in a calculation with a vector, the same calculation is applied to every element in the array. For example, to add one to every element of vector **a**, we could use the following:

```
>> a
a =
     1     2     3     4     5     4     3     2     1
>> a+1
ans =
     2     3     4     5     6     5     4     3     2
>>
```

We see that the scalar value of 1 was added to every element in the vector. To multiply every element by 3, we could use the following commands:

```
>> a
a =
     1     2     3     4     5     4     3     2     1
>> 3*a
ans =
     3     6     9    12    15    12     9     6     3
>>
```

We see that each element in the array was multiplied by the scalar value of 3. We can do fairly complex calculations with scalars, and the calculation will be done for each element in the array. For example, we will add 1 to **a** and then multiply the result by 3:

```
>> a
a =
     1     2     3     4     5     4     3     2     1
>> (a+1)*3
ans =
     6     9    12    15    18    15    12     9     6
>>
```

We can also define variables as scalars and then use them in calculations as well. In the calculations below, **q** is a scalar with a value of 5.

```
>> q=5;
>> a
a =
     1     2     3     4     5     4     3     2     1
>> a/q
ans =
    0.2000    0.4000    0.6000    0.8000    1.0000    0.8000    0.6000    0.4000
0.2000
>>
```

Lastly, we will demonstrate scalar subtraction:

```
b =
    10     9     8     7     6     7     8     9    10
>> b-q
ans =
     5     4     3     2     1     2     3     4     5
>>
```

C. Vector Calculations

MATLAB is design to do matrix calculations. This is something you will not learn until your college days. In MATLAB when you specify multiplication and division, they are actually matrix calculations. In the example below, **K** is a row vector and **L** is a column vector:

```
>> K=[1 2 3];
>> L=[1; 2; 3];
>> K*L
ans =
    14
>> L*K
ans =
     1     2     3
     2     4     6
     3     6     9
```

We see that $K*L$ is not the same as $L*K$, and $K*L$ results in a scalar value and $L*K$ results in a matrix. Thus, $*$ is a matrix operation and beyond what we need for this project. **By default, if you are working with arrays, MATLAB performs matrix operations.** We do not want to do matrix algebra for anything in this course. Thus, we need to use the element by element operators in MATLAB. They are $+$, $-$, $.*$, $./$, and $.^$. Note that the last three used a period. So $.*$ is multiplication, $./$ is division, and $.^$ is power. Always use the period for these operators. Note that you can also use $.+$ and $.-$, but these operations are the same with and without the period.

We will first demonstrate addition and subtraction:

```
>> a
a =
     1     2     3     4     5     4     3     2     1
>> b
b =
    10     9     8     7     6     7     8     9    10
>> a+b
ans =
    11    11    11    11    11    11    11    11    11
>> a-b
ans =
    -9    -7    -5    -3    -1    -3    -5    -7    -9
>>
```

With `a+b` we see that the first element of `a` is added to the first element of `b`. The second element of `a` is added to the second element of `b`, and so on. The only requirement is that `a` and `b` have the same number of elements. The same is true with `a-b`. The first element of `b` is subtracted from the first element of `a`. The second element of `b` is subtracted from the second element of `a`, and so on. Again, the only requirement is that `a` and `b` have the same number of elements.

For multiplication and division, use the `.*` and `./` operators:

```
>> a
a =
     1     2     3     4     5     4     3     2     1
>> b
b =
    10     9     8     7     6     7     8     9    10
>> a.*b
ans =
    10    18    24    28    30    28    24    18    10
>> a./b
ans =
    0.1000  0.2222  0.3750  0.5714  0.8333  0.5714  0.3750  0.2222  0.1000
```

With `a.*b` we see that the first element of `a` is multiplied with the first element of `b`. The second element of `a` is multiplied with the second element of `b`, and so on. The only requirement is that `a` and `b` have the same number of elements. The same is true with `a./b`. The first element of `b` is divided by the first element of `a`. The second element of `b` is divided by the second element of `a`, and so on. Again, the only requirement is that `a` and `b` have the same number of elements.

Finally the power (`.^`) operator

```
a =
     1     2     3     4     5     4     3     2     1
>> a.^2
ans =
     1     4     9    16    25    16     9     4     1
```

We see that each element is squared.

D. Plotting

MATLAB has a large number of functions available for plotting data. Here we will just discuss a basic plot. The syntax for this command is **plot(x,y)** where **x** and **y** are vectors that contain the values of the points. **x** and **y** must contain the same number of points.

As an example, we will plot the graph of the parabola $y=x^2-2x+1$. First, we must define the values of **x**. We will let **x** go from -5 to 7 in steps of 0.1:

```
>>x=-5:0.1:7;
```

Note that instead of using the colon operator, we can also use the **linspace** function. The syntax of the **linspace** function is **linspace(first_value, last_value, number_of_points)**

```
>>x=linspace(-5,7,100);
```

This **linspace** function generates 100 equally spaced points from -5 to 7. The step between the points is

$\frac{7-(-5)}{100} = 0.12$. We can use either method to generate the points for our **x** values. Next, we need to calculate the **y**-values. This is easy to do in MATLAB:

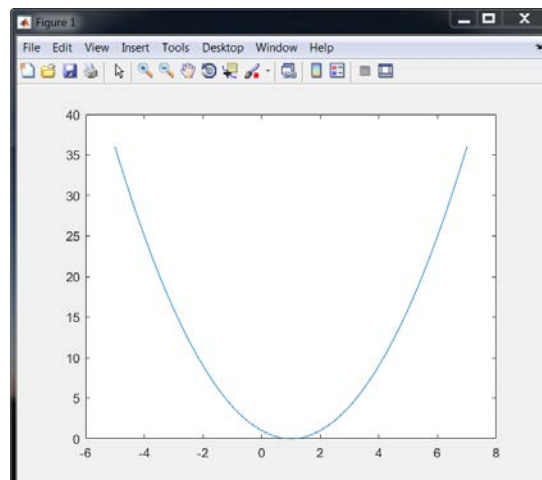
```
y=x.^2-2*x+1;
```

Note that we are doing element-by-element operations. The **.**^ is the element-by-element operation for power. Thus each element is squared. With **2*x**, each element is multiplied by 2. And with the **-1**, we subtract -1 from each element. In short, the equation **x.^2-2*x+1** is calculated for every value of **x**. We only write the equation once, but it will be executed 100 times in this example.

Now that we have **x** and **y** values, we can plot them. The entire code sequence is shown below:

```
>> x=linspace(-5,7,100);
>> y=x.^2-2*x+1;
>> plot(x,y)
```

The generated plot is:

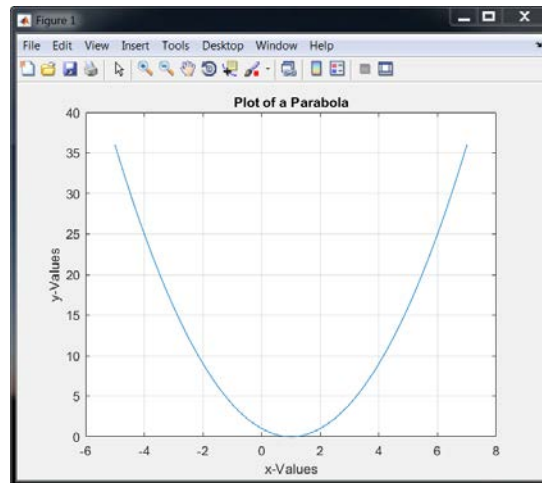


We can annotate the plot with the commands below:

```
>> title('Plot of a Parabola');
>> xlabel('x-Values')
```

```
>> ylabel('y-Values')
>> grid on
```

The resulting plot is



You could come up with more interesting labels for the x-axis and y-axis labels if you wish.

E. Questions

Question VII-1: Is the mean of a vector the same as the average?

Question VII-2: What is a scalar?

Question VII-3: What is a^2 ? Is this scalar power? Does this work?

Question VII-4: What is the difference between `.` and `+` when doing array calculations?

Question VII-5: What is the difference between `.` and `-` when doing array calculations?

F. Exercises

Exercise VII-1: What is the sum of all of the odd numbers between 1 and 100?

Exercise VII-2: What is the average of all of the numbers between 1 and 1000?

Exercise VII-3: What is the average of all of the numbers between 1 and 999?

Exercise VII-4: Three ways to find the sum of the numbers between 1 and 100 are shown below. Verify that the three methods yield the same answer and then explain why they are the same.

- a. `sum(1:100)`,
- b. `y=1:100; x=100:-1:1; sum(x+y)/2`
- c. `100*101/2`

Exercise VII-5: If `a=[1 2 3 4 5 4 3 2 1]`, what is the numerical value of `mean(a(4:6))`. Verify by hand and with MATLAB. Explain how this command works.

Exercise VII-6: Continuing with the previous question, what is `3* mean(a(4:6))/4`?

Exercise VII-7: Continuing with the previous question, what is `0.75* mean(a(4:6))`? Do you think this method is more or less efficient than the method used in the previous exercise?

For the next few problems, use the following variables:

$A1=[1 \ 8 \ 3 \ -7, \ -2, \ 6]$

$B1=[1 \ 99 \ 2 \ 33 \ 55 \ 99]$

Calculate the quantities by hand and then check with MATLAB.

Exercise VII-8: $A1+B1$

Exercise VII-9: $A1-B1$

Exercise VII-10: $A1.*B1$

Exercise VII-11: $A1./B1$

Exercise VII-12: $(A1-B1)./B1$

Exercise VII-13: $(A1+B1).*B1$

Exercise VII-14: Generate a plot of the parabola $y=-x^2-2x+2$. Use 200 points and let x vary from -7 to $+5$. Label the title, x -axis, and y -axis of the plot. Display the grid on the plot.

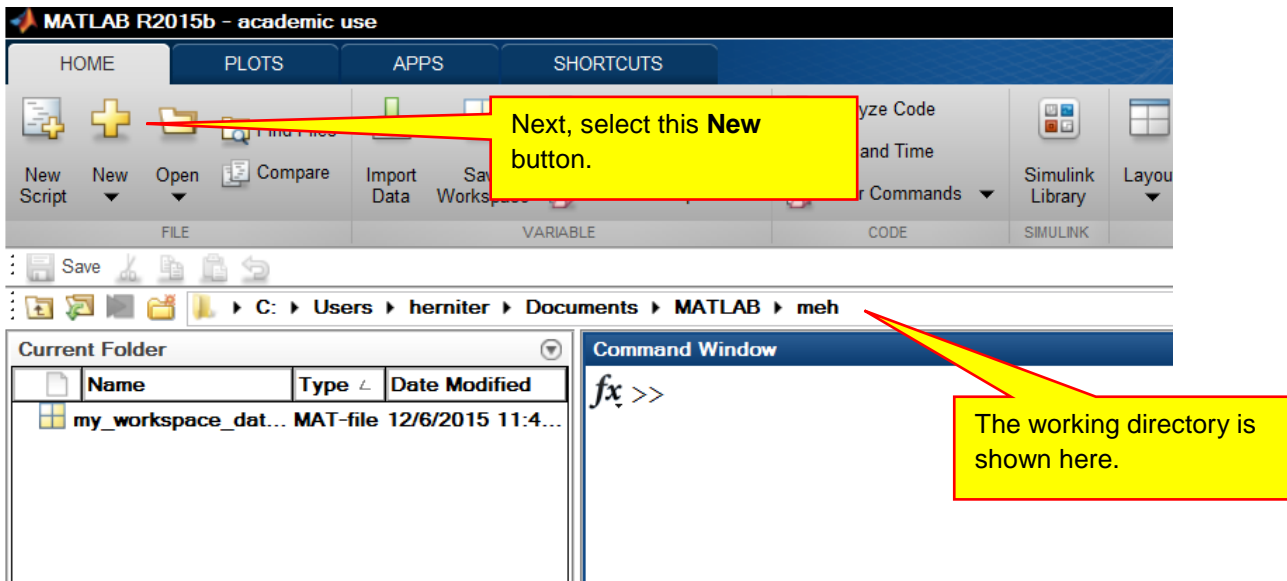
Exercise VII-15: Generate a plot of the function $y=x^3-x^2-2x+2$. Use 200 points and let x vary from -7 to $+7$. Label the title, x -axis, and y -axis of the plot. Display the grid on the plot.

Exercise VII-16: Generate a plot of the function $y=2x+2$. Use 200 points and let x vary from -3 to $+3$. Label the title, x -axis, and y -axis of the plot. Display the grid on the plot.

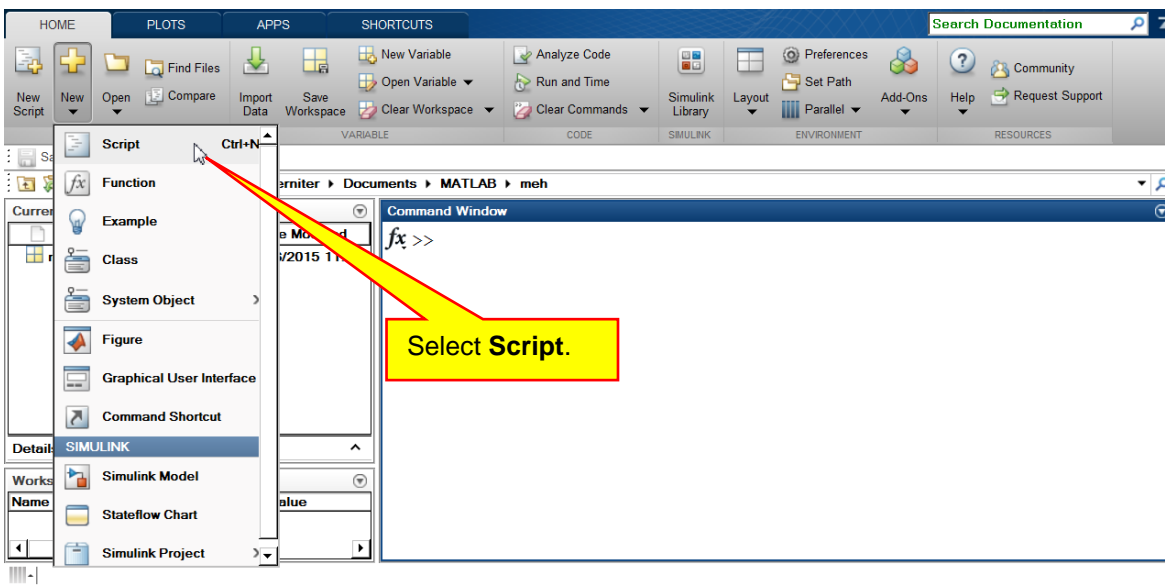
Lesson VIII: Script Files and Control Flow

A. Script Files

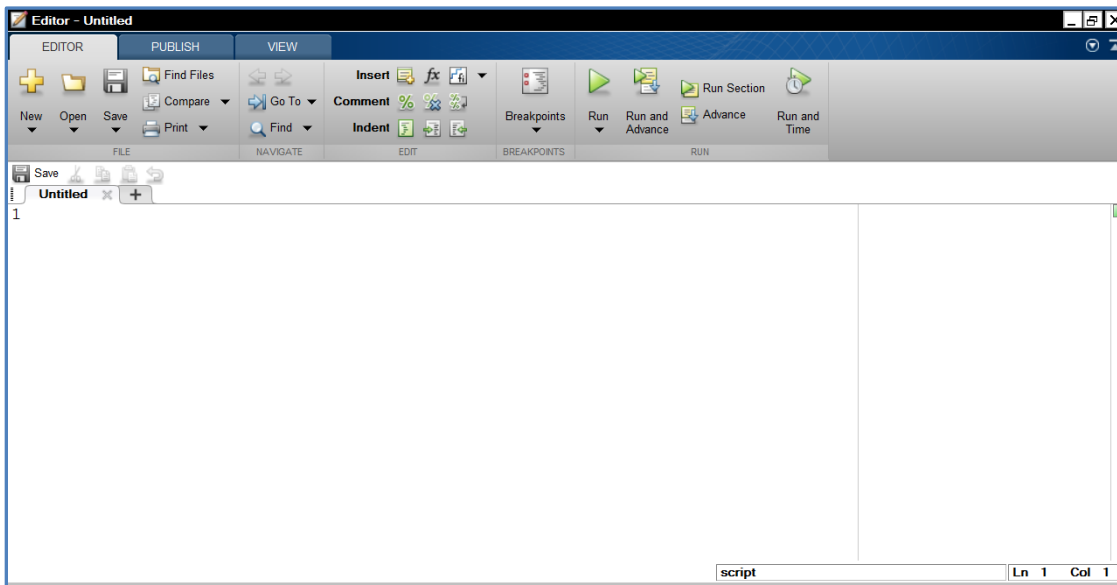
Script files are text files that contain a sequence of MATLAB commands. If you have a large number of commands that you type in at the command prompt repeatedly, instead of typing in the commands at the command prompt over and over, you can place the commands in a script file and then run the script file. As an example, we will place the plotting commands from Lesson VII.D in a script file. Note that when we create a script file, it is placed in the current directory. Thus, make sure that your current directory is the directory that you created to save your files:



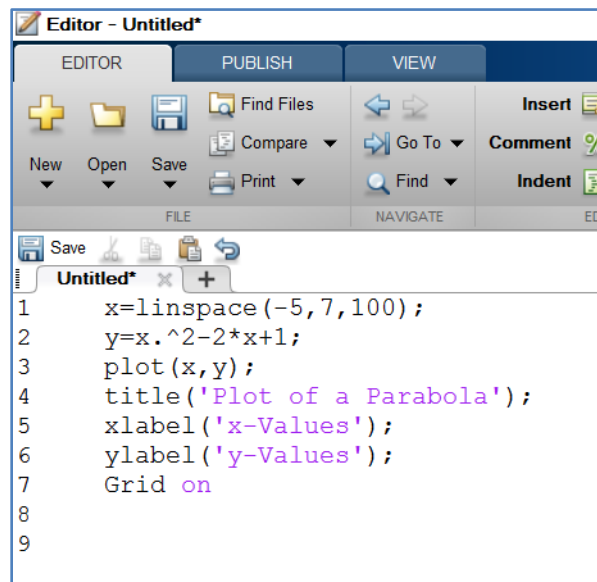
Select the **New** button as shown above and then select **Script**:



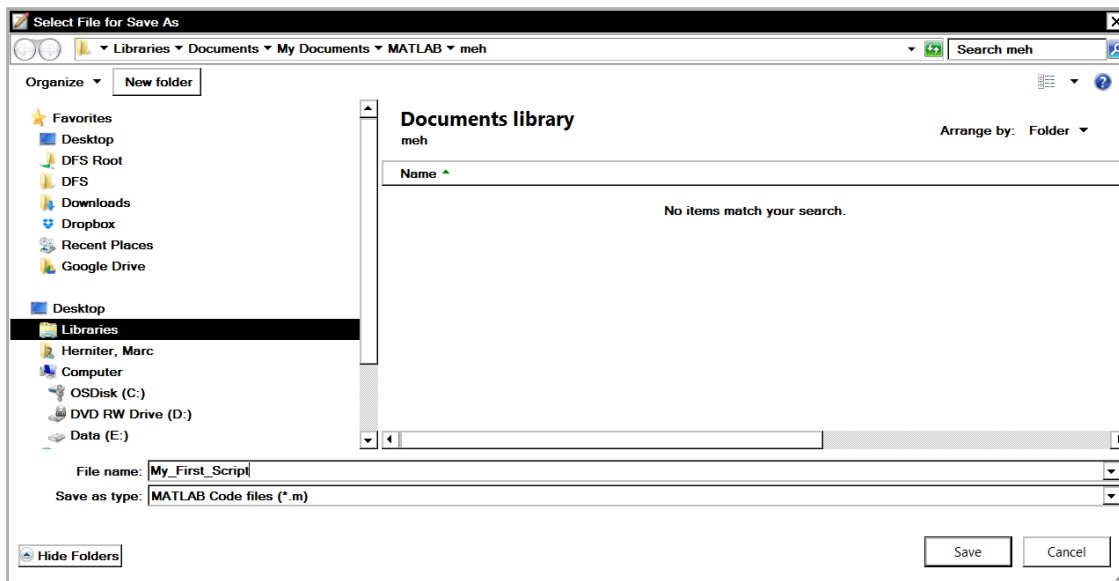
An empty **Editor** window will open:



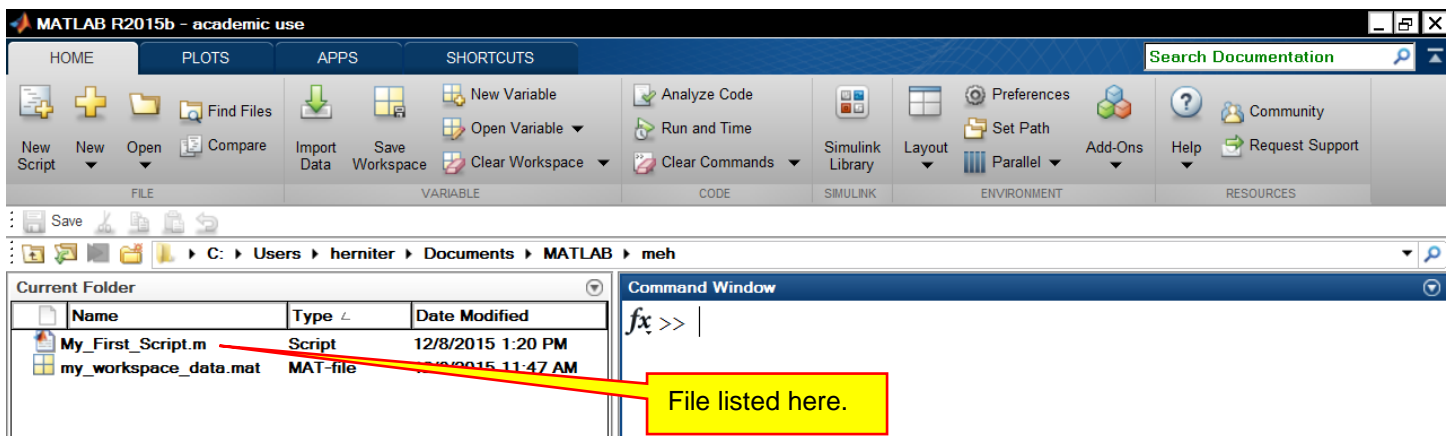
Type in the commands that we used to generate the plot in Lesson VII.D:



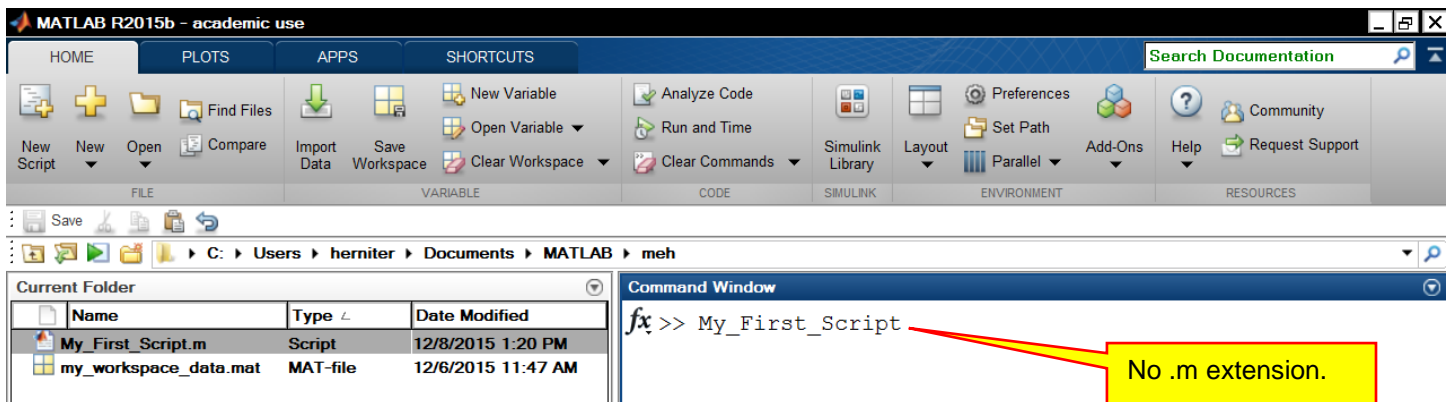
Next, save the script by clicking on the **Save** button. When you name the file, it cannot have any spaces in the name. The naming convention is the same as for variables; you can use alphanumerical characters and the underscore. I will name my file "My_First_Script"



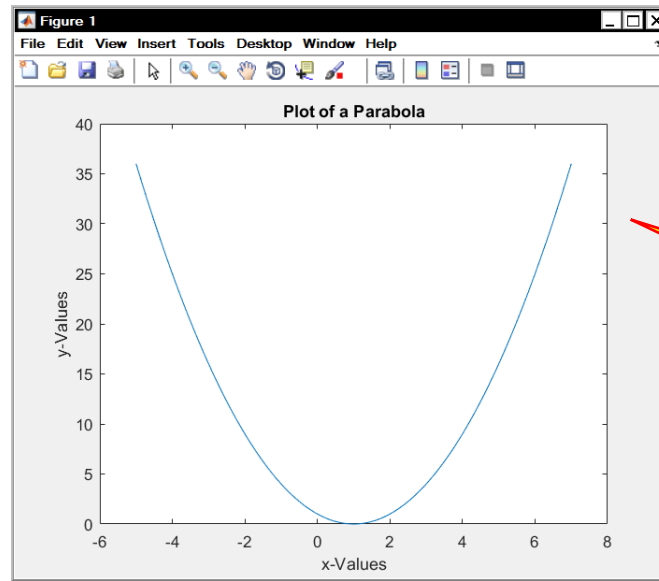
When you save the file, MATLAB will add a `.m` file extension to the file name. (The file will be saved as `My_First_Script.m`.) The file should be listed in your current directory:



We will show three ways to run your file. The first way is to type the name of your script at the MATLAB prompt. Note that you type the name without the `.m` extension:



When you press the **ENTER** key, the script will run. In our case, if we have no errors, the plot will show up with all of the annotations and a grid:



My script mostly ran. However there is no grid. Close the plot window. The MATLAB Command window shows an error highlighted in red:

Current Folder

Name	Type	Date Modified
My_First_Script.m	Script	12/8/2015 1:20 PM
my_workspace_data.mat	MAT-file	12/6/2015 11:47 AM

My_First_Script.m (Script)

Workspace

Name	Value
x	1x100 double
v	1x100 double

```
>> My_First_Script
Cannot find an exact (case-sensitive) match for 'Grid'

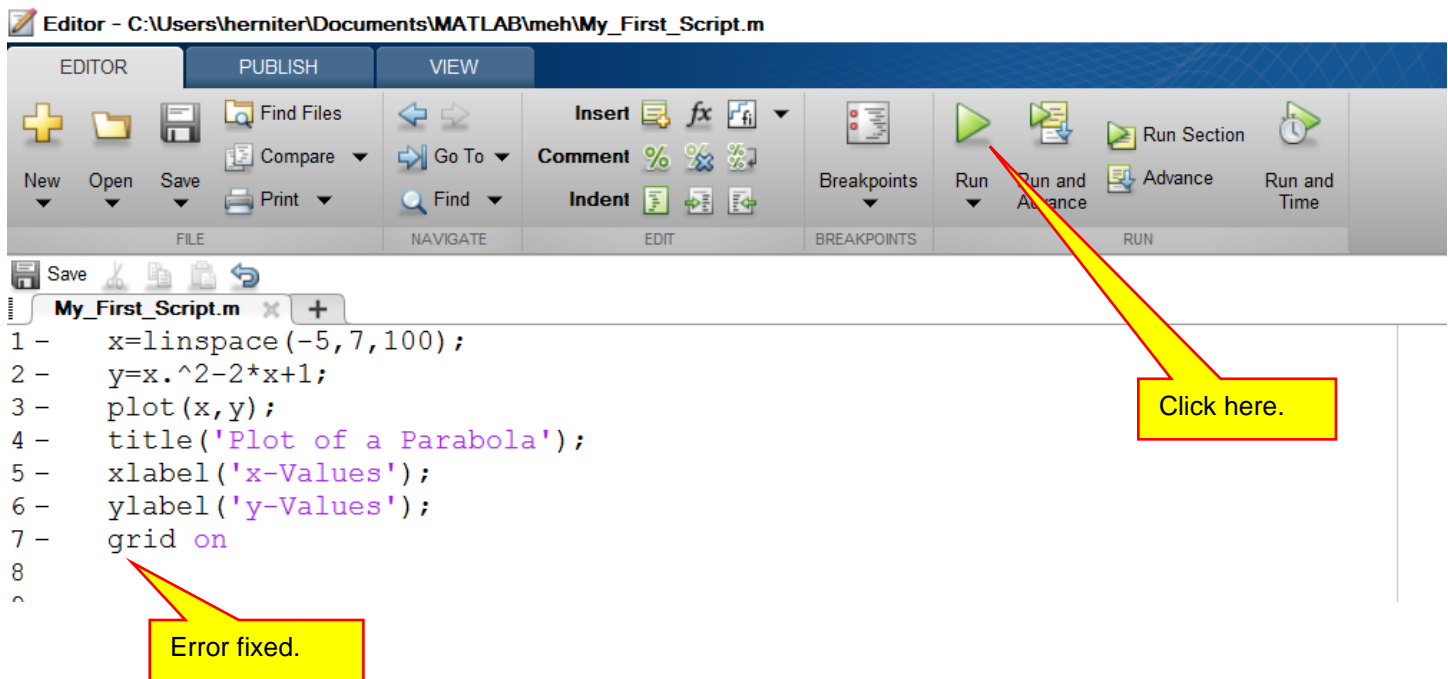
The closest match is: grid in
C:\MATLAB\R2015b_64\toolbox\matlab\graph2d\grid.m

Error in My_First_Script (line 7)
Grid on

fx >> |
```

The error says that line 7 of my script generated an error and that MATLAB does not know what the Grid function is. When I look at my script, I notice that I have “Grid on” and not “grid on.” (It turns out that MS Word auto capitalized the command while writing this manual. I just copied and pasted that command into the script file.)

I will go back to the script file and correct line 7:



Editor - C:\Users\herniter\Documents\MATLAB\meh\My_First_Script.m

EDITOR PUBLISH VIEW

FILE NAVIGATE EDIT BREAKPOINTS RUN

My_First_Script.m

```

1 - x=linspace(-5,7,100);
2 - y=x.^2-2*x+1;
3 - plot(x,y);
4 - title('Plot of a Parabola');
5 - xlabel('x-Values');
6 - ylabel('y-Values');
7 - grid on
8
^

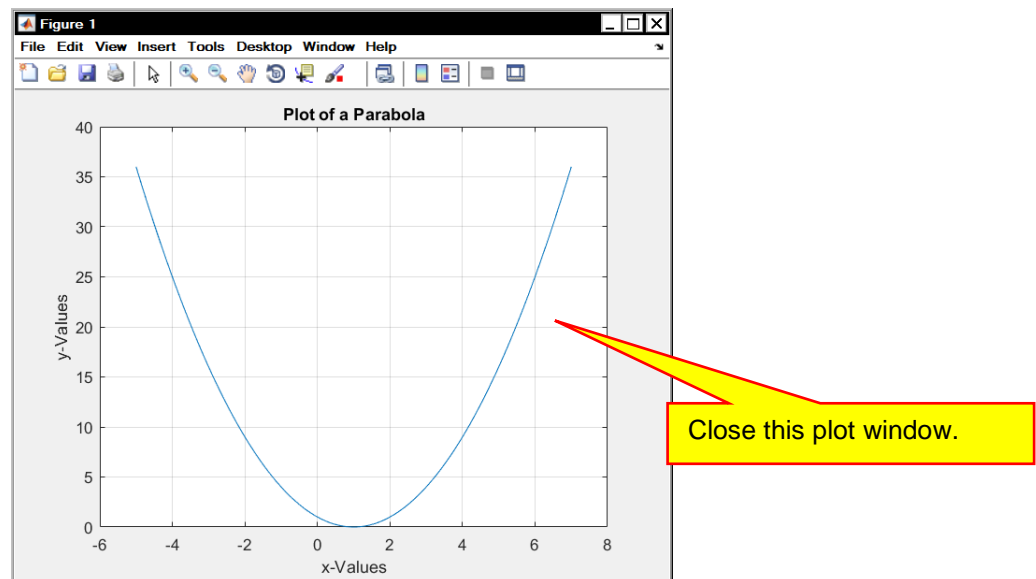
```

Click here.

Error fixed.

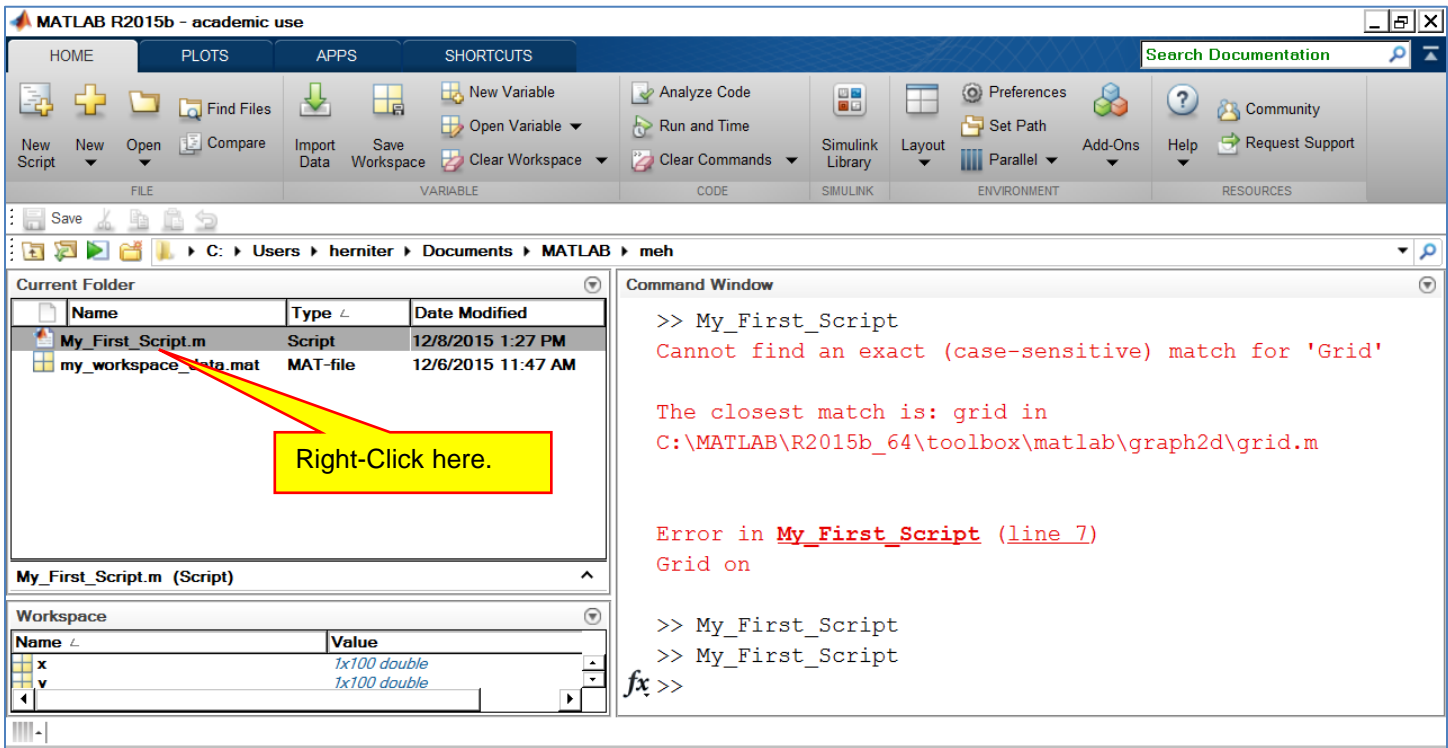
Save the script file before continuing.

A second way to **Run** the script is to click on the **Run** button as shown above. The script will run and the plot should now be correct.

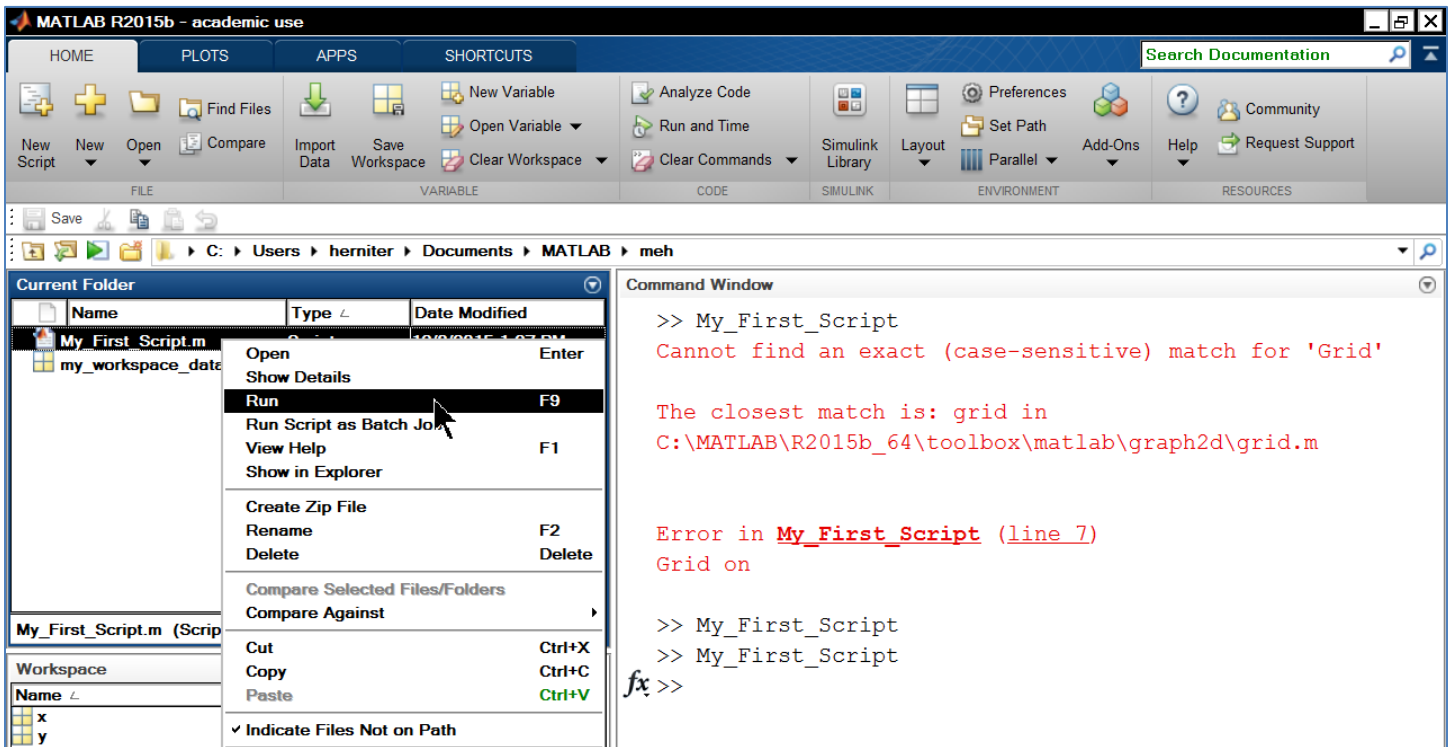


The plot now appears to be correct. Close the window.

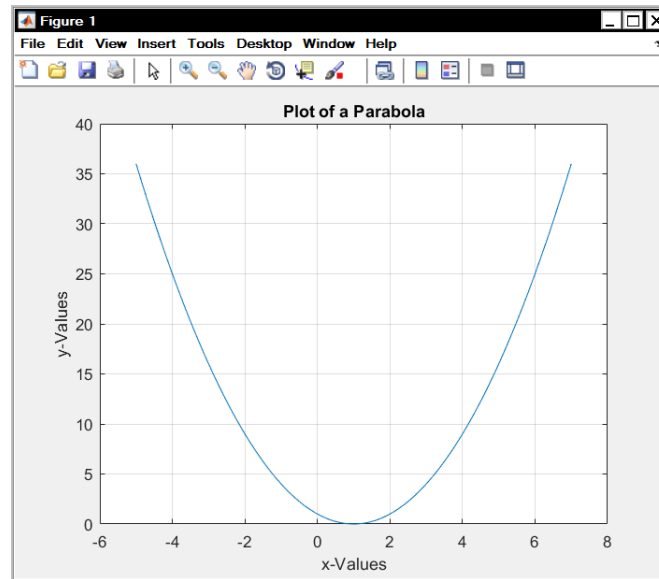
A third way of running the script file is to right-click (Mac OSX: Control-Click) on the file name in the file **Current Folder** window:



Select **Run** in the menu that appears:



The script should run and the figure window should open:



B. The If Statement

The **IF** statement is used to choose between different options based on a logical decision. This is called control flow. For example, if x is 5 you can do one thing, and if x is not 5 you can do something else. Since control flow usually involves several MATLAB commands, the commands are usually placed inside a script file. The syntax of the IF statement is the following:

```
if expression1
    statements1
elseif expression2
    statements2
else
    statements3
end
```

In the statement above, if expression1 is true, then only the statements in section **statements1** are executed. The remainder of the **IF** statement is ignored. **IF** expression1 is false, then expression2 is checked. If expression2 is true, then only the statements in section **statements2** are executed and the remainder of the **IF** statement is ignored. If expression1 is false and expression2 is false, then the statements inside the **ELSE** clause (statements3) are executed. Thus, the statements inside the **ELSE** clause are executed only if all of the other conditions are false.

The **ELSEIF** and **ELSE** portions of the statement are optional. Thus, we could have the statement:

```
if expression1
    statements1
end
```

So, if expression1 is true statements1 are executed. If expression1 is false, then the statement are not executed.

You can have multiple **ELSEIF** statements:

```
if expression1
    statements1
elseif expression2
    statements2
```

```
elseif expression3
    statements3
elseif expression4
    statements4
elseif expression5
    statements5
else
    statements6
end
```

In this example, the expressions are checked sequentially. The statements that are executed are the ones with the first true expression. If none of the expressions are true, then the statements in the **ELSE** clause are executed.

As an example, we will execute the following set of commands:

```
if x==5
    x=x+1;
end
% Display the value of x
x
```

This code segment tests if x is equal to 5. If so, 1 is added to x . If x is not 5, then MATLAB continues executing the statements after the end statement. In this case, the value of x is displayed. A few comments about the code are in order. First, the operator to test equality is `==`. Thus `x==5` tests if x is equal to 5. Note that this is different than `x=5` which sets x to 5. The valid operators for conditions in IF statements are listed in the table below. Note that in MATLAB they are referred to as relational operators:

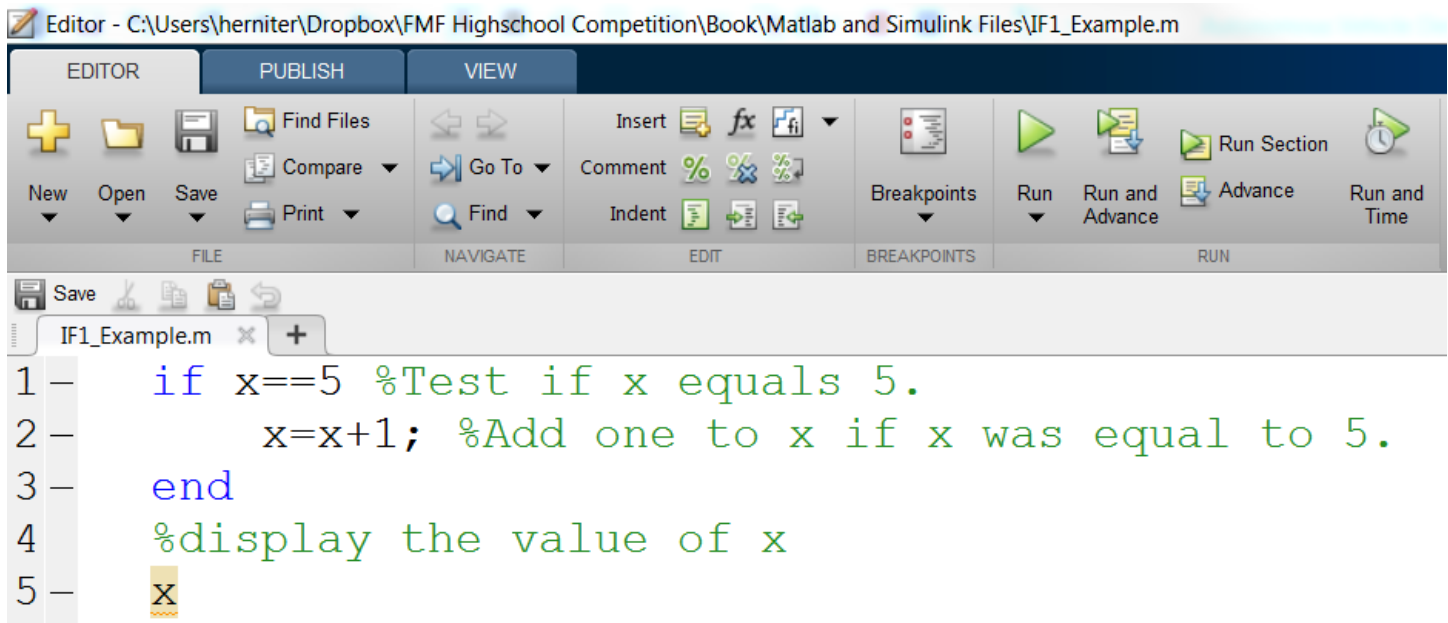
Relational Operators	
<code>==</code>	equal
<code>>=</code>	Greater than or equal to
<code>></code>	Greater than
<code><=</code>	Less than or equal to
<code><</code>	Less than
<code>~=</code>	Not equal to

We also see that the % sign specifies a comment. Any text that follows the % is ignored. The example below shows the previous code with numerous comments:

```
if x==5 %Test if x equals 5.
    x=x+1; %Add one to x if x was equal to 5.
end
%Display the value of x
x
```

Note that a comment can take up an entire line or can appear after MATLAB statements.

Since **IF** statements usually involve multiple lines of code, we will place the code inside a MATLAB script file. We covered script files in Section VIII.A. Note that a script file has a `.m` extension. Place this code inside a script file and name it `IF1_Example.m`. My file is shown below:

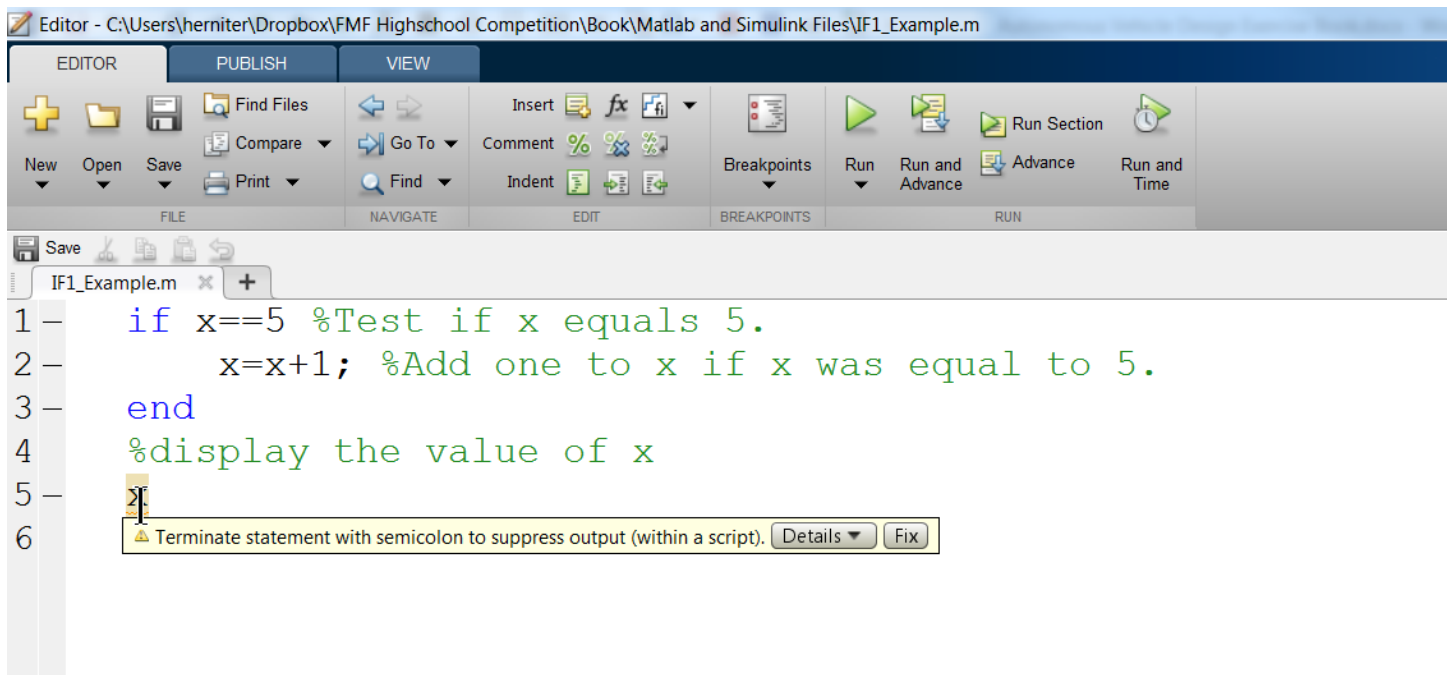


```

1 -   if x==5 %Test if x equals 5.
2 -       x=x+1; %Add one to x if x was equal to 5.
3 -   end
4 -   %display the value of x
5 -   x

```

Note that the MATLAB text editor helps us by highlighting different portions of the script file. For example, MATLAB keywords like `if` and `end` are shown in blue. Comments are shown in green. Possible errors are underlined in red and highlighted. Note that in line 5, the `x` is highlighted as a possible error. The editor assumes that all statements will be terminated with a semicolon so that the output of the statement is suppressed. If you hover the mouse pointer over the highlighted statement, the editor will specify the error:



```

1 -   if x==5 %Test if x equals 5.
2 -       x=x+1; %Add one to x if x was equal to 5.
3 -   end
4 -   %display the value of x
5 -   x
6 -

```

Terminate statement with semicolon to suppress output (within a script). Details Fix

In our case, this is not an error because we want to use this statement to display the value of `x`. Save the script. Next we will set `x` to a value and then run the script. In the command window, set `x` to 3 and then run the script by typing its name:

```

Command Window
>> x=3;
>> IF1_Example

x =

     3

fx >>

```

Since x was 3, the condition $x==5$ was false, so 1 was not added to x . Next, we will set x to 5 and then run the script:

```

Command Window
>> x=5;
>> IF1_Example

x =

     6

fx >> |

```

In this case, the condition was true, so the statements inside the if-end statement were executed, and 1 was added to x .

Note that we can have multiple statements inside the if-end keywords. For example, if the condition is true, we will add 1 to the number, then divide it by 2, and then subtract 1 from it. (Note that we could do this with a single statement had we wanted to. The code segment is shown below:

```

if x==5 %Test if x equals 5.
    x=x+1; %Add one to x if x was equal to 5.
    x=x./2; %Divide x by 2
    x=x-1; %Subtract 1 from x.
end
%Display the value of x
x

```

Note that we could have done this all in the single statement $x=(x+1)./2-1$; However, we are illustrating the fact that we can have multiple statements inside an if-else statement. Save this modified code segment as a new script file, say IF2_Example. and then test the script with the same inputs as before:

```
Command Window
>> x=3;
>> IF2_Example

x =

     3
|
>> x=5;
>> IF2_Example

x =

     2

fx >> |
```

As an example of the `if-elseif-else-end` statement, we will implement the following logic. If the number is equal to 5, add one to it. If that is not true, check if the number is greater than or equal to 10. If so, add 2 to it. If that is not true, check if the number is less than -5. If so, set the number to zero. If none of the conditions are true, set the value to 100. The code segments is shown below:

```
if x==5 %Test if x equals 5.
    x=x+1; %Add one to x if x was equal to 5.
elseif x>=10
    x=x+2;
elseif x < -5
    x=0;
else
    x=100;
end
%display the value of x
x
```

Tests of the script are shown below:

```
Command Window
>> x=3;
>> IF3_Example

x =

    100

>> x=5;
>> IF3_Example

x =

     6

>> x=75;
>> IF3_Example

x =

    77

>> x=-10;
>> IF3_Example

x =

     0
```

For the numbers shown, the script appears to work correctly.

Note that the IF statement does not have to have any elseif clauses or it can have several elseif clauses. Furthermore, the IF statement does not have to have the else either.

As a last example, we will show an example of an expression that uses a nested IF statement and a compound expression. A compound expression can test multiple logical expressions at the same time. An example is $(x==5) \&\& (y==2)$. $\&\&$ is the logical AND operator. Thus both conditions must be true for the entire expression to be true. Thus, x must equal 5 AND y must equal 2 in order for the expression to be true. Another example is $(x==5) \|\| (y==2)$. In this example, $\|\|$ is the logical OR operator. The overall expression will be true if x equals 5, or if y equals 2, or if x equals 5 and y equals 2. (One of both of the expressions being true results in the total expression being true.)

C. For Loops

For loops allow us to loop through a code segment a specified number of times. The syntax is

```
for index = values
    statements
end
```

An example is

```
for i = 1:5 % Loop 5 times.
    i % Display the value of i.
    pause(1); % Pause for 1 second.
end
```

In this example, the index is *i*. the values are 1:5. If you remember the colon operator from section VI.C.1, 1:5 generates the numbers 1, 2, 3, 4, 5. In this code segment, *i* is set to 1 and then the statements are executed. Then *i* is set to 2, and the statements are executed, all the way up to *i* = 5.. Thus, the code segment above loops five times. Each time through the loop, the value of *i* is displayed and MATLAB pauses for 1 second. Save this code segment in a script file named For1_Example. Executing the code segment yields the following output.

```
>> For1_Example
i =
    1
i =
    2
i =
    3
i =
    4
i =
    5
>>
```

An example of combining the `for` loop with an **IF** statement is:

```
for i = 1:5 %Loop 5 times.
    if i==3 %If i equals 3, print the statement below.
        fprintf('The value of i is 3!\n');
    else
        i %If i is not equal to 3, display the value of i.
    end
end
```

Here we loop 5 times with *i* set to values from 1 to 5. If the value of *i* is three, we print the text 'The value of i is 3!'. If the value is not 3, then the value of *i* is displayed. Note that the `\n` in the `fprintf` statement causes the line to be printed, generates a new line, and goes to the beginning of a new line. (To see the effect of the `\n`, try running the script with the `\n` removed!)

```
>> For2_Example
i =
    1
i =
    2
```


The value of i is 3!

```
i =
    4
i =
    5
```

As a last example, we can count down with a **for** loop.

```
for i = 5:-1:1 %Loop 5 times. Count down.
    if i==3 %If i equals 3, print the statement below.
        fprintf('The value of i is 3!\n');
    else
        i %If i is not equal to 3, display the value of i.
    end
end
```

The only difference is in the use of the colon operator. 5:-1:1 generates numbers from 5 down to 1 with a step of -1. (This was also covered in section VI.C.1.) This code segment generates the following output:

```
>> For3_Example
i =
    5
i =
    4
The value of i is 3!
i =
    2
i =
    1
```

In our Autonomous Vehicle project, we will be using **FOR** loops to step through the values of an array. An example of this is shown below:

```
x=[15,3, 0,-99, 88]; %Define the array with 5 elements.

for i = 1:5 %i goes from 1 to 5. This generates the indices 1,2,3,4,5.
    x(i) %Display the value of the element.
    if x(i) >0 %Check if the element is positive.
        fprintf('This element of x is positive.\n');
    elseif x(i)<0 %Check if the element is negative.
        fprintf('This element of x is negative.\n');
    else %If we get here, the element is zero.
        fprintf('This element of x is zero.\n');
    end
end
```

Here, x has 5 values, x(1)=15, x(2)=3, x(3)=0, x(4)=-99, and x(5)=88. The **for** loop is used to step through each element of x. Since i goes from 1 to 5, we step through all of the values of x. For each value of i, the code segment

```
x(i) %Display the value of the element.
if x(i) >0 %Check if the element is positive.
    fprintf('This element of x is positive.\n');
elseif x(i)<0 %Check if the element is negative.
    fprintf('This element of x is negative.\n');
else %If we get here, the element is zero.
    fprintf('This element of x is zero.\n');
end
```

is executed. Thus, we repeat the code segment for each individual value of x . This code segment displays the value of $x(i)$ and then states whether the element is positive, negative, or zero. Running the entire script yields the output below:

```
>> For4_Example
ans =
    15
This element of x is positive.
ans =
     3
This element of x is positive.
ans =
     0
This element of x is zero.
ans =
   -99
This element of x is negative.
ans =
    88
This element of x is positive.
>>
```

D. Questions

Question VIII-1: What is a script file?

Question VIII-2: What is a file extension?

Question VIII-3: What is the file extension for script files?

Question VIII-4: What is the naming convention for script files?

Question VIII-5: Are spaces allowed in script file names?

Question VIII-6: What are the alphanumeric characters?

Question VIII-7: Specify three ways of running a script file.

Question VIII-8: What is control flow?

Question VIII-9: Specify three different forms of the IF statement.

Question VIII-10: True or False: In an IF statement, you do not need to use the ELSEIF portion of the statement.

Question VIII-11: True or False: In an IF statement, you do not need to use the ESLE portion of the statement.

Question VIII-12: What are relational operators?

Question VIII-13: How do you specify a comment in MATLAB code?

Question VIII-14: How are errors displayed in the MATLAB Editor?

Question VIII-15: How do you test if a script works correctly?

Question VIII-16: What are logical operators?

Question VIII-17: What is the AND operator? Give an example.

Question VIII-18: What is the OR operator. Give an example.

Question VIII-19: What is the function of a for loop?

Question VIII-20: Can for loops count up, down, or both?

Question VIII-21: Specify how for loops can step through the elements of an array.

Question VIII-22: What is an index?

Question VIII-23: What is the colon operator?

E. Exercises

Exercise VIII-1: Create a script that tests if a variable named y is equal to 17. If it is equal to 17, add 2 to it.

Exercise VIII-2: Create a script that tests if a variable named y is equal to 17. If it is equal to 17, add 2 to it. If it is not equal to 17, subtract 3 from it.

Exercise VIII-3: Create a script that sets x to a number between 1 and 20. If x is prime, output the text string "The number is prime." If the number is not prime, output the message, "The number is not prime." Solve this problem using an if and fprintf statements.

Exercise VIII-4: Create a script that sets x to a number between 1 and 20. If x is prime, output the text string "The number is prime." If the number is not prime, output the message, "The number is not print." Solve this problem using an array, for loops, if statements, and fprintf statements.

Exercise VIII-5: Define x as the row vector [1, 3, 5, 8, 88, 72, 6]. Create a script that defines this array, and then displays the values of the elements in reverse order.

Exercise VIII-6: Define x as the row vector [1, 3, 5, 8, 88, 72, 6]. Create a script that defines this array, then displays the values of the elements in reverse order, and then displays the values of the elements in their original order.

Exercise VIII-7: Define x as the row vector [1, 3, 5, 8, 88, 72, 6]. Use a for loop to display the values of 1st, 3rd, 5th, and 7th elements.

Exercise VIII-8: Define x as the row vector [1, 3, 5, 8, 88, 72, 6]. Use a for loop to display the values of 7th, 5th, 3rd, and 1st elements, in that order.

Exercise VIII-9: The MATLAB mod function finds the remainder after dividing two numbers. For example, $\text{mod}(5,2)$ is the remainder when we divide 5 by 2. Note that $\text{mod}(5,2)=1$ and $\text{mod}(4,2)=0$. We can use this function to see if a positive integer is odd or even. Use this function to determine if a number x is odd or even. Use an IF statement to print out an appropriate text message stating whether the number is odd or even.

Lesson IX: Assembling the Vehicle Steering

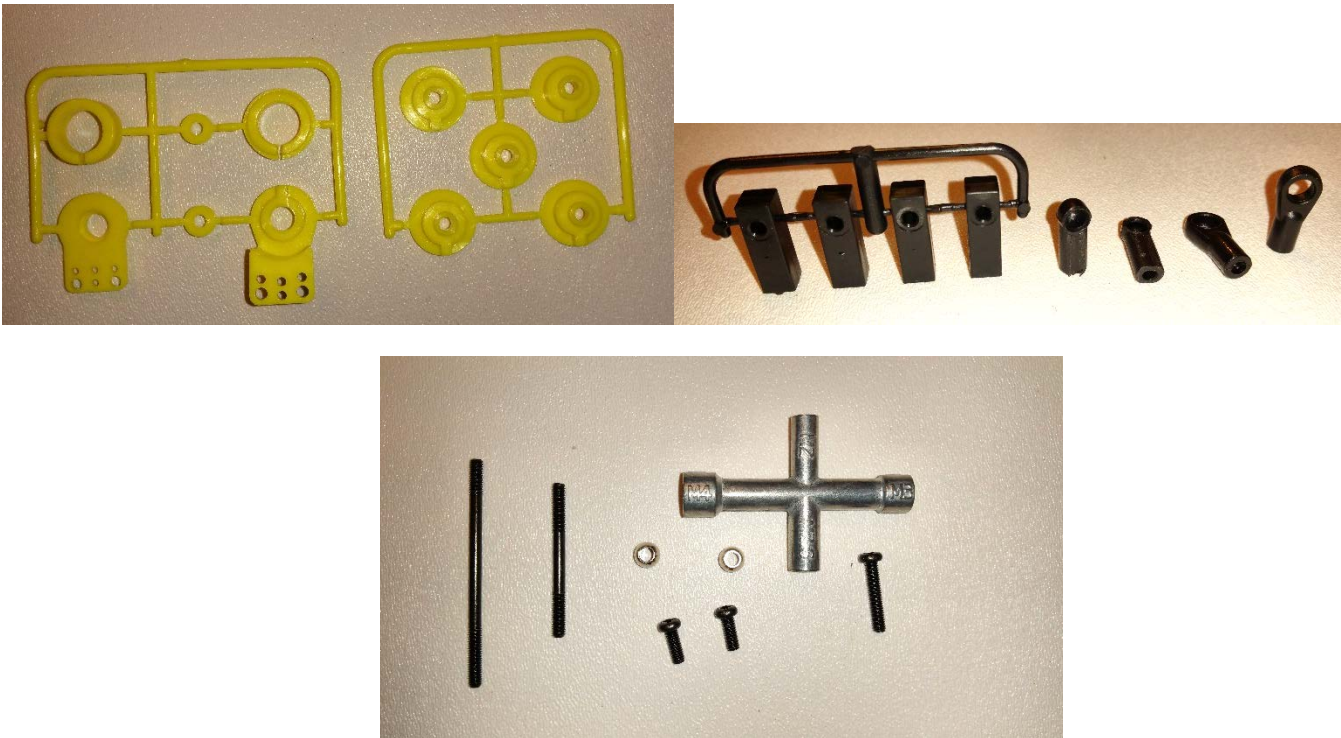
A. Zeroing the Servo Motor

When we assemble the steering, we will adjust the tie rods assuming that the servo motor is at an angle of zero. If this is not true, the steering will have a large offset and it will be difficult to have the vehicle drive straight. To avoid this problem, we will set the servo motor angle to zero now. Repeat Section IV.B on page 57 before continuing.

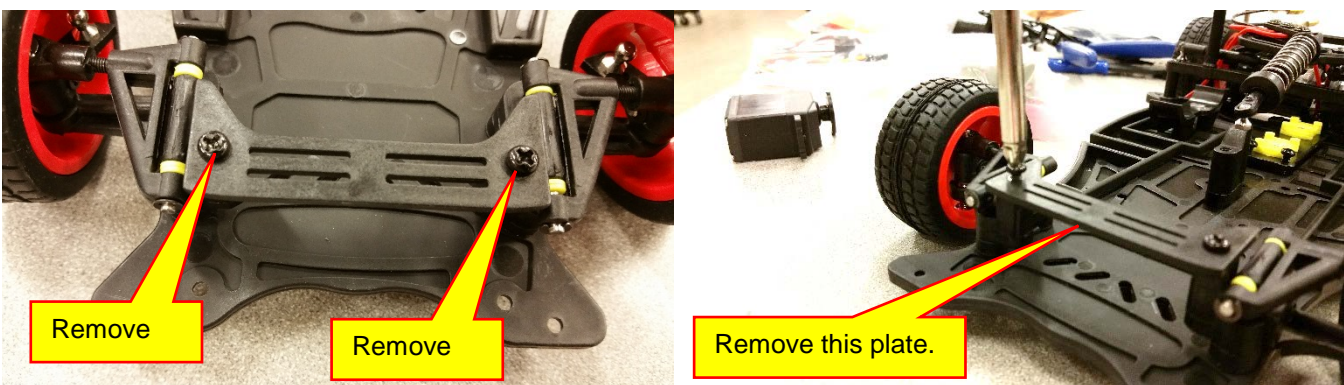
B. Assembling the Steering

As a first step in assembling the car, we will put together the steering. It is important that you zero the servo motor first, or your car will always power up with a steering offset, meaning that it will never be pointed straight.

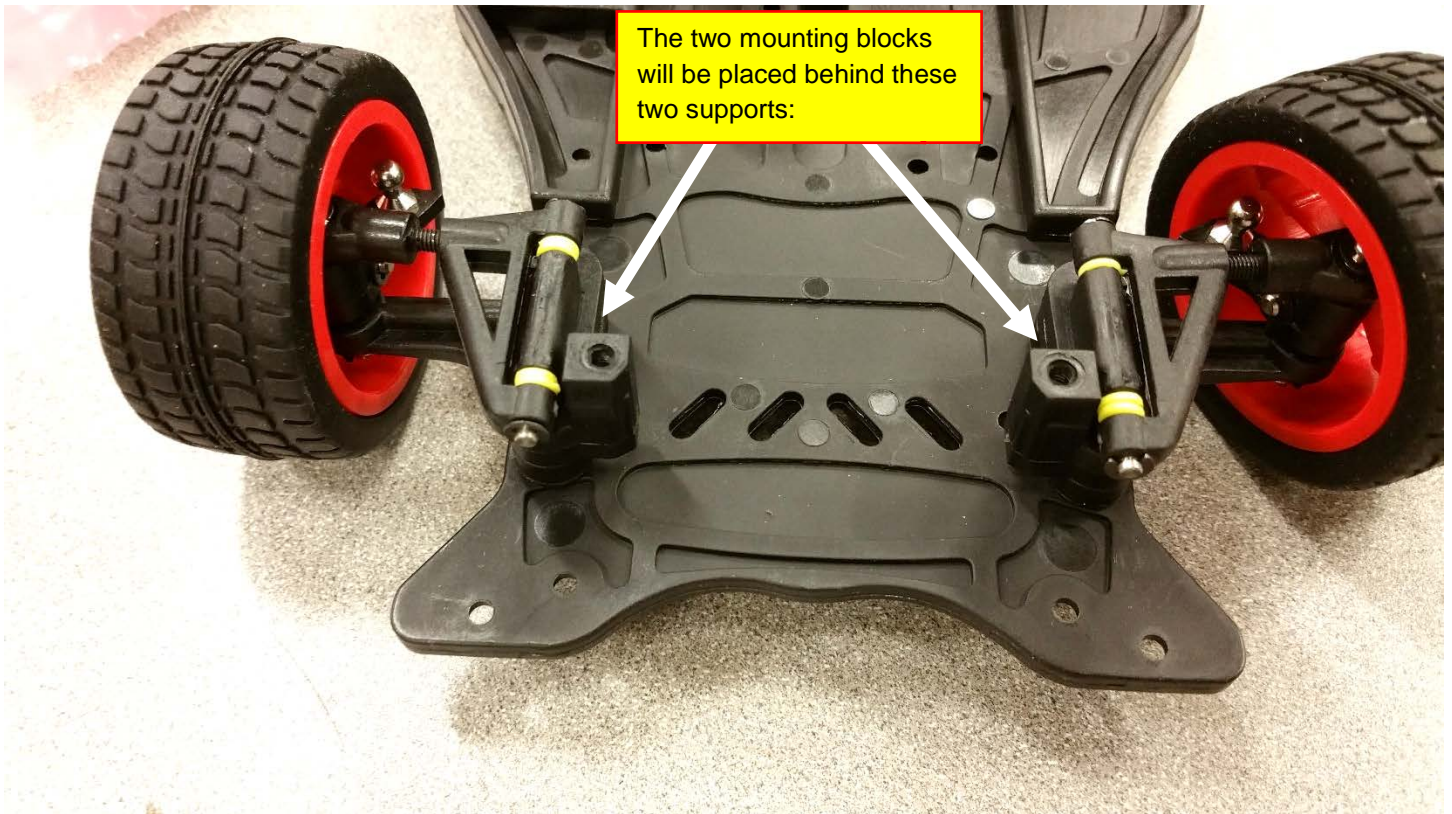
We will be using the parts shown below:



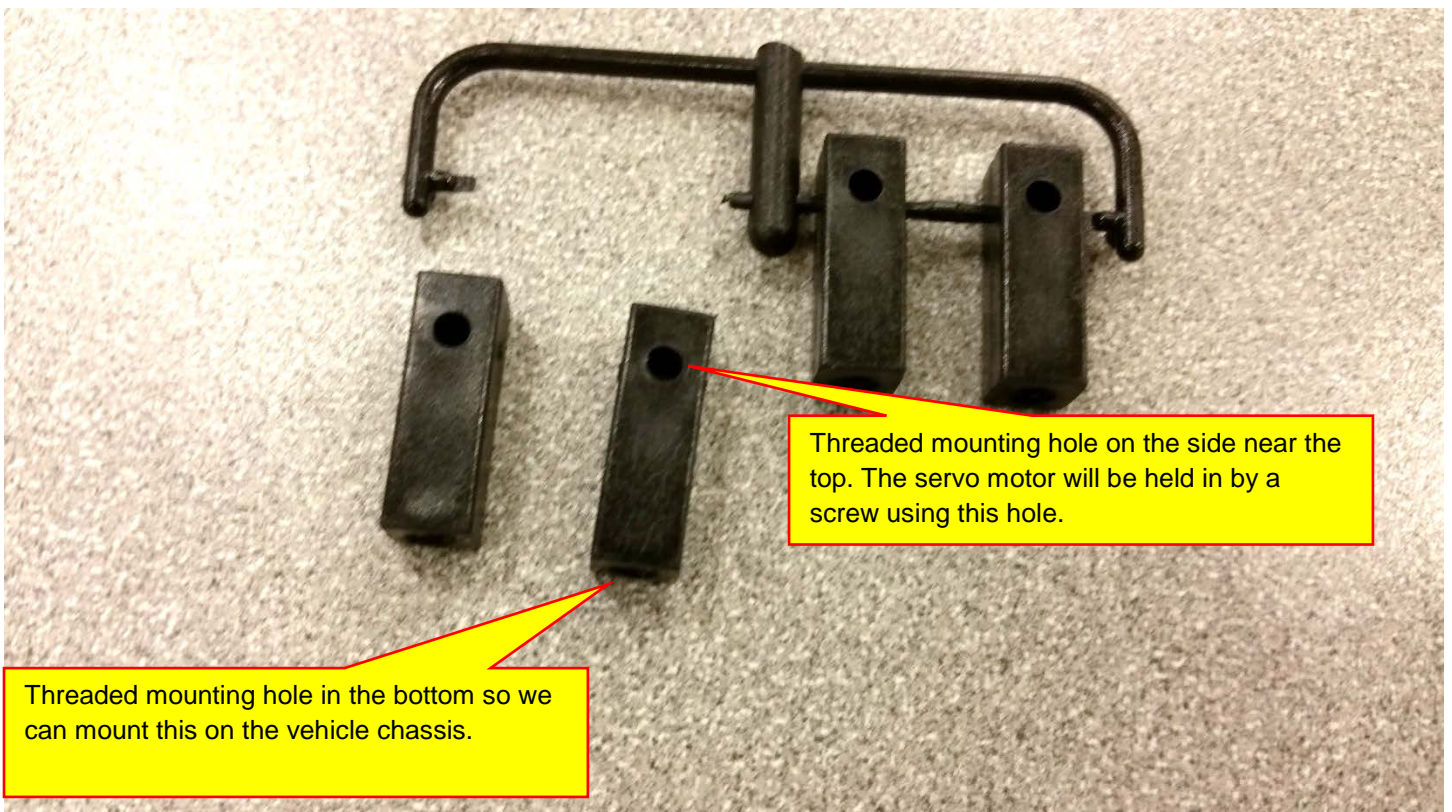
Remove the two screws from the front mounting plate that holds down the steering servo motor:



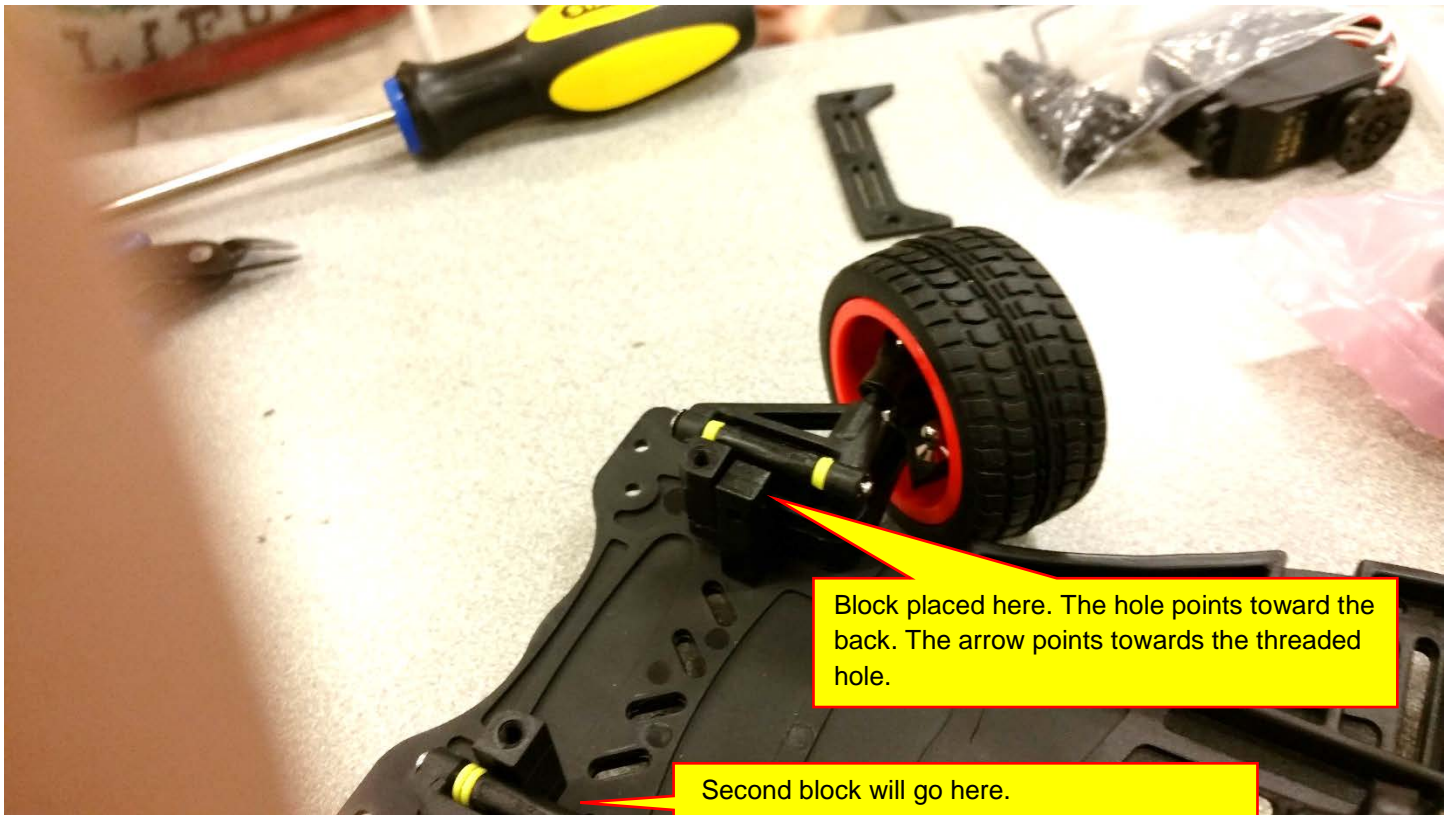
The front should look as shown:



Locate the servo motor mounting blocks and detach two of them:



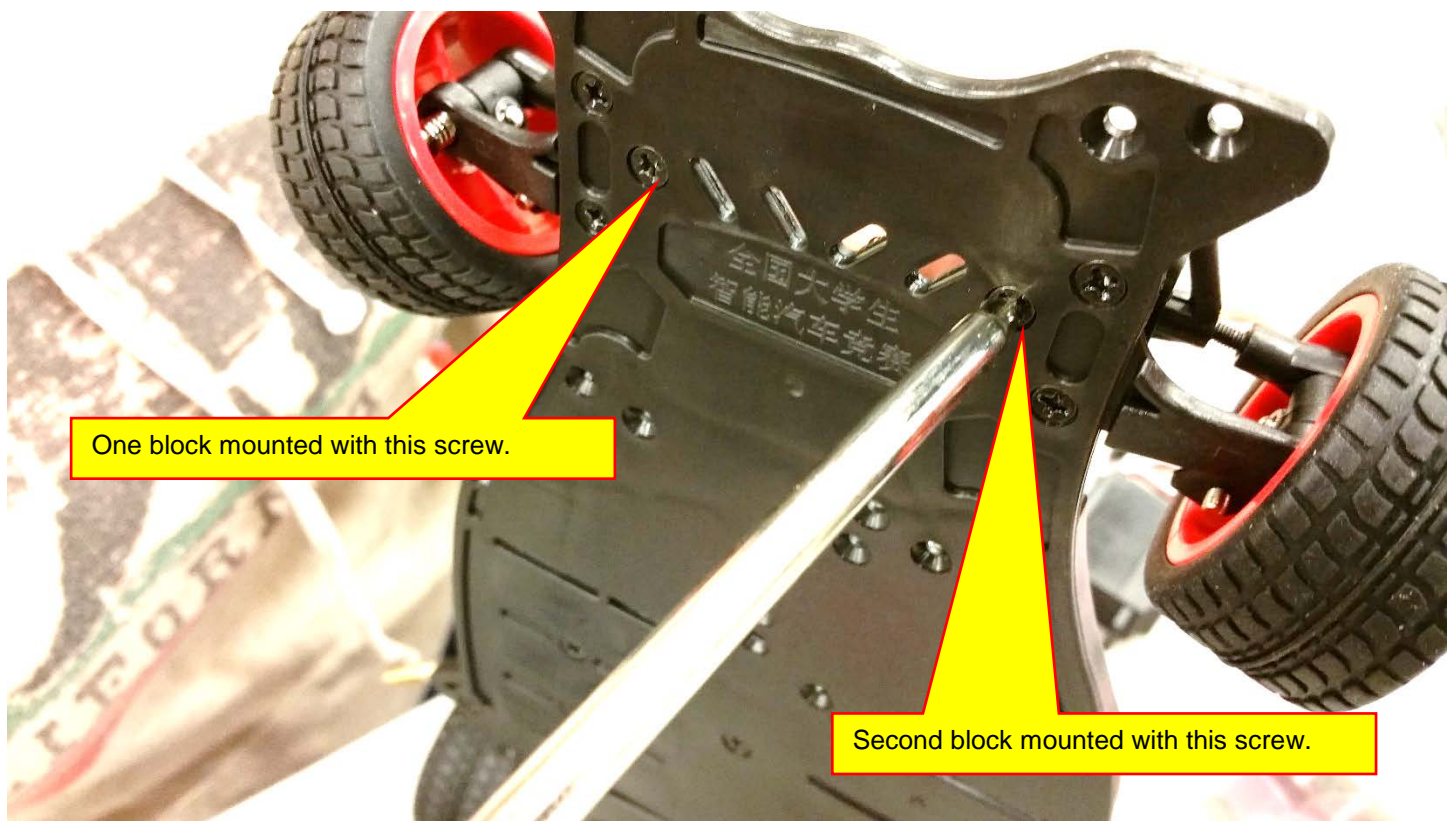
These mounting blocks have a hole on the side and a hole in the bottom. This block will be placed on each side of the vehicle as shown below:



Use a flat pan head screw to attach the mounting block to the chassis. On the bottom side of the chassis, you will see the mounting hole:



Use a flat pan head screw to attach the block. Mount both blocks:



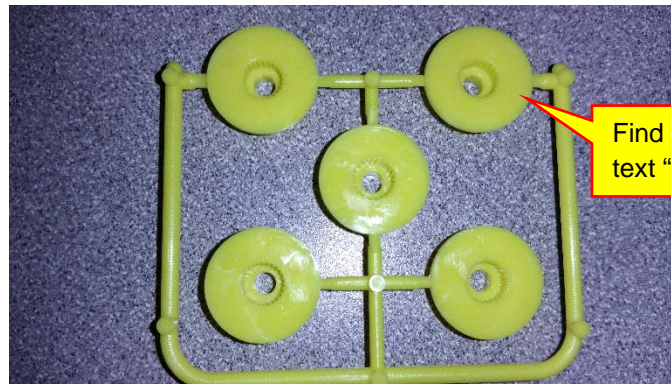
The servo motor should fit as shown below. Do not attach the servo motor yet:



Next, remove the wheel that comes attached to the servo motor:



Find the appropriate splined wheel for our motor. It should have the text 'FU' written on it:

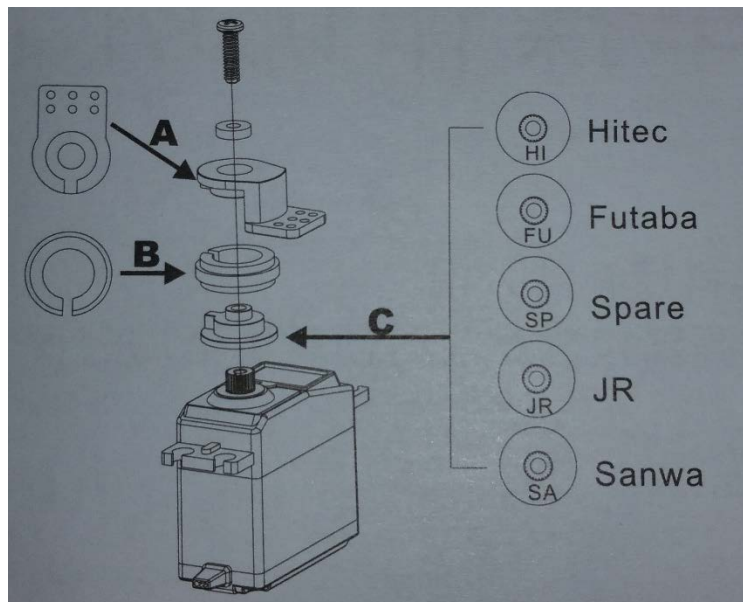


Find the wheel that has the text "FU" on it.

Next, we will assemble the steering plate. You will need the following parts. Note that the order they are shown in this picture is the order they will be placed next to each other.



Assemble them as shown in the instructions that accompany your kit (which is copied below):



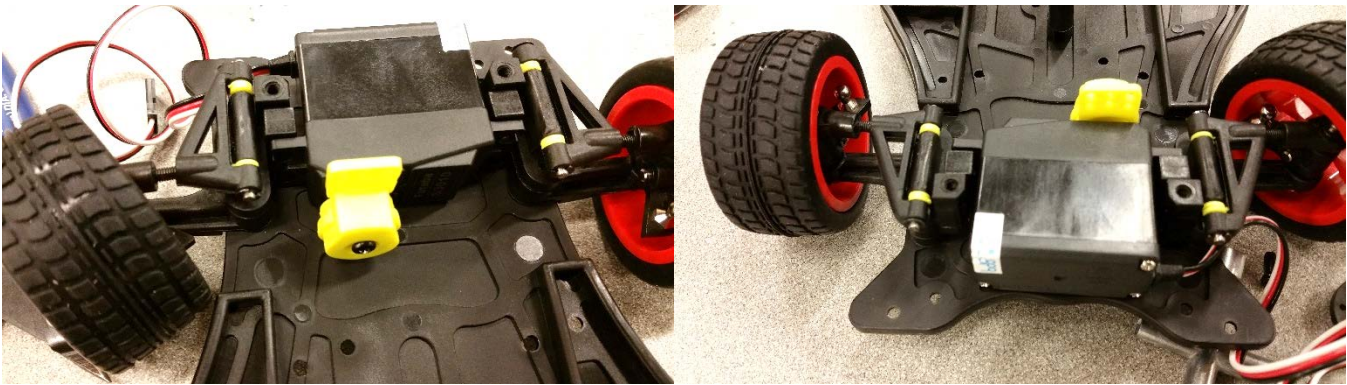
When the parts are placed together, it should look as shown below:



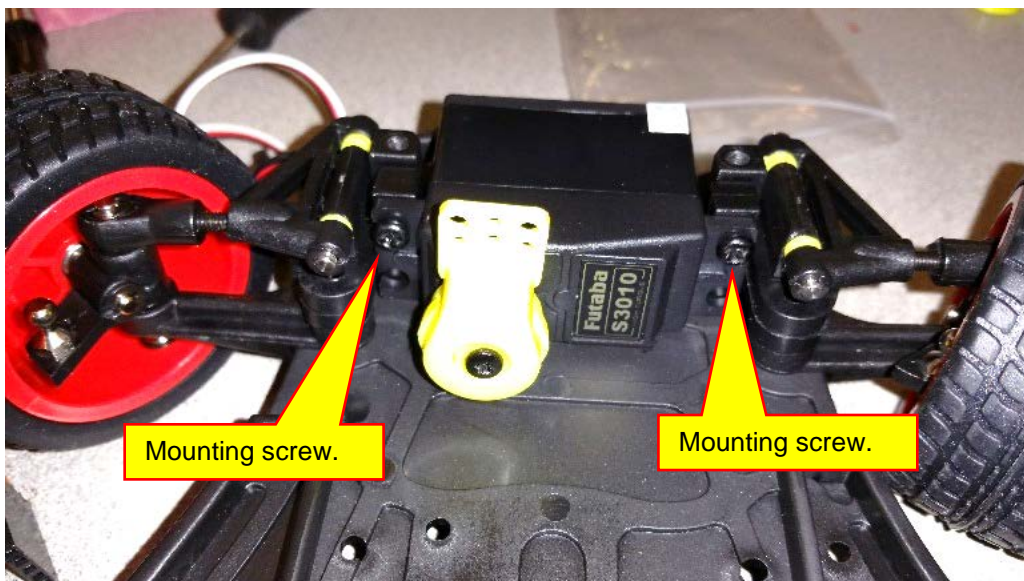
Mount the steering plate on the servo as shown.



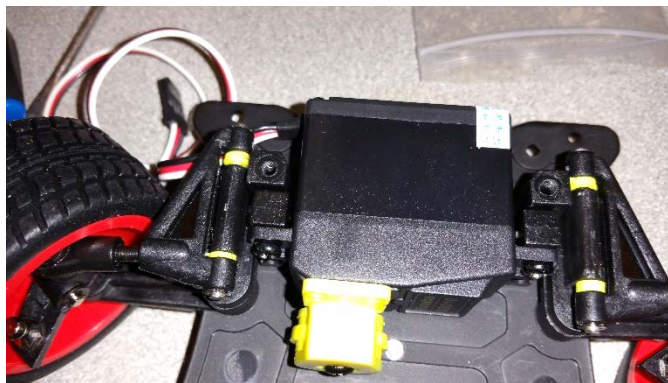
Mount the servo motor in the vehicle chassis:



Secure the servo motor to the chassis with two Philips head screws:

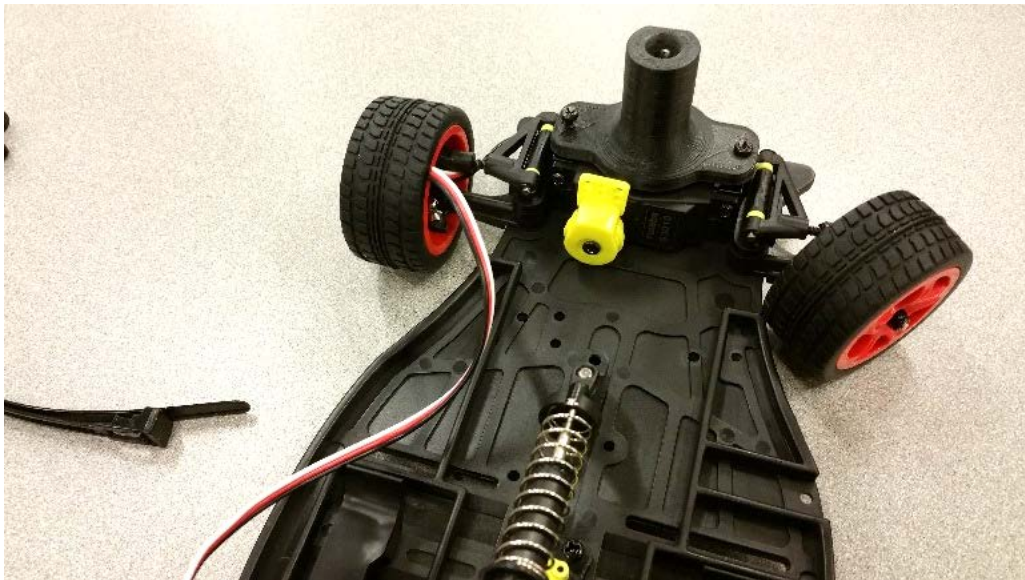


Next, we will attach the camera mount. Use the same screws that we removed from the front plate. Do not reuse the plate that we removed on page 107:





The servo motor assembly is shown below:



C. Assembling the Tie Rods

Next, we will assemble the tie rod arms. Locate the following parts:



Screw on the plastic ends onto to the two tie rods:

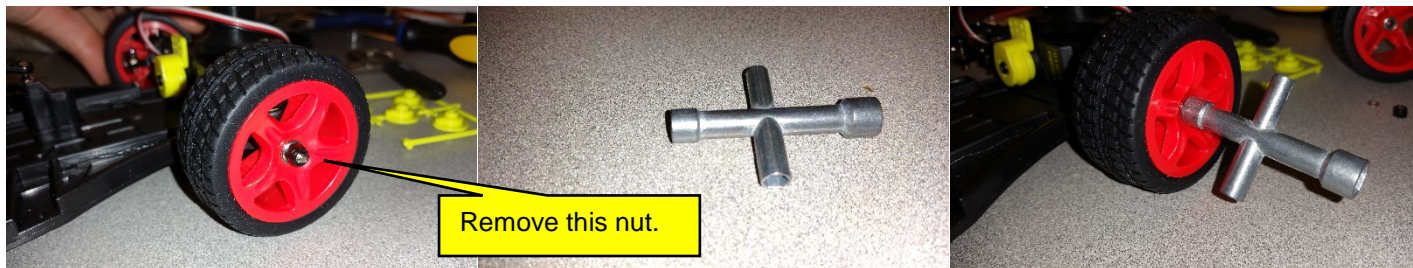


Next, locate the ball bearing/pivot bearing and then snap it into the plastic holder using a needle node pliers:

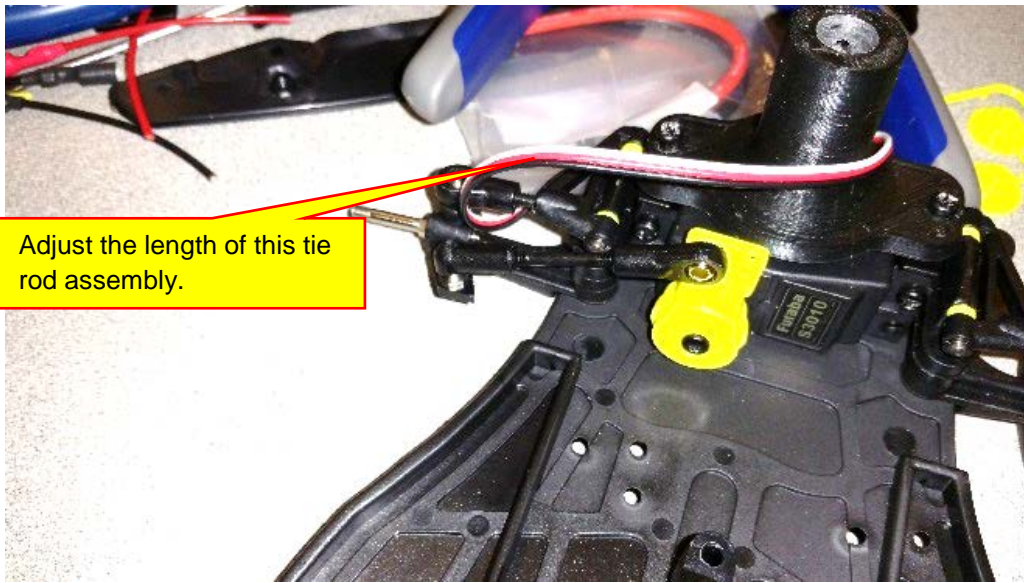


Do this for both tie rod assemblies.

Next, remove the two front wheels from the vehicle by removing the tire nut with the tool provided:

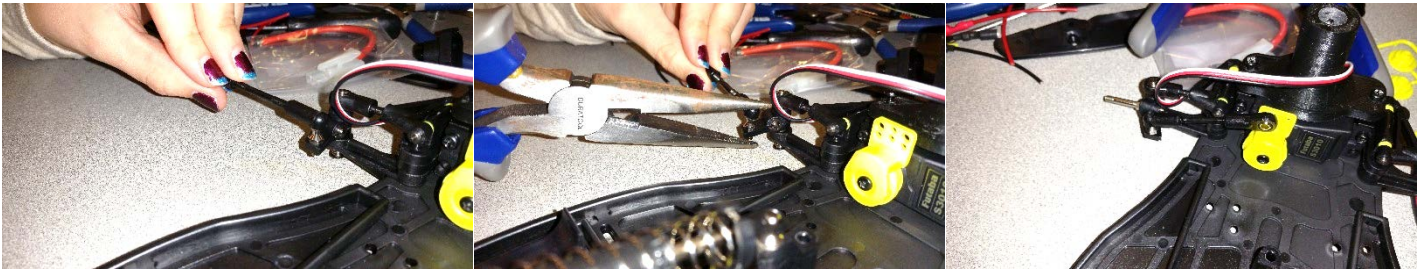


With the tires remove, adjust the length of the two tie rod assemblies so that the two front wheels will point straight when connected to the servo motor mounting plate:



Make sure that you adjust both tie rods and then you remember which tie rod is on which side. The two tie rods are not the same length.

Next, place the tie rod on the steering ball and snap it on to the ball with a pair of needle nose pliers. It will take a fair amount of force to do this:



Repeat the process for the other side:



Finally, reattach the front wheels and then screw the other end of the tie rods into the servo motor plate as shown:

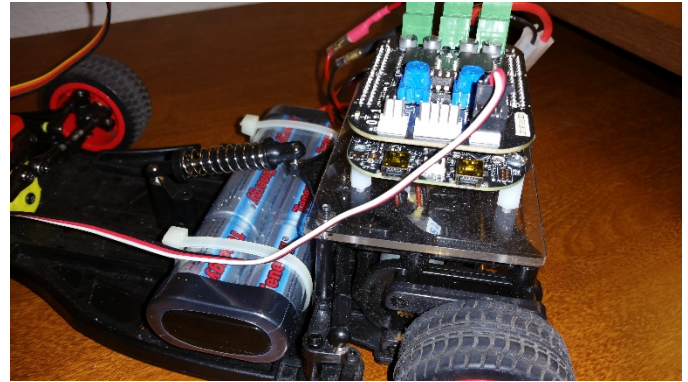
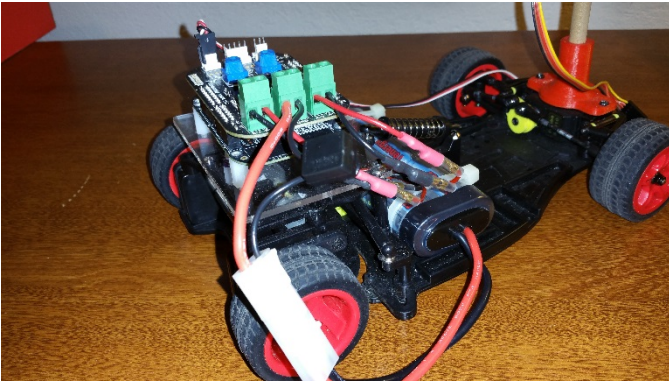


Lesson X: Mounting the KL25Z

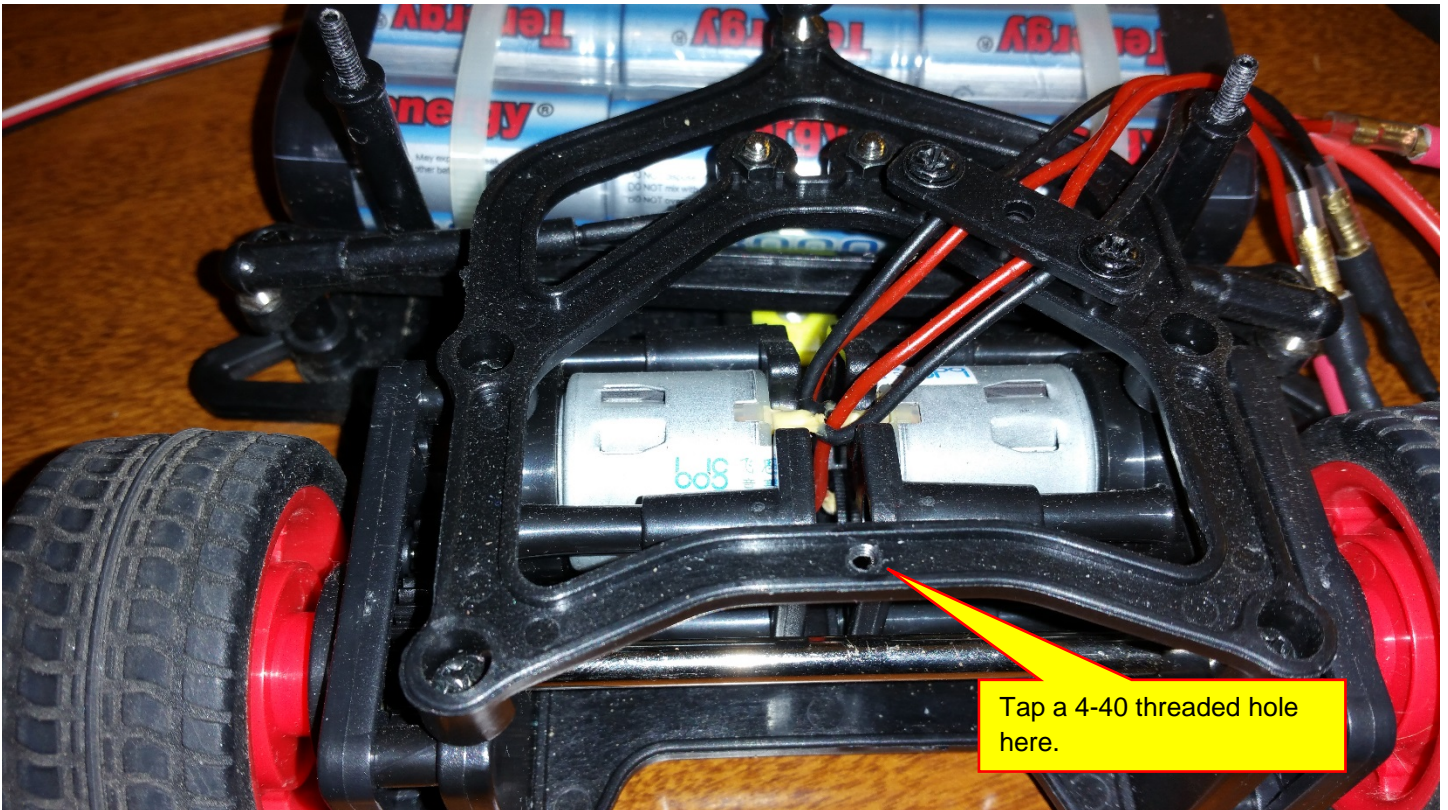
We will now mount the KL25Z and the Freescale Cup Shield on the vehicle and then test the various components.

A. Mounting the KL25Z and TFC Shield

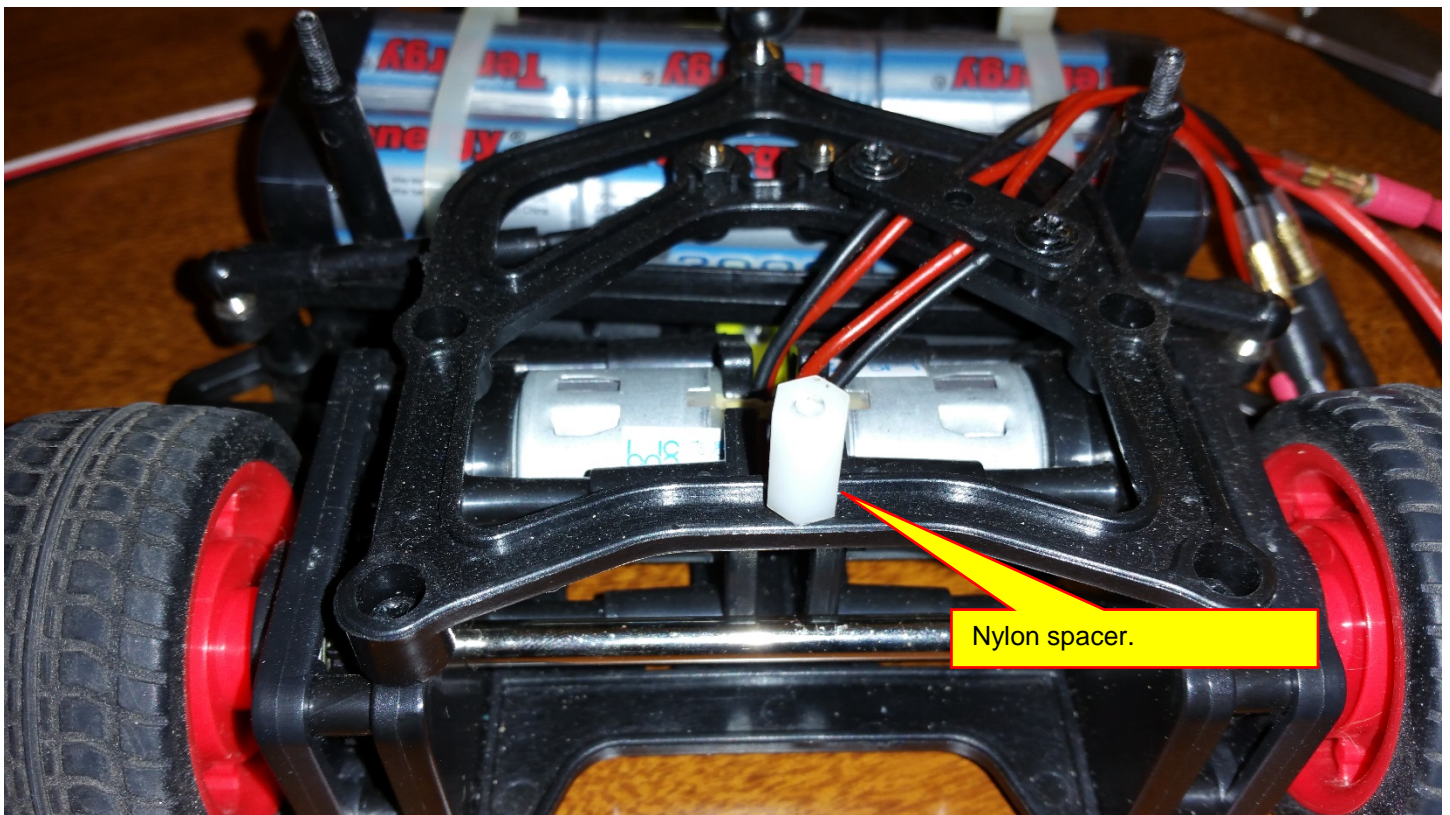
Our final design will look at shown:



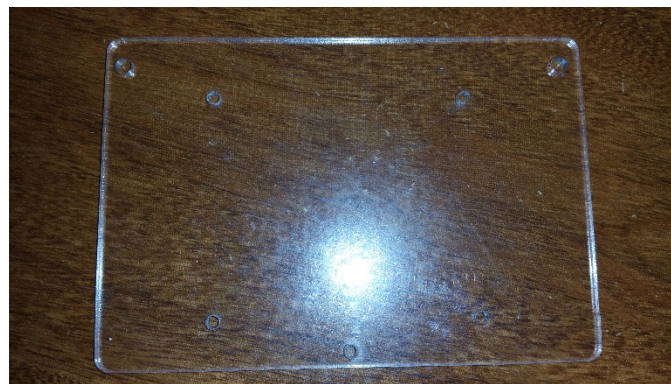
On your vehicle chassis, you will need to tap a 4-40 threaded hole as shown:



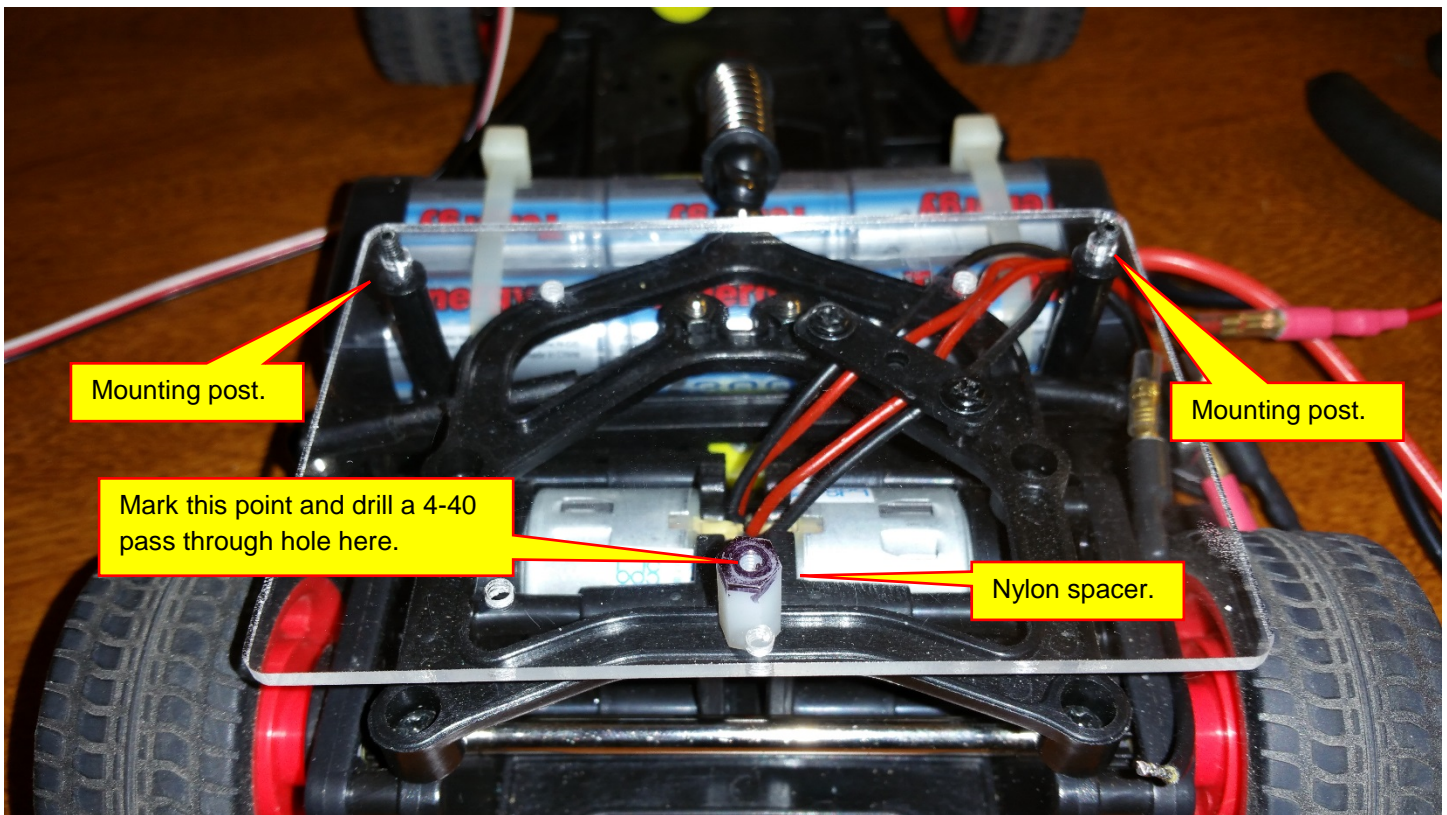
Screw in one of the $\frac{1}{2}$ - inch nylon spacers into the threaded hole:



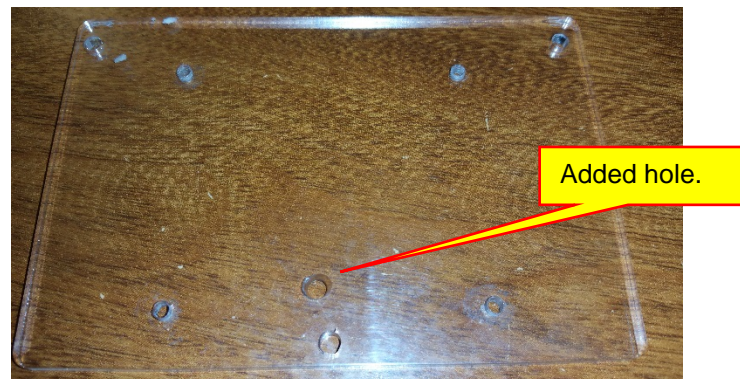
Locate the Plexiglas mounting plate:



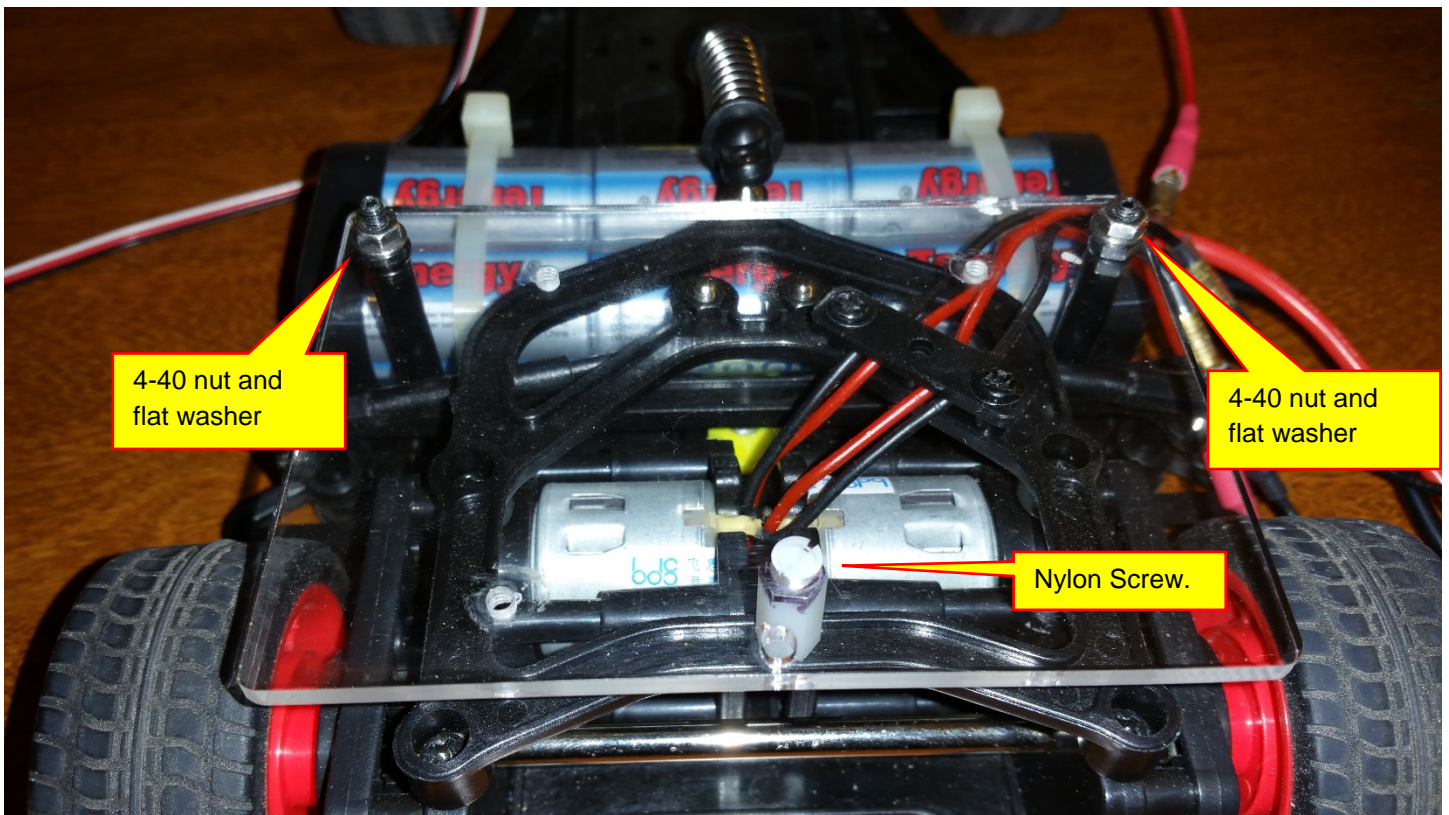
Mount the plate on the two mounting posts and the nylon spacer as shown:



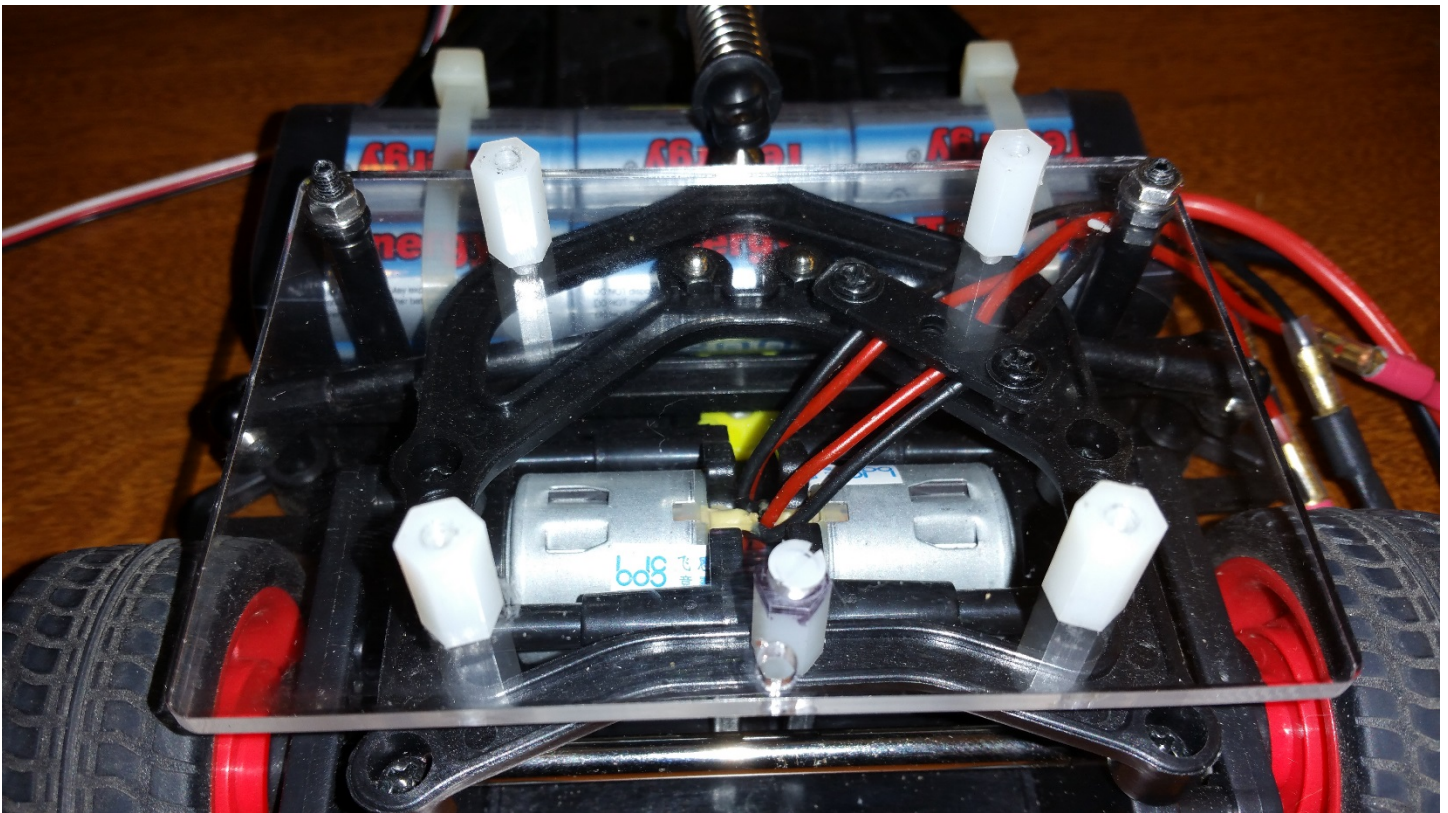
Locate the location of the 4-40 pass through hole as shown above. Drill a hole in the Plexiglas mounting plate so that you can attach the plate to the vehicle chassis. The new hole is shown below:



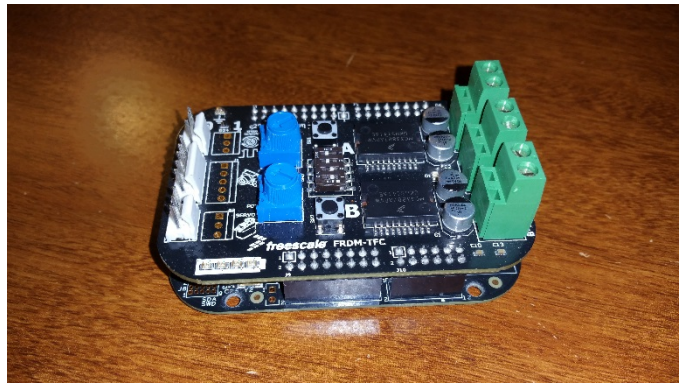
Mount the plate on your vehicle as shown:



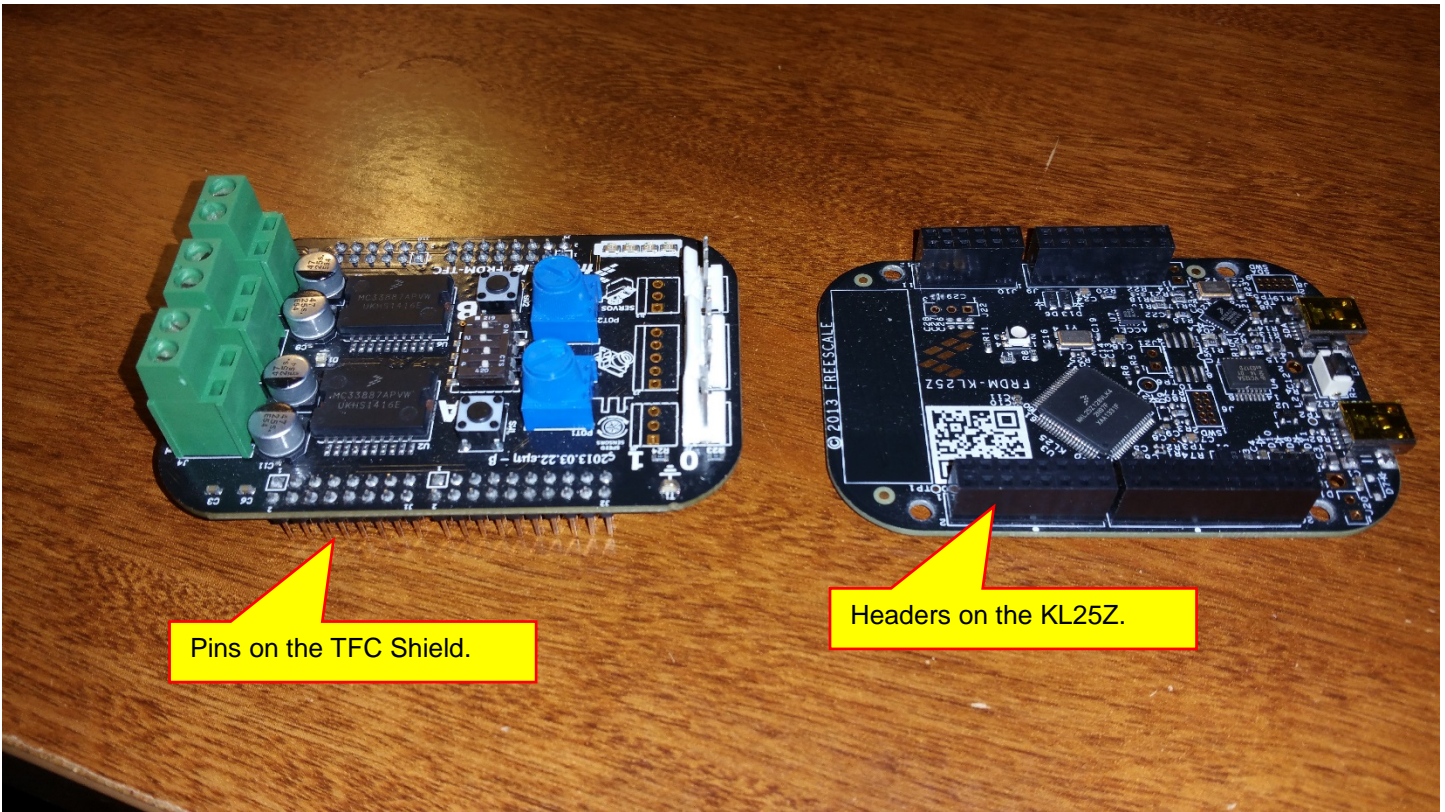
Thread in four ½-inch nylon spacers:



Next, locate the KL25Z board and the TFC Shield. Note that the TFC Shield is mounted on the KL25Z board and both are packaged inside an antistatic bag:



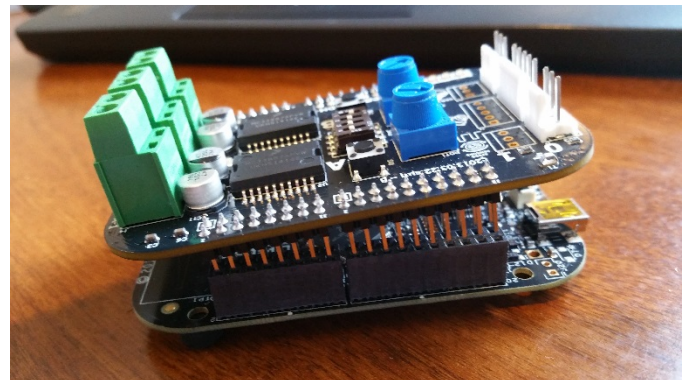
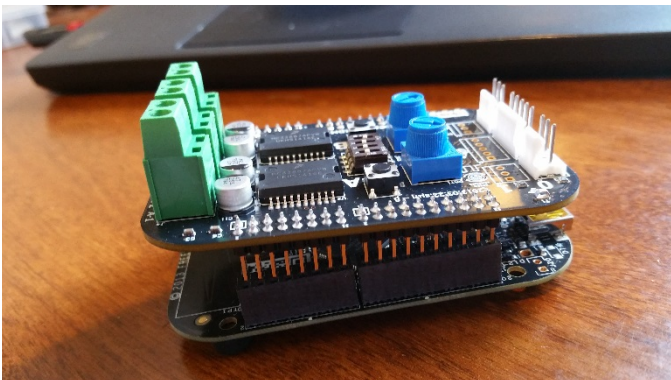
The TFC Shield is plugged into the KL25Z Board. Both Boards are shown separately below:



Pins on the TFC Shield.

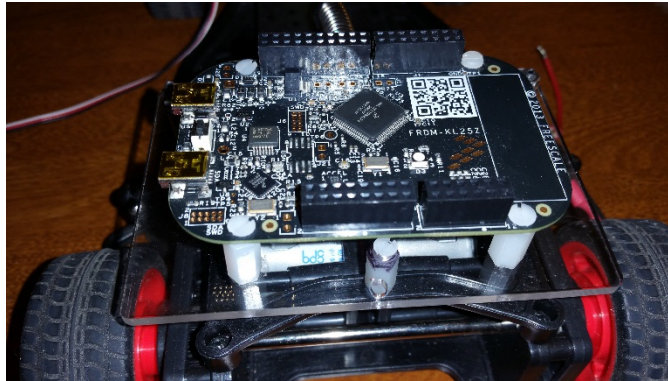
Headers on the KL25Z.

We see that the TFC Shield has pins that plug into the KL25Z headers. Gently pull apart the two boards. Do not twist the boards. Alternately pull up one side then the other until the boards pull apart:

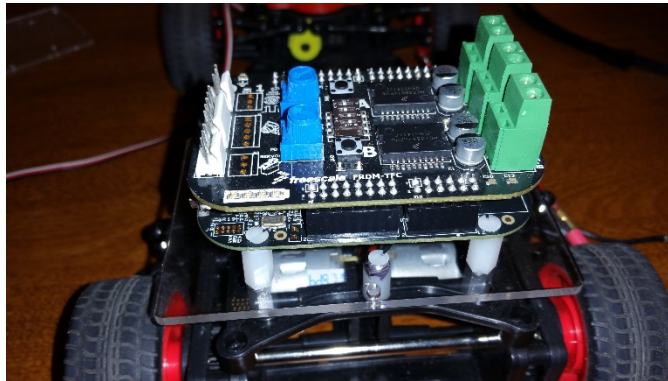


Pull up each side a small amount to avoid bending the pins.

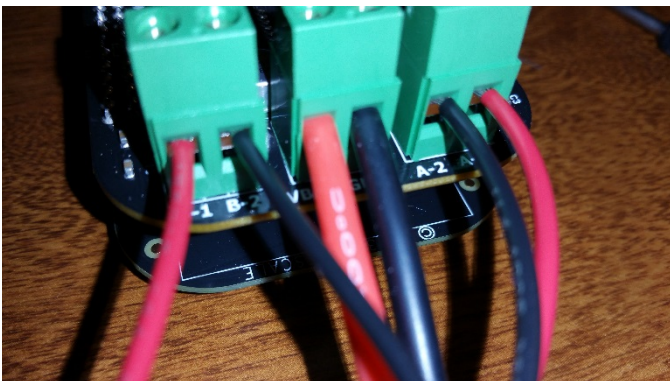
Mount the KL25Z on the four nylon standoffs using 4-40 nylon screws:



Next, mount the TFC Shield on the KL25Z board. Be sure to align the pins correctly or you will damage the pins:



When you tested the motors in Section IV.C you should have marked the wires and written down the connections. If not, the correct wiring is shown on page 61. If you did not mark the wires and you cannot find page 51, the correct wiring is shown below:

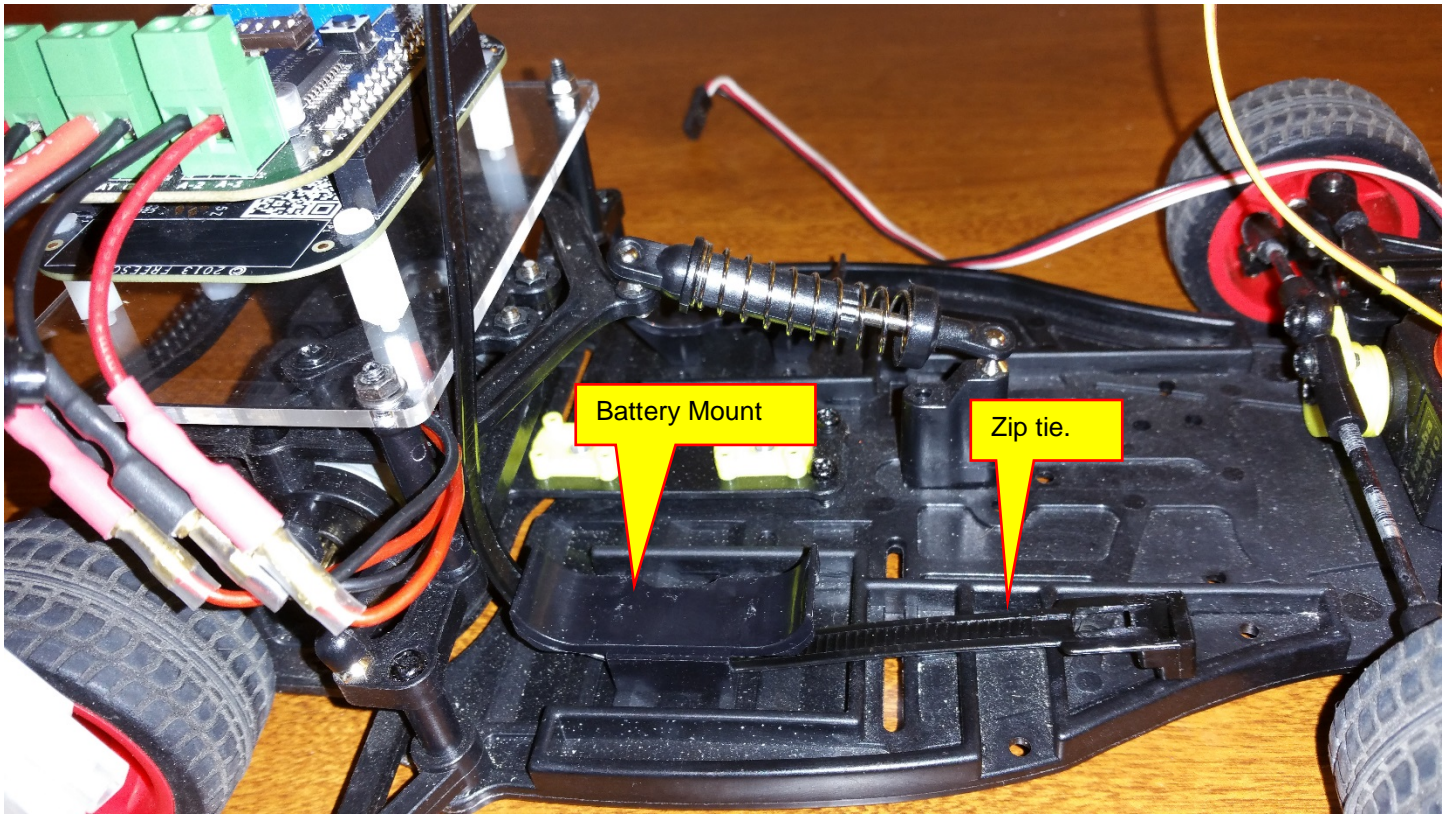


Remember that motor A is the right motor and motor B is the left motor. The two center wires are for the battery.

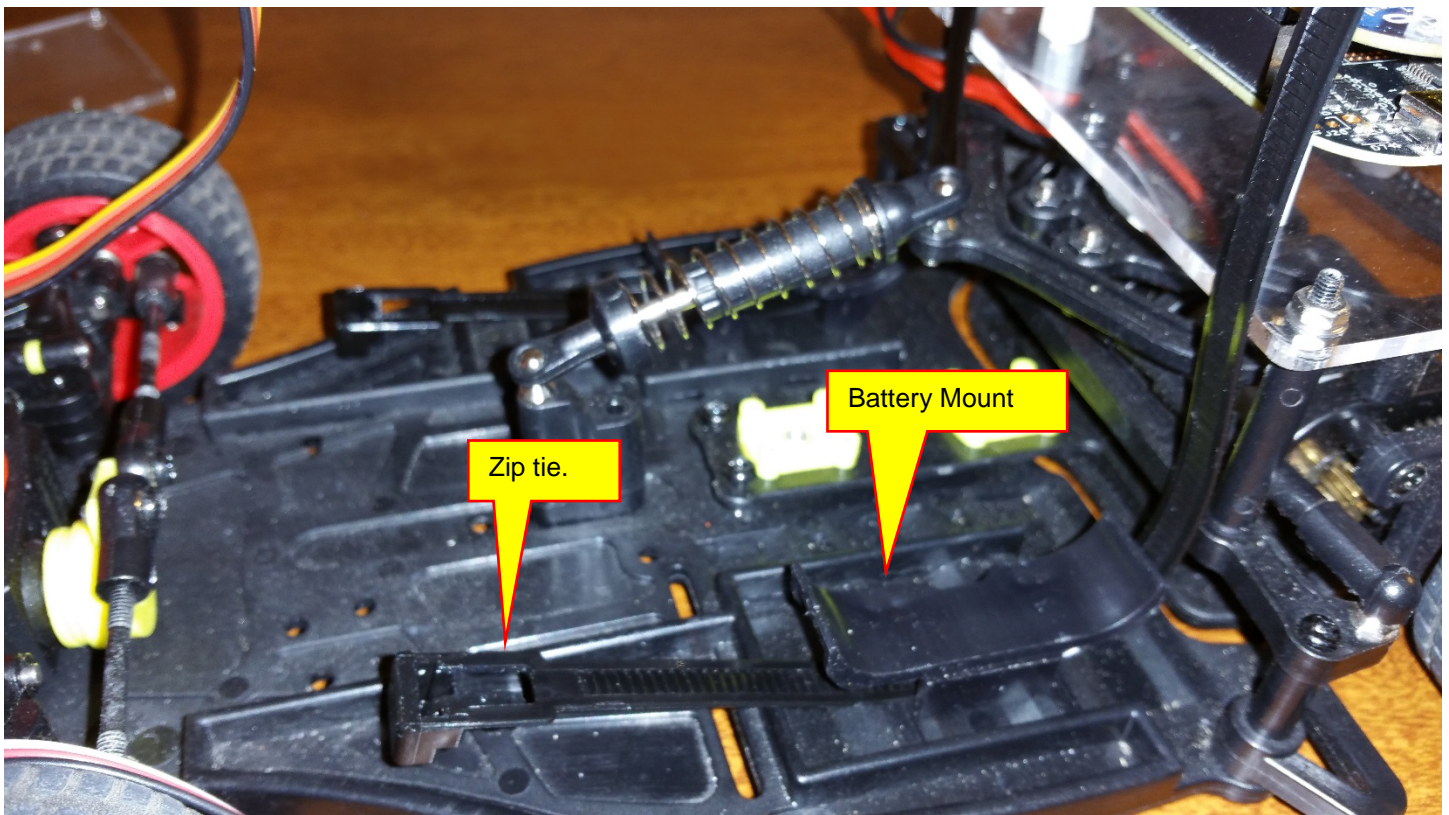
Next, we need to mount the battery. Locate the two zip ties:



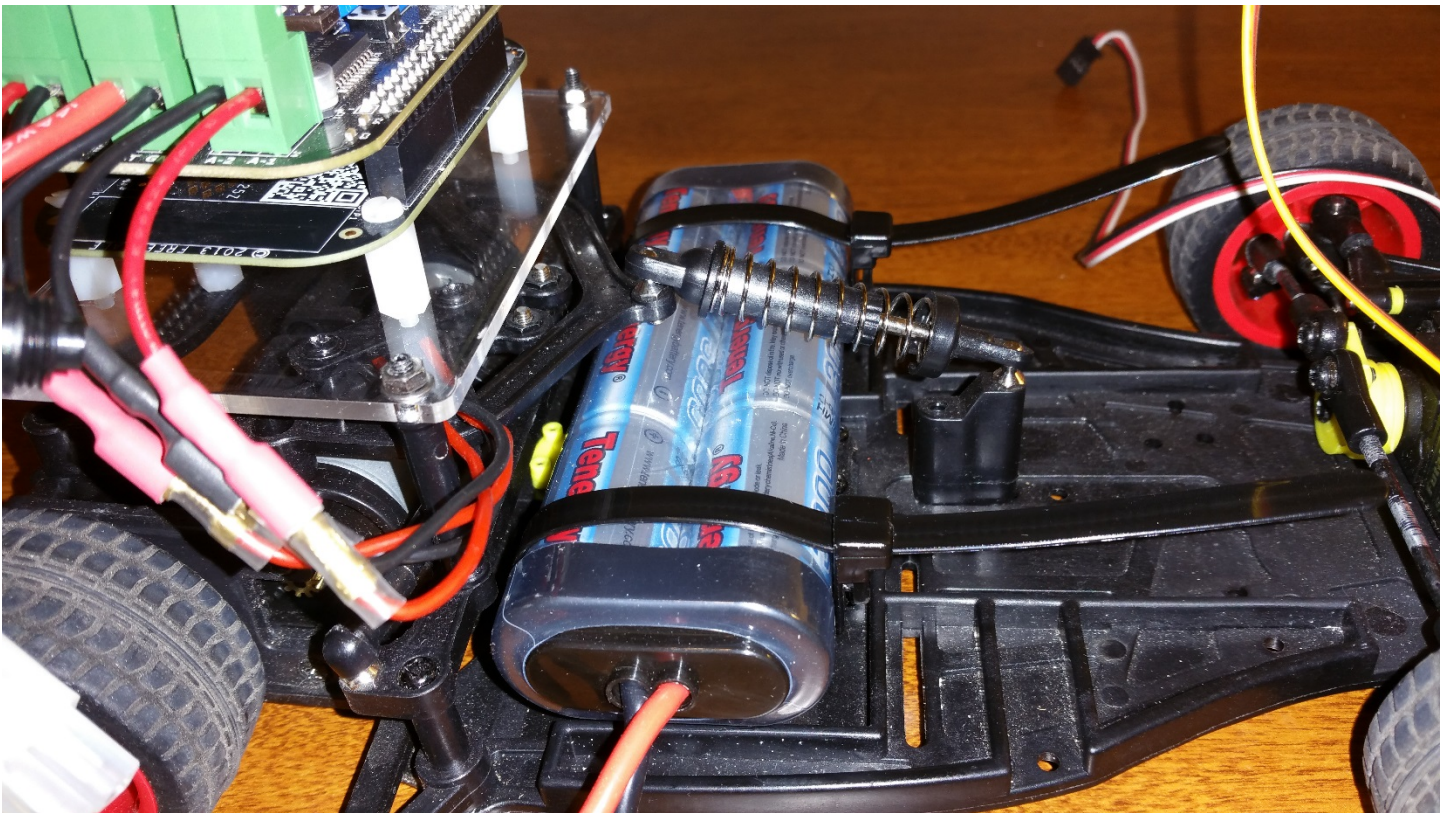
Thread the two zip ties through the slot below the two battery mounts:



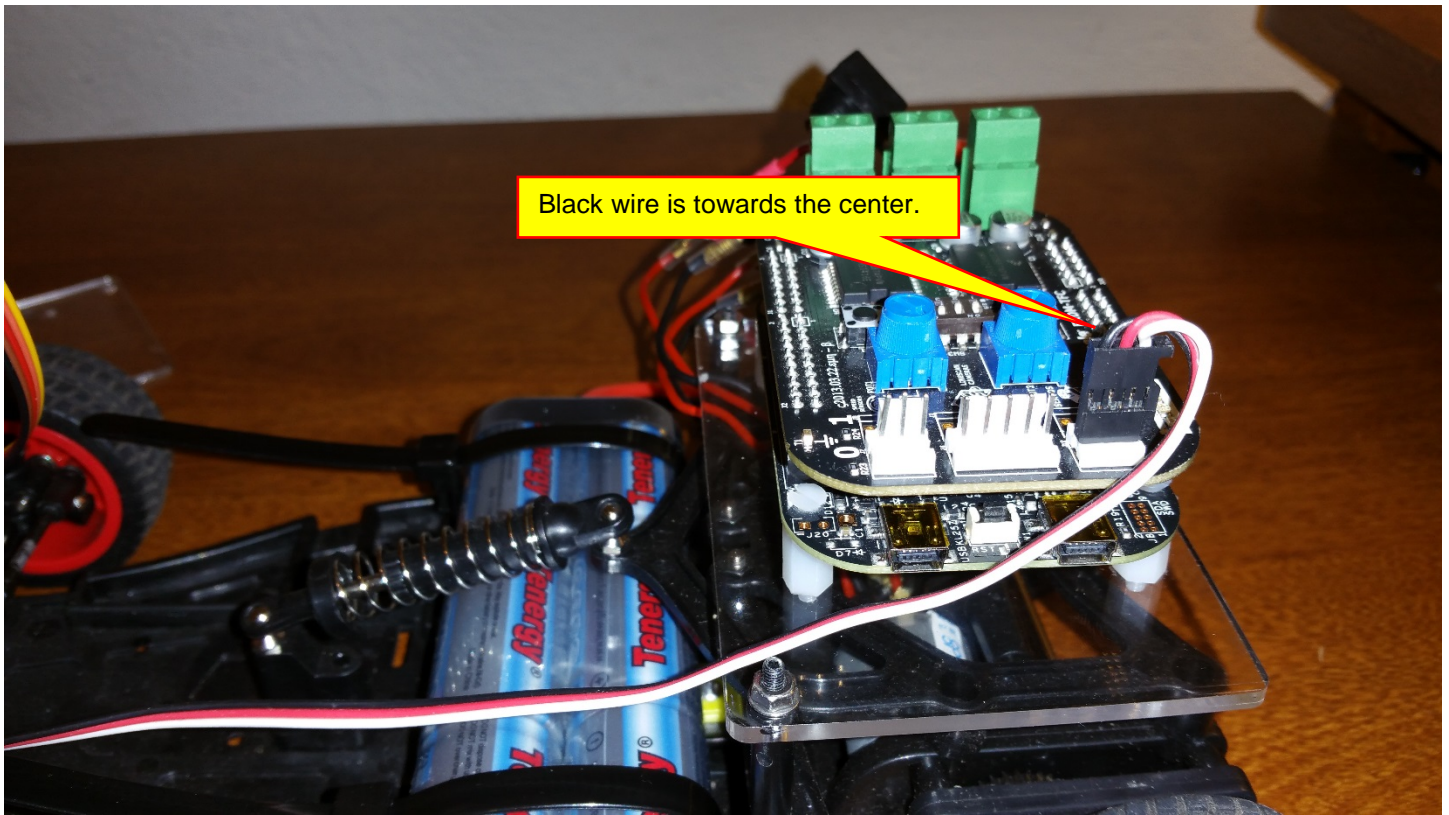
And, the other side:



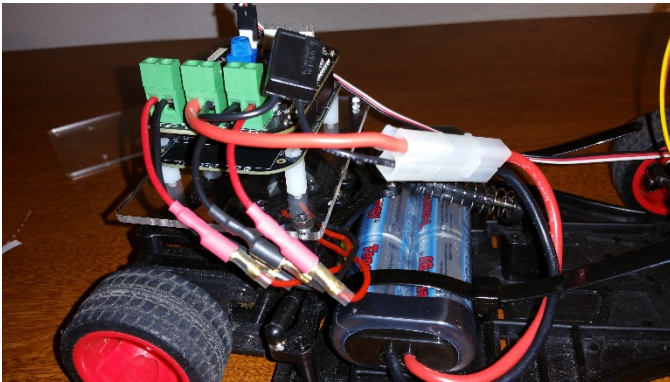
Mount the battery and secure it with the zip ties:



Next, connect the cable for the servo motor to TFC Shield:



Finally, connect the battery:

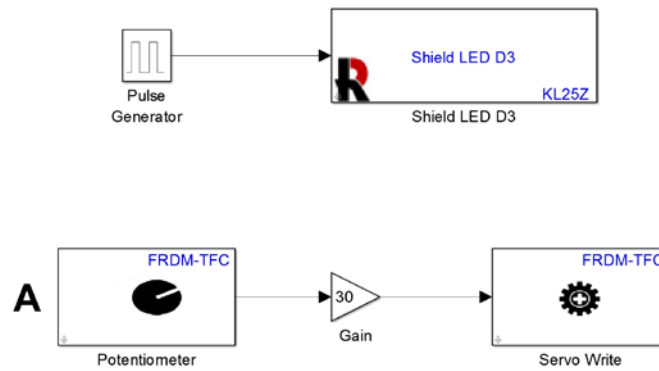


B. Testing

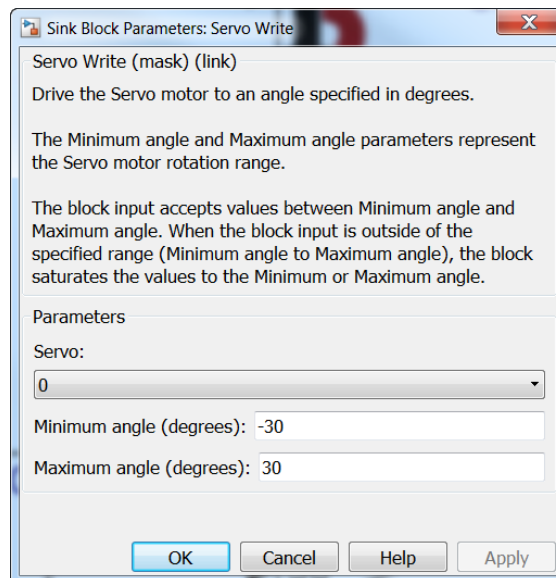
We will now test the two drive motors and servo motor. All of the models shown here were developed previously. **Remember, when you program the KL25Z, the battery switch should be turned off. There is a bug in the TFC shield that causes the motors to spin at full speed during programming if the shield is powered by the battery.**

1. Servo Motor and Steering

We will now test and calibrate the servo motor and steering. Build the model shown below:



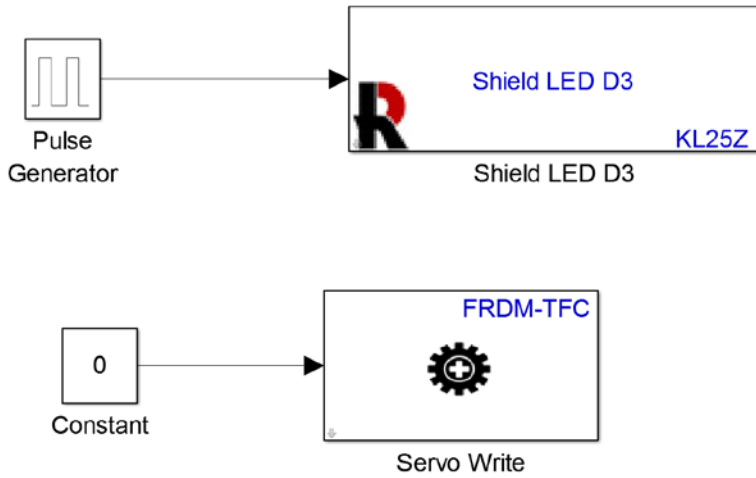
Note that the steering gain was set to 30. Since the potentiometer outputs a signal between -1 and +1, the signal to the **Servo Write** block will be -30 to 30. Double-click on the **Servo Write** block and change the limits to -30 and 30:



Make sure that the battery power to the vehicle is turned off. Build and download the model. Turn on the battery power to the vehicle. Verify:

1. That the steering changes as you change the position of Potentiometer A.
2. Make sure that as you hit the right and left limits of the steering motion, the servo motor does not "buzz." If it buzzes, then +30 or -30 is too large and you will need to change the limits in the **Servo Write** block.

Next, open the model that zeroed the servo motor:

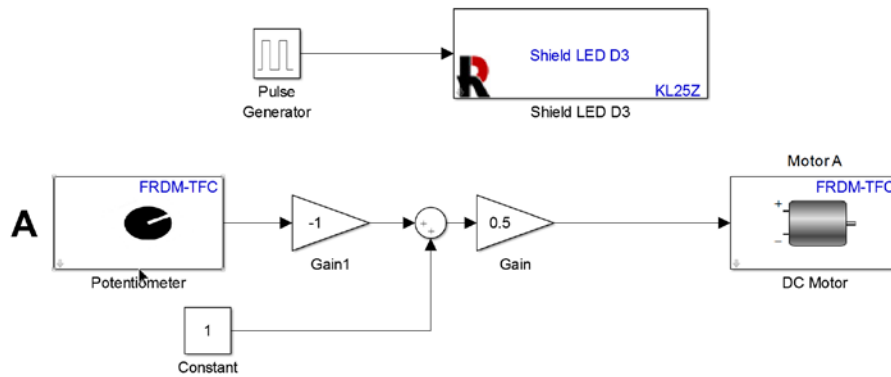


Build and download this model. Verify:

- That the wheels point straight. This verifies that the vehicle drives (mostly) straight when the steering angle is zero.

2. Right Drive Motor

Build and download the following. Make sure that when you build the model, battery power to the vehicle is off:

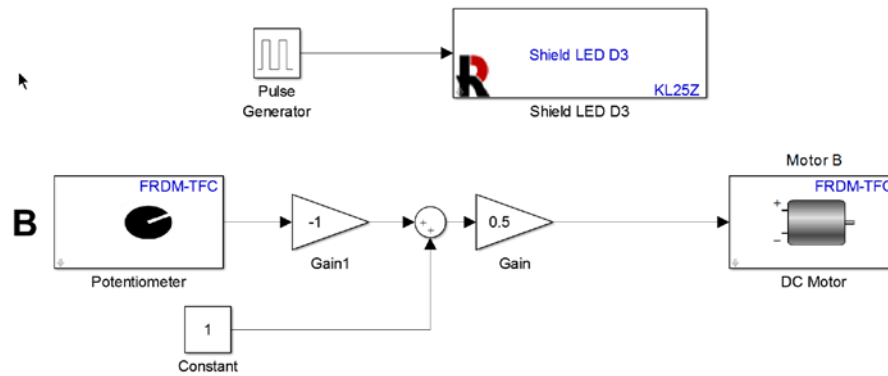


Verify:

1. The right motor spins from 0 to maximum speed as potentiometer A is turned clockwise.
2. That motor A only spins in the direction that will move the vehicle forward.

3. Left Drive Motor

Build and download the following. Make sure that when you build the model, battery power to the vehicle is off:



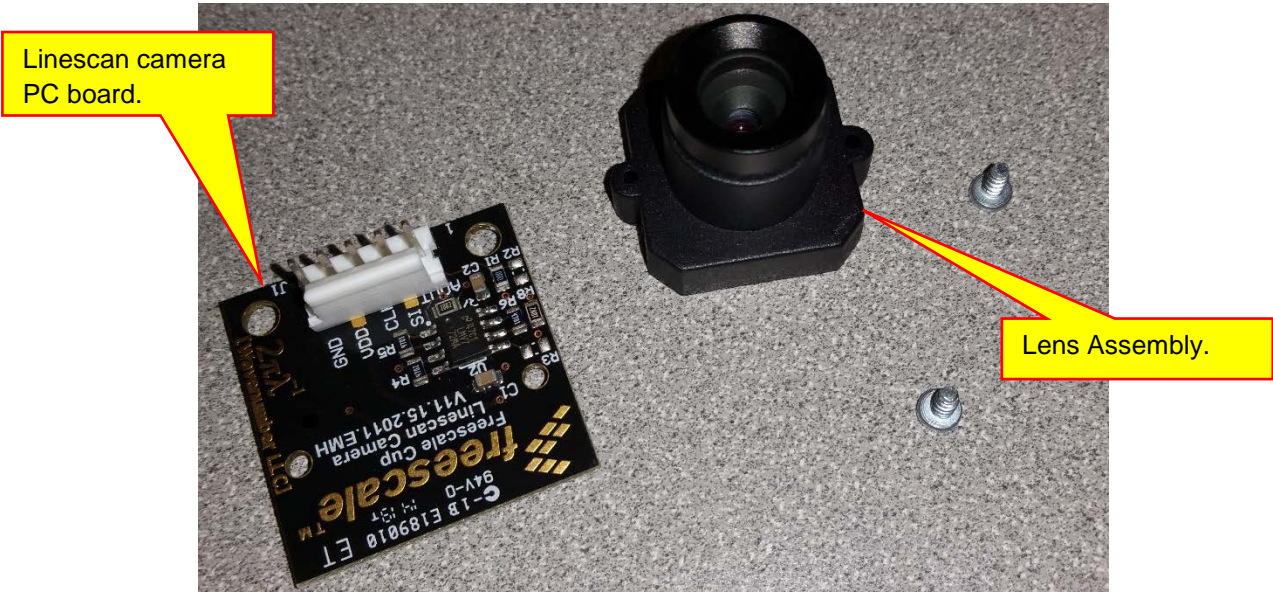
Verify:

1. The left motor spins from 0 to maximum speed as potentiometer B is turned clockwise.
2. That motor B only spins in the direction that will move the vehicle forward.

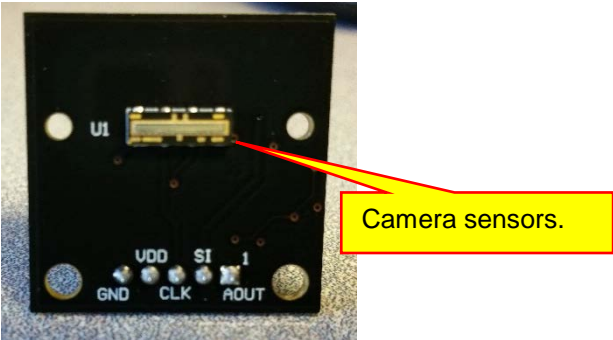
Lesson XI: Assembling and Mounting the Camera

A. Assembling the Camera

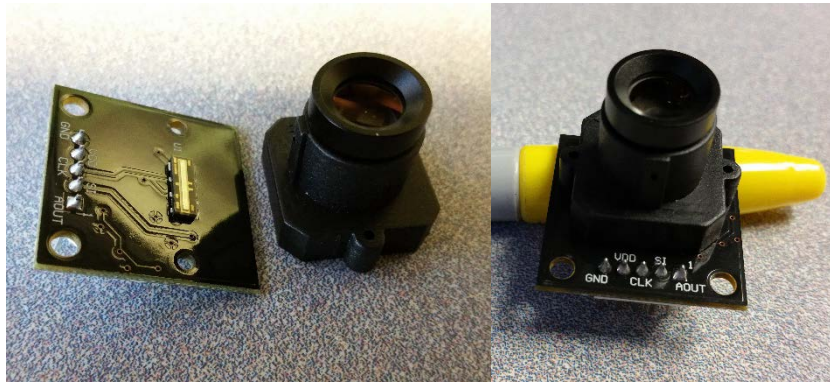
The Freescale Cup Linescan Camera does not come preassembled, so we must put it together. Locate the camera lens assembly, pc board, and screws:



The lens assembly mounts on the back side of the Line Scan Camera PC board. The back side is shown below:



The lens assembly goes on this side of the PC board:



Use the two screws to attach the lens mount to the Linescan Camera PC board:



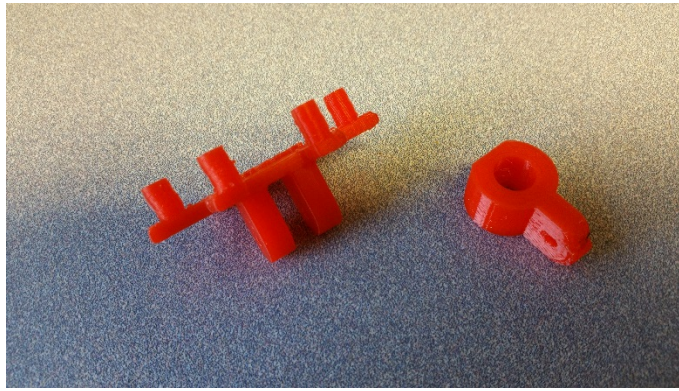
The front and back of the completed assembly are shown below:



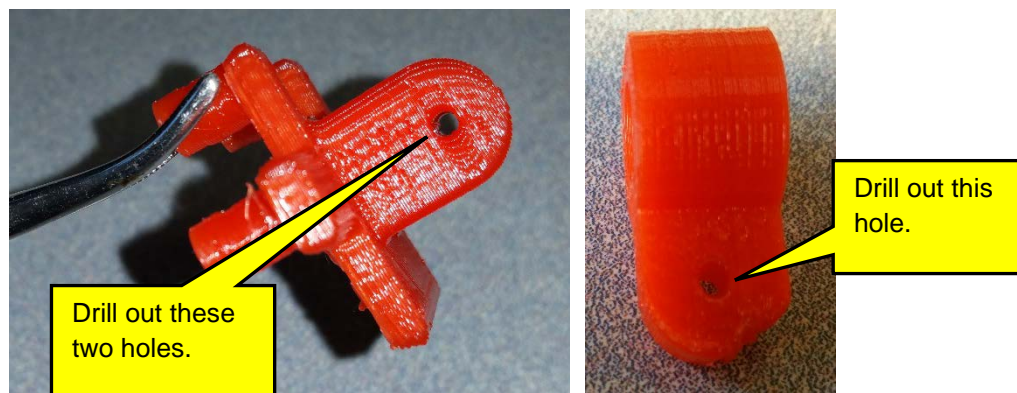
Note which pin on the connector is the GND pin.

B. Assembling the Camera Mount

Locate the parts for the camera mount:

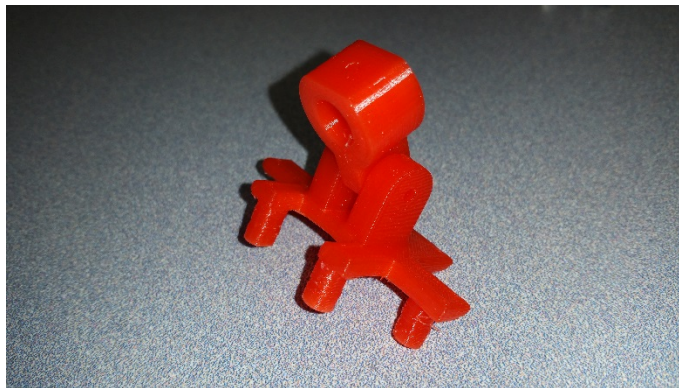


The holes in the mount are not large enough for our hardware. Drill out the holes shown below so that they pass a 4-40 screw:

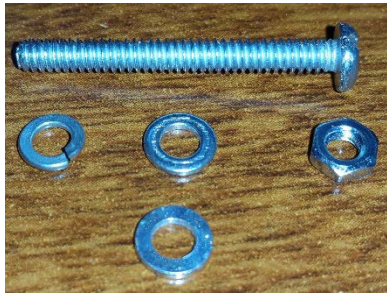


A #32 drill should be sufficient since we don't want it to be completely loose.

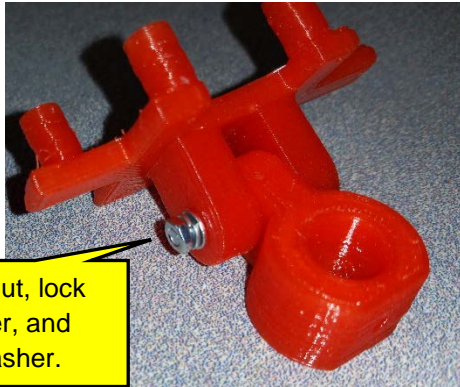
Assemble the two pieces as shown. Make sure that the holes in the two pieces line up:



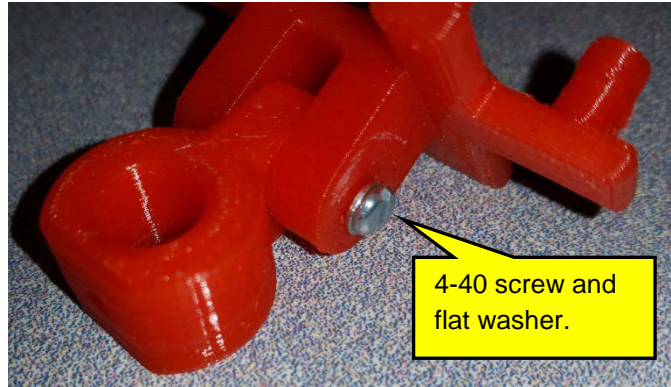
Use 4-40 hardware to secure the two pieces. Use a 1-inch 4-40 screw, two flat washers, a lock washer, and a nut. The hardware is shown below:



Assemble the parts as shown below:

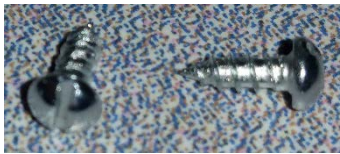


4-40 nut, lock washer, and flat washer.



4-40 screw and flat washer.

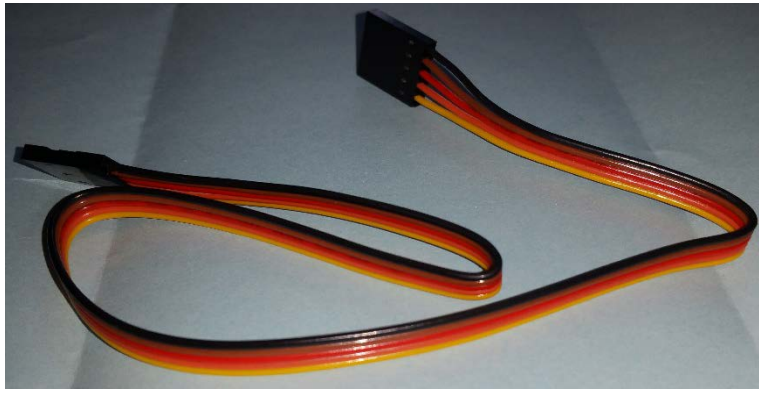
Locate the two screws for attaching the camera:



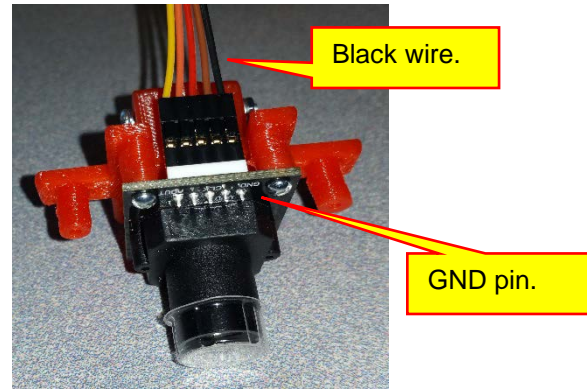
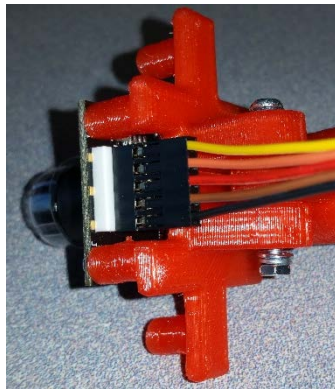
The camera should be attached to the camera mount as shown:



Note which pin of the Linescan PC board is marked as ground (GND). Locate the camera cable:



Attach the camera cable with the black wire connected to the GND pin on the Linescan PC board:



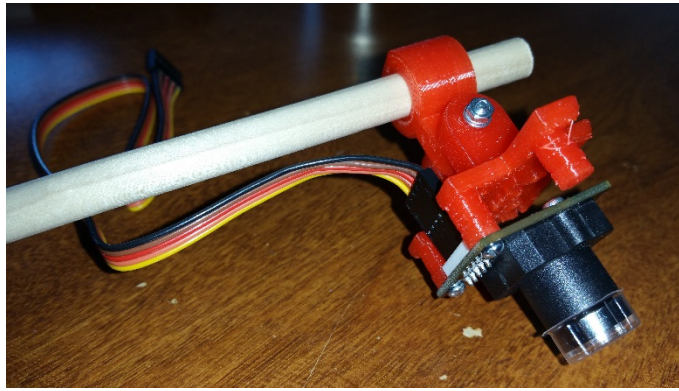
The camera assembly is now complete.

C. Mounting the Camera

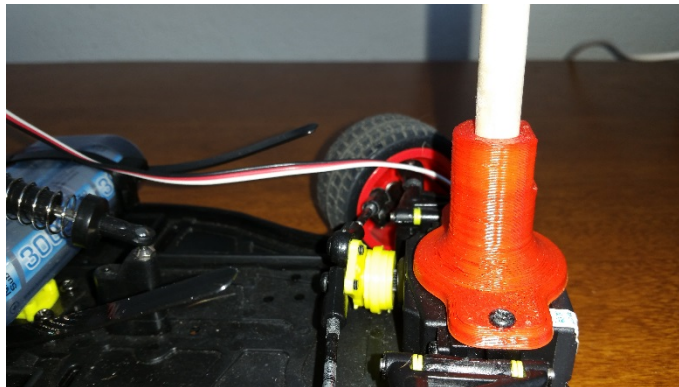
We will now mount the camera on the vehicle. The orientation of the camera is important. If you mount it upside down, the steering control system will turn the wrong direction. Locate the 3/8-inch dowel:



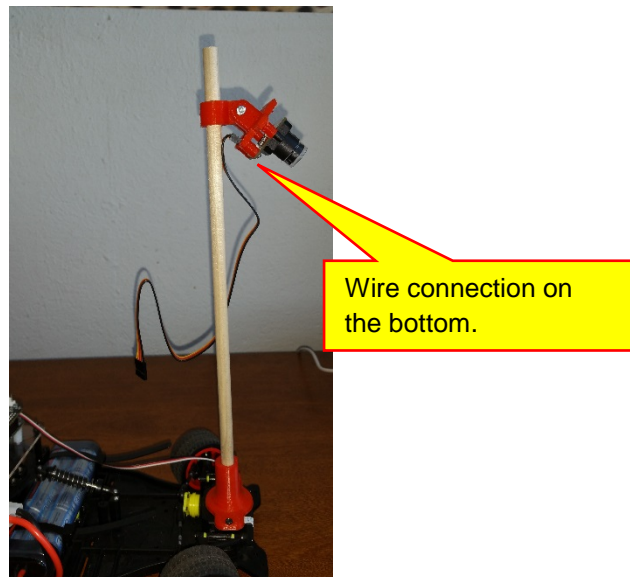
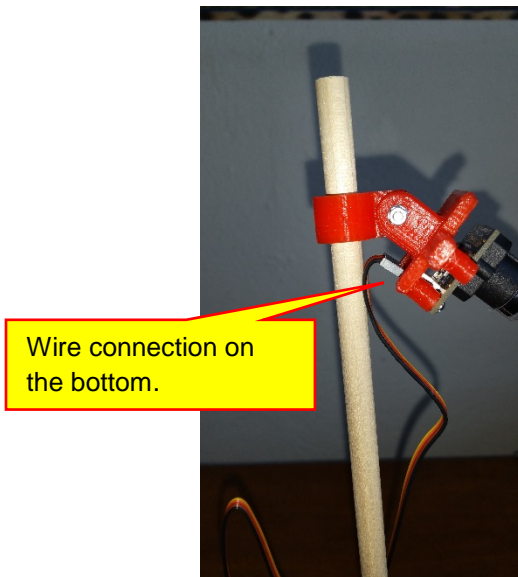
The length of the dowel is your choice. One of the design parameters of the vehicle is the height of the camera above the track. The height is adjustable but limited by the size of your dowel. **Note that you can always change the dowel if needed.** Cut the dowel to an appropriate size. Typically, a foot long works well, but you can choose a different size. Slide the camera on to the dowel. This may take a bit of force. Rotate the camera mount back and forth to slide it on. Be careful not to put any force on the camera or wires as this will damage the camera:



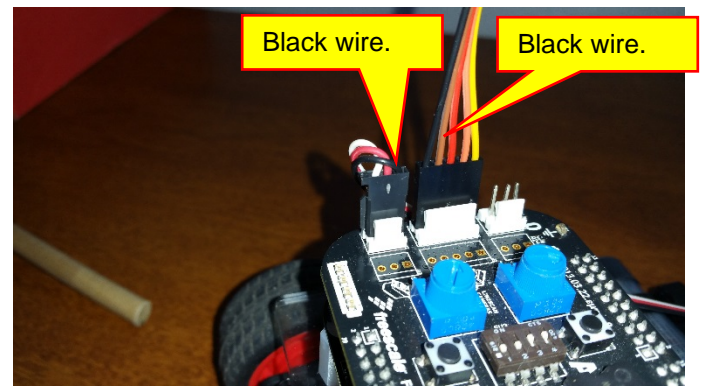
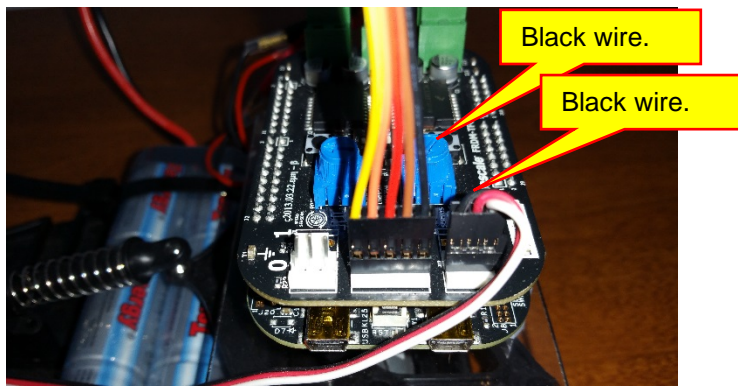
Next, mount the dowel on the vehicle. The dowel goes in the camera mount at the front of the vehicle:



Note that the camera wires should be on the bottom. If this is not the case, the vehicle will steer in the wrong direction:



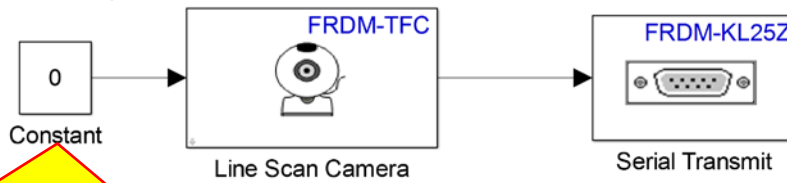
Finally, connect the other end of the camera cable to the camera receptacle on the TFC Shield. Note that the ground wire (black) of the camera cable is next to the ground (black) wire of the servo motor:



The assembly of the vehicle is complete!

D. Camera Testing

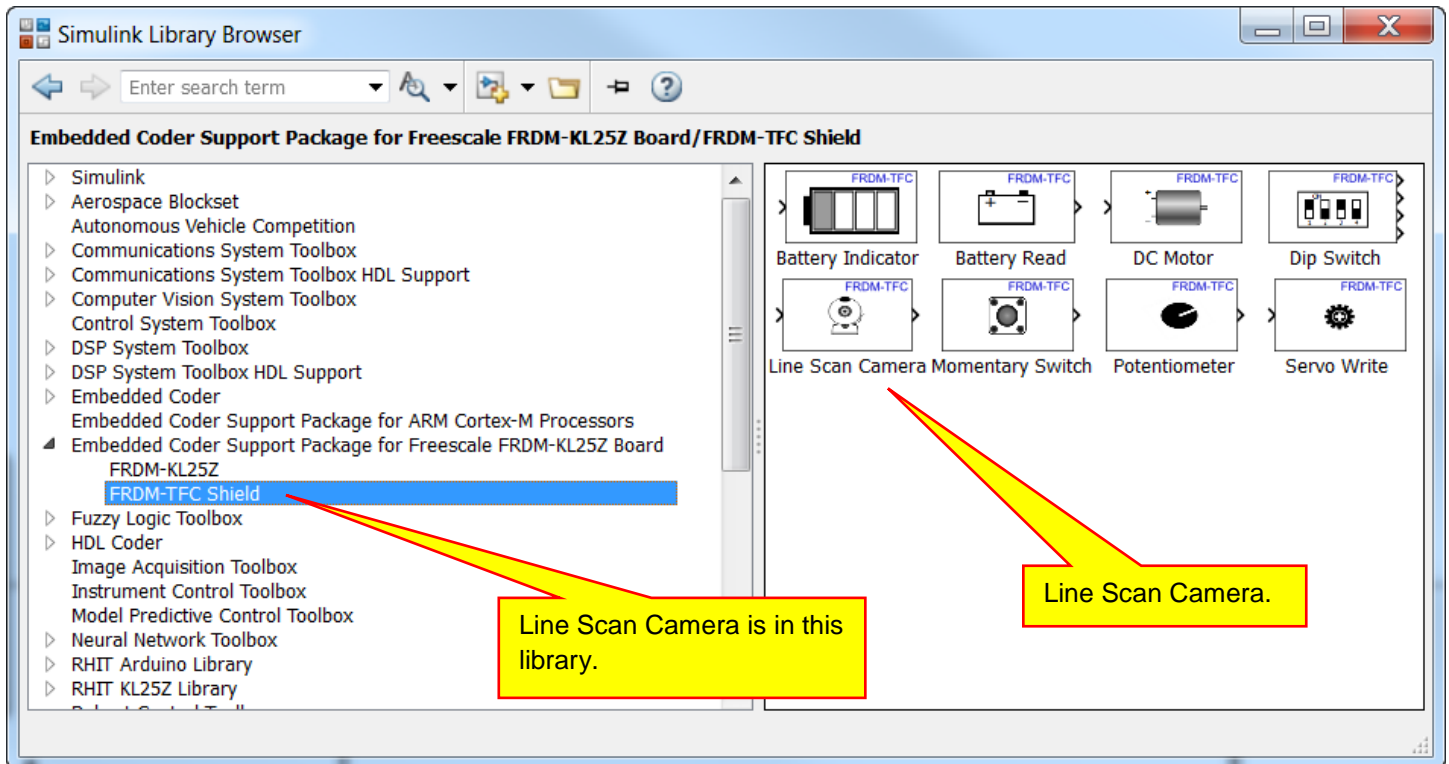
We are now ready to test the camera. Create the following model:



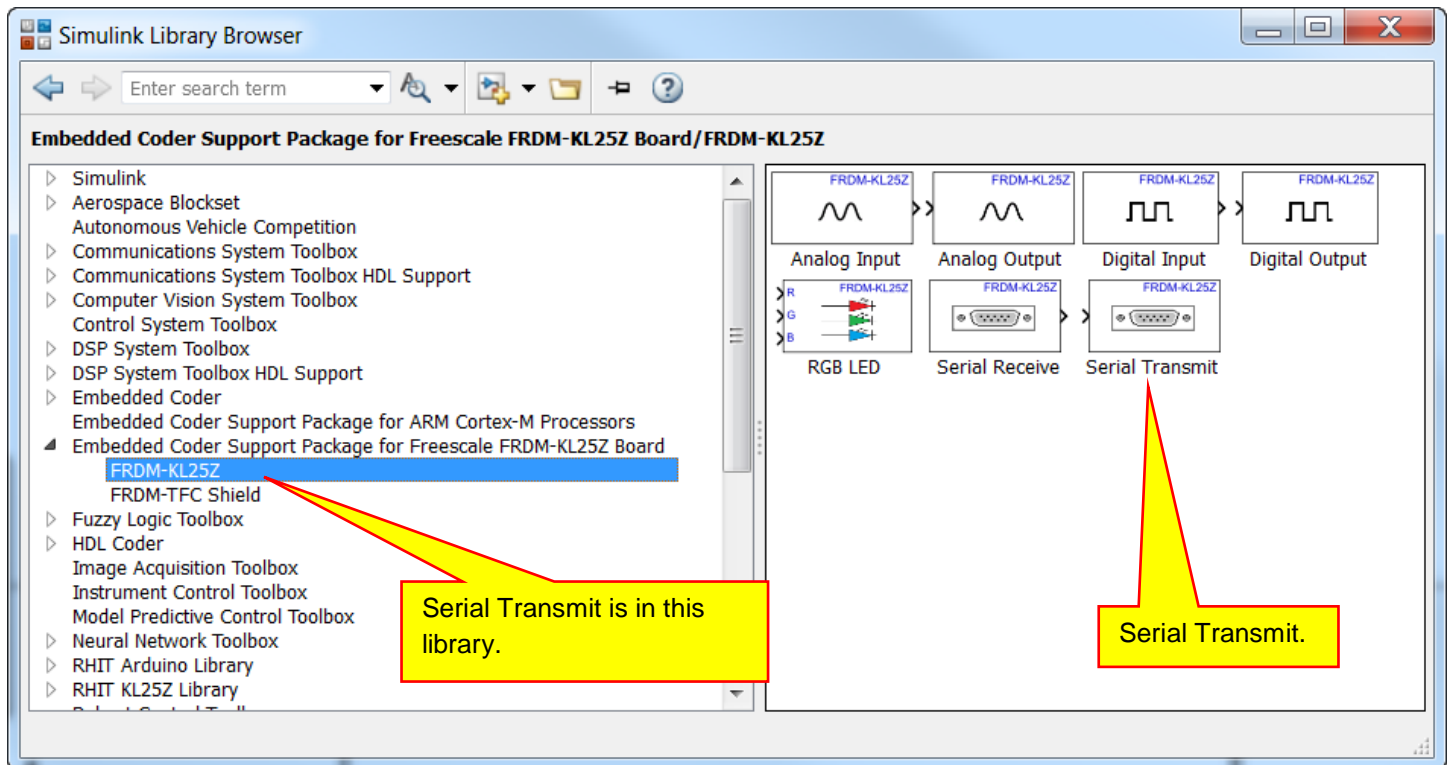
Set this number between 1 and 9. Do this for all future models.

The new parts are located in the following libraries:

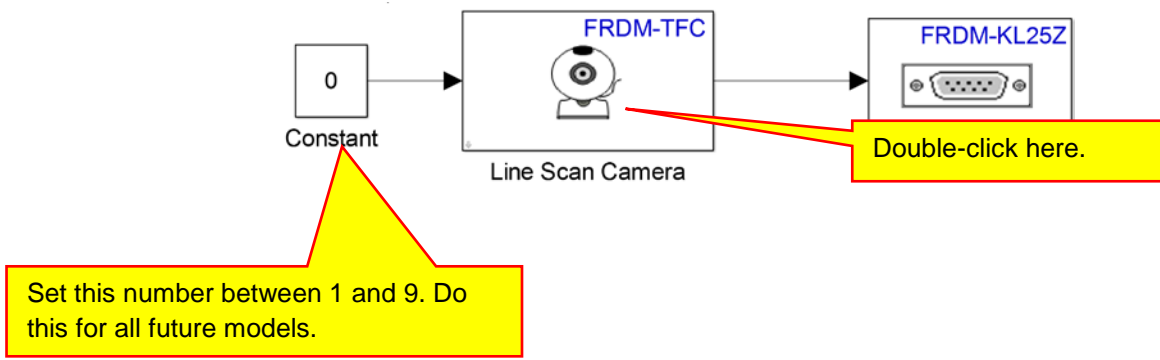
- Line Scan Camera: **Embedded Coder Support Package for Freescale FRDM-KL25Z Board / FRDM-TFC Shield:**



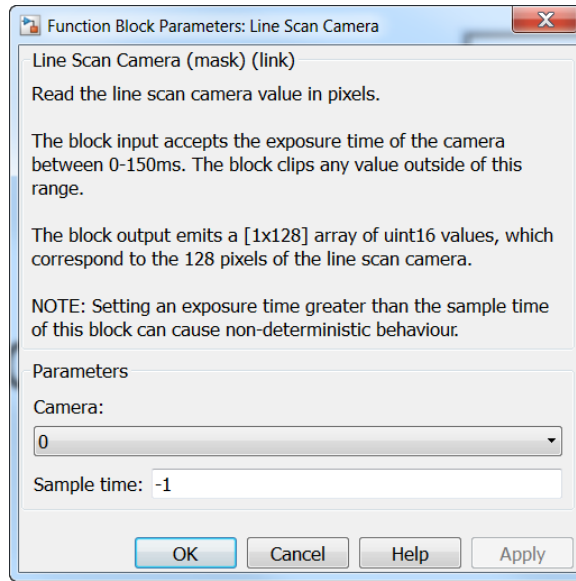
- Serial Transmit: **Embedded Coder Support Package for Freescale FRDM-KL25Z Board / FRDM-KL25Z:**



We need to set up the **Line Scan Camera** block and the **Serial Transmit** block. Double-click on the **Line Scan Camera** block:

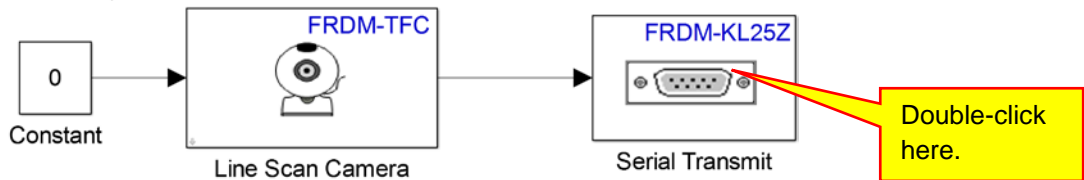


Change the parameters as shown below:

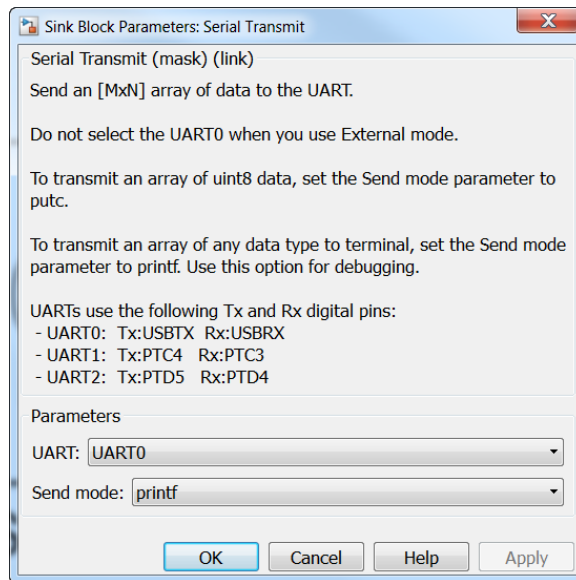


Note that we are using **Camera 0** and that there are two cameras available. If you decide that you want to use a second camera in the future, one is available. (The camera takes a lot of processing time, so if you decide to use a second camera, you will need to design a very efficient algorithm.) The **Sample time** is set to -1. This means that the camera will take a “picture” every time the model runs. Later in this part, we will set the model to run with a fixed time step of 0.25 seconds meaning that we will take a picture every 0.25 seconds, or four times per second. In the vehicle model that controls the car, we will take a picture every 0.02 seconds. Click the **OK** button to accept the changes.

Next, double-click on the **Serial Transmit** block:

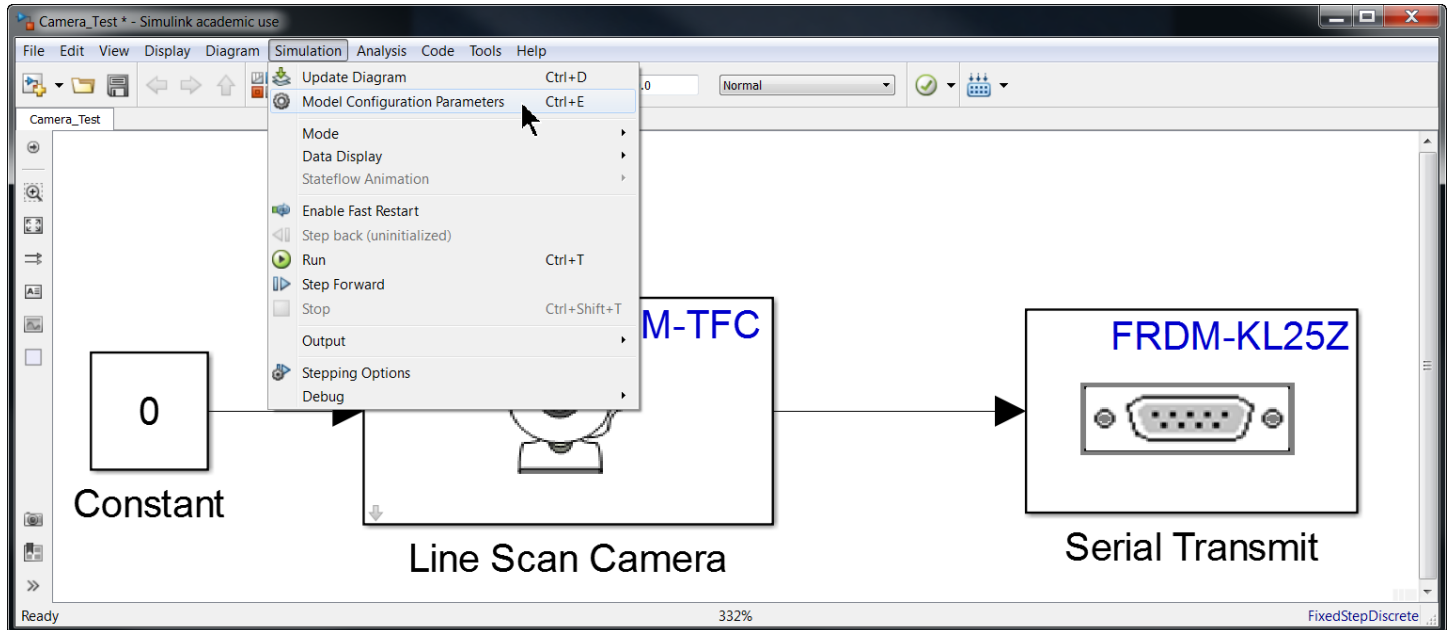


Set the parameters as shown:

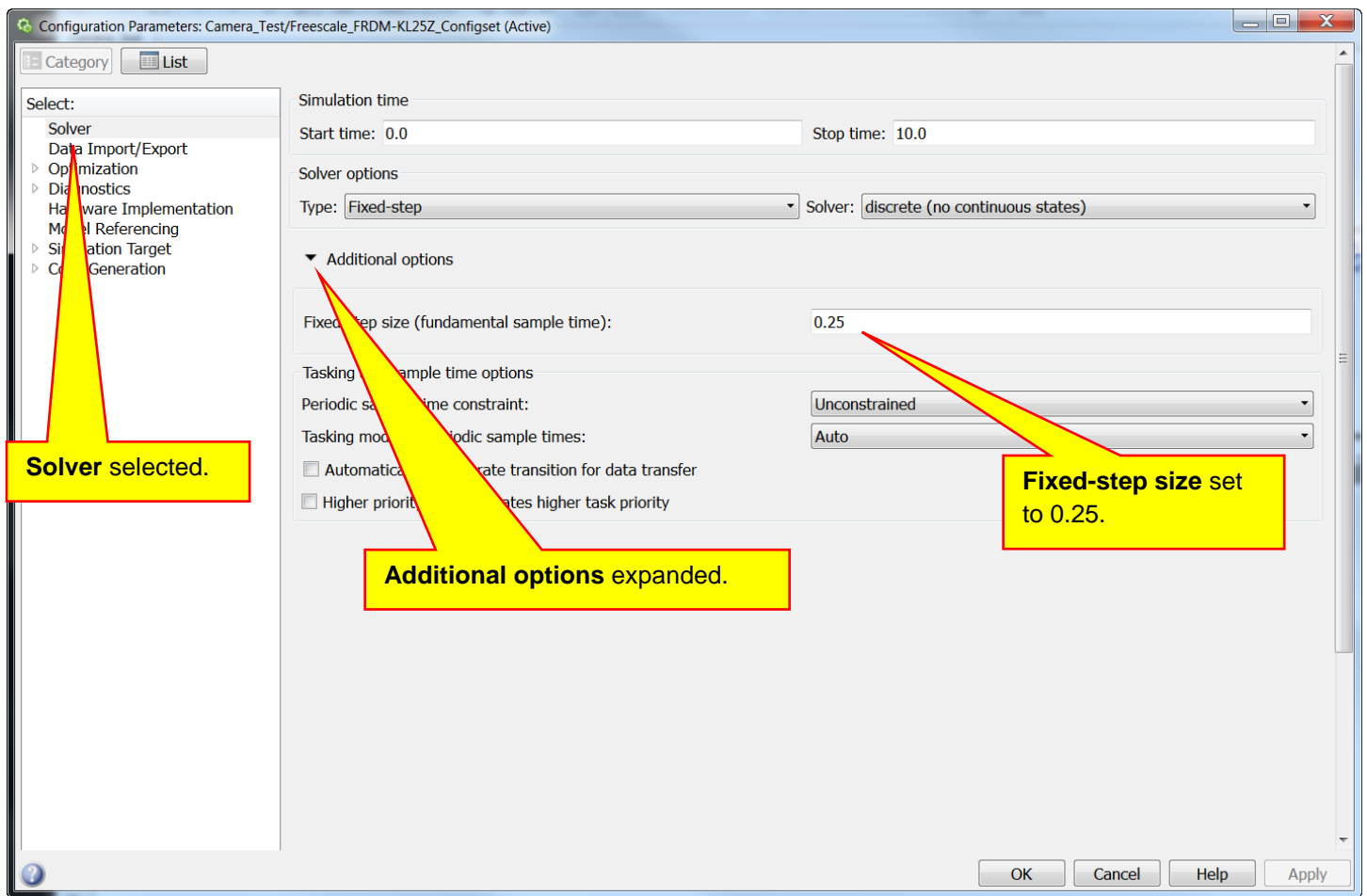


The KL25Z has three serial ports available. However, the way the TFC-Shield is wired, we can only use UART0 as the pins for UART1 and UART2 are used for different functions in the vehicle. Note that UART0 is connected to the SDA port. This allows us to view the camera data by connecting the SDA port to a USB port on our computers. (This is the same port that we program the KL25Z with.) Click the **OK** button to accept the changes.

Next, select **Simulation** and then **Model Configuration Parameters** from the Simulink menus as shown below:

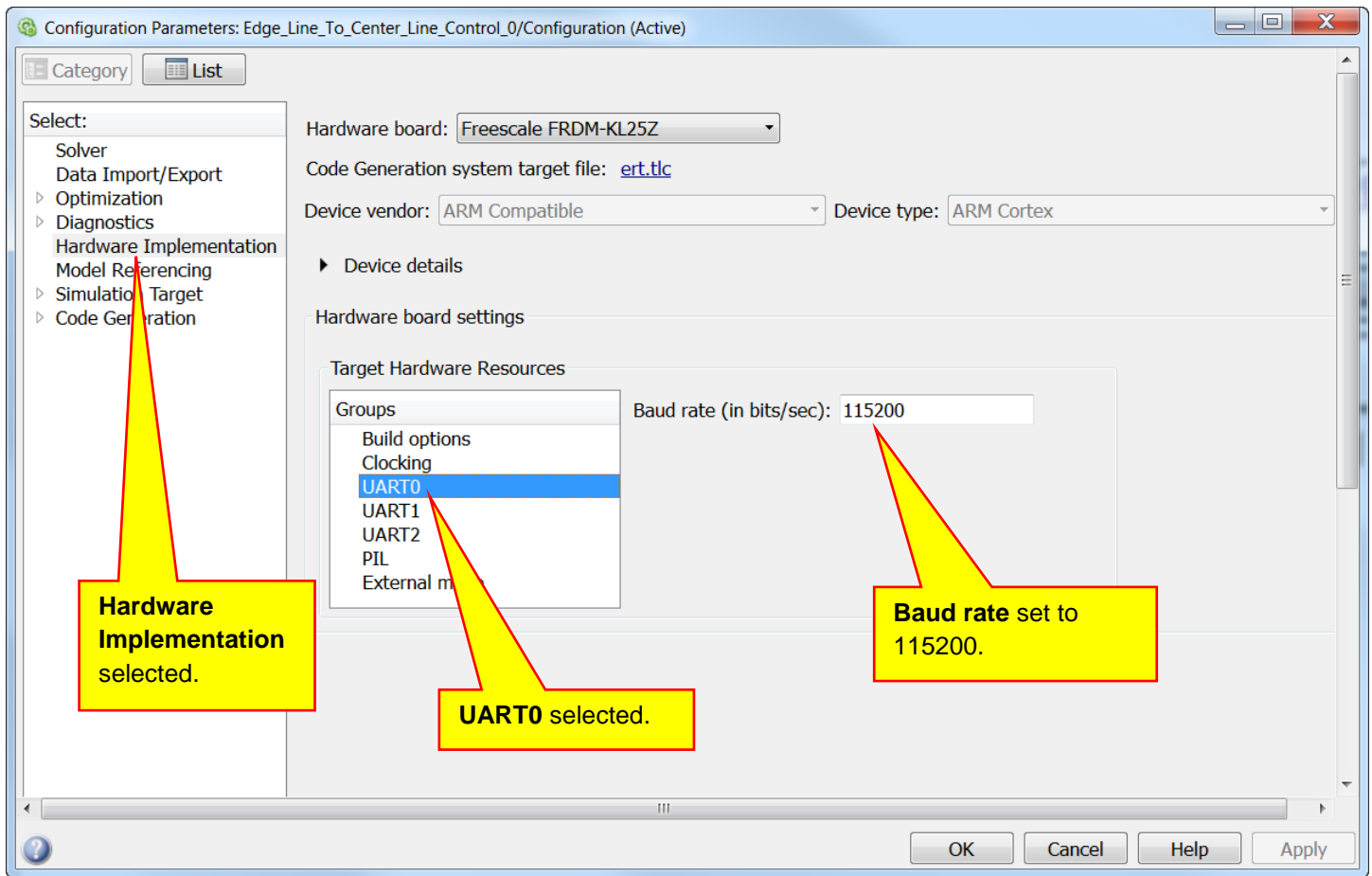


First, select **Solver**, expand the **Additional options**, and then specify the **Fixed-step size** as **0.25** seconds:



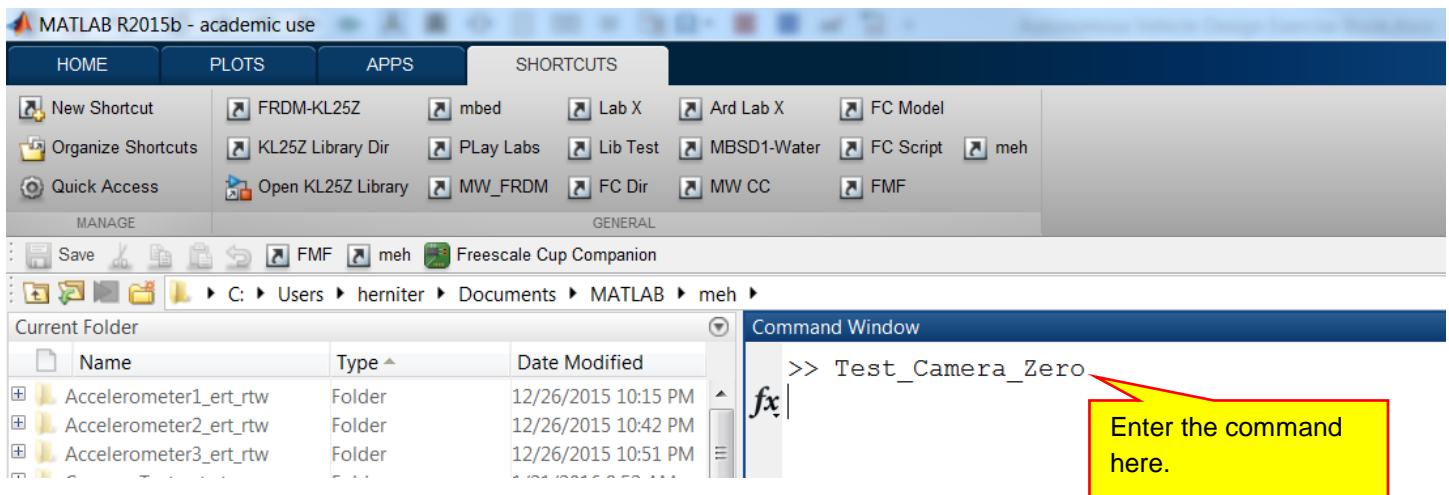
After setting these options. Click the **Apply** button. Do not close this window.

Next, select **Hardware Implementation, UART0** and set the **Baud rate to 115200**:

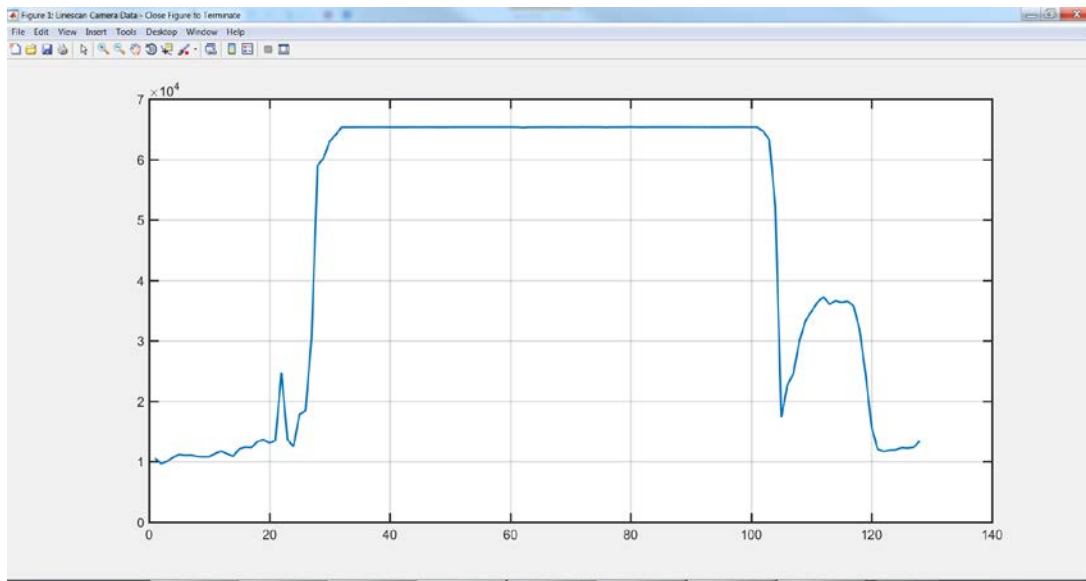


The Baud rate sets the communication speed of the serial port we are using. Since the camera sends a lot of data to our laptop computers, we need to specify a high communication speed. (High for serial ports.) Click the **Apply** button and then click the **OK** button. Build and download this model to your vehicle.

At the MATLAB command prompt, type the command **Test_Camera_Zero**:



Press the **Enter** key to run the command. After a few moments, a window will open and display a plot of the camera data:



Move the camera around to see how the image changes. You might also test the camera on your track to see how the camera views the track and the edge lines. To terminate the plot, close the figure. (The above camera plot was take with the car on a straight portion of the track.)

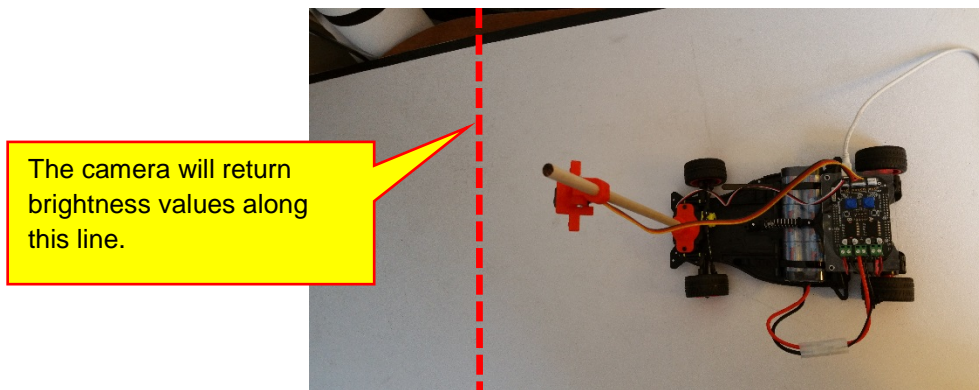
Congratulations! You have now successfully tested the complete vehicle!

Lesson XII: Processing Data from the Linescan Camera

Now that we have all of the hardware working, we will try to understand the basic operations of the control system that makes this an autonomous vehicle. There are three basic systems: steering, propulsion, and vehicle enabling and safety. In this section, we will look at the first component of the steering system which is obtaining data from the camera and processing that data.

A. Camera Data

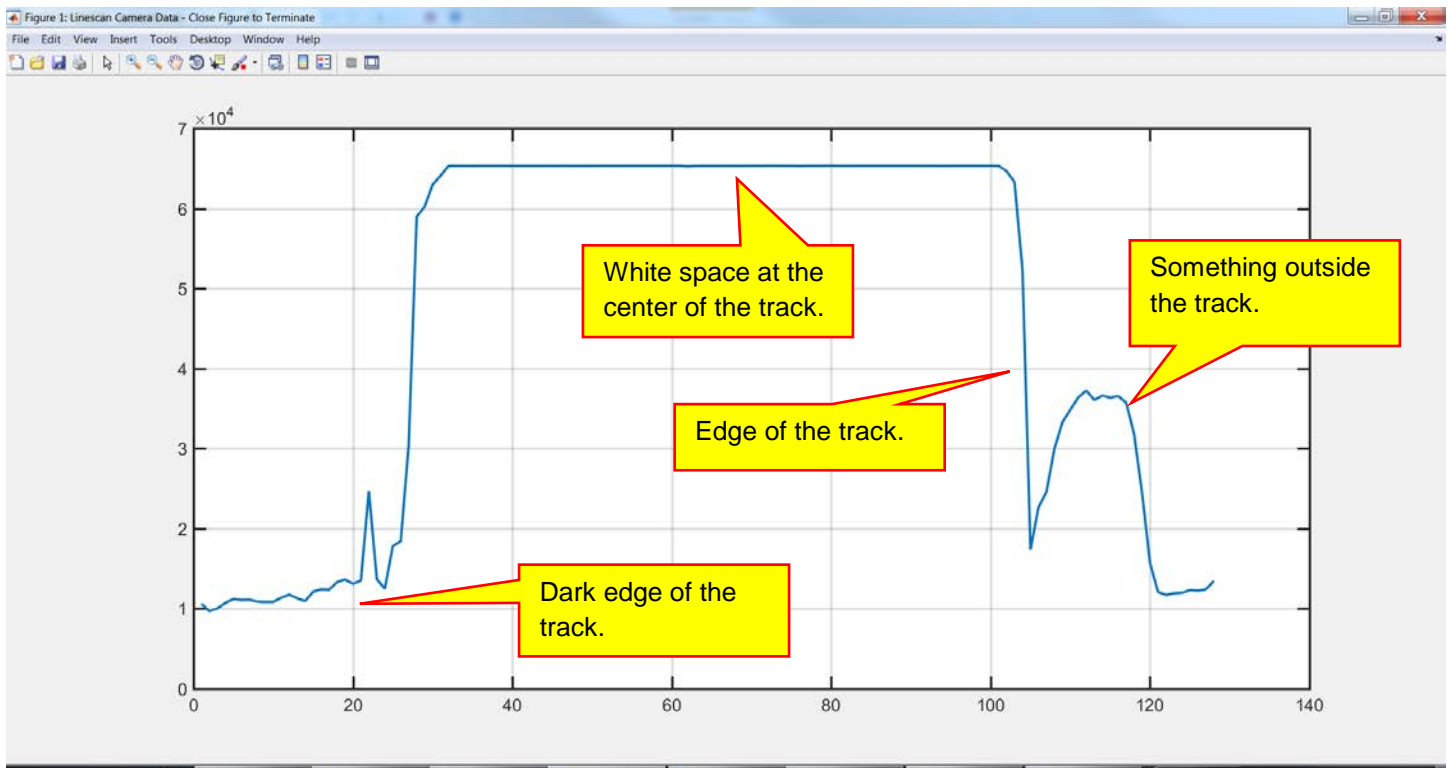
The camera returns an array of data. If you recall, a scalar value is a single value. An example of a scalar is the number 5. If variable x is a scalar, then $x=-3$ is an example of a scalar value. An array has multiple values. An example of an array is $[5\ 8\ 3\ 9\ 7]$. This is a 1-dimensional array with 5 values. This type of array is also called a row vector. The camera returns a row vector of 128 values. If you draw a line across the track as shown below:



the values returned by the camera are the brightness values along this line. If we use the function `Test_Camera_Zero`, we can see an example of the brightness values. **(On the car, you should still have the model from Section XI.D. If the model is not on your car, build and download that model onto your vehicle.)** Type the command `Test_Camera_Zero` at the MATLAB command prompt:

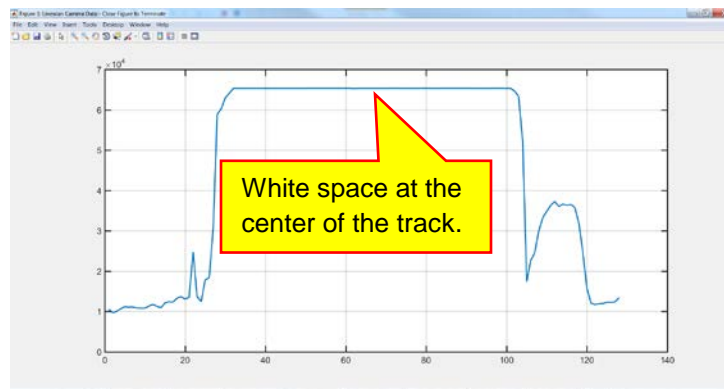
```
Command Window
>> Test_Camera_Zero
fx
```

After a few moments, a plot window will open:

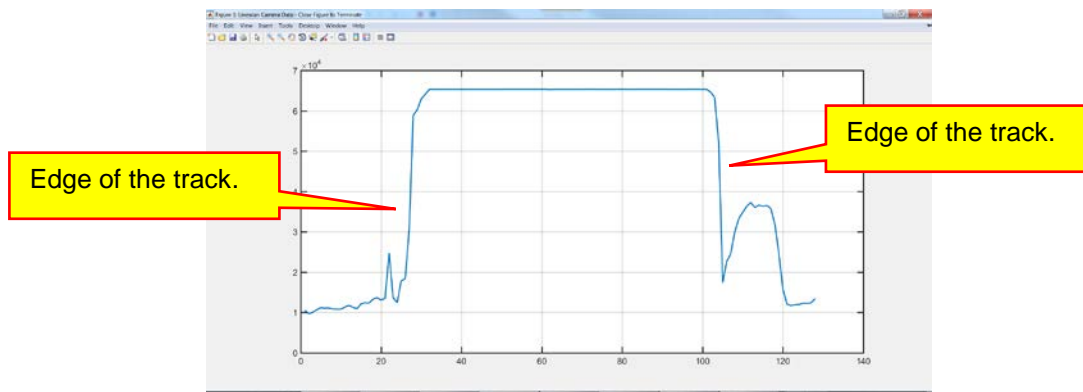


There are 128 elements in the array and each element can have a value from 0 to 65535. On a well-lit white track, white space usually has values close to the maximum. The edges of the track have values close to 10,000.

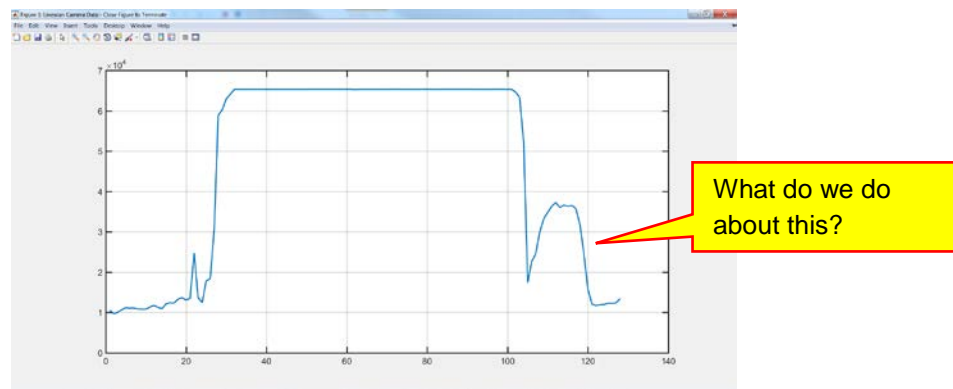
The screen capture above is from a car in the center of the track. It is a great example of what the camera sees. The white space in the center of the track is easily identifiable and basically flat with brightness values close to the maximum value:



There is a large drop off between the high values at the center and the lower values at the edges of the track:



In this example, it is pretty easy to differentiate between the center of the track and the edges. However, as in all data we will receive from the camera, there is some data that we need to ignore:



It is our job as engineers, given the data above, to determine where the center of the track is. The above is a great example of what the camera sees. The center and edges are identifiable and there is some data that needs to be neglected as it should not be used to find the center of the track.

The screen capture above is actually quite ideal. The car was in the center of a straight piece of track, the camera was pointed appropriately, and the lighting was good. Some things that we need to consider:

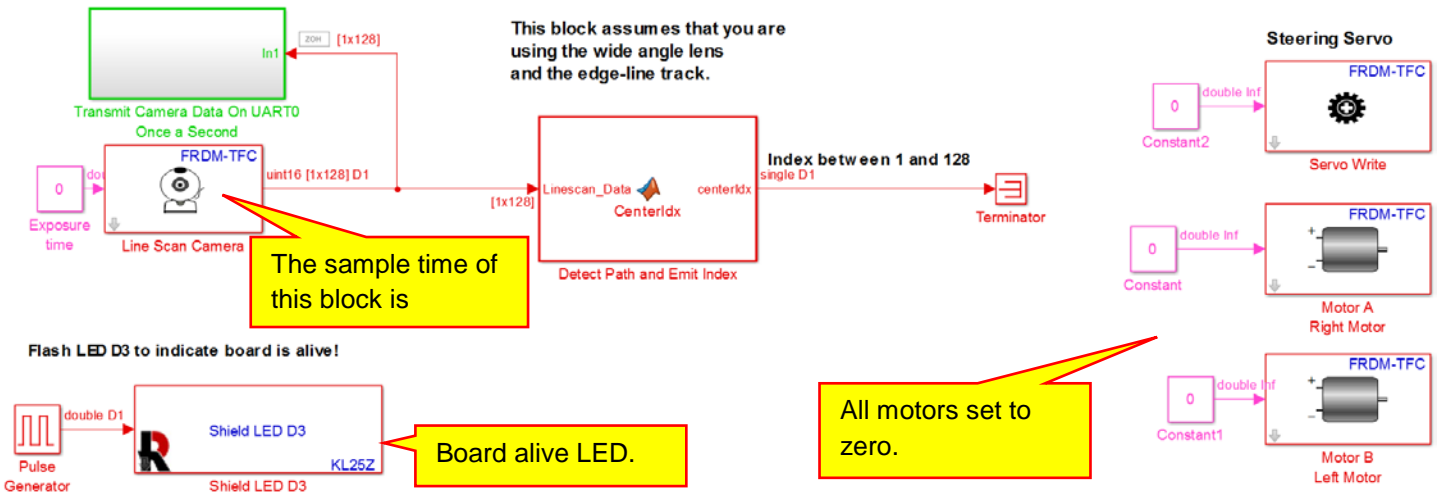
- 1) What if the car is not in the center of the track?
- 2) What if the track is not well lit?
- 3) What if lighting is not uniform? Every ceiling has multiple lights. These lights cast shadows.
- 4) What if there is something casting a shadow near the track or dark space near the track. (Like you stand close to the track while racing your car. Or the track is laid over a floor that has dark and light colors.)
- 5) What if the vehicle is going through a curve? (A vehicle that goes straight only is not too useful.)

These are all possibilities that you much consider when you think about how to improve the center finding algorithm of your camera.

B. Processing the Camera Data

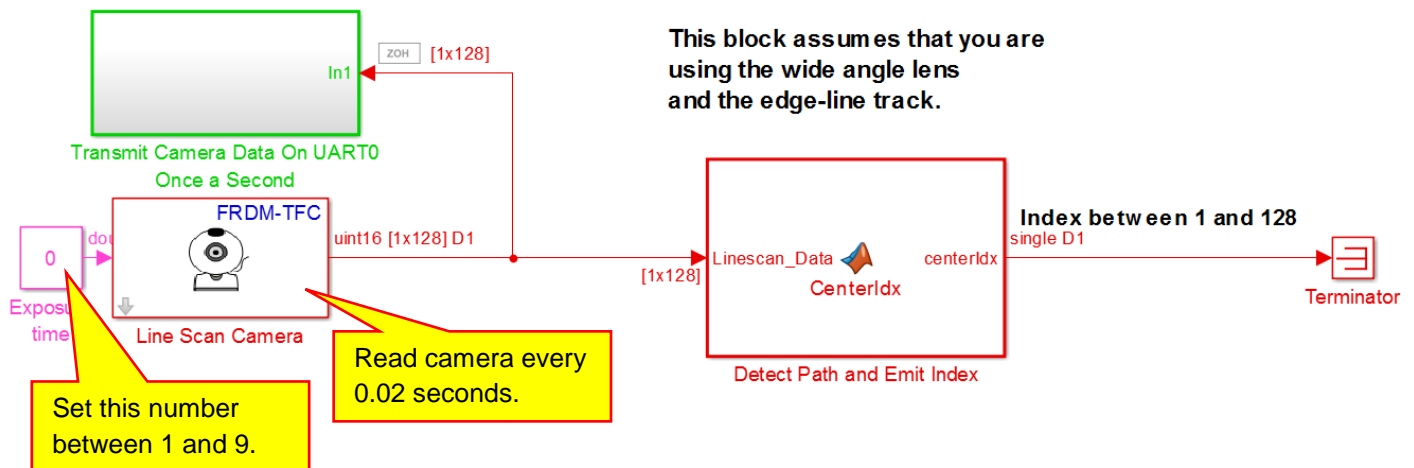
The high school model comes with a very simple algorithm for finding the center of the track. This algorithm is good enough to get your car around the track, but has room for a lot of improvement. Understanding and improving this will greatly increase the speed of your vehicle.

Change to directory, "Lesson 12 Files" Open model "Vehicle_Camera_Read_0.slx." **(Use this model as the starting point for all future models.)** This model contains the blocks to properly read the camera data.



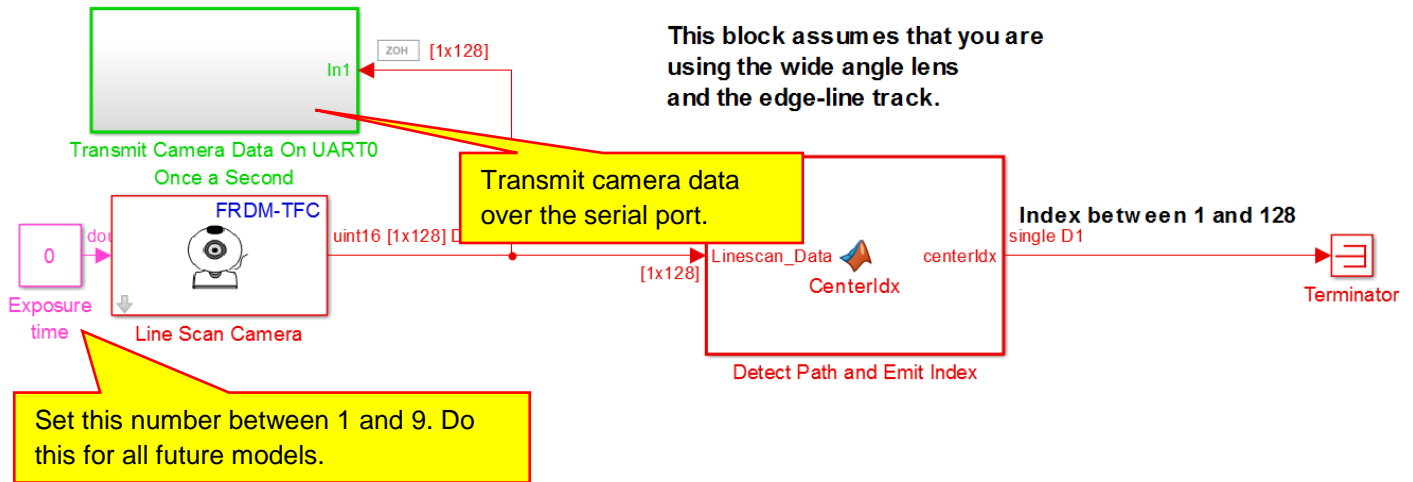
There are a few features of this model irrelevant to the camera at the moment. First, LED D3 blinks at a 1 Hz rate to indicate that the model is running correctly and that the board is alive. Second, the servo and drive motors are set to zero. This prevents them from doing anything unintended. (If we do not set them to zero, then the input signals to those devices may be undefined. When we power the board, the motors could do something unexpected. Examples could be full power to a drive motor causing your vehicle to fly off the table or a large signal to the servo motor causing it to rail against the side of the vehicle, thus damaging the servo motor.) To prevent this possibility, it is far easier to just set the unused outputs to zero.

The remainder of the model is for the reading the camera data:



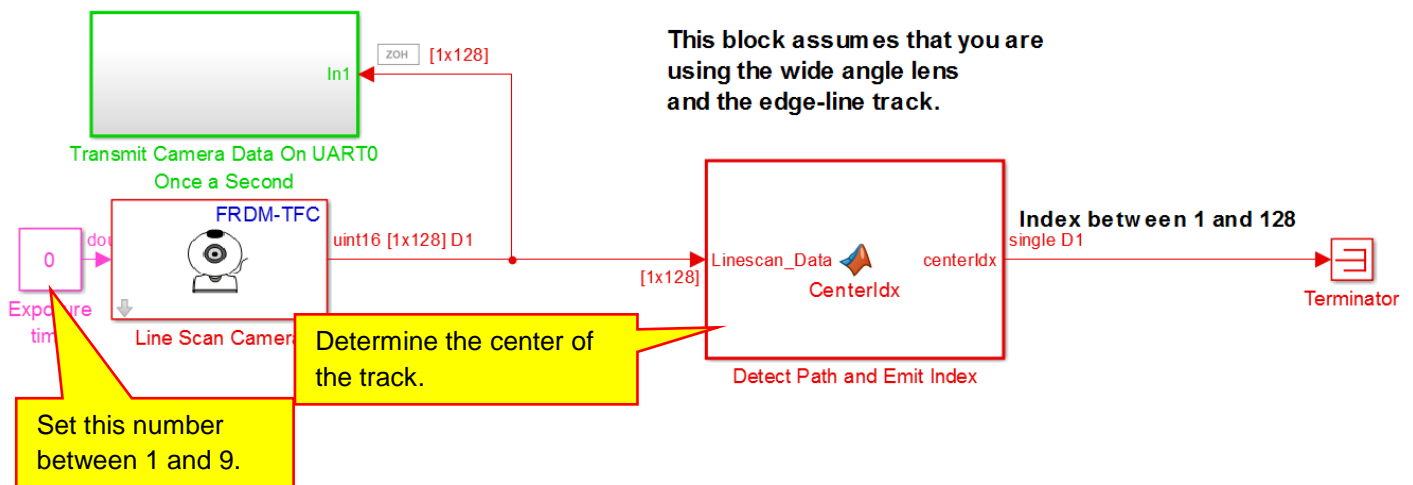
This portion of the model does three things. Data is read from the camera every 0.02 seconds. **(The sample time of this block is 0.02.)** This fast rate is necessary in order for the vehicle to navigate the track quickly. Reading the camera every 0.02 seconds will allow us to change the steering every 0.02 seconds. The faster the car moves, the more often we have to adjust the steering. We can't adjust the steering until we read new data from the camera. Thus, we need to read the camera frequently. Reading the camera every 0.02 seconds is as fast as we can read the data.

An important diagnostic (something we humans want to know about the vehicle) is what the camera data looks like. In order to view the camera data, we need to send it to our laptop computer and plot the data. The data is sent through the serial port to our computer using the block shown below:

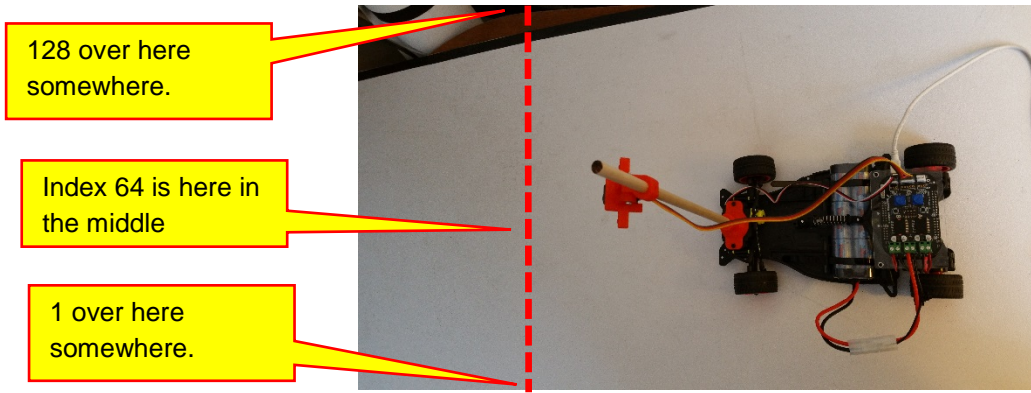


This block sends data once every second to the serial port. We will then plot the data using a MATLAB script so that we can view the data. Since the camera outputs a lot of data and it takes a lot of time to transmit that data, we only send data once a second. We only need to send the data once a second because humans will be looking at the data. The data transmitted over the serial port will not be used to adjust the vehicle steering because once a second is far too slow to adjust the steering of a fast moving vehicle. Instead, the data will be viewed by humans to understand what the data looks like. When we look at this data, we must remember that we only see changes once every second. Thus, when we view this data, the vehicle must be moving very slowly or not at all. (Imagine that you are navigating your way through a maze and are walking very slowly. If you open your eyes once every second, you might be able to make your way through the maze without bumping into any obstructions. However, if you attempt to run quickly through the maze while opening your eyes once every second, you will probably smash into the first obstruction (that you fail to see) and become disabled.)

The third block of this model processes the camera data and determines the center of the track:

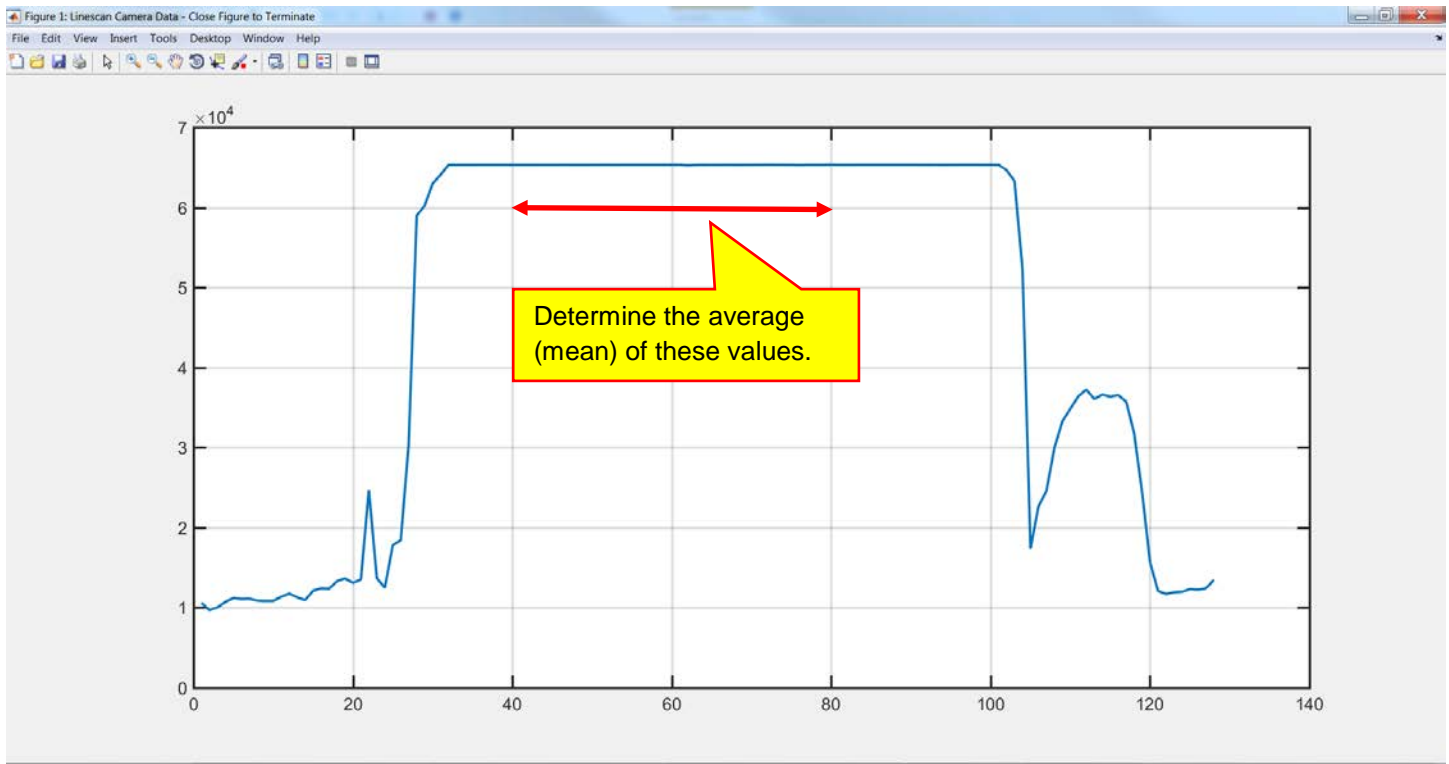


Remember that the camera returns 128 values representing the brightness going across the track. We will define the center of the track as the index 64:



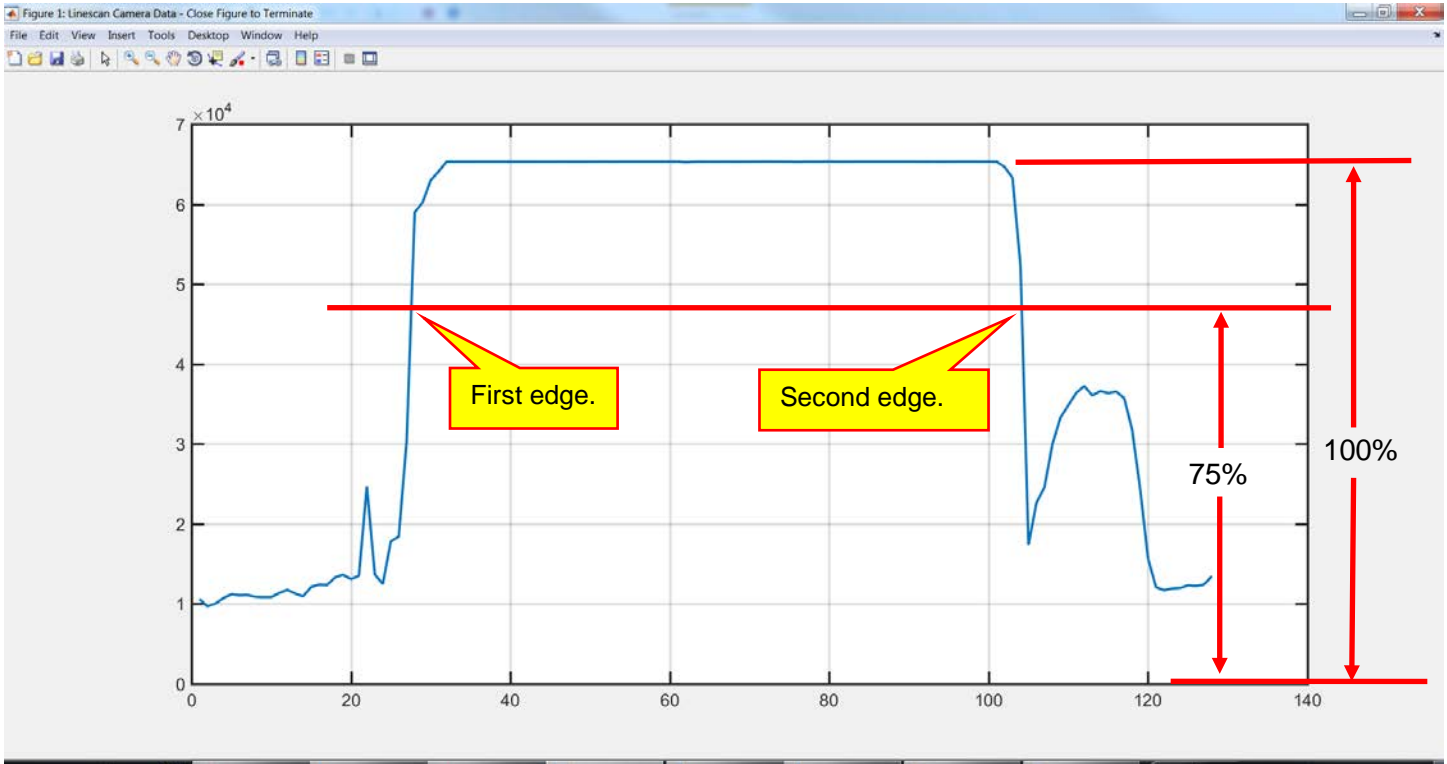
Note that we are using a wide angle lens and that the position of the camera makes a big difference. Indices 1 and 128 may be the edges of the track or past the edges of the track. (We don't really care.) However, index 64 is defined as the center of the track. The block labeled **Detect Path and Emit Index** finds where we think the center of the track is relative to the vehicle position. If the block emits 64, then the car is in the center of the track. If the block emits a different value, say 68 or 60, then the car is off-center. If the block emits 255, then the vehicle is lost and we better send out a search party. (Or, ignore the value and hope the next value is better – 255 indicates that the center finding algorithm did not find the center.)

Before opening the block, we will look at our “ideal” plot of data and discuss a simple algorithm for finding the center. First, find an average value of the “top” portion of the data:

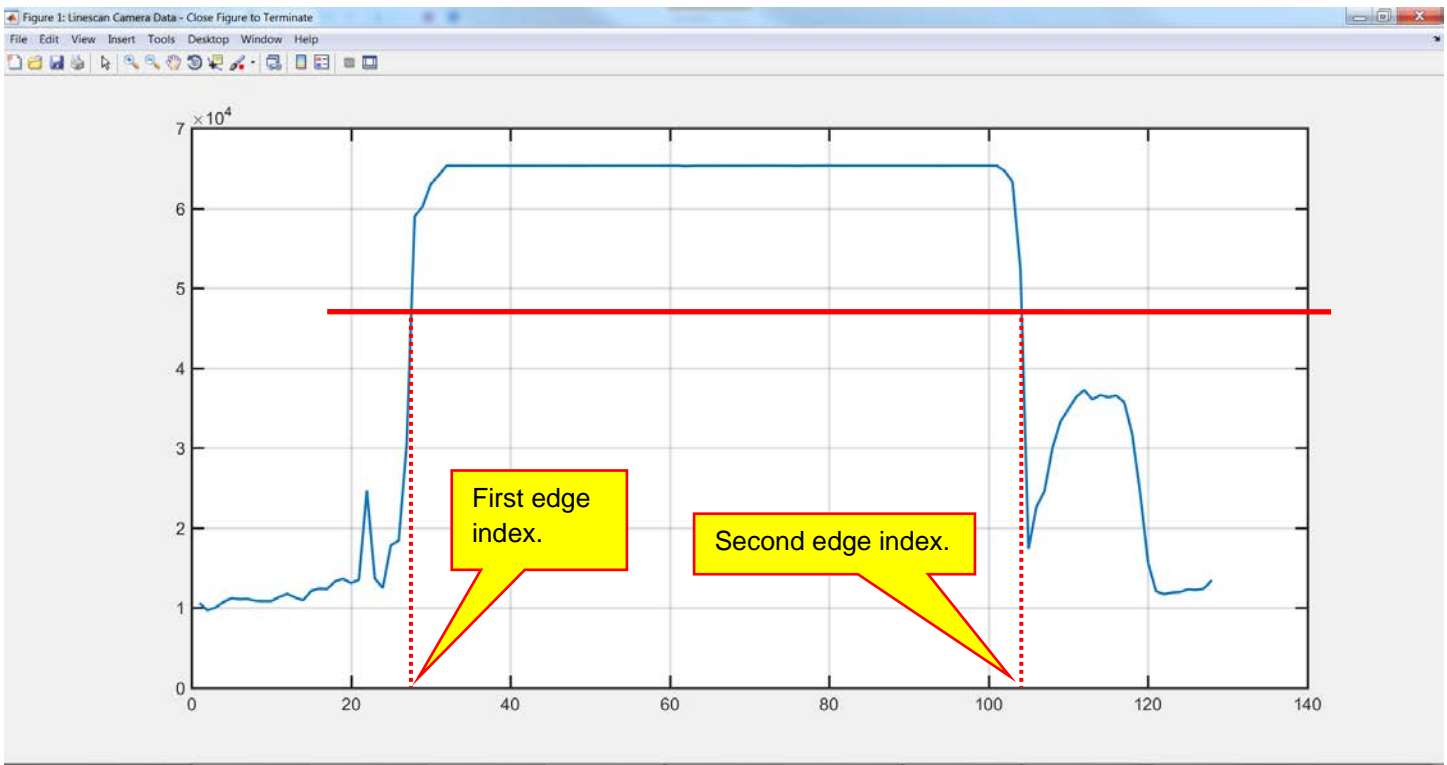


We will take the mean of the values for indices 40 to 80, because these are roughly the middle forty data values returned by the camera. The camera returns 128 values. The data from indices 1 to 10 and the data from indices 118 to 128 will probably be junk outside the track. Thus, indices 40 to 80- roughly represent the center of the track.

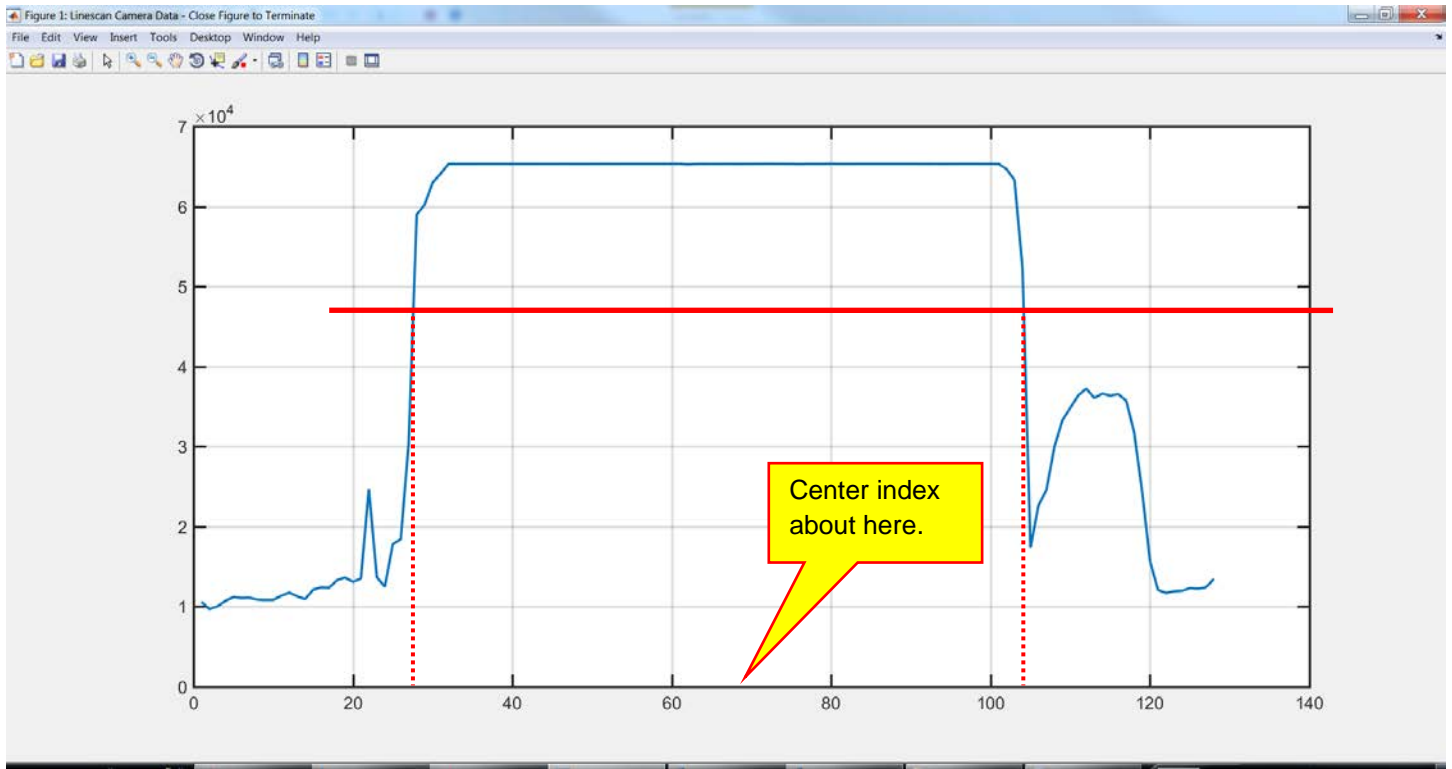
Next, we will recognize an “edge” as when the brightness is down to 75% of the mean value:



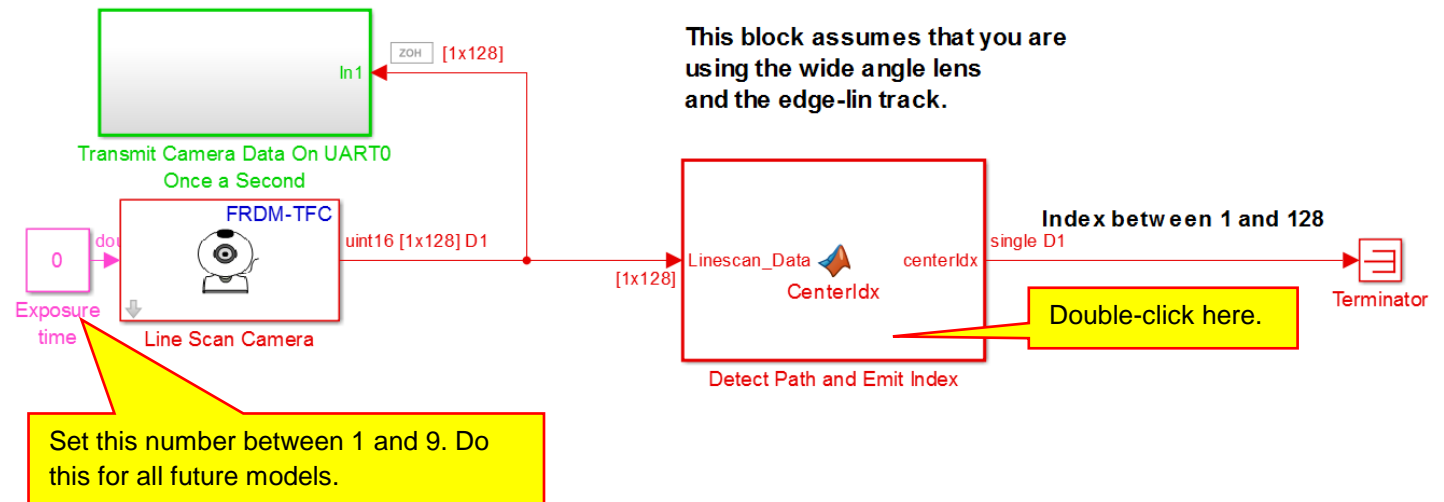
The location of the edges is the indices of where the brightness values are down to 75% of the average brightness value of the center:



The center of the track is calculated as the first edge index plus the second edge index divide by 2, (First + Second)/2:



Now that we have a simple algorithm, we will realize it using MATLAB code. MATLAB code can be placed inside a Simulink model using a MATLAB Function block. In our model, block **Detect Path and Emit Index** is a MATLAB function block that contains MATLAB code to implement our center finding algorithm. Double-click on the block to open it and view the code:



The MATLAB editor will open and display the code contained within this block:

```

1 function centerIdx = CenterIdx(Linescan_Data)
2 % function centerIdx = CenterIdx(Linescan_Data)
3 % This function finds the current position of the track's
4 % center.
5
6 -----
7 % The following code will take camera data and also the edges and center of
8 % the track using the algorithm the vehicle uses. If you modify the
9 % lines below, you must also modify the same lines in file
10 % "Full_High_School_Model_0_View_Camera.m".
11 %
12 %
13
14 First_Edge=single(1);
15 Second_Edge=single(128);
16
17 Mean_Value=3*mean(Linescan_Data(40:80))/4;
18 %Find the First Edge
19 for i=64:-1:1
20     if Linescan_Data(i)<Mean_Value;
21         First_Edge=single(i);
22         break;
23     end
24 end

```

The line `function centerIdx = CenterIdx(Linescan_Data)` defines the name of the function as `Center_Idx`. The input to the function is contained in variable `Linescan_Data`. This is the camera data passed to the block from the Line Scan Camera block. Thus, the input to this function is the 128 element array of brightness values from the camera. The output of the block is the index representing where we think the center of the track is. We will now look at the algorithm in parts.

Note that you will see many commands that look like `single(1)` or `single(i)`. This means that values are saved as single-precision real numbers on the microcontroller. This specifies the data type we are using. For the moment you can just ignore the word “single” and can think of `single(1)` as `1` and `single(i)` as `i`, or for that matter `single(anything)` as `anything`.

The first thing we do is initialize the first and second edges to the maximum and minimum possible indices. If values are found for the two edges, these values will be changed.

```

First_Edge=single(1);
Second_Edge=single(128);

```

Next, we calculate the threshold as 75% of the mean value of the Linescan camera data:

```

% Calculate the threshold as 75% of the mean value of the bright section of
% the track.
Threshold=0.75*mean(Linescan_Data(40:80));

```

Recall that the command `40:80` generates the values from 40 to 80 incremented by 1. That is, 40, 41, 42, 43, ..., 79, 80. Thus, `Linescan_Data(40:80)` is the middle portion of the linescan data and `mean(Linescan_Data(40:80))` finds the average value of the middle portion. Finally.

`0.75*mean(Linescan_Data(40:80))` takes 75% of the mean of the middle portion. We call this value the **Threshold** because we will be looking for when the Linescan Camera data falls below this value.

Next, we will try to find the first edge. We will start in the middle and work backwards looking for when the Linescan data falls below the threshold. Remember that the command `64:-1:2` starts at 64 and counts down by 1 until it reaches 2:

```
%Find the First Edge
for i=64:-1:2
    if Linescan_Data(i)<Threshold;
        First_Edge=single(i);
        break;
    end
end
```

We are using a `for` loop:

```
for i=64:-1:2
    (Lines inside)
end
```

This command sets `i` to 64 and then evaluates the lines inside. Then it sets `i` to 63 and evaluates the lines inside. Then it sets `i` to 62, and so on. `i` will continue counting down until `i` equals 2 or we hit the `break` command, at which point the `for` loop stops.

Inside the `for` loop, we have the lines

```
if Linescan_Data(i)<Threshold;
    First_Edge=single(i);
    break;
end
```

We check each value of the linescan data. Once we find a value less than the threshold, we save the index and break the `for` loop. Thus, we look at the linescan camera data starting at the middle and we search backwards looking for when the value is less than the threshold. When we find a value less than the threshold, we save the index of that value as our first edge. **Important note: If we do find an edge, the first edge will be set to something other than 1. Thus, if the first edge is equal to one, we never actually found the first edge.**

The code for finding the second edge is similar to the first edge except that we start looking at the middle of the data and search forward. When we find a value less than the threshold, we save the index of that value as our second edge:

```
% Find the Second Edge
for i=64:1:127
    if Linescan_Data(i)<Threshold;
        Second_Edge=single(i);
        break;
    end
end
```

Important note: If we do find an edge, the second edge will be set to something other than 128. Thus, if the first edge is equal to 128, we never actually found the second edge.

Finally, if we did find two edges, the center is calculated as the average of the two edges:

```
% If the edge indices are valid, then calculate the
% center as the average of the two edge indices.
if First_Edge>1 && Second_Edge<128
    centerIdx=(Second_Edge+First_Edge)/2;
else
    centerIdx=single(255);
end
```

Note that if the `centerIdx` is set to 255 when we do not find two edges. This is the code we use to indicate that our algorithm could not find the center of the track.

At the moment, do not modify this code. (In the future you may improve this algorithm, but not now.) Build and download this model to your vehicle.

C. Plotting the Camera Data

We have created a script that you can use to view the camera data, the calculated edges, and the center index. This script is located in file `Full_High_School_Model_0_View_Camera.m`. Open this file. If you scroll down to lines 111 to 140, you will notice that the code in this m-file is the same as in our vehicle model:

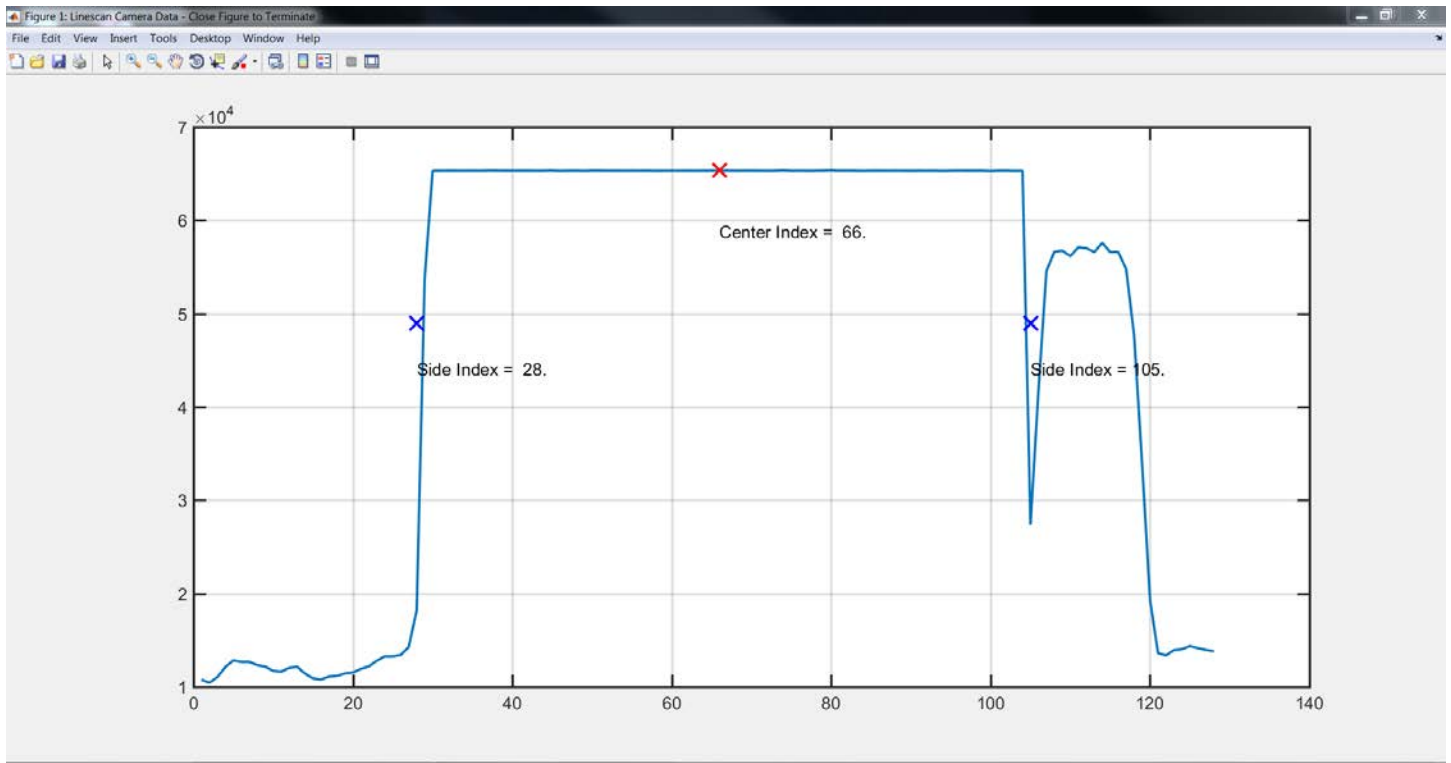
```
110
111 -     First_Edge=single(1);
112 -     Second_Edge=single(128);
113
114     % Calculate the threshold as 75% of the mean value of the bright section of
115     % the track.
116 -     Threshold=0.75*mean(Linescan_Data(40:80));
117
118     %Find the First Edge
119 - □ for i=64:-1:2
120         if Linescan_Data(i)<Threshold;
121             First_Edge=single(i);
122             break;
123         end
124 - end
125
126     % Find the Second Edge
127 - □ for i=64:1:127
128         if Linescan_Data(i)<Threshold;
129             Second_Edge=single(i);
130             break;
131         end
132 - end
133
134     % If the edge indices are valid, then calculate the
135     % center as the average of the two edge indices.
136 -     if First_Edge>1 && Second_Edge<128
137         centerIdx=(Second_Edge+First_Edge)/2;
138     else
139         centerIdx=single(255);
140 - end
```

This is because we want this script to use the same algorithm as our vehicle, but we want to plot the information so we can visualize the data and our algorithm. If you modify the center finding algorithm in your vehicle model, then you should also make the same changes here so that the generated plot matches the algorithm in your vehicle. Close the file as we will not make any changes.

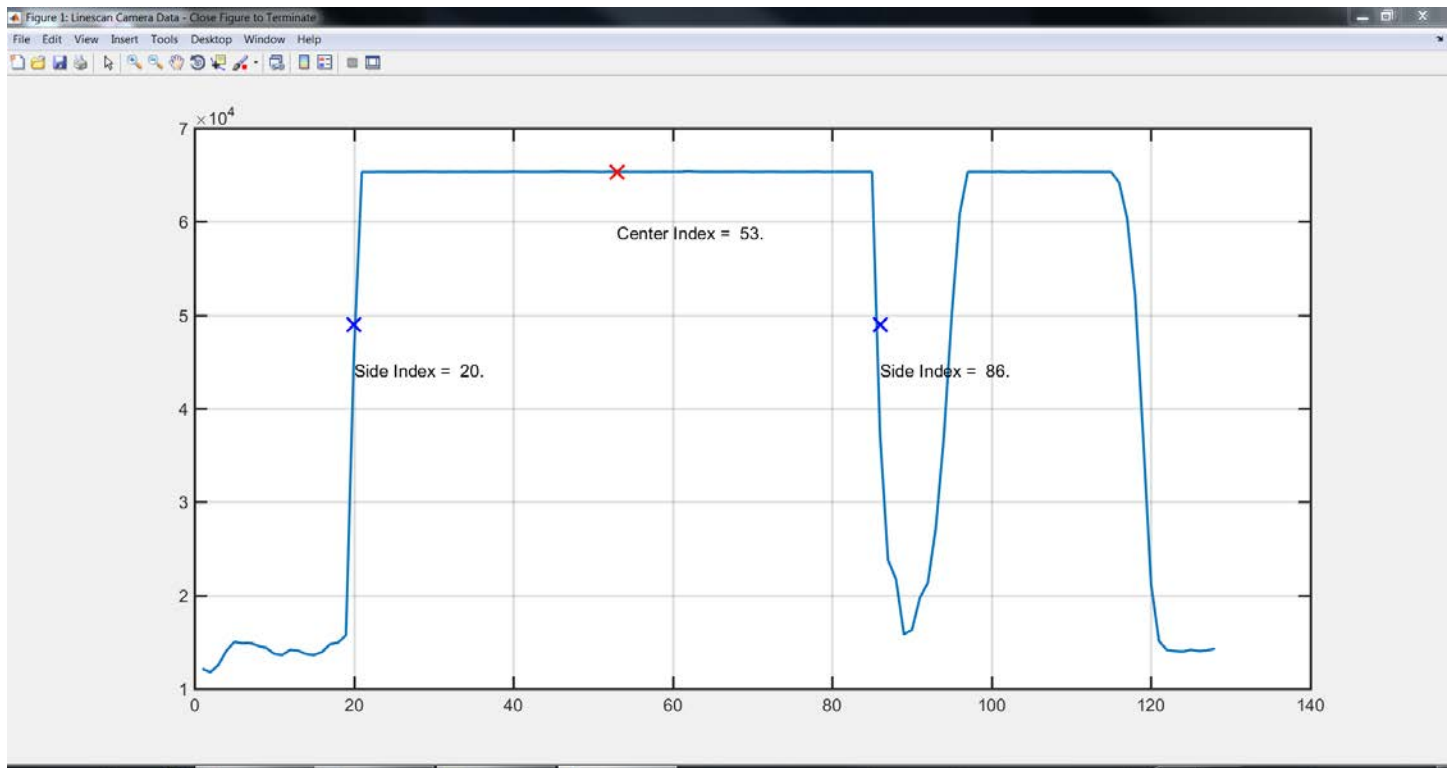
We now want to view the camera data and see how our center finding algorithm works:

- 1) Build and download model `Vehicle_Camera_Read_0` to your vehicle. (You should have already done this in the previous section.)
- 2) Place your vehicle in a straight section of the track midway between the two edge lines.
- 3) Connect your vehicle to your laptop using the SDA port on the KL25Z and a USB port on your laptop.
- 4) Run the script file `Full_High_School_Model_0_View_Camera.m`.

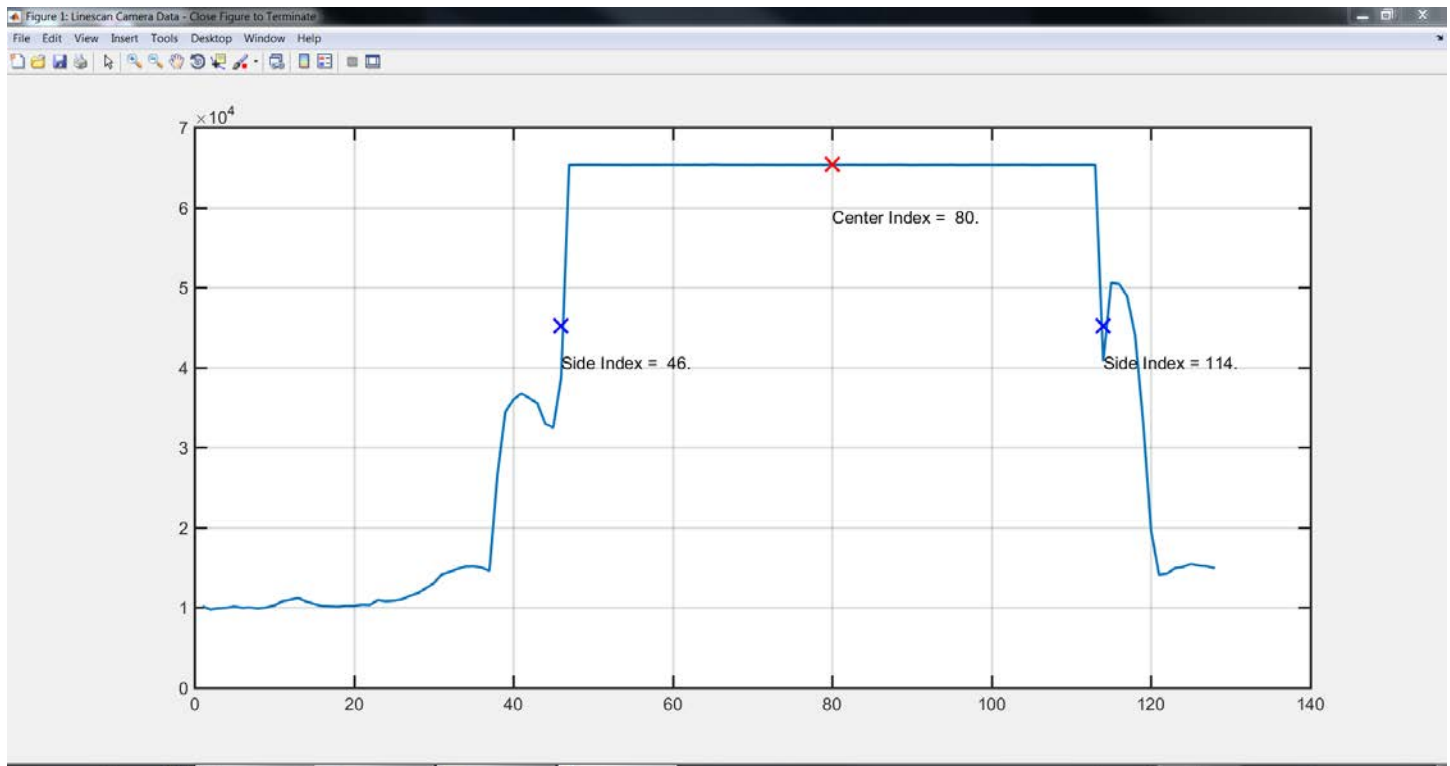
Depending on lighting, you should see a plot similar to the one below:



Note that the plot displays where we calculate the two edges and the center index. If we move the vehicle toward one side of the track, the plot and displayed indices will change:



Next, move the vehicle near the other side of the track:



In these screen captures, we display the side indices only to show us how well our algorithm is working. That information is not used by our vehicle (at the present moment – maybe you will use this information). The vehicle uses the center index. This value is compared to 64 and will determine if we turn the wheels right or left, and by how much (in the next lesson).

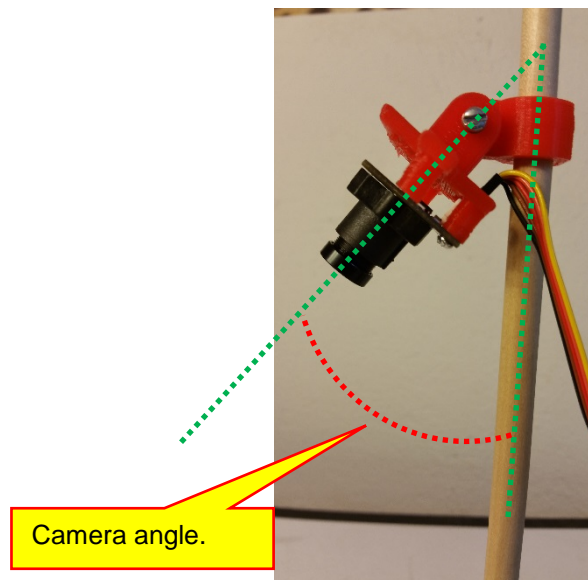
Note that the data you display and the accuracy of the algorithm will depend on the lighting, camera angle, and shape of the track. In the next section, we will observe the camera data and algorithm for various conditions and decide if we need to modify our algorithm.

D. Characterizing the Linescan Camera Data

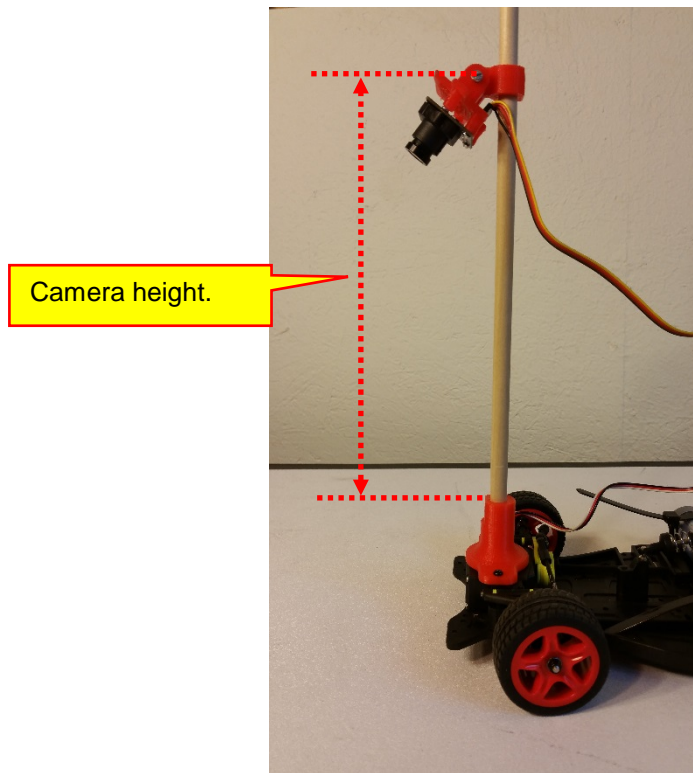
Now that we have the tools to observe what the linescan camera is seeing, we would like to make a lot of measurements so that we become familiar with what the camera sees and how it changes under various conditions. Our goals in the endeavor are many:

1. Obtain a basic understanding of what the camera sees.
2. How do the plots and indices change under different lighting conditions?
3. How is a curve different than a straight section?
4. How does the plot change with camera angle?
5. How does the plot change with camera height?
6. How does the plot change with camera focus?
7. In a straight section, how does the plot change if we are closer to one side of the track than the other.
8. In a curved section, how does the plot change if we are closer to one side of the track than the other.
9. After we choose a camera angle, height, and focus, is there a way to detect that we have set these correct in the future? (The camera can move with use and handling. We will always need to readjust it.)

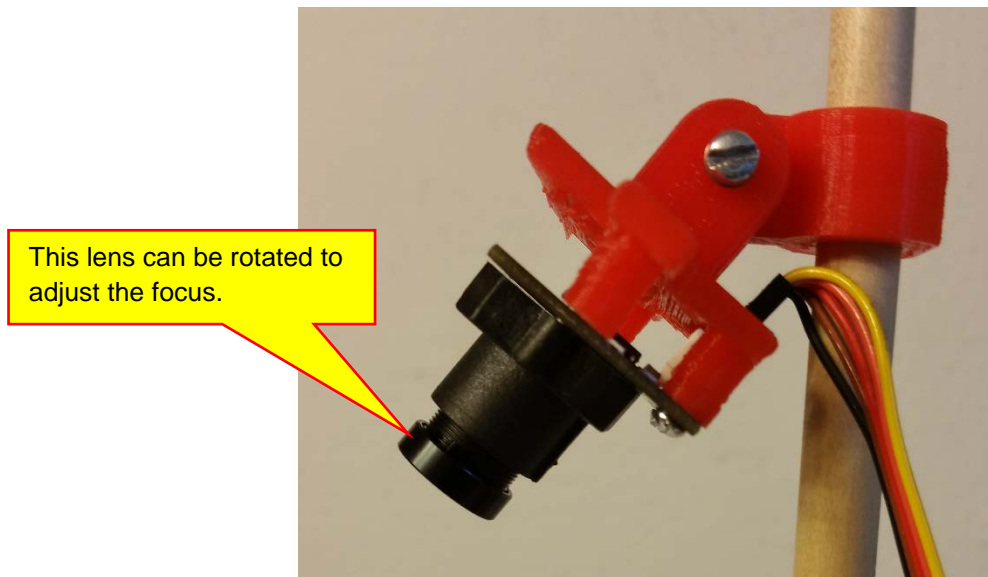
We will now define the camera height and angle. The picture below defines the camera angle:



You should be able to measure or calculate this camera angle. The height of the camera is defined below. Note that the height is the distance from the top of the lower mount to the pivot screw in the camera mount:



The camera focus can be adjusted by rotating the camera lens clockwise or counter-clockwise:



1. Effect of Camera Focus

We will now see how the camera focus affects the various image properties. Place the car in the center of a straight portion of the track. Rotate the focusing ring clockwise as much as you can without being too forceful. Measure the indices and note anything important about the plot. Then rotate the focusing ring two full turns counter-clockwise (CCW) and repeat the measurements. Repeat again for four and six full turns counter clockwise. While collecting this data, you might want to take screen captures of the images for future reference. If you do, note the name of the file where you save the screen capture. In Windows 7, you can make screen captures using a program called the Snipping Tool. In the last row of the table, note which focus setting you think is best. For these measurements, set the camera height to 22 centimeters and the camera angle to 45 degrees.

Title: Camera Focus				
Constant Properties				
Camera Height 22 cm	Camera Angle 45 Degrees		Straight Track	Centered Vehicle
Measured Properties				
Focus (Turns CCW)	Side Index 1	Center Index	Side Index 2	Screen Capture #
Max - Clockwise				
2				
4				
6				
Which Is Best?				

Repeat the above measurements for a curved section of track. Place the vehicle at the beginning of the curved section of track where the vehicle would first enter the curve. Make sure that the vehicle is centered between the edge lines. Fill in the following table:

Title: Camera Focus				
Constant Properties				
Camera Height 22 cm	Camera Angle 45 Degrees		Curved Track	Centered Vehicle
Measured Properties				
Focus (Turns CCW)	Side Index 1	Center Index	Side Index 2	Screen Capture #
Max - Clockwise				
2				
4				
6				
Which Is Best?				

From these two measurements, determine what you think is the best focus setting. Use this setting for the remainder of the measurements. Note that you can add more measurements if you do not think these two tables are sufficient to determine what the best focus is.

2. Camera Angle

With the focus set to what you have determined as the best focus, we now wish to see the effect of camera angle. We will keep the camera height at 22 cm. Due to the camera wires, the minimum camera angle is about 45 degrees. Do not try an angle less than 45 degrees or you may damage the camera. Perform this measurement on both straight and curved sections of track:

Title: Camera Angle				
Constant Properties				
Camera Height 22 cm	Camera Focus – User Determined Best		Straight Track	Centered Vehicle
Camera Angle (Degrees)	Measured Properties			
	Side Index 1	Center Index	Side Index 2	Screen Capture #
45				
50				
55				
60				
65				
Which Is Best?				

Title: Camera Angle				
Constant Properties				
Camera Height 22 cm	Camera Focus – User Determined Best		Curved Track	Centered Vehicle
Camera Angle (Degrees)	Measured Properties			
	Side Index 1	Center Index	Side Index 2	Screen Capture #
45				
50				
55				
60				
65				
Which Is Best?				

From these two measurements, determine what you think is the best camera angle. Use this setting for the remainder of the measurements. Note that you can add more measurements if you do not think these two tables are sufficient to determine the best camera angle.

3. Camera Height

With the focus and camera angle set to what you have determined as the best of each, we now wish to see the effect of camera height. Perform this measurement on both straight and curved sections of track:

Title: Camera Height				
Constant Properties				
Camera Height – User Determined Best	Camera Focus – User Determined Best		Straight Track	Centered Vehicle
Camera Height (cm)	Measured Properties			
	Side Index 1	Center Index	Side Index 2	Screen Capture #
22				
19				
16				
13				
10				
Which Is Best?				

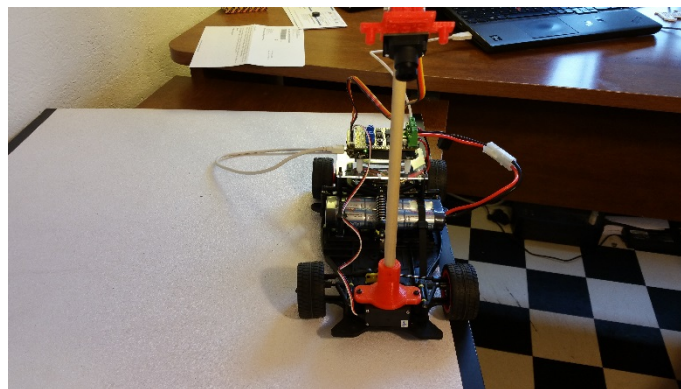
Title: Camera Height				
Constant Properties				
Camera Height – User Determined Best	Camera Focus – User Determined Best		Curved Track	Centered Vehicle
Camera Height (cm)	Measured Properties			
	Side Index 1	Center Index	Side Index 2	Screen Capture #
22				
19				
16				
13				
10				
Which Is Best?				

4. The Best of All Settings

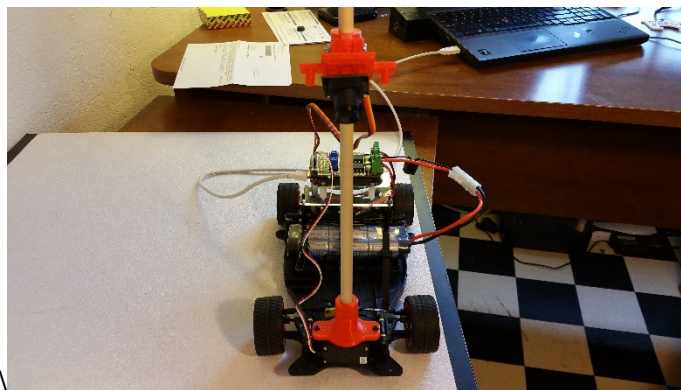
It turns out that camera height, angle, and focus are not all mutually exclusive. Changing one does affect the others. However, after making these measurements we have a better sense of how the individual settings affect the camera vision and our center finding algorithm. At the moment, set the angle, height, and focus to what you determine as the best. Note that these values will change in the future based on vehicle speed and how you want to anticipate curves. Thus, the settings we choose are only the initial settings. However, we have learned how to make measurements to get a qualitative feel of how the vision system works.

5. Index versus Vehicle Position

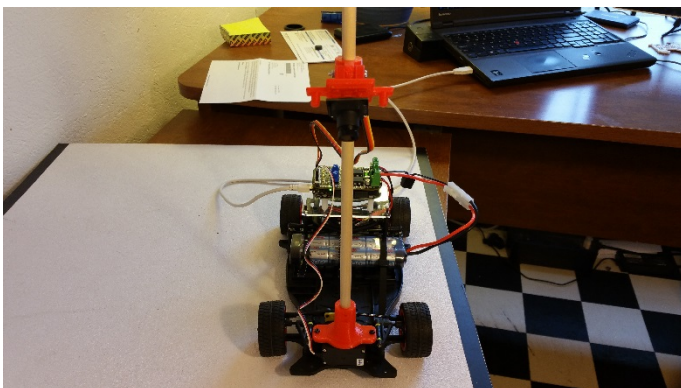
As a last exercise, we want to measure how the indices change based on the position of the vehicle in the track. We will use a straight track. Start with the left two wheels of the vehicle on the left black line:



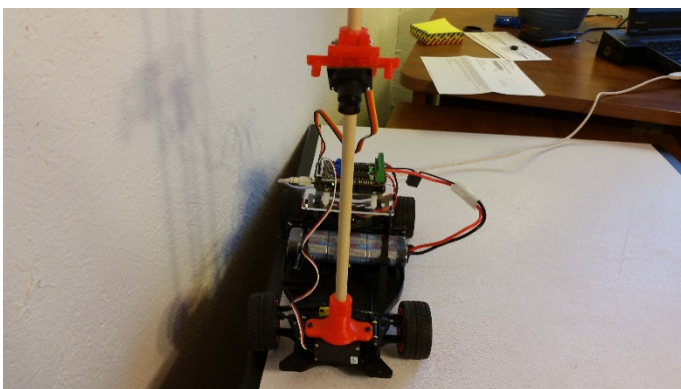
Measure the indices. Then move the vehicle over one inch:



Measure the indices. Then move the vehicle over 1 inch again:



Repeat the procedure every inch until the right tires reach the right line:



Measure the indices. Fill in the table below:

Title: Indices versus Vehicle Position				
Constant Properties				
Camera Height – User Determined Best	Camera Focus – User Determined Best	Camera Angle – User Determined Best	Straight Track	
Distance from Left Black Line (in)	Measured Properties			
	Side Index 1	Center Index	Side Index 2	
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				

Next, we wish to plot this data. For example, suppose the data looked as shown (abridged):

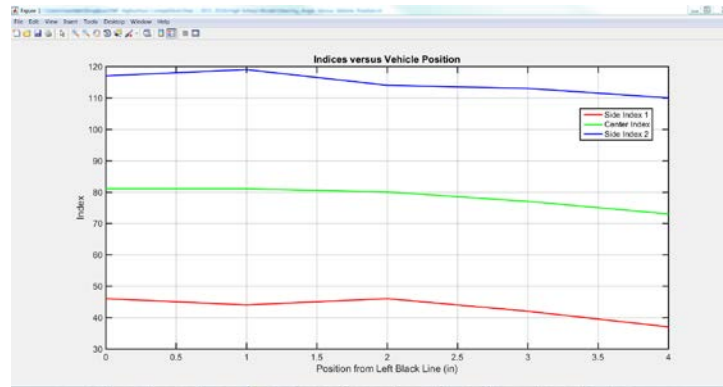
Distance from Left Black Line (in)	Measured Properties			
	Side Index 1	Center Index	Side Index 2	
0	46	81	117	
1	44	81	119	
2	46	80	114	
3	42	77	113	
4	37	73	110	

To plot this data, we would use the following MATLAB commands:

```
%Steering angle versus vehicle position script
Position = [0, 1, 2, 3, 4];
Side_Index_1 = [46, 44, 46, 42, 37];
```

```
Center_Index = [81, 81, 80, 77, 73];
Side_Index_2 = [117, 119, 114, 113, 110];
plot(Position, Side_Index_1, 'r', Position, Center_Index, 'g', Position,
Side_Index_2, 'b');
grid on
title('Indices versus Vehicle Position');
xlabel('Position from Left Black Line (in)');
ylabel('Index');
legend('Side Index 1', 'Center Index', 'Side Index 2');
```

For the data given, this script generates the plot below:



This script is saved as file `Steering_Angle_Versus_Vehicle_Position.m` in the high school vehicle model directory, if you wish to use it. A slightly different way to view the data is the script below:

```
%Steering angle versus vehicle position script
```

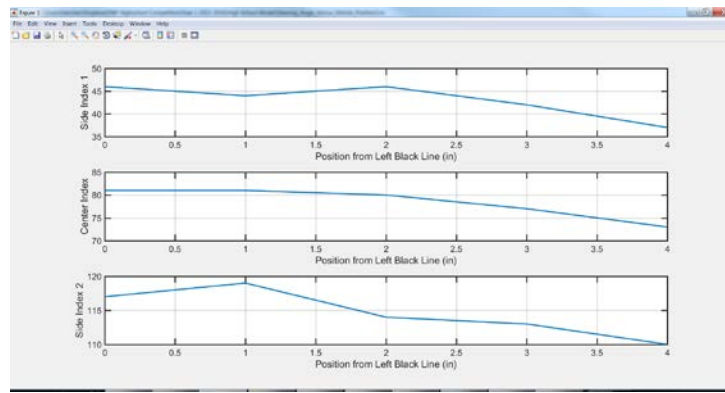
```
Position = [0, 1, 2, 3, 4];
Side_Index_1 = [46, 44, 46, 42, 37];
Center_Index = [81, 81, 80, 77, 73];
Side_Index_2 = [117, 119, 114, 113, 110];
%
```

```
subplot(3,1,1)
plot(Position, Side_Index_1);
grid on
xlabel('Position from Left Black Line (in)');
ylabel('Side Index 1');
```

```
subplot(3,1,2)
plot( Position, Center_Index);
grid on
xlabel('Position from Left Black Line (in)');
ylabel('Center Index');
```

```
subplot(3,1,3)
plot(Position, Side_Index_2);
grid on
xlabel('Position from Left Black Line (in)');
ylabel('Side Index 2');
```

This script generates the plot below:



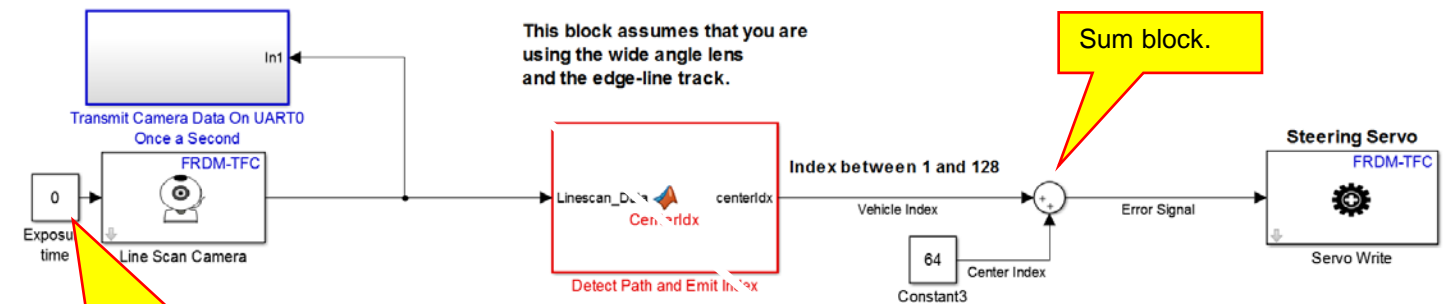
This script is saved as file `Steering_Angle_Versus_Vehicle_Position2.m` in the vehicle model directory, if you wish to use it.

Lesson XIII: Autonomous Steering

Now that we have a signal from the Linescan Camera that determines where it thinks the center of the track is, we can create a system that adjusts the steering to automatically center the vehicle. Here, we will learn how to turn the wheels to center the vehicle based on the index determined by center finding algorithm.

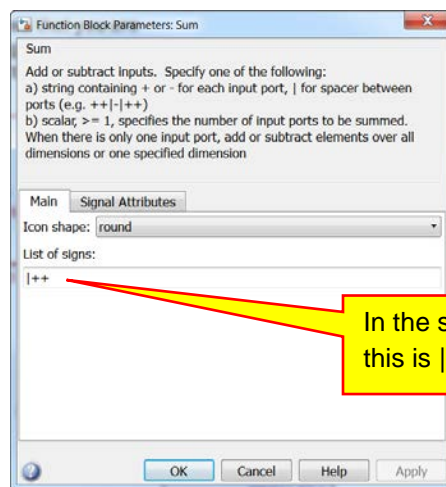
A. Turning the Wheels Automatically

In Lesson XII we learned that block **Detect Path and Emit Index** analyzes the data from the Linescan camera and emits an index between 1 and 128. This index, which we will now refer to as the **Vehicle Index** represents where the vehicle thinks the center of the track is relative to where the vehicle is on the track. We will define the center of the track as the index 64. The difference between 64 and the Vehicle Index is the **Error Signal**. We can calculate this error using the model below. **(Do not build this model from scratch. Start with model "Vehicle_Camera_Read_0.slx" which is located in directory "Lesson 12 Files." Modify this model as shown below. Do not start from scratch.)**

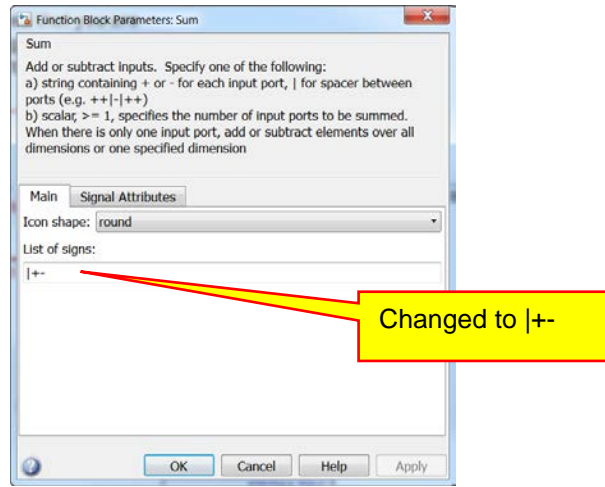


Set this number between 1 and 9. Do this for all future models.

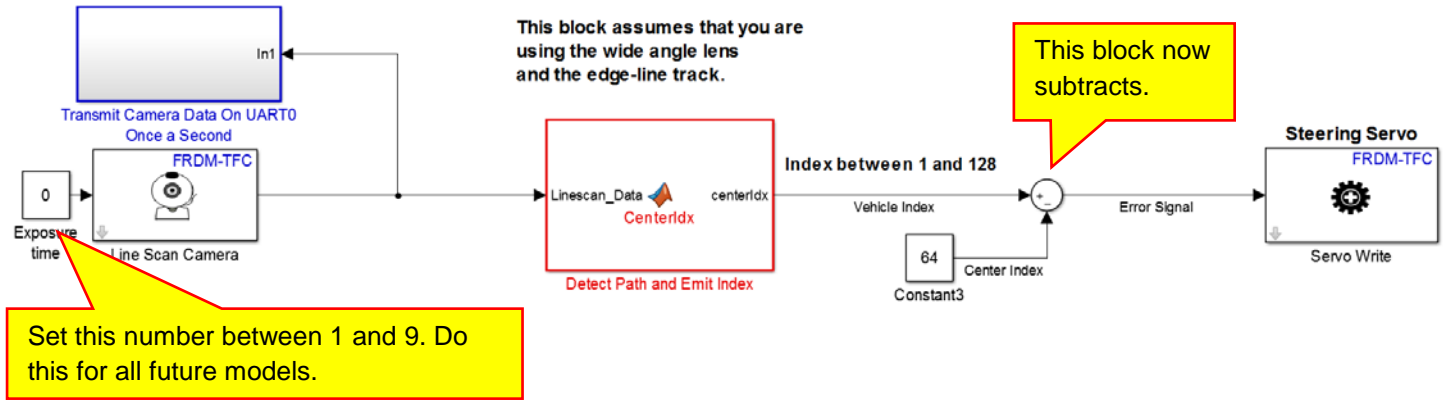
We need to change the sum block to a subtraction block. If you double-click on the **Sum** block, you will see the following:



When you place the **Sum** block, the default **List of signs** is |++. The | means no sign at the top of the circle. The ++ means that the left input is a + and the bottom input is a +. To change the block function to subtraction, change the **List of signs** to |+-:



The block should now show the bottom input being subtracted from the left input:

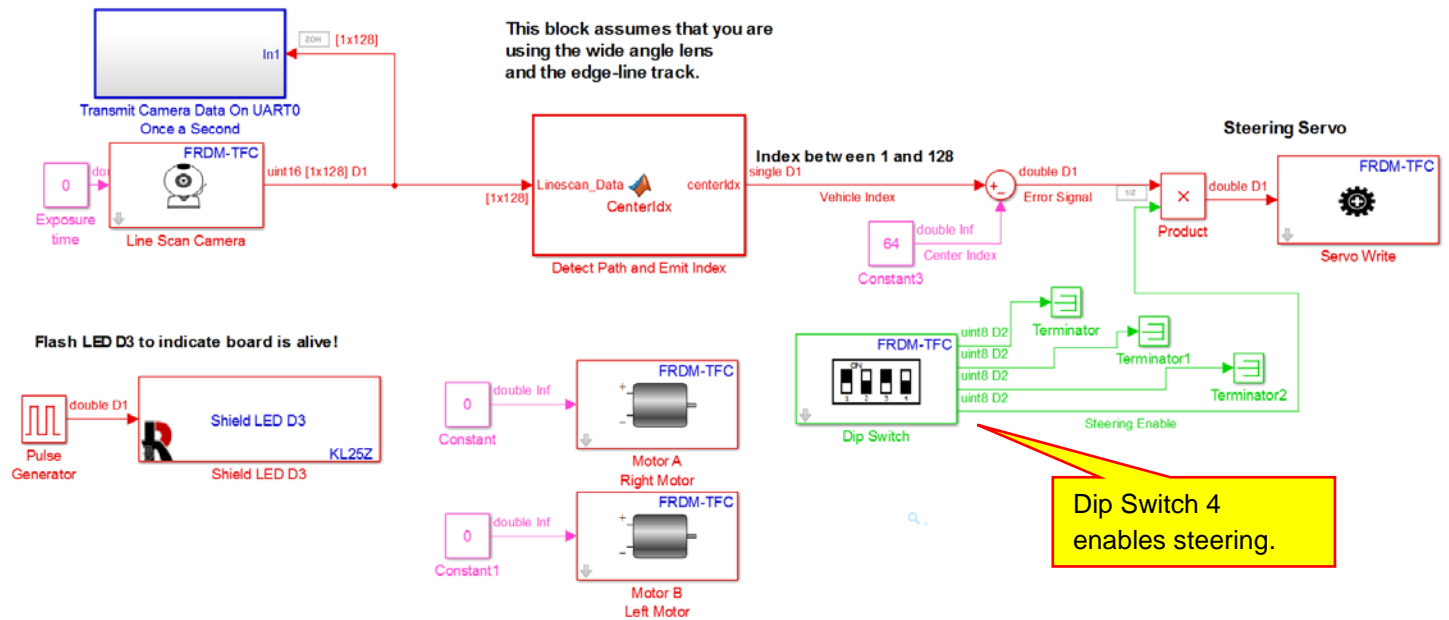


Note that the Error Signal above goes directly to the Steering Servo motor and controls vehicle steering. We have defined the center of the track as index 64. If the Vehicle Index is 64, then the Error Signal is zero. A zero signal to the servo motor causes the steering to go straight. Thus if the vehicle index is 64, the vehicle drives straight. If the Vehicle Index is greater than 64, the Error Signal is positive and the steering will turn the wheels, which way, we do not know. If the Vehicle Index is less than 64, the Error Signal is negative and the steering will turn the wheels the other way. This is the basic operation of the vehicle steering.

We have to answer two questions for good steering.

- 1) The steering should turn the vehicle such that it keeps the vehicle in the center of the track. Do the wheels turn the correct way to do this? A positive signal to the servo motors turns the wheels one way, and a negative signal turns the wheels the other way. The question is, do they turn the wheels the correct way to center the vehicle.
- 2) Assuming that the steering turns the vehicle in the correct direction, do the wheels turn the vehicle enough?

We will answer the first question now. The complete model is shown below:



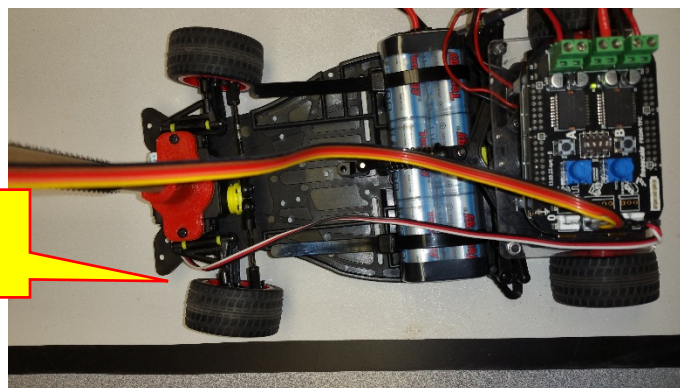
We notice two changes to the model. First, the drive motors are set to zero. We do not want the vehicle to move under its own power until we have the steering working well. Second, we are using the **Dip Switch** block to enable the steering. When Dip Switch 4 is zero, the signal to the servo motor is zero and the wheels will point straight. When we transport the vehicle and do not want the steering to bounce all over the place, we set Dip Switch 4 to zero. When Dip Switch 4 is a 1, the signal to the Servo Motor is equal to the Error Signal, and the Error Signal will control the steering. Thus, Dip Switch 4 is used to enable the steering. B

We can now test the steering. Build and download this model. Place the vehicle on a **straight** portion of the track, turn on the battery power. Adjust your camera to the best position as determined in Lesson XII. Verify the following operations.

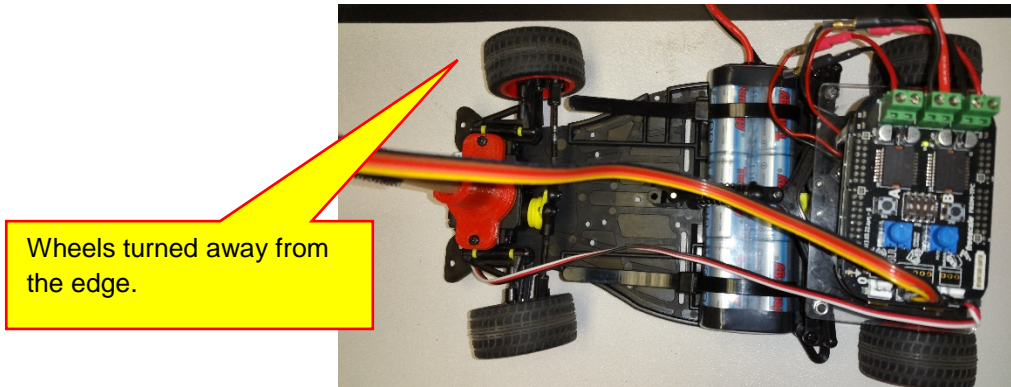
Demo XIII-1: With the Dip Switch 4 set to zero, verify that the front wheels do not move as the vehicle is moved from one side of the track to the other. The steering wheels should always point straight.

Demo XIII-2: With the Dip Switch 4 set to one, verify that the front wheels turn as the vehicle is moved from one side of the track to the other.

Demo XIII-3: With the Dip Switch 4 set to one, verify that the front wheels turn slightly to the right when the vehicle is close to the left edge of the track. Notice that the wheels turn away from the edge to which they are closest.

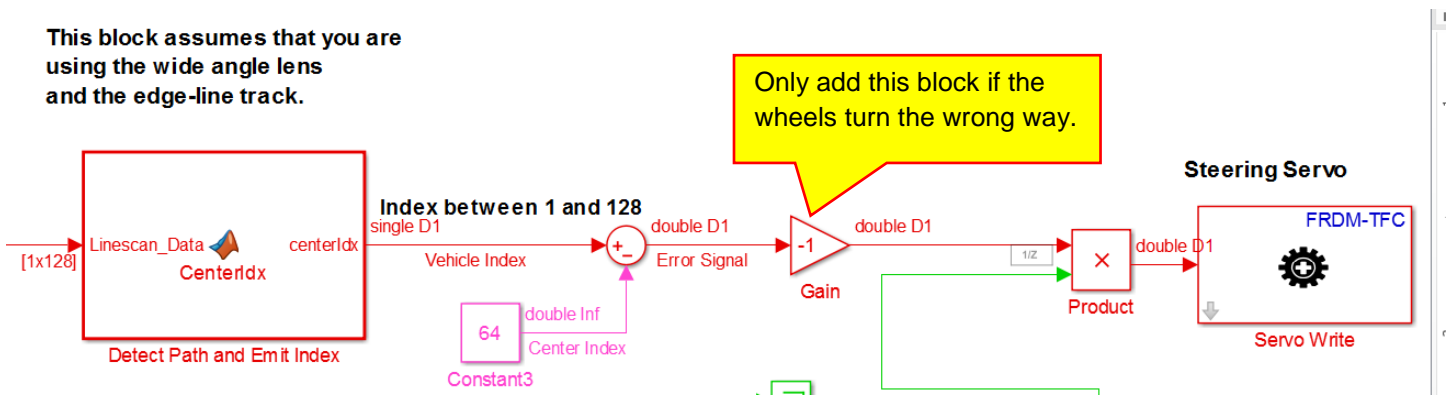


Demo XIII-4: With the Dip Switch 4 set to one, verify that the front wheels turn slightly to the left when the vehicle is close to the right edge of the track. Notice that the wheels turn away from the edge to which they are closest.



Disable the vehicle steering by setting Dip Switch 4 to zero.

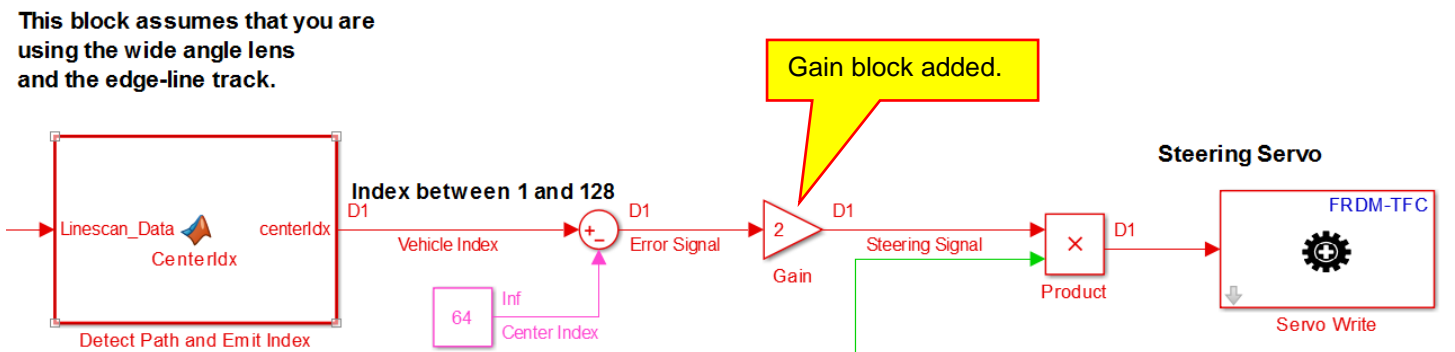
If your wheels turn the correct way, go on to the next section. If your wheels turn the wrong way, we need to add the **Gain** block shown below to correct the steering:



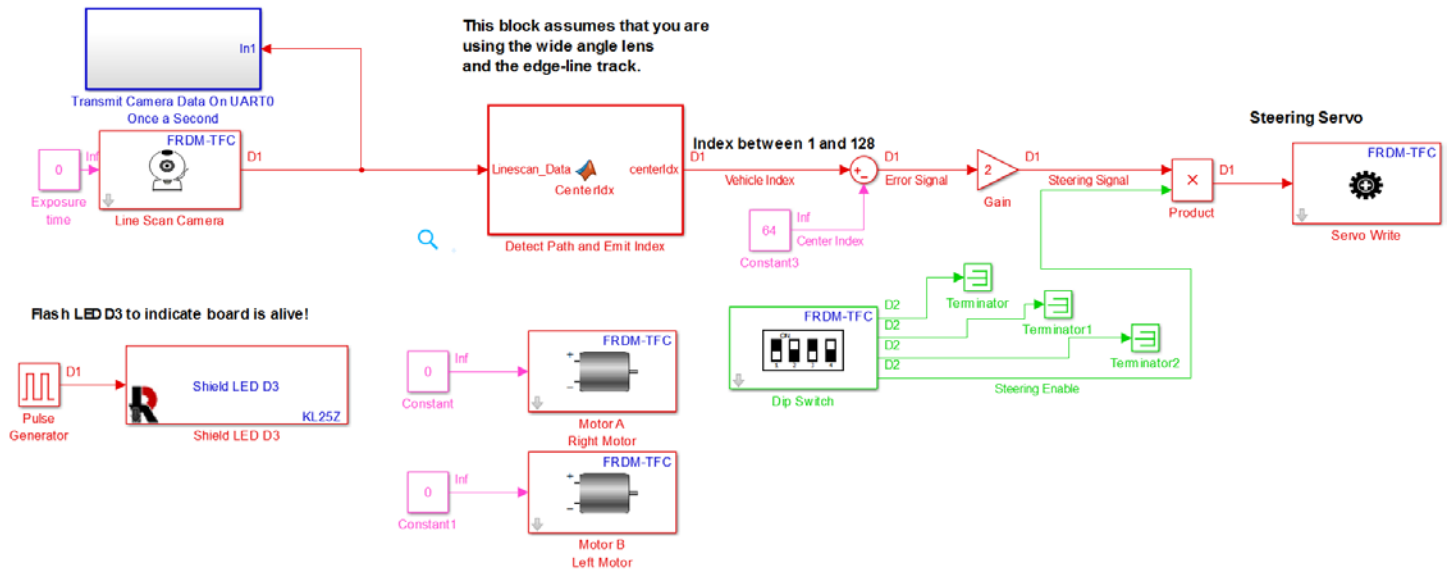
If you added the **Gain** block above, repeat Demo XIII-1 through Demo XIII-4 above to verify the steering.

B. Fixed Gain Steering Amplifier

In the previous section, we were able to make the front wheels turn in the correct direction to move the vehicle toward the center of the track. What you probably noticed is that the wheels do turn, but not enough. We can fix this by adding gain to the system to make the Error Signal larger. This is shown below:



We see that we multiply the Error Signal by 2. Thus the signal passed to the Servo motor is now twice what it was before. This means for the same error, the front wheels turn twice as much. (Note that if your wheels turned the wrong way in the previous section, this gain should be -2.) The rest of the model is the same:



Repeat the previous testing we did on a straight track with the updated model. The testing is repeated below.

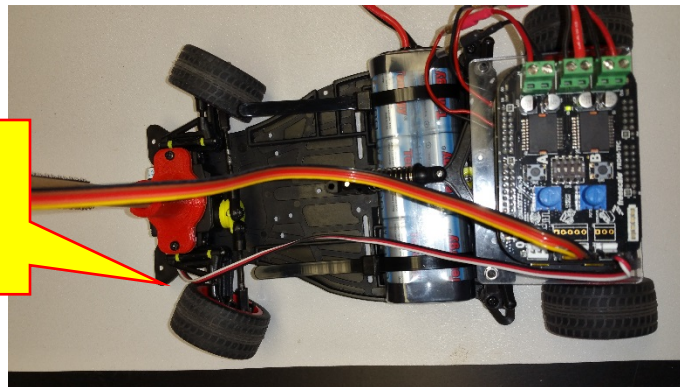
Build and download this updated model. Place the vehicle on a **straight** portion of the track, turn on the battery power. Adjust your camera to the best position as determined in Lesson XII. Verify the following operations.

Demo XIII-5: With the Dip Switch 4 set to zero, verify that the front wheels do not move as the vehicle is moved from one side of the track to the other. The steering wheels should always point straight.

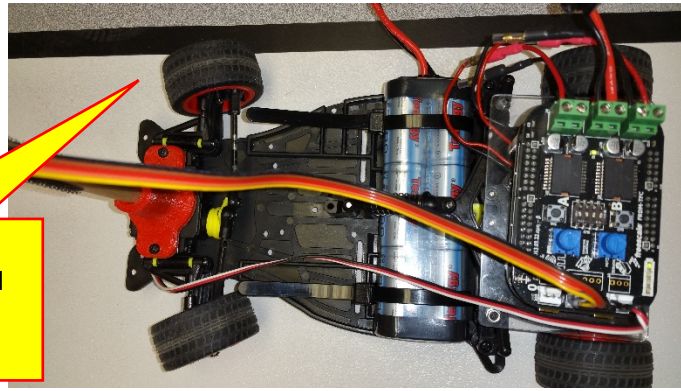
Demo XIII-6: With the Dip Switch 4 set to one, verify that the front wheels turn as the vehicle is moved from one side of the track to the other. Note that they move more than in the previous set of tests.

Demo XIII-7: With the Dip Switch 4 set to one, verify that the front wheels turn to the right when the vehicle is close to the left edge of the track. Notice that the wheels turn away from the edge to which they are closest, and they turn a lot more than in the previous set of tests.

Wheels turned away from the edge. They are turned much more than in the previous set of tests.



Demo XIII-8: With the Dip Switch 4 set to one, verify that the front wheels turn to the left when the vehicle is close to the right edge of the track. Notice that the wheels turn away from the edge to which they are closest.

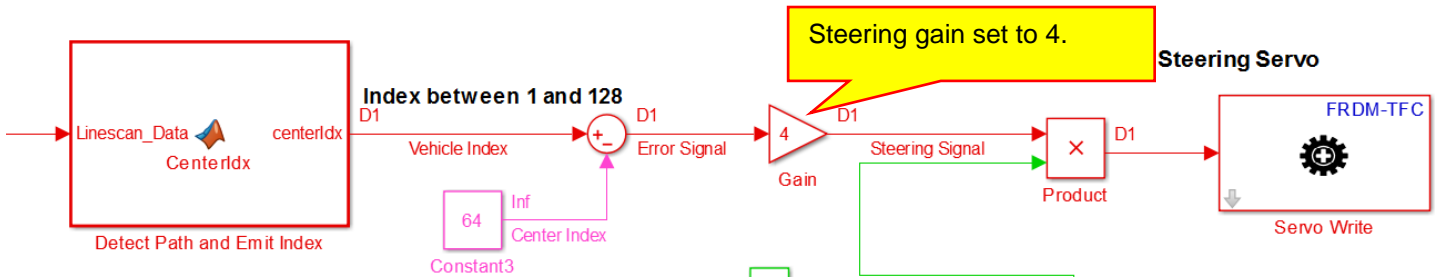


Wheels turned away from the edge. They are turned much more than in the previous set of tests.

Disable the vehicle steering by setting Dip Switch 4 to zero.

Repeat the same set of tests with the steering gain set to 4:

This block assumes that you are using the wide angle lens and the edge-line track.

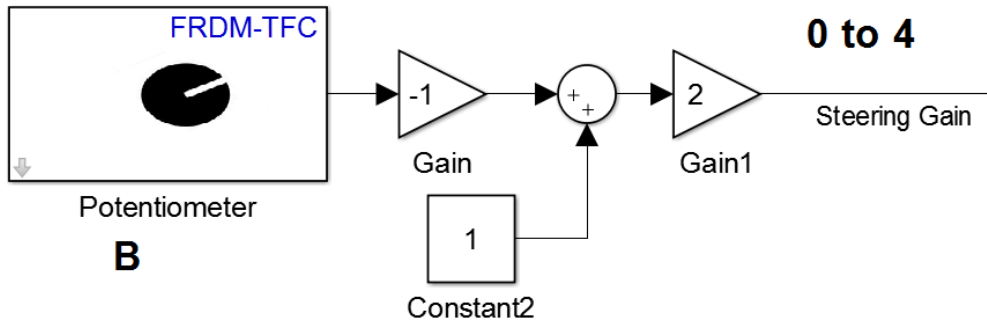


For straight sections, this gain may seem a little high. However, we may need this high gain for high vehicle speeds and sharp turns. Repeat the previous testing for the steering gain of 4.

C. Adjustable Gain Steering Amplifier

As we have seen from the previous section, the steering gain determines how sharply the front wheels turn for a given error. The error being how far away we are from the center of the track. Although you may think that you have a good idea of how to set the gain, it turns out that we will need to change it frequently because the steering performance of the vehicle will depend on vehicle speed, camera height, camera angle, and steering gain. We studied the effect of each of these parameters individually and with a stationary vehicle. We may choose to change these parameters once the vehicle starts moving. It turns out that the steering performance will be based on vehicle speed and our choices of camera angle, camera height, and steering gain. This is why we studied these parameters individually. It is much easier to study each parameter separately and see its effect. Once the vehicle starts moving, all of these parameters may need to be adjusted. Our previous experience gives us a small idea of how to adjust them.

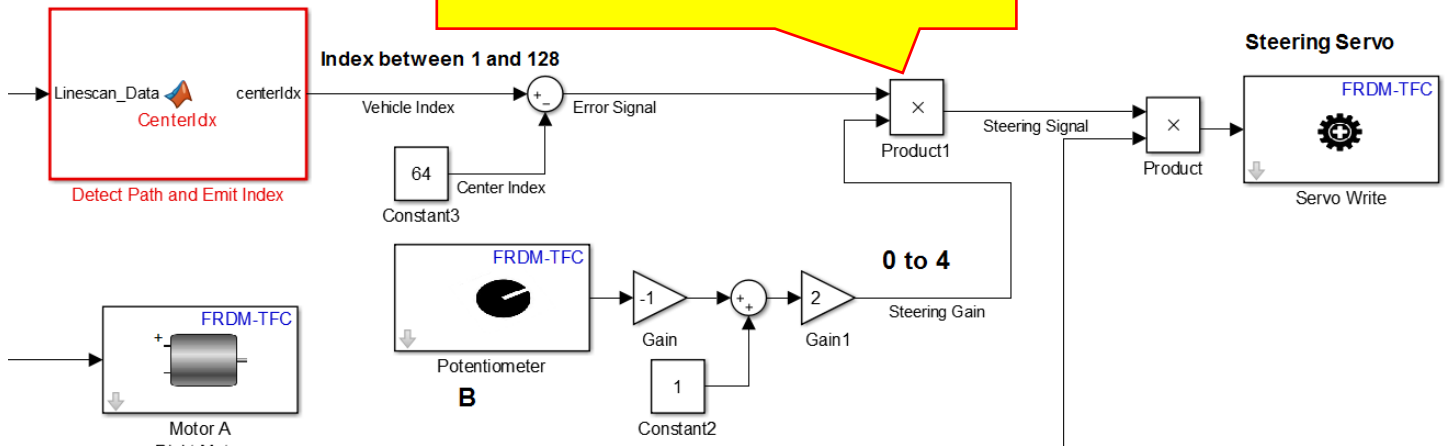
Because the steering gain has such a large effect on the steering performance, we would like to be able to easily change it. Instead of changing the value of the gain block and then rebuilding the model, we will use potentiometer B to change the gain. The set of blocks below output a signal from 0 to 4 as Potentiometer B is rotated clockwise:



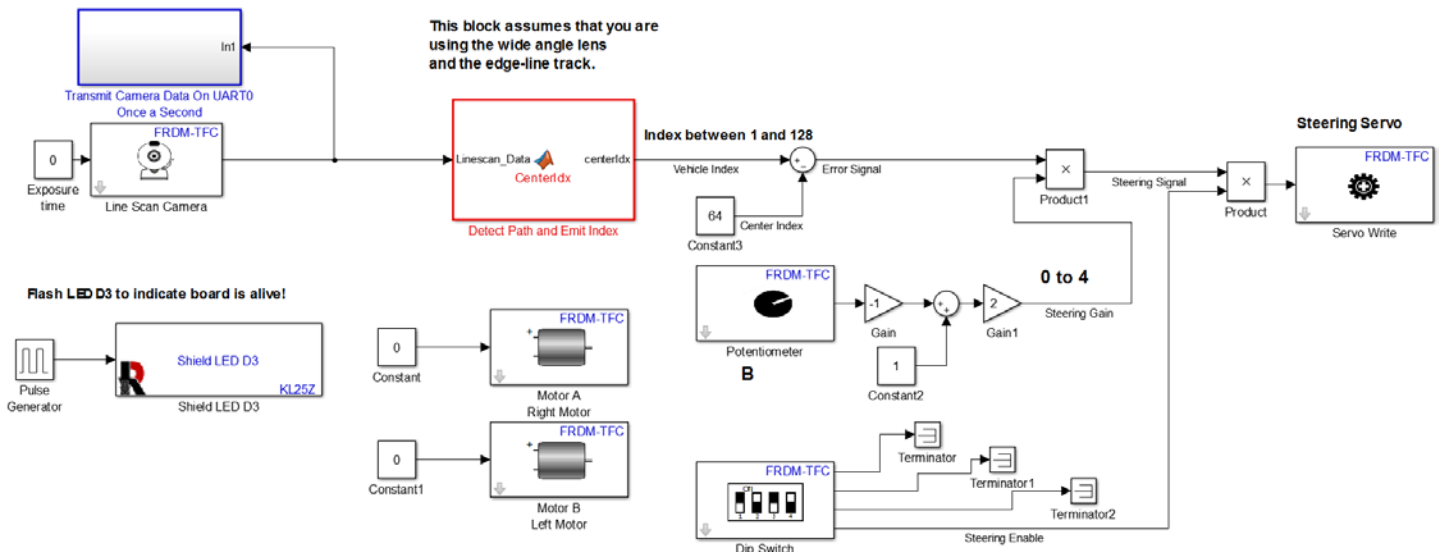
We studied a similar set of blocks in Section III.B on page 41. We will use this set of blocks instead of the fixed gain block of the previous section. This signal is multiplied by the error signal using a **Product** block:

This block assumes that you are using the wide angle lens and the edge-line track.

This is now a product block. It used to be a gain block with a fixed value.



The complete model is shown below. (The remainder of the model is unchanged.)



We now need to test our changes. Build and download this updated model. Place the vehicle on a **straight** portion of the track, turn on the battery power. Adjust your camera to the best position as determined in Lesson XII. Verify the following operations.

Demo XIII-9: With the Dip Switch 4 set to zero, verify that the front wheels do not move as the vehicle is moved from one side of the track to the other. The steering wheels should always point straight.

Demo XIII-10: With the Dip Switch 4 set to one and potentiometer B turned all the way counter clockwise, verify that the front wheels **do not turn** as the vehicle is moved from one side of the track to the other.

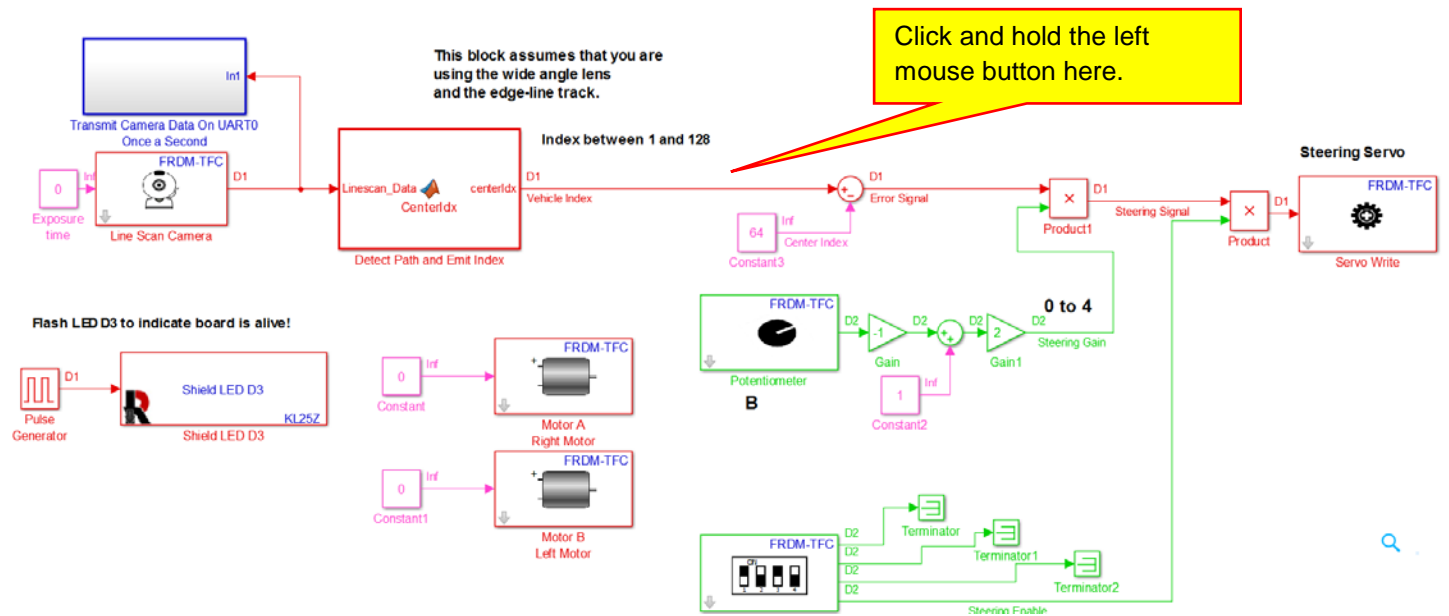
Demo XIII-11: With the Dip Switch 4 set to one and potentiometer B turned 25% clockwise, verify that the front wheels turn as the vehicle is moved from one side of the track to the other.

Demo XIII-12: With the Dip Switch 4 set to one and potentiometer B turned 50% clockwise, verify that the front wheels turn as the vehicle is moved from one side of the track to the other. Note that they move more than in the previous test.

Demo XIII-13: With the Dip Switch 4 set to one and potentiometer B turned fully clockwise, verify that the front wheels turn as the vehicle is moved from one side of the track to the other. Note that they move more than in the previous test.

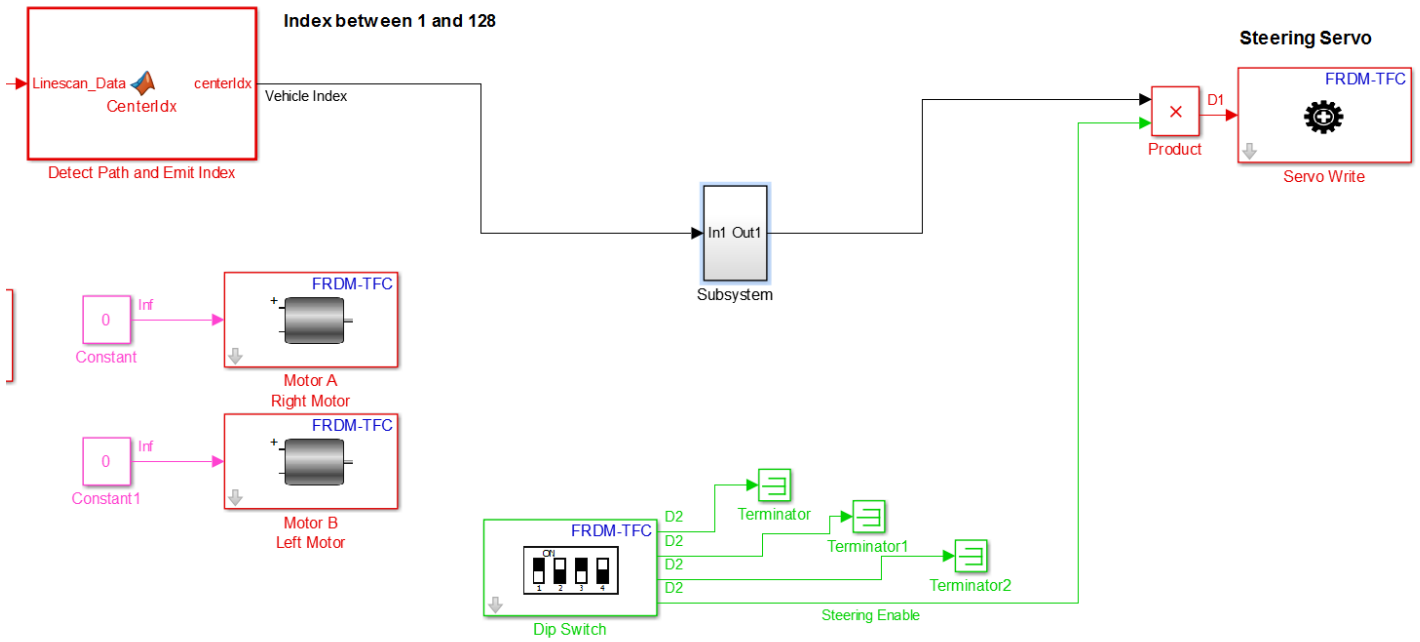
D. Subsystems

The model we have built is starting to get rather large and complex. To make the system more readable, we can group blocks that implement a specific function into a subsystem. This will make the system easier to understand and also group blocks together that perform a specific function. To illustrate this, we will place all of the blocks used for steering into a subsystem. First, move the blocks around so that we have a little more space around the blocks used for the steering. This is done to make it easier to select the steering blocks:

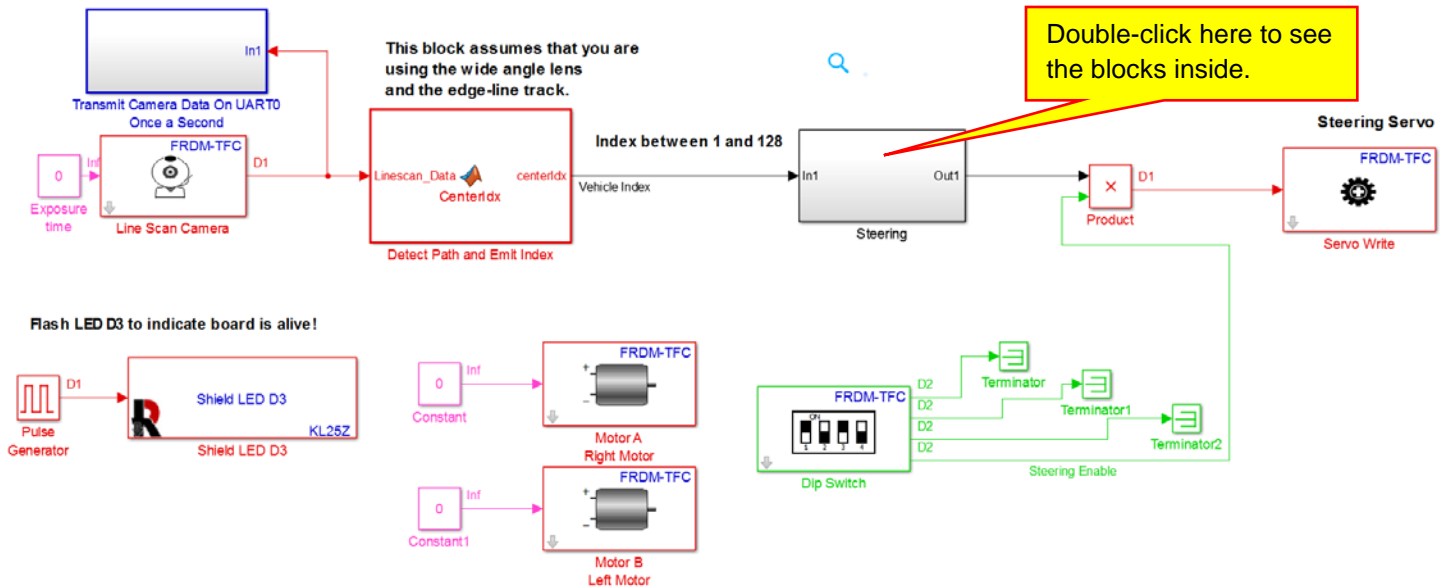


Next, click and hold the left mouse button as shown above. As you move the mouse, a highlighting rectangle will appear. Drag the rectangle so that it highlights all of the blocks in our steering system as shown:

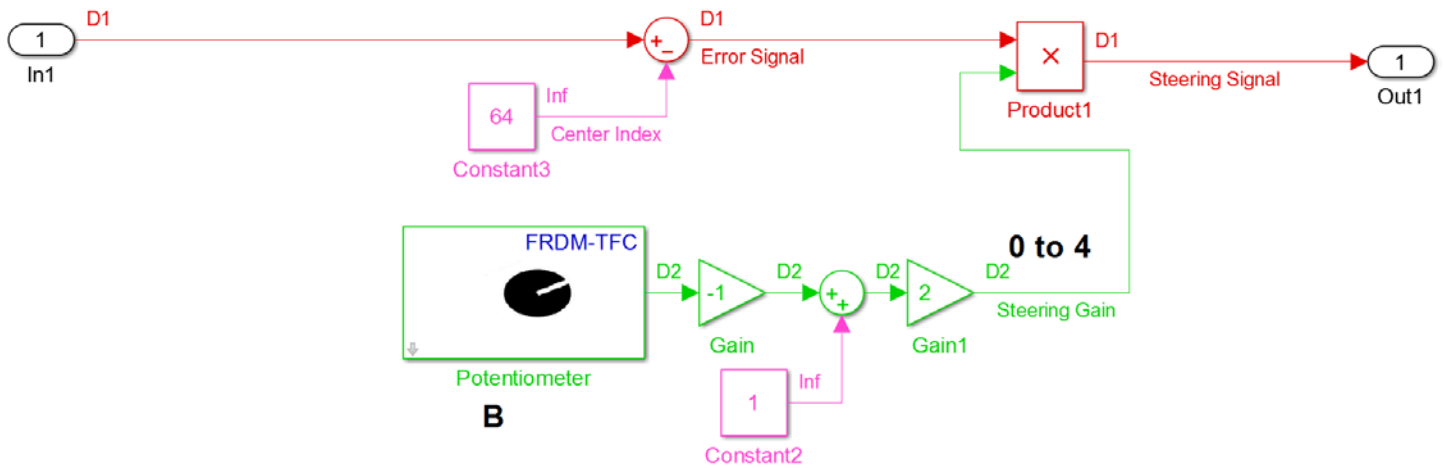
This block assumes that you are using the wide angle lens and the edge-line track.



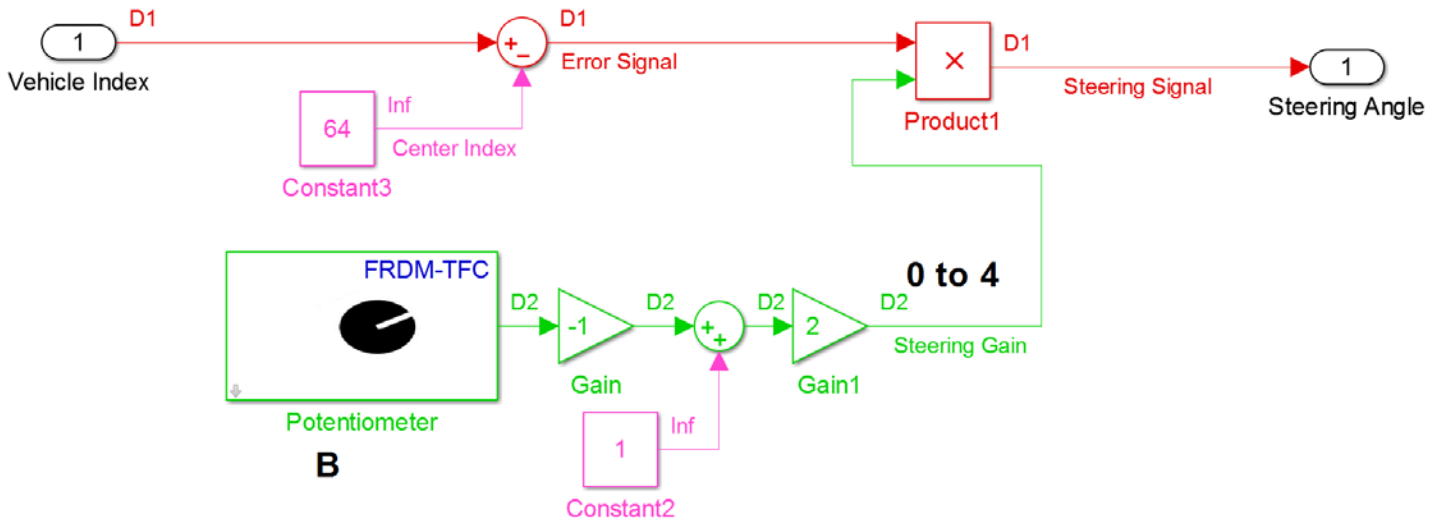
Rename the subsystem as **Steering**, resize the block, and then rearrange your block diagram to look nice:




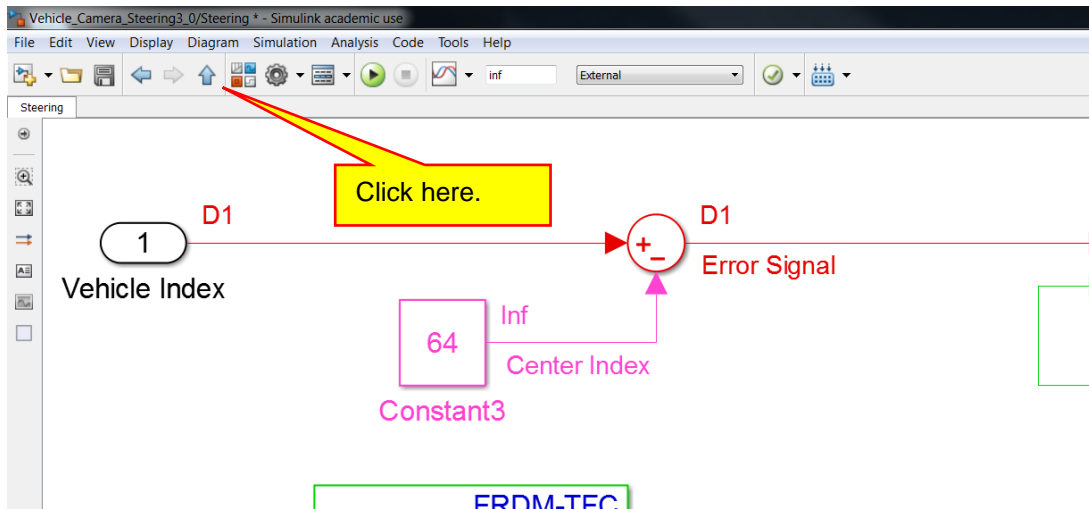
Double-click on the **Steering** subsystem to see the blocks inside. It should look very familiar:



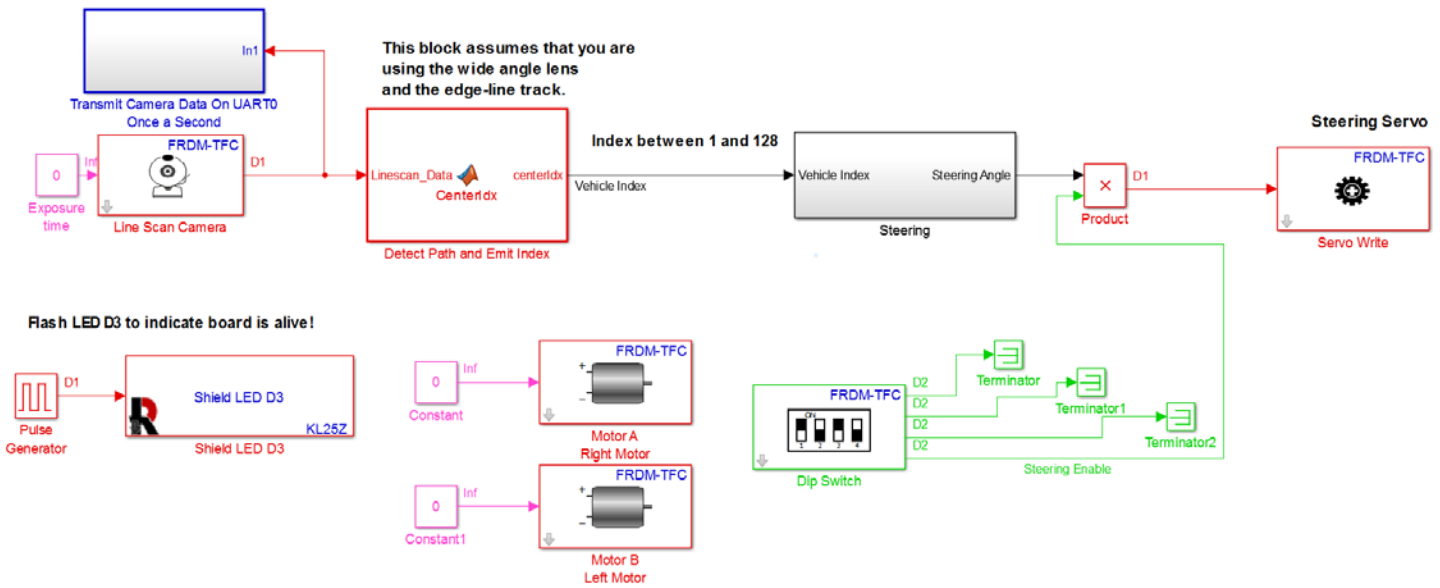
Notice that we have an **In1** port and an **Out1** port. We can rename these to be more meaningful. Rename the In1 port to **Vehicle Index** and rename the Out1 port to **Steering Angle**:



Click on the up arrow  as shown below to see the top level model:



The top level model is now much cleaner. You can change the size of the **Steering** subsystem so that the port labels can be easily seen:



Build and download this model to your vehicle. Perform the same testing as you did in the previous section to verify that the model still works.

E. Push Testing of the Vehicle

We now have a vehicle where we can adjust the steering gain, but the vehicle does not move. This is ideal for testing the basic operation of the steering. If the steering fails, the vehicle will not speed away out of control. What we will do is place the vehicle on the oval track, enable the steering, adjust the steering gain, and push the vehicle around the track. If the vehicle cannot steer as we push it, it will not steer when the vehicle motors are enable. If the vehicle steers as we push it, we have confidence that the vehicle will steer when we use the drive motors. Layout the oval track. Place your car on the track, turn on the power, enable the steering, increase the steering gain and push the vehicle around the track.

Demo XIII-14: Verify that the vehicle can navigate around the track **clockwise** as you push it. When adjusted properly, the vehicle should steer by itself and you should not have to manually reposition the vehicle in order for it to stay on the track. All you should need to do is push the vehicle. You will need to adjust the steering gain in order for the vehicle to steer properly and stay on the track. You may also find that you need to change the camera angle and camera height to make the steering perform properly.

Demo XIII-15: Verify that the vehicle can navigate around the track **counter** clockwise as you push it. When adjusted properly, the vehicle should steer by itself and you should not have to manually reposition the vehicle in order for it to stay on the track. All you should need to do is push the vehicle. You will need to adjust the steering gain in order for the vehicle to steer properly and stay on the track. You may also find that you need to change the camera angle and camera height to make the steering perform properly.

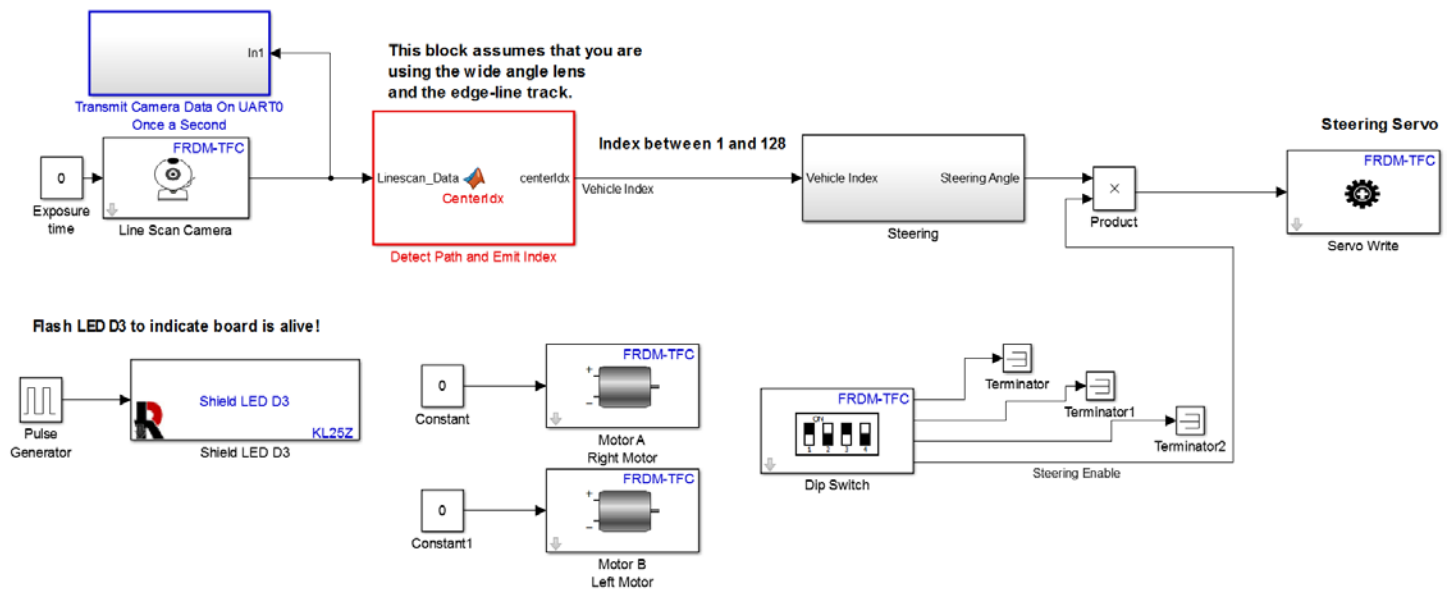
Demo XIII-16: The track is 24 inches wide. Determine values for the steering gain, camera height, and camera angle such that as you push the vehicle around the track, the vehicle never comes closer than 6 inches to either edge of the oval track.

Lesson XIV: Motoring

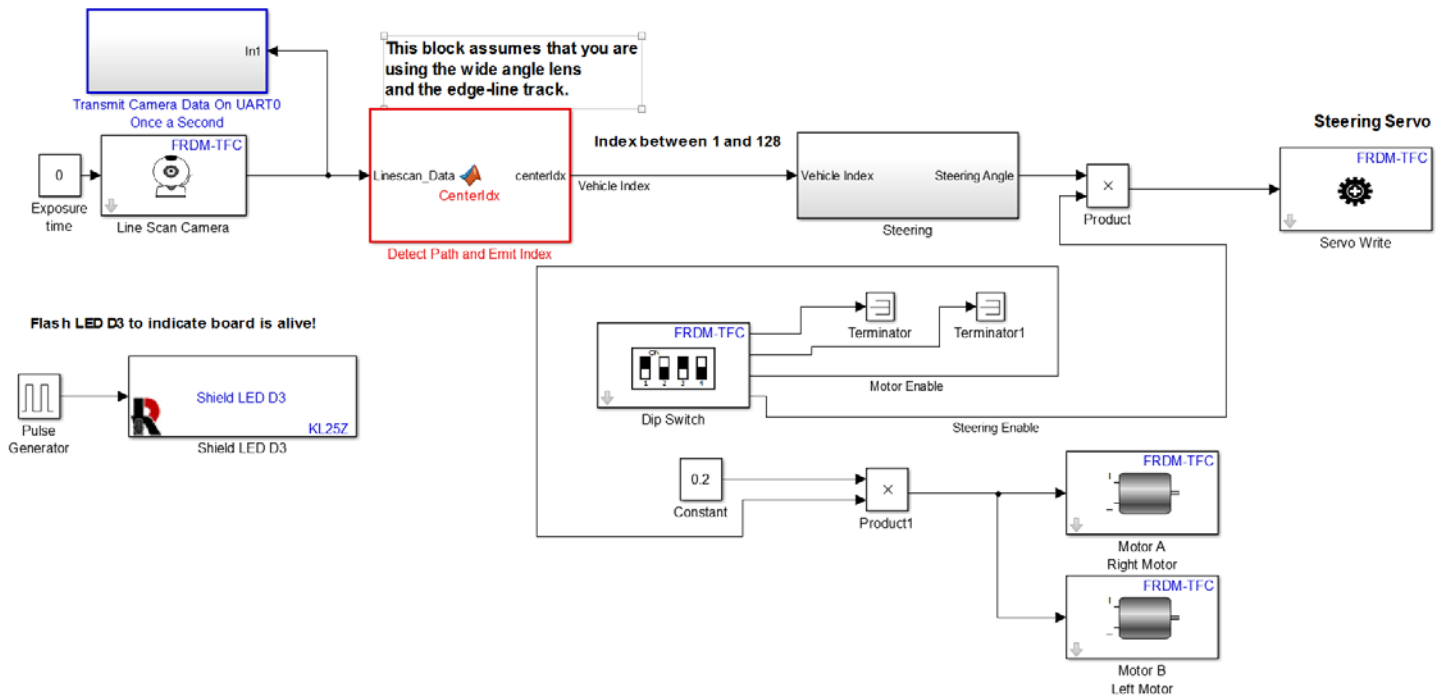
Now that we have worked out basic steering and our vehicle can navigate the track when pushed, it's time to take the leap and let the vehicle move under its own power. This will be a larger step that you may think. We will find that the vehicle speed, camera angle, steering gain, and lighting will play a role in how well your vehicle navigates the track. You might even find that your presence near the vehicle makes a difference! (This is because if you stay too close to the track, your shadow might affect the lighting and cause the camera to behave differently.) This complexity means that we need to come up with a systematic way of testing our vehicle to determine the effect of all of these control parameters (speed, steering gain, camera angle, etc...)

A. Constant Speed Motoring and Motor Enable

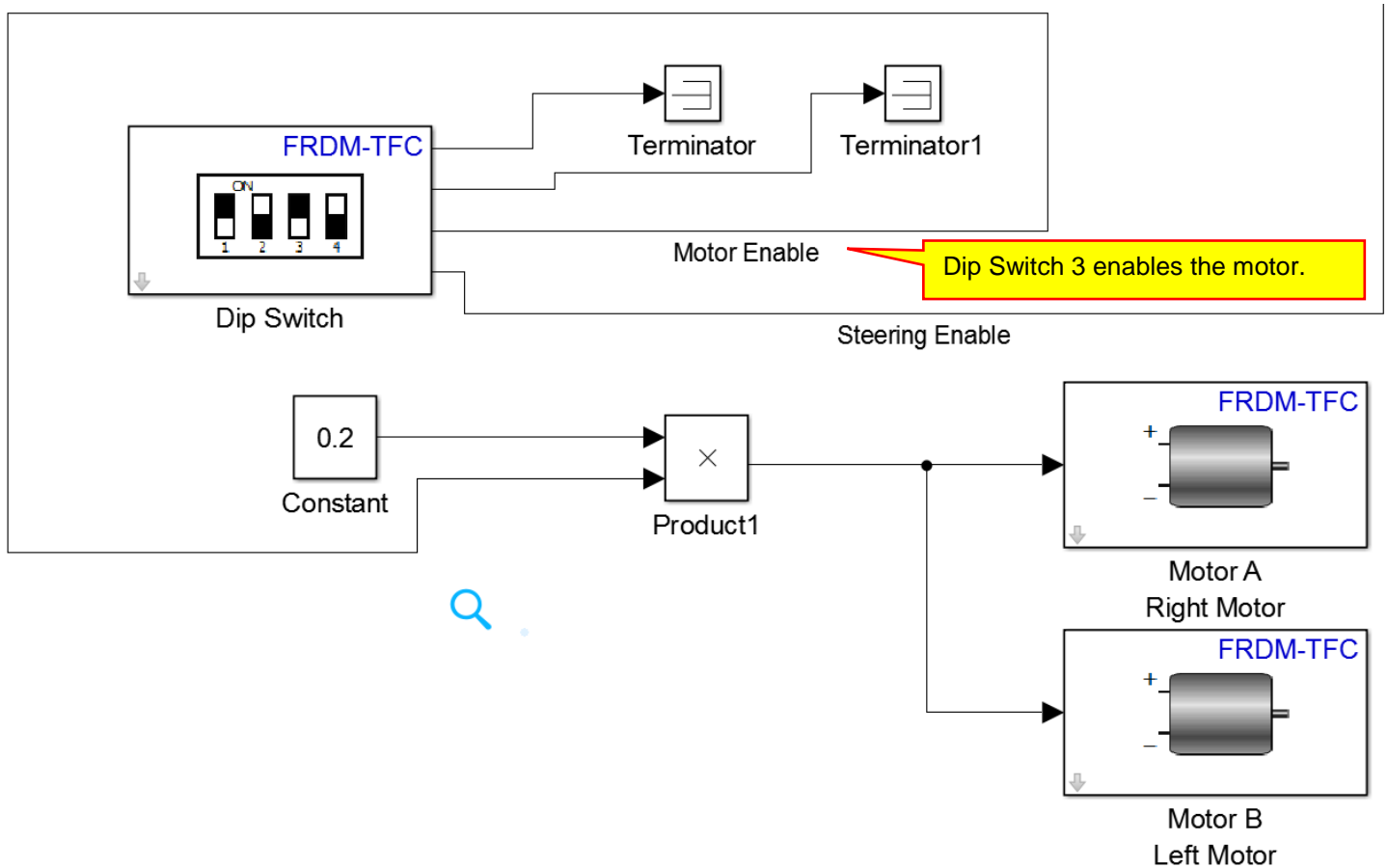
In order to make the speed of the vehicle a known parameter, we will set the motor control signal to a constant. To change the speed, you will need to change the constant and then rebuild the vehicle. With this constant speed, we can perform a number of tests to see the effects of various parameters such as the steering gain and camera angle. As we did with the steering, we will use one of the Dip switches to enable the motors. Both motors will receive the same motor signal. We will start with the model shown on page 176, which is repeated below:



Modify the motor section of the model as shown:



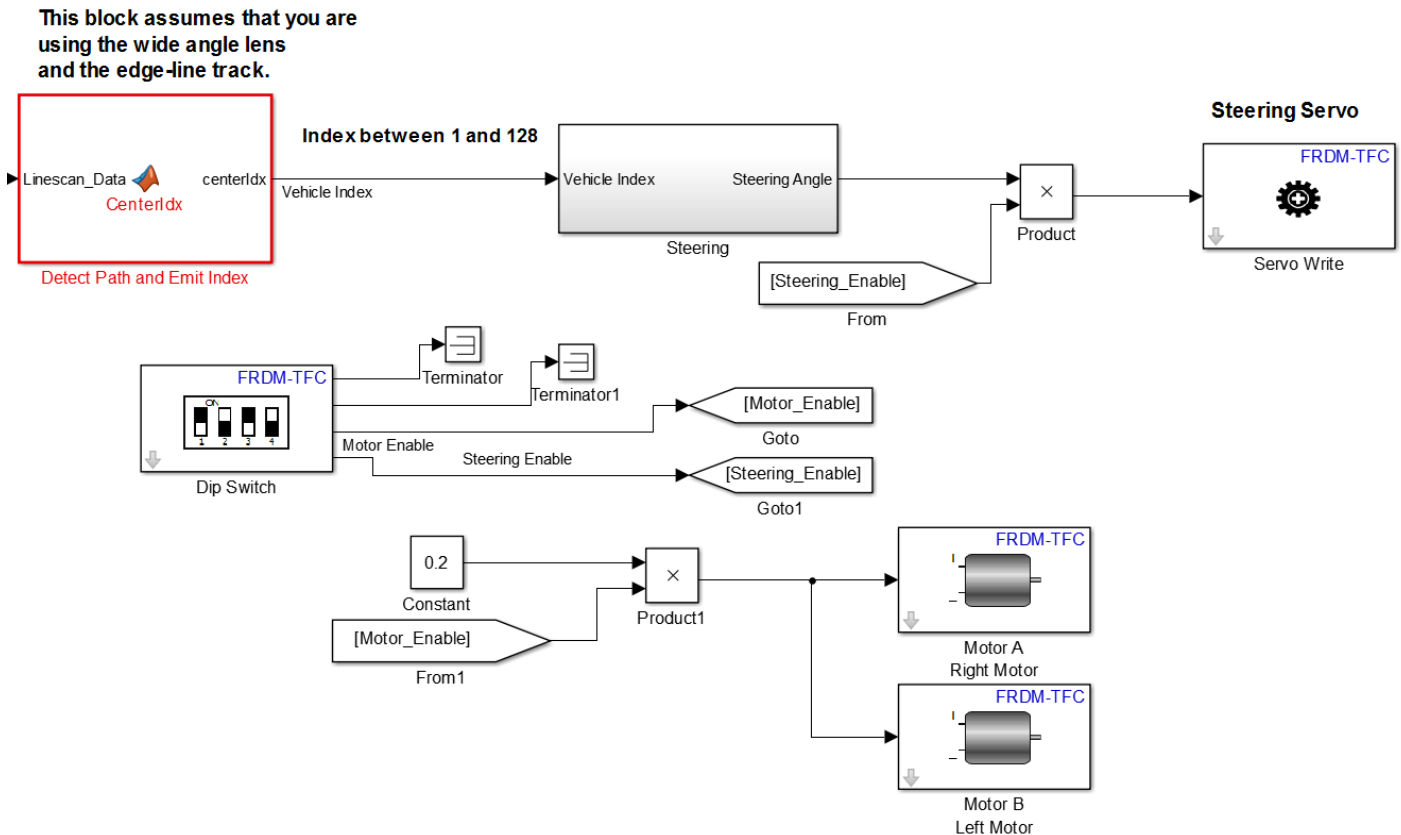
Zooming in on the motor section that we modified, we have:



Dip Switch 3 is used to enable the motors in a similar manner to what we used to enable the servo motor. When Dip Switch 3 is a zero, the signal to the motors is zero. When Dip Switch 3 is a 1, the signal to the

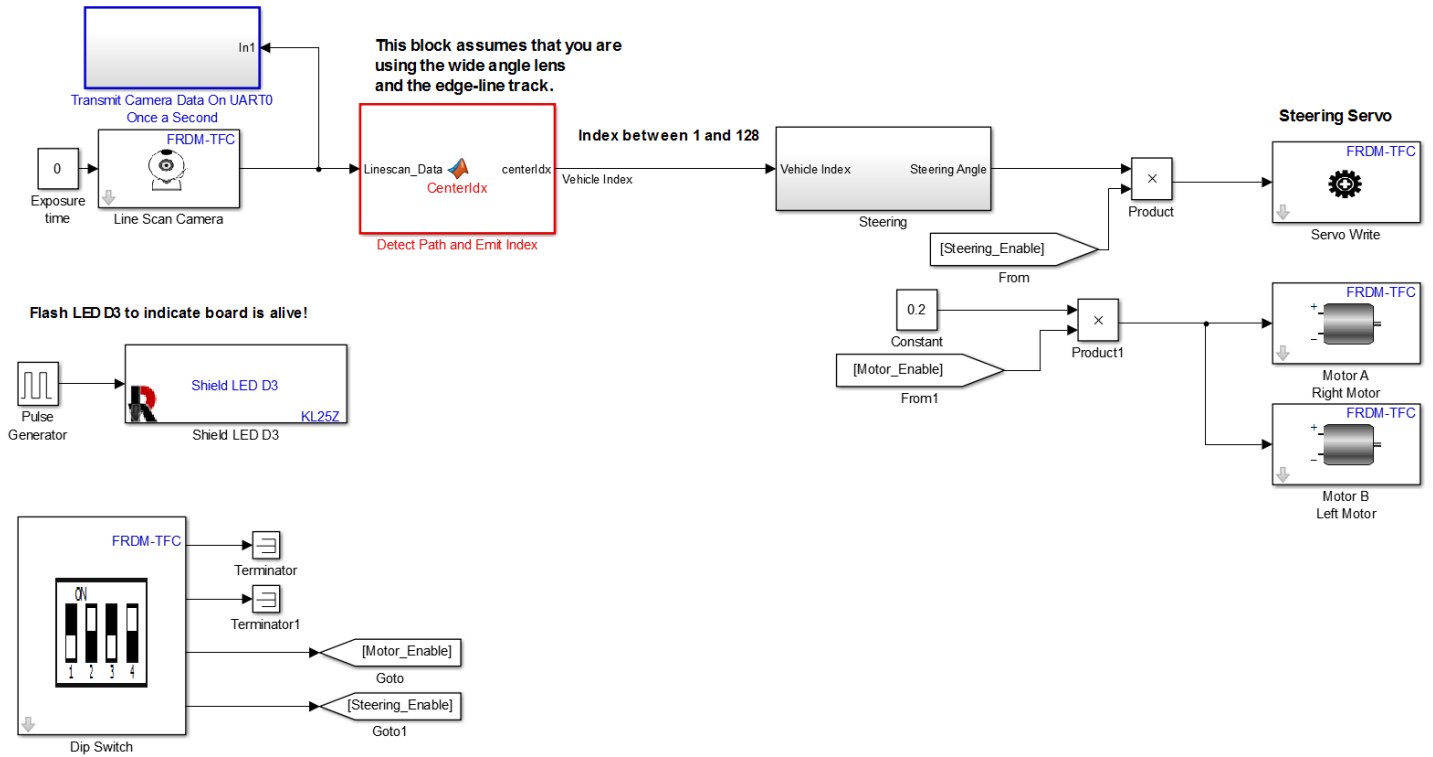
motors is equal to the value of the constant, in this case 0.2. Note that the motors require a signal of -1 to +1. Thus a constant signal of 0.2 represents using the motors at 20% full power in the forward direction.

The above wiring is a bit messy with the enable signals running all over the place. To fix this, we can use blocks called **Goto** and **From** tags. These blocks are located in the **Simulink / Signal Routing** library. The block diagram below is equivalent to the one we made earlier, just a bit cleaner. Note that **Goto** and **From** blocks with the same label are connected together:



Note that the labels for the **Goto** and **From** blocks are referred to as tags. Tags cannot have spaces. Thus, we use underscores in the tag names, “Steering_Enable” and “Motor_Enable”

Now that we have the tags, we can rearrange the model so it looks a little more organized:



You do not need to use the **Goto** and **From** blocks, but for complex models, it does tend to make them more readable.

We will now use the following procedure to build the model. In the next section, we will change the constant for motor speed. Every time you change this constant and need to rebuild the model, follow this procedure.

- 1) Set the Steering Enable Dip Switch to 0.
- 2) Set the Motor Enable Dip Switch to 0.
- 3) Turn off battery power using the pushbutton power switch.
- 4) Build the model.

You must turn off battery power when you build a model. There is a bug in the TFC shield that powers the motors full on while you program the KL25Z. Thus, if you do not turn off the battery power while building a model, when the model is downloaded to the KL25Z, your vehicle will be fully powered and speed away for a few seconds. Since you have the USB cable plugged in, this will may damage the connector and destroy your KL25Z microcontroller.

B. Controller Tuning

With this basic self-powered vehicle, we now wish to do some testing. Our main goal is to tune the vehicle parameters so that it stays on the track while moving as fast as possible. This is not as easy as it may sound as both of these performance metrics (staying on the track and moving fast) are affected by the steering gain and camera angle. We need to come up with a process that allows us to understand the effect of these parameters and also not destroy our vehicle. Note that we can break these cars! Too much speed combined with a vehicle going off course may result in braking parts of the vehicle. It is best to come up with a testing process that allows to understand the effects of each parameter while reducing risk of damage to the vehicle.

We will use the following steps to start a test. We will assume that the battery power is off.

- 1) Place the vehicle on the track. Always place it in the same starting point with the same orientation. The starting position should be repeatable.
- 2) Adjust any parameters that you intend to vary, such as camera angle or steering gain.
- 3) Turn on battery power using the pushbutton power switch.
- 4) Set the Steering Enable Dip Switch to 1.
- 5) Set the Motor Enable Dip Switch to 1.
- 6) Release the vehicle and observe its performance.

1. Testing Procedure

Set the motor speed constant to 20%. (This was done the last time we built the model, so we should not have to change anything.) Set the camera angle to 45 degrees. (This was the minimum camera angle when we did our testing in Section XIII.E.) Using Potentiometer B, set the steering gain fully clockwise. This is maximum steering gain. We will run the following tests on the oval track:

- 1) Does it navigate the track in the clockwise direction around the oval? (Yes or no.) If not, no other tests are necessary. However, you should test this failure a few times.
- 2) During a turn while moving clockwise, how close to the inside of the turn did the vehicle come in inches?
- 3) During a turn while moving clockwise, how close to the outside of the turn did the vehicle come in inches?
- 4) During a straightaway, did the vehicle drive straight or weave back and forth. Weaving back and forth is referred to as an oscillation.
- 5) Time to complete 10 laps clockwise around the oval. Use a stop watch to time your vehicle and manually count the laps.

These same tests need to be performed while the vehicle is navigating the track in the counter-clockwise direction as well.

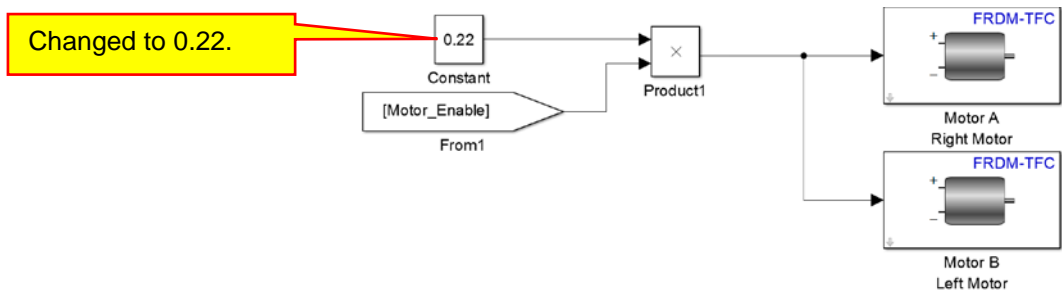
2. Motor Speed Based on Camera Angle

For the first set of tests, we will keep the motor speed at 0.20 and the steering gain at maximum (100% Clockwise) and vary the camera angle. The table has open rows for unspecified camera angles if you want to do more measurements. Note that not all of these tests may be possible because the vehicle speed is too high. If you find that the vehicle speed is too high and the vehicle cannot stay on the track, skip that set of tests! **You do not want to damage your vehicle.**

Model Tuning: Variable Camera Angle						
Motor Speed: 0.20		Steering Gain 100% CW		Vehicle Direction Around Oval: CW		
Steering Angle (%)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
45						
50						
55						

Model Tuning: Variable Camera Angle						
Motor Speed: 0.20		Steering Gain 100% CW		Vehicle Direction Around Oval: CCW		
Steering Angle (%)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
45						
50						
55						

Next, we will increase the vehicle speed and preform the same set of tests. Change the motor speed constant from 0.2 to **0.22** as shown below:



We will perform the same set of tests at this faster vehicle speed. Build and download your model. (Make sure you follow the steps given on page 180 when making changes and building the model. Fill in the tables below with this higher motor speed.

Model Tuning: Variable Camera Angle						
Motor Speed: 0.22		Steering Gain 100% CW		Vehicle Direction Around Oval: CW		
Steering Angle (%)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
45						
50						
55						

Model Tuning: Variable Camera Angle						
Motor Speed: 0.22		Steering Gain 100% CW		Vehicle Direction Around Oval: CCW		
Steering Angle (%)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
45						
50						
55						

Repeat the same set of tests for motor speeds of 0.24, 0.26, 0.28, and 0.30. Note that if the vehicle fails at a lower speed, do not test it at a higher speed.

Model Tuning: Variable Camera Angle						
Motor Speed: 0.24		Steering Gain 100% CW		Vehicle Direction Around Oval: CW		
Steering Angle (%)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
45						
50						
55						

Model Tuning: Variable Camera Angle						
Motor Speed: 0.24		Steering Gain 100% CW		Vehicle Direction Around Oval: CCW		
Steering Angle (%)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
45						
50						
55						

Model Tuning: Variable Camera Angle						
Motor Speed: 0.26		Steering Gain 100% CW		Vehicle Direction Around Oval: CW		
Steering Angle (%)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
45						
50						
55						

Model Tuning: Variable Camera Angle						
Motor Speed: 0.26		Steering Gain 100% CW		Vehicle Direction Around Oval: CCW		
Steering Angle (%)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
45						
50						
55						

Model Tuning: Variable Camera Angle						
Motor Speed: 0.28		Steering Gain 100% CW		Vehicle Direction Around Oval: CW		
Steering Angle (%)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
45						
50						
55						

Model Tuning: Variable Camera Angle						
Motor Speed: 0.28		Steering Gain 100% CW		Vehicle Direction Around Oval: CCW		
Steering Angle (%)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
45						
50						
55						

Model Tuning: Variable Camera Angle						
Motor Speed: 0.30		Steering Gain 100% CW		Vehicle Direction Around Oval: CW		
Steering Angle (%)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
45						
50						
55						

Model Tuning: Variable Camera Angle						
Motor Speed: 0.30		Steering Gain 100% CW		Vehicle Direction Around Oval: CCW		
Steering Angle (%)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
45						
50						
55						

After making all of these measurements, we need to make a few observations which we shall pose as questions:

Question XIV-1: What is the tradeoff between vehicle speed and camera angle?

Question XIV-2: Does changing the camera angle change where the vehicle begins making a turn? Explain your answer.

Question XIV-3: What happens when you make the camera angle too large and you get too close to the outside of a turn?

Question XIV-4: Explain how you might adjust the vehicle speed so that the vehicle turns closer to the outside edge of a turn.

Question XIV-5: Explain how you might adjust the vehicle speed so that the vehicle turns closer to the inside edge of a turn.

Question XIV-6: Explain how you might adjust the camera angle so that the vehicle turns closer to the outside edge of a turn.

Question XIV-7: Explain how you might adjust the camera angle so that the vehicle turns closer to the inside edge of a turn.

3. Motor Speed versus Steering Gain

We now have an idea of how the vehicle speed and camera angle effect the vehicle’s performance. We now want to do the same with vehicle speed and steering gain. Based on your results from the previous section, choose a camera angle that you think yields the best overall performance for all vehicle speeds tested. Use this camera angle for all of these tests.

In this testing procedure, we will set a fixed vehicle speed, the optimal steering angle that we choose, and then see the effects of changing the steering gain. We will then repeat the procedure at different speeds. In this way, we will get a feel for how vehicle speed and steering gain interact. To change the steering gain, we will use potentiometer B. We will use settings of full clockwise (100%CW), 75% of full clockwise (75% CW), 50% of full clockwise, and 25% of full clockwise. As in the previous testing, we will test the vehicle around the oval track in both directions. Make the measurements in the tables provided below:

Model Tuning: Variable Steering Gain						
Motor Speed: 0.20		Camera Angle: (User Optimal: _____)		Vehicle Direction Around Oval: CW		
Steering Gain (% CW)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
100						
75						
50						
25						

Model Tuning: Variable Steering Gain						
Motor Speed: 0.20		Camera Angle: (User Optimal: _____)		Vehicle Direction Around Oval: CCW		
Steering Gain (% CW)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
100						
75						
50						
25						

Model Tuning: Variable Steering Gain						
Motor Speed: 0.22		Camera Angle: (User Optimal: _____)		Vehicle Direction Around Oval: CW		
Steering Gain (% CW)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
100						
75						
50						
25						

Model Tuning: Variable Steering Gain						
Motor Speed: 0.22		Camera Angle: (User Optimal: _____)		Vehicle Direction Around Oval: CCW		
Steering Gain (% CW)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
100						
75						
50						
25						

Model Tuning: Variable Steering Gain						
Motor Speed: 0.24		Camera Angle: (User Optimal: _____)		Vehicle Direction Around Oval: CW		
Steering Gain (% CW)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
100						
75						
50						
25						

Model Tuning: Variable Steering Gain						
Motor Speed: 0.24		Camera Angle: (User Optimal: _____)		Vehicle Direction Around Oval: CCW		
Steering Gain (% CW)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
100						
75						
50						
25						

Model Tuning: Variable Steering Gain						
Motor Speed: 0.26		Camera Angle: (User Optimal: _____)		Vehicle Direction Around Oval: CW		
Steering Gain (% CW)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
100						
75						
50						
25						

Model Tuning: Variable Steering Gain						
Motor Speed: 0.26		Camera Angle: (User Optimal: _____)		Vehicle Direction Around Oval: CCW		
Steering Gain (% CW)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
100						
75						
50						
25						

Model Tuning: Variable Steering Gain						
Motor Speed: 0.28		Camera Angle: (User Optimal: _____)		Vehicle Direction Around Oval: CW		
Steering Gain (% CW)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
100						
75						
50						
25						

Model Tuning: Variable Steering Gain						
Motor Speed: 0.28		Camera Angle: (User Optimal: _____)		Vehicle Direction Around Oval: CCW		
Steering Gain (% CW)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
100						
75						
50						
25						

Model Tuning: Variable Steering Gain						
Motor Speed: 0.30		Camera Angle: (User Optimal: _____)		Vehicle Direction Around Oval: CW		
Steering Gain (% CW)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
100						
75						
50						
25						

Model Tuning: Variable Steering Gain						
Motor Speed: 0.30		Camera Angle: (User Optimal: _____)		Vehicle Direction Around Oval: CCW		
Steering Gain (% CW)	Navigate Track (Y/N)	Distance to Inside Rail (In)	Distance to Outside Rail (In)	Oscillations in Straight Section (Y/N)	Time to Complete 10 Laps (Seconds)	Comments
100						
75						
50						
25						

Again, we need to make a few observations:

Question XIV-8: What is the tradeoff between vehicle speed and steering gain?

Question XIV-9: Does changing the steering gain change where the vehicle begins making a turn? Explain your answer.

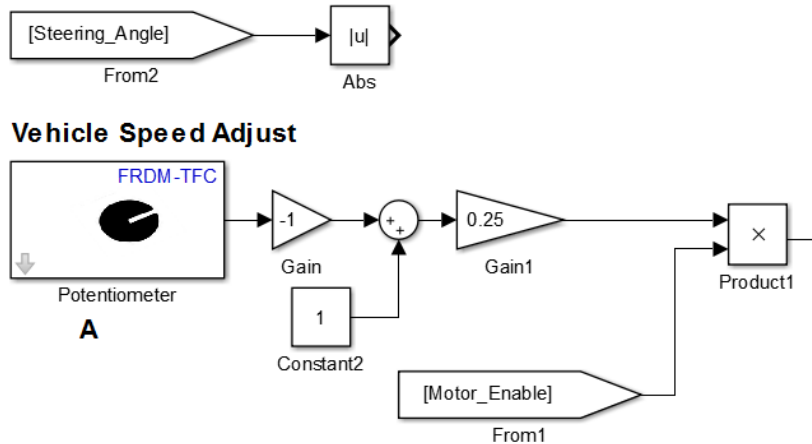
Question XIV-10: What happens when you make the steering gain too small and you get too close to the outside of a turn?

Question XIV-11: At a constant speed, what does changing the steering gain affect?

C. Adjustable Speed Motoring

Now that we have done a lot of fixed speed testing, we have gained an appreciation of how sensitive the vehicle is to changing the speed. A small change can make the vehicle uncontrollable. Also, we have gained some discipline and we won't test the vehicle a full power because we know it won't work, and we know that we could damage the vehicle. We have also learned that we do need to try different vehicle speeds and that changing a constant and building the model takes a lot of time. To reduce this time, we will make the speed variable by using potentiometer A. To limit the speed, we will limit the signal to be 0 to 0.5 because we know that higher speeds will probably cause the car to crash. The model below adds blocks to make the vehicle speed variable with potentiometer A:

Thus, we can take the absolute value of the Steering angle. Note that the **Abs** block is located in the **Simulink / Math Operations** library:



Next, we need to implement our mathematical function:

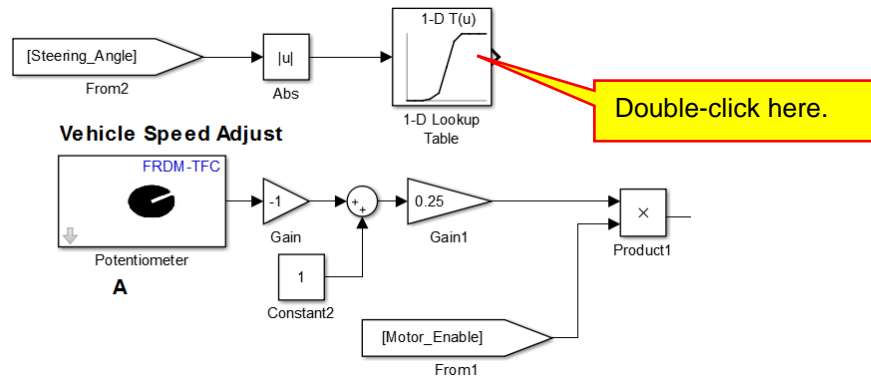
- If the absolute value of the steering angle is between 0 and 10, the speed should be 100%.
- If the absolute value of the steering angle is between 10 and 30, the speed should be reduced linearly from 100% down to 50%.

Linear means a straight line. One way to implement this functionality would be an **IF** statement and the equation of a straight line. An equation that implements the line is:

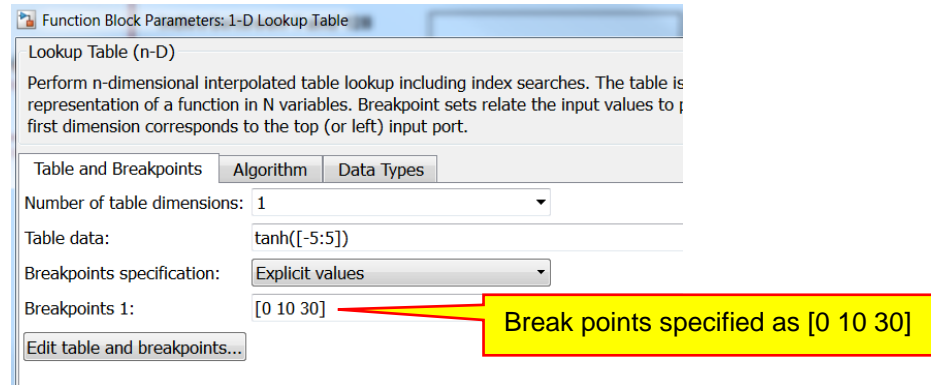
$$y = 1 - 0.5 \frac{(x - 10)}{(30 - 10)}$$

When X is 10, y is 1 (or 100%). When X is 30, y is 0.5 (or 50%). We can implement this functionality with a MATLAB function block:

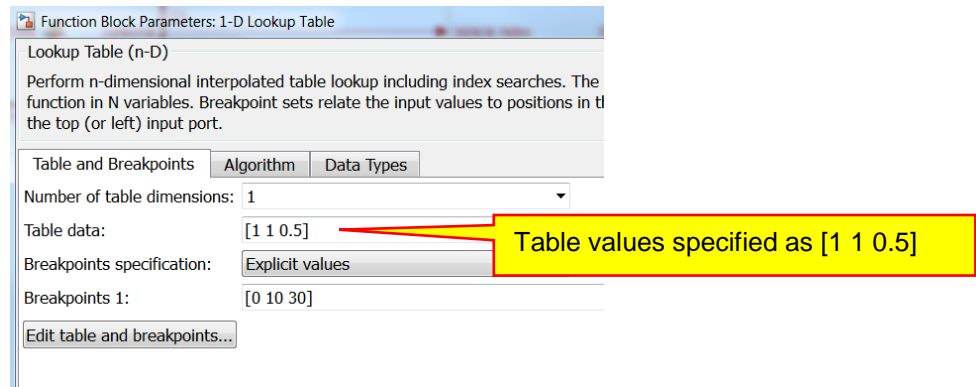
We will not do it this way. (Also note that this function is incorrect if x is greater than 30!) Instead, we will use a lookup table. Lookup tables are basically algebra in a picture. We create a plot of what we want the function to look like and then Simulink creates the math to implement that function. Place a part called **1-D Lookup Table** in your model. This part is located in the **Simulink / Lookup Tables** library:



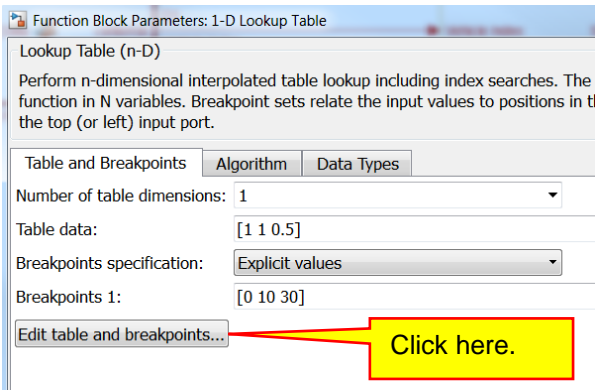
Double-click on the **1-D Lookup Table** block to open it. Change the **Breakpoints** to the vector [0 10 30]:



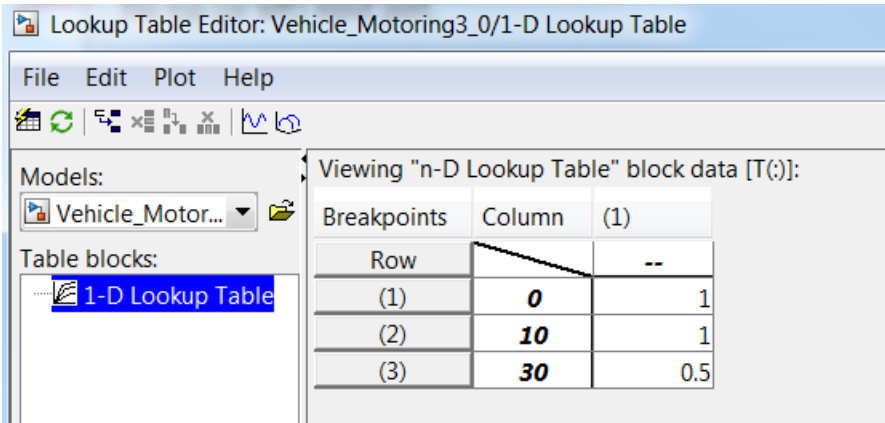
The **Breakpoints** are values of the input where the function changes. Thus, the points specify a steering angle of 0, 10, and 30. Next, specify the **Table data** as [1 1 0.5]




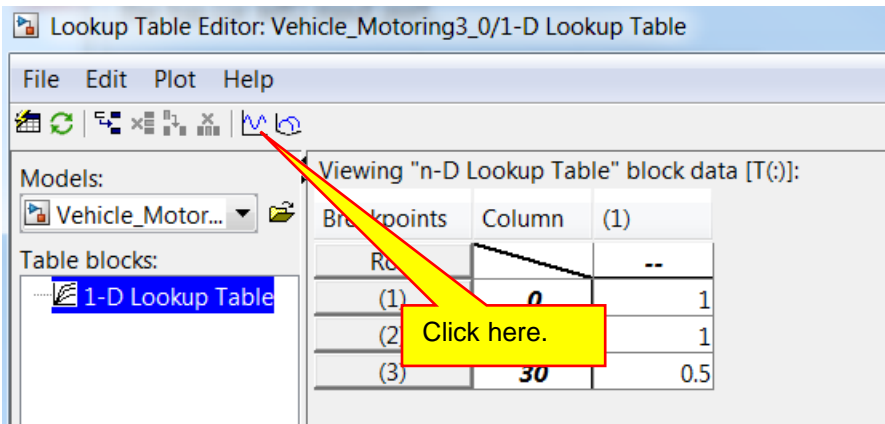
The **Table data** are the output values of the block. What we have specified is the following. Remember that the absolute value of the steering angle is the input to the block. For steering angles of 0 to 10, the output of the block is 1. For steering angles of 10 to 30, the output of the block will vary linearly from 1 down to 0.5, or 100% down to 50%. To see our data, click the **Edit table and breakpoints** button:



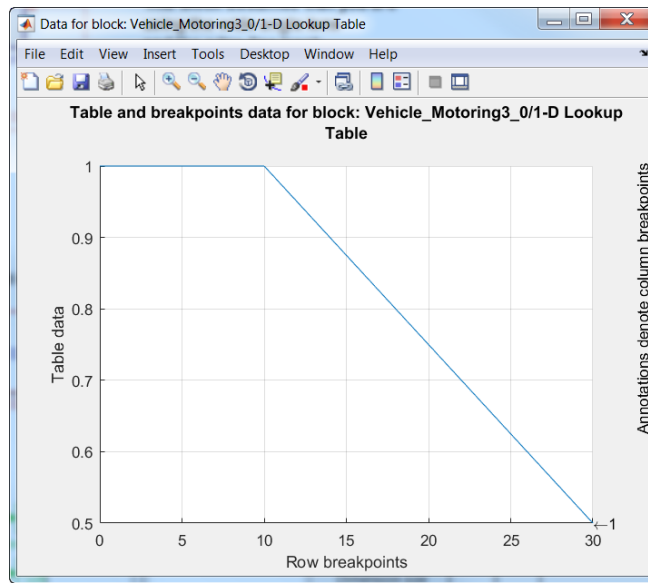
A spreadsheet view of the table data will be shown:



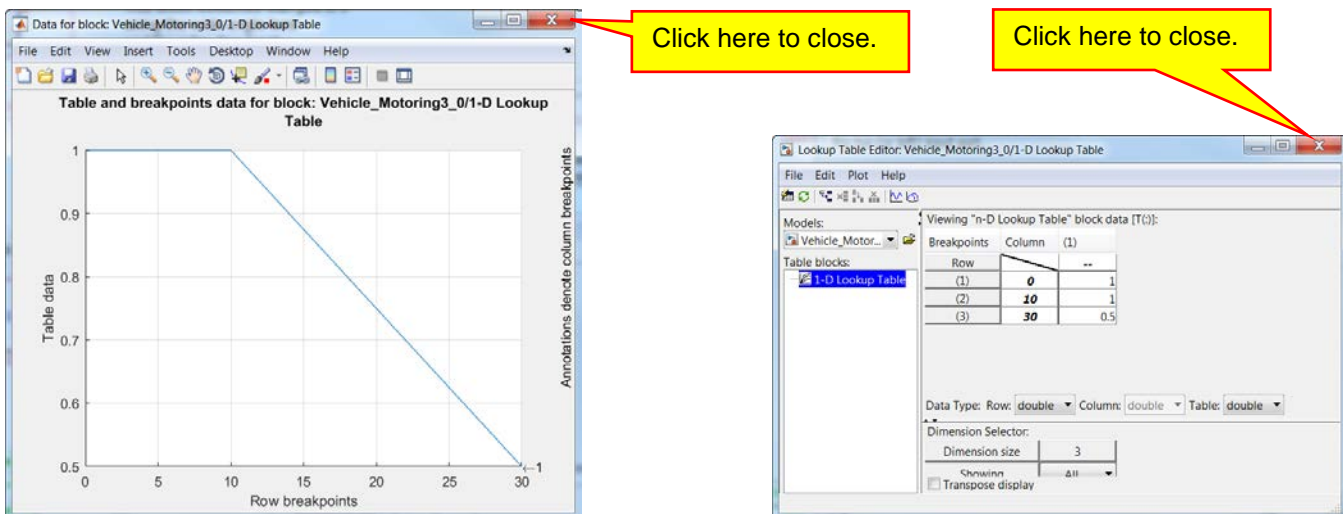
If you want to modify the table, you could do it here. To see a plot of the table we are creating, click the Linear Plot button 



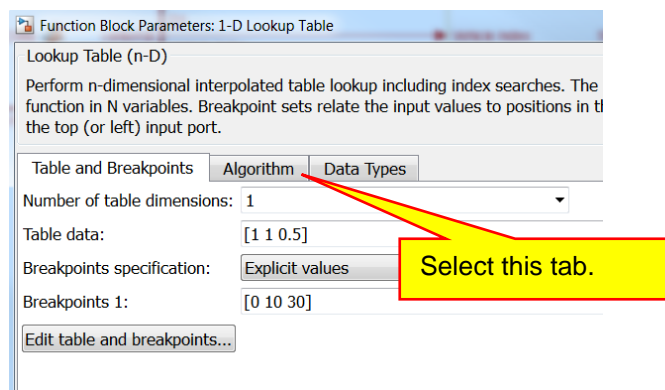
A plot of the lookup table we are implementing will be shown:



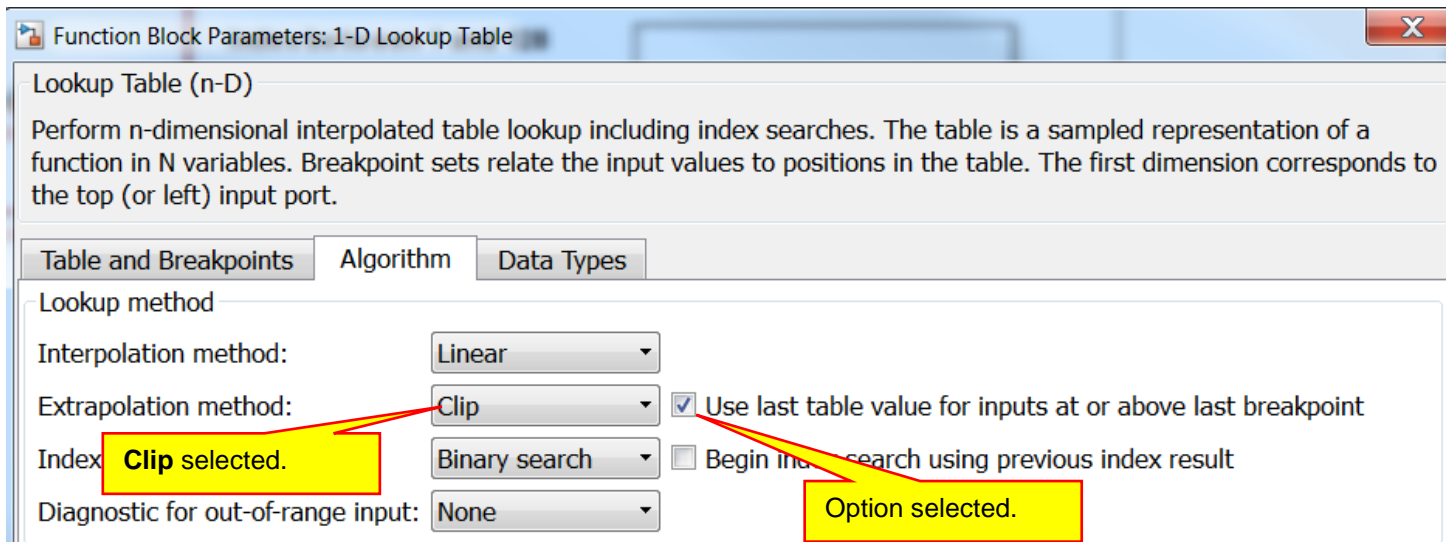
This is a nice picture of the functionality we are implementing. From 0 to 10, the output is 1. From 10 to 30, the output decreases linearly from 1 down to 0.5. Close the plot window and close the spreadsheet view:



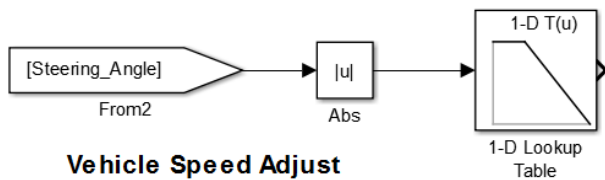
There is one more change that we need to make. Click the **Algorithm** tab as shown below:



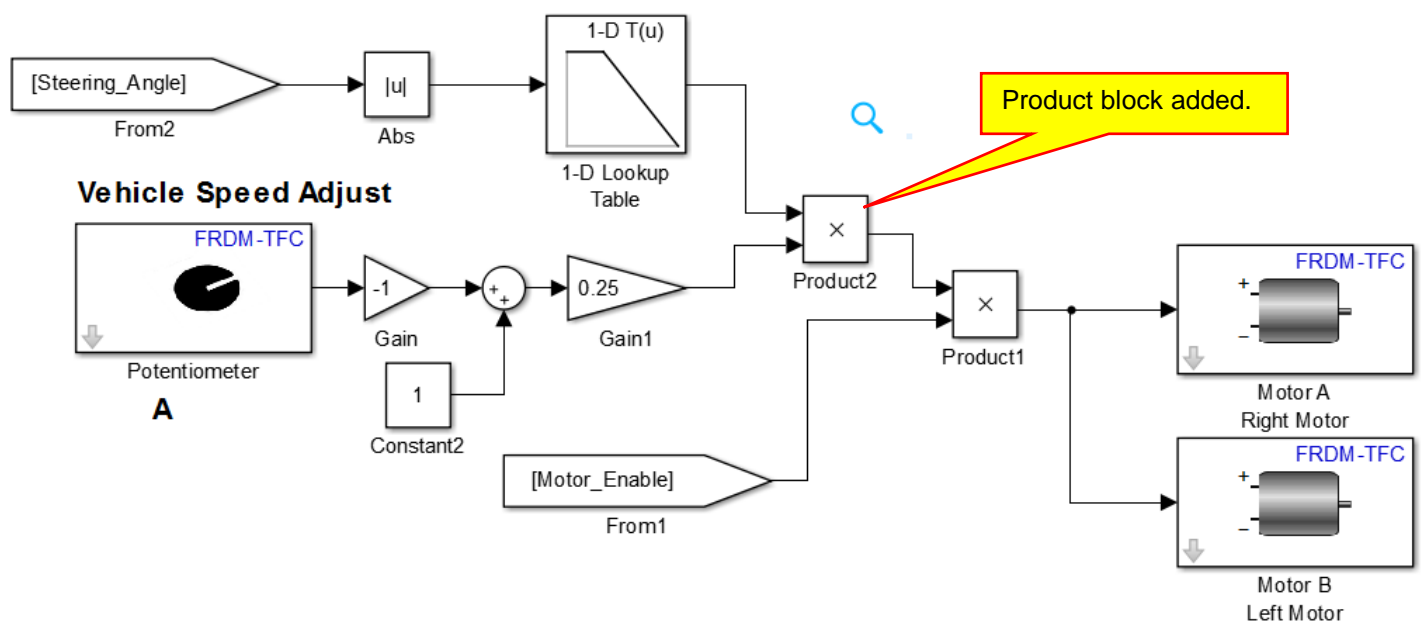
Change the **Extrapolation method** to **Clip** and select the option shown:



These options mean that when the input is less than zero, the output of the table will be 1, and when the input is greater than 30, the output will be 0.5. In other words, if the input is outside the domain specified by our table, the endpoints of the table will be used. Click the **OK** button to select the values we have specified. The block will display a plot of the function we have implemented:

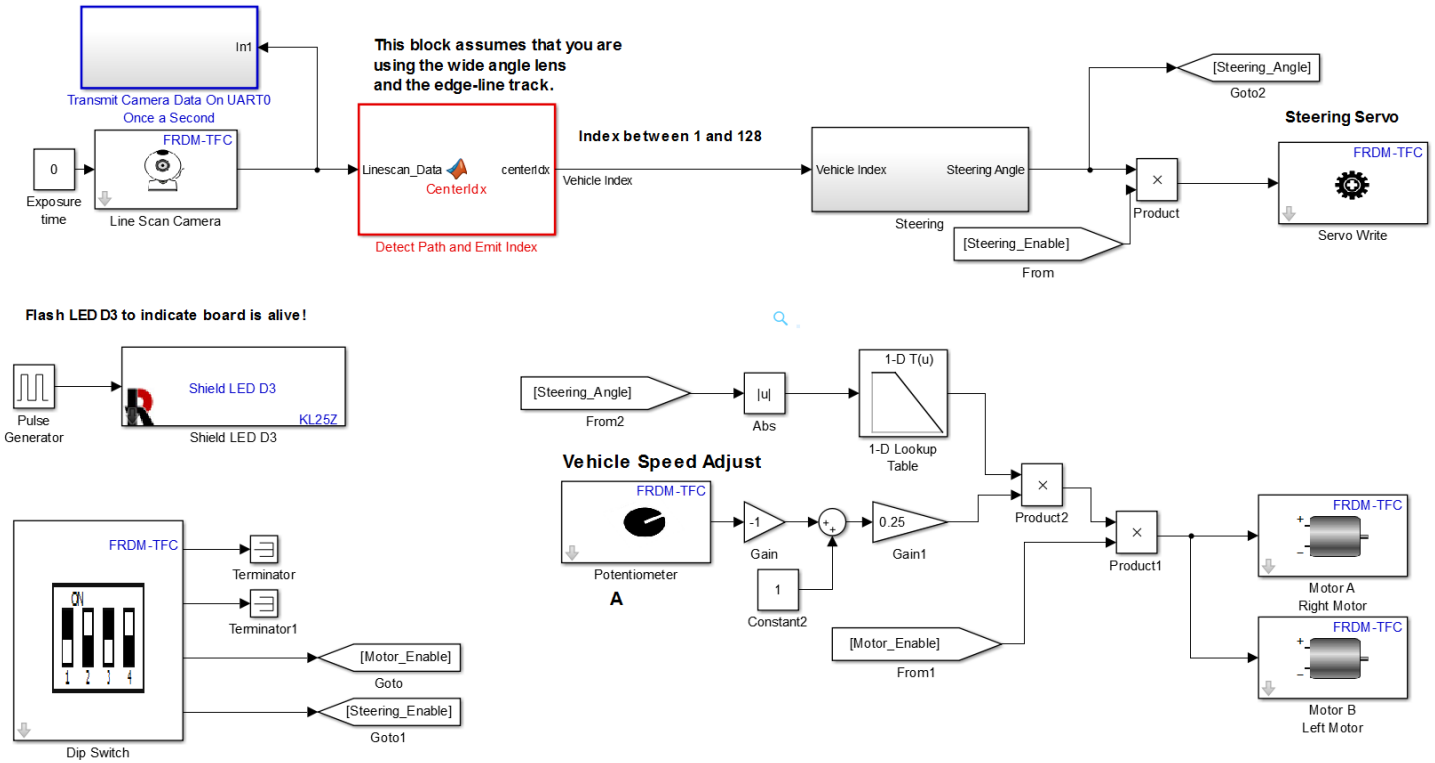


As a final step, we need to modify the speed signal with the values output by the lookup table:



When the steering angle is between 0 and 10, the output of the lookup table is 1, so the signal going to the motor is the speed determined by Pot A. When the steering angle is between 10 and 30, the output of the

lookup table decreases linearly from 1 down to 0.5, reducing the value from the Pot accordingly, and therefore reducing the vehicle speed around a turn. The complete model is shown below:



Build and download this model to your vehicle. Verify the following functionality in the demonstrations listed below:

Demo XIV-1: Verify that the vehicle speed is reduced as the vehicle makes a turn to the right.

Demo XIV-2: Verify that the vehicle speed is reduced as the vehicle makes a turn to the left.

Demo XIV-3: Verify that Potentiometer A determines the maximum speed when the vehicle is moving straight.

F. Optimizing Vehicle Speed – Steering Angle Based Vehicle Speed

We now have a vehicle that we can increase the speed in the straightaways because we know it will slow down in the turns. We will repeat the same optimization that we did in Section XIV.D with this added functionality. We should be able to achieve overall faster vehicle speeds. After filling in these tables, compare your results to the vehicle that did not have steering angle based vehicle speed. You will need to vary the vehicle speed, camera angle, steering gain to achieve the fastest vehicle speed. If you are brave, you can even modify the lookup table, although we recommend that you do this as a separate optimization.

Lesson XV: Vehicle Improvements

In this lesson we will suggest various improvements that you could make to the vehicle in an attempt to increase its performance. We will not discuss in detail how to make those improvements. The implementation is an assignment for the student. Hopefully you can use all of the knowledge you have gained in the previous lessons to implement your own solution to the suggested improvement. Also note that this list is not exhaustive. You may have your own ideas that you may want to attempt. It is never wrong to come up with an idea. It is only wrong to not separate the good ideas from the bad ideas. Always think your ideas through before testing them, and discuss them with your peers. There may be an obvious flaw that you or someone else can identify. If the ideas still sound feasible, give it a try.

Note that this section could take several weeks as the process is to try ideas that may or may not improve the vehicles performance. The amount of time depends on the number of ideas you want to test and the amount of time left until the competition. You do not need to try all of the ideas in this section, and you can come up with your own ideas. Not all of the ideas listed in this section will yield performance improvements.

Before choosing a method to implement, you should read this entire chapter and decide which methods you want to investigate. There is no specific order in which you should proceed. You can test as few or as many as you wish. However, you should read this entire lesson to see which methods interest you.

A. Metrics, Testing, and Evaluation

In order to evaluate your ideas and their implementation, we need to come up with metrics to gauge our performance. You will need to come up with a number of performance metrics, figure out how to test those metrics, record the results, and then compare those results.

First we must come up with some performance metrics. Some examples are given below. You should come up with additional metrics as well:

- Time for your vehicle to complete 10 laps around the oval.
- Minimum distance from the inside edge.
- Minimum distance from the outside edge.
- Does the vehicle drive straight?
- How deep into the corner does the vehicle travel before it starts a turn?
- What is the turning radius of your vehicle?
- What is the top speed of your vehicle in a straightaway?
- What is the minimum speed of your vehicle in a curve?

You will probably add metrics as you find that unwanted things and/or good things happen when you make changes, and you decided that those things are important. The more you work with your vehicle, the more things you will notice. If these things are important, you will come up with a metric for that thing and figure out how to test and measure it.

In order to test our ideas, we need to benchmark our original vehicle. Create a set of performance metrics, design a set of tests, and record those results for the vehicle that you completed at the conclusion of the last lesson. This is our baseline. We can then compare this baseline to the ideas we try in this section.

Also note that we are suggesting several new methods in this section. In addition to testing and varying these methods, you will also need to test the steering gain, vehicle speed, and camera angle in conjunction with these new methods. Although the exercises may not mention this testing, many of these new methods will affect how you set these parameters, and variations in the steering gain, vehicle speed, and camera settings should be tested in conjunction with the new methods mentioned.

Exercise XV-1: Create a list of metrics that you wish to use to evaluate the Performance of your vehicle. This should be a bulleted list. Specify how you would test and measure each metric.

Exercise XV-2: Create a table in which to record the performance metrics listed in **Exercise XV-1**.

Exercise XV-3: Measure the performance of your baseline vehicle. Record the information in your table. Keep the table in a safe location such as a notebook or file folder. You will need to compare the baseline results to all of the ideas you attempt in this section.

B. Basic Adjustments

In previous lessons, we have discussed the steering gain, camera adjustments, and vehicle speed. The team that understands these basic adjustments will achieve the greatest performance improvements without making any additional changes when compared to a team that takes these items lightly and focuses only on the improvements suggested in this lesson. Steering gain, camera adjustments, and vehicle speed are the fundamentals of the vehicle operation. Finding the best combination of these items will yield the greatest performance benefits.

Come up with a logical way to vary each item (steering gain, camera settings, and vehicle speed) and then measure the performance. Note that they are interdependent, but if you vary one at a time and carefully observe the vehicle's operation, you can determine its affect. Vary these three, record the performance, and attempt to achieve the best vehicle performance before attempting the items below. Document and record all changes and results. (Fill out a lot of tables and make notes of what you have changed.)

C. Cleaning the Tires

You may have noticed that as you use the vehicle, the track and the vehicle become covered with dust. This layer of dust reduces the coefficient of static friction between the tires and the track, which can greatly reduce how fast you can navigate a turn and how fast you can accelerate. The suggested method for cleaning the tires is to use isopropyl alcohol and a paper shop towel. You might come up with another method.

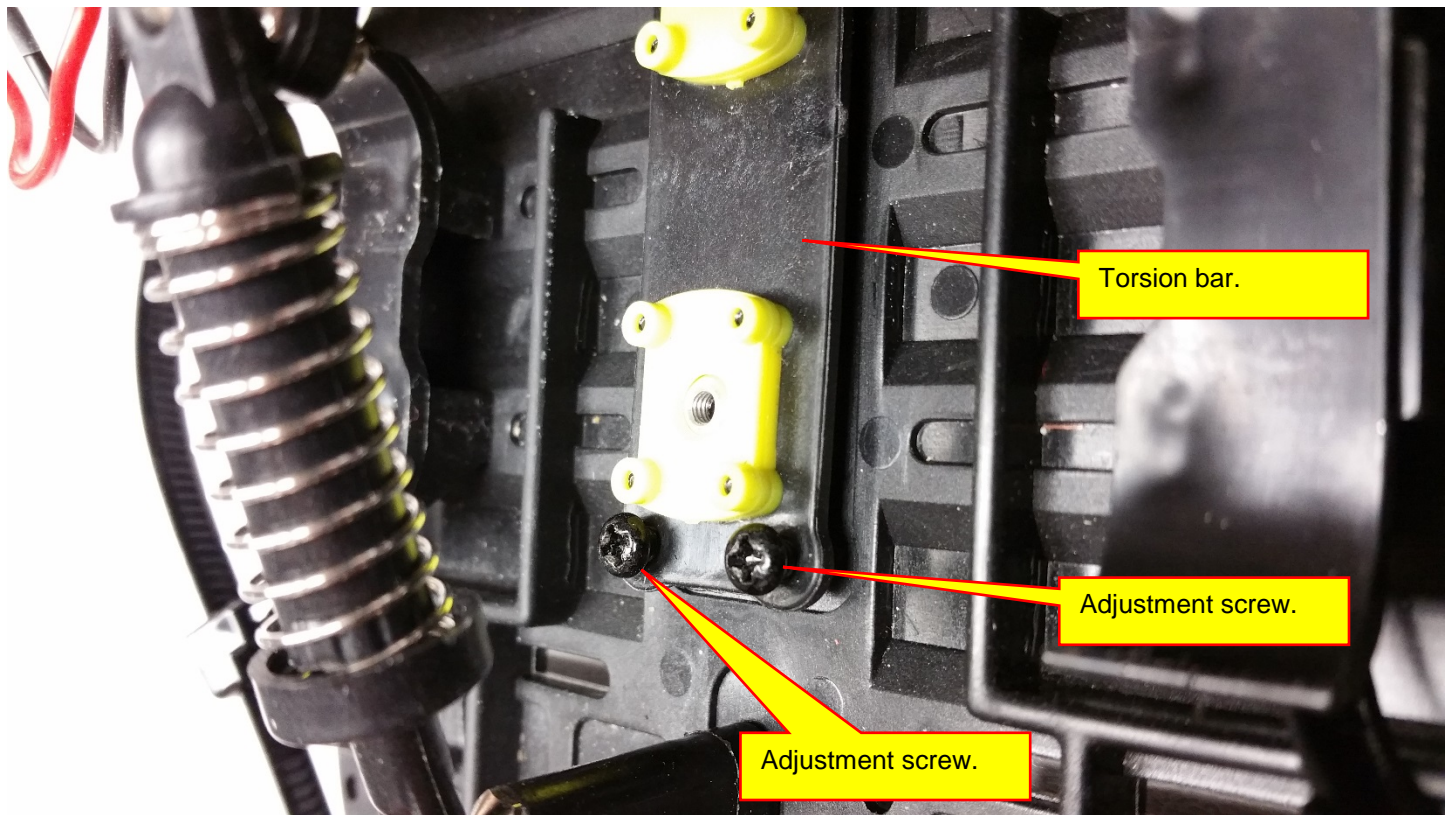
Exercise XV-4: Measure the performance of your vehicle before you clean the tires.

Exercise XV-5: Come up with a procedure for cleaning your vehicle tires. Document this procedure so that it will be repeated whenever the vehicle is used for testing or competition. Your procedure should include the materials used, any special methods that you employ, and how often the tires should be cleaned.

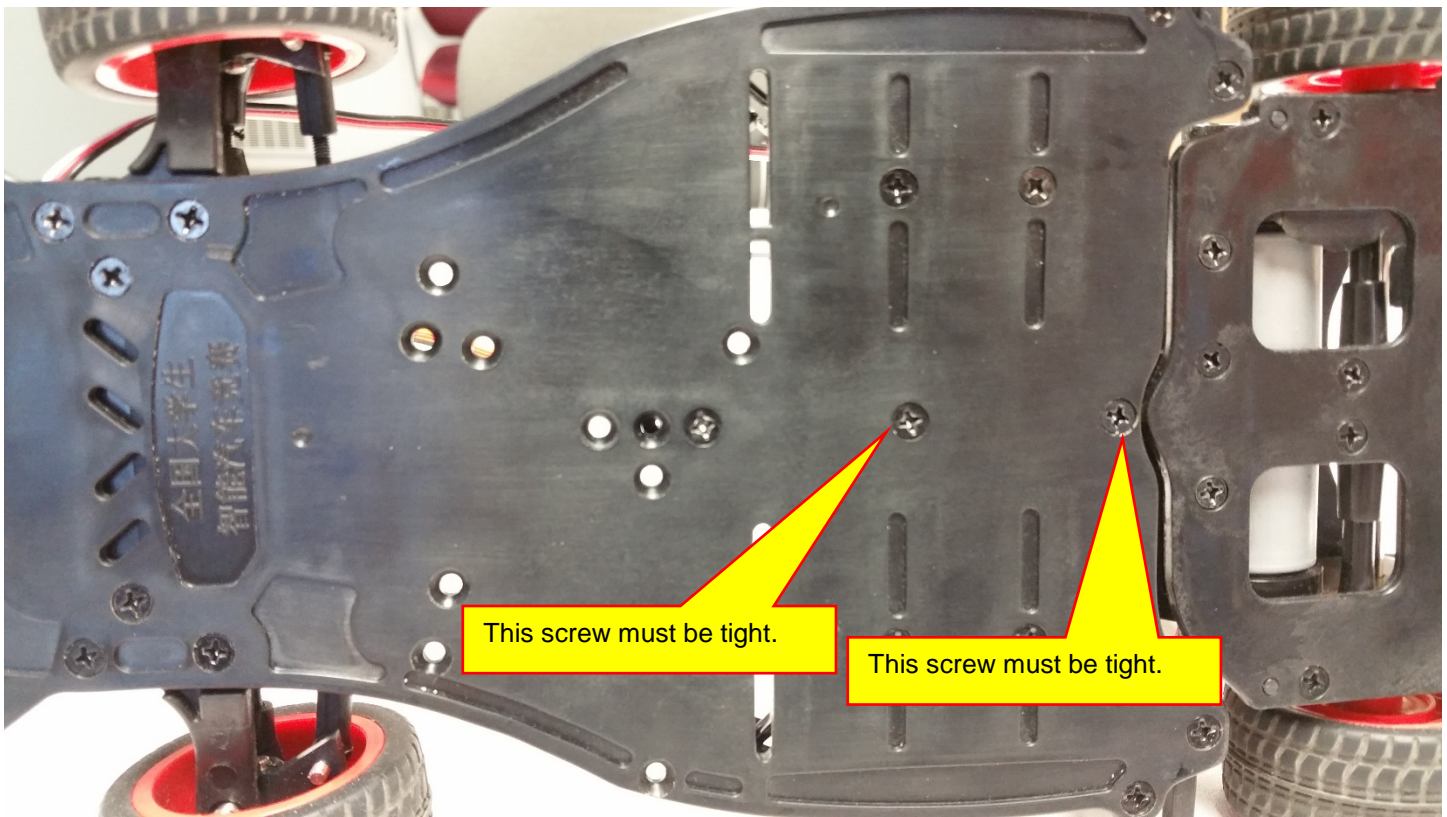
Exercise XV-6: After cleaning the tires, measure the vehicle's performance and compare to that measured in **Exercise XV-4**.

D. Torsion Bar Adjustment

If you remove the battery, you will notice that there are two screws that you can use to adjust how much the back wheels are allowed to twist:



These two screws can be tightened or loosened to make twisting the back end harder or easier. This type of spring or bar is referred to as a torsion bar or torsion spring and adjusts how much something can twist or resist twisting. In our case, the torsion bar adjusts how much the back end can twist relative to the front of the vehicle. Tightening or loosening the screws adjusts the stiffness of the torsion bar. Also note that there are two screws on the bottom of the vehicle that secure the bar to the vehicle. These two screws must be tight, and are not for adjustment:



Actually, all of the underside screws should be tight. Now is a good time to check off of the screws.

Adjust the two topside screws on the torsion bar. Grasp the front of the vehicle with your left hand and the rear of the vehicle with your right hand. Twist the vehicle gently. Make an adjustment to the vehicle and notice that the stiffness changes.

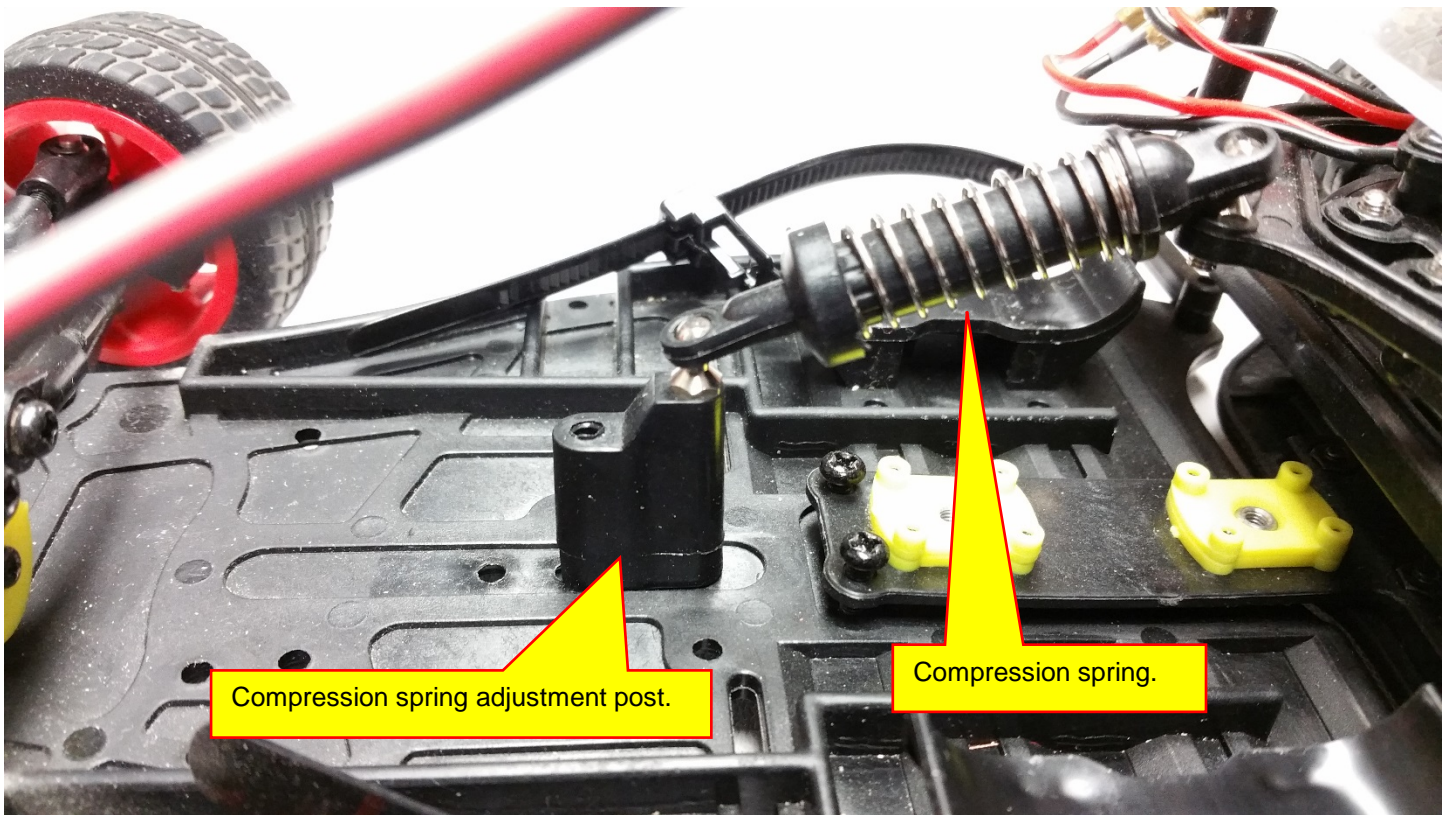
Exercise XV-7: Adjust the torsion bar to minimum stiffness. This means that you can twist it with very little force. Measure your vehicle's performance.

Exercise XV-8: Adjust the torsion bar to maximum stiffness. Measure your vehicle's performance and compare to **Exercise XV-7**.

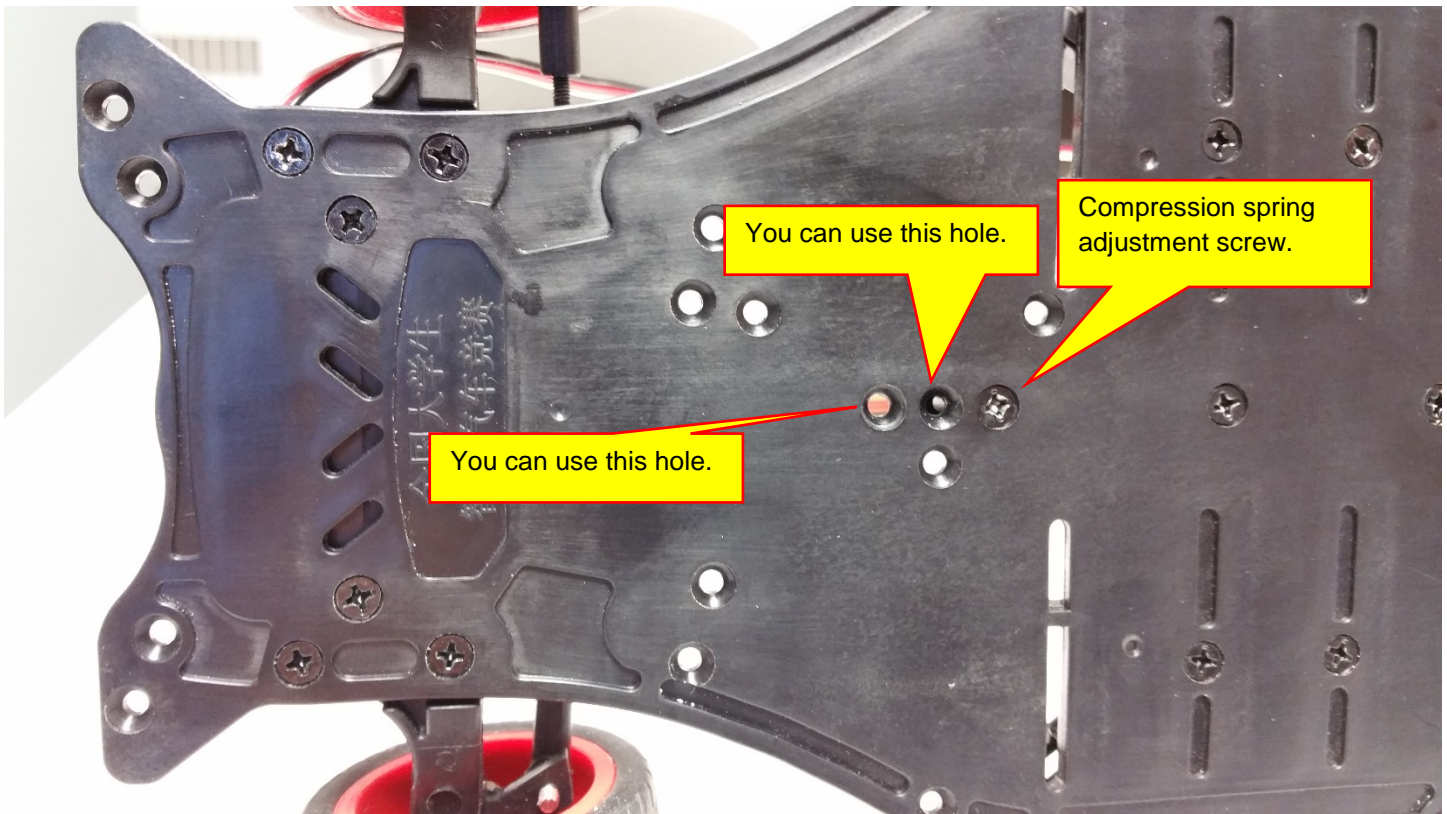
Exercise XV-9: Adjust the torsion for maximum performance. This may take several adjustments and several measurements. Make sure that you record a table of data for every adjustment that you make. Also make sure that you have a way of knowing the position of the adjustment screws.

E. Compression Spring Adjustment

The compression spring and adjustment post are shown below:



This spring adjusts how much the vehicle flexes in the center. (Flexes, not twists.) The compression can be changed by rotating the post by 180 degrees or moving the post forwards or backwards. As shown above, the spring is compressed the maximum amount. Yours will be set differently. The post is moved using the screw on the bottom side of the vehicle:



There are six positions you can use. The three holes that are lined up can be used to move the post forward or backward. You can also rotate the post by 180 degrees to make further adjustments.

Exercise XV-10: Adjust the compression spring to minimum compression. (It will be completely uncompressed with no hope of actually ever being compressed. Thus, it will do nothing.) Measure your vehicle's performance.

Exercise XV-11: Adjust the compression spring to maximum compression. Measure your vehicle's performance and compare to **Exercise XV-10**.

Exercise XV-12: Adjust the compression spring for maximum performance. This may take several adjustments and several measurements. Make sure that you record a table of data for every adjustment that you make. Also make sure that you have a way of knowing the position of the adjustment post.

F. Front-Rear Weight Distributions and Adjustment

Right now the weight on each tire is unknown and was not really a design concern. The camera was mounted on the front and the hardware was mounted on the rear. It appears that we have placed more weight on the rear of the vehicle. This might increase the traction on the rear wheels but also might decrease the traction of the front wheels, and therefore affect the steering. You can change the weight distribution of the car by adding steel weights at various locations in the vehicle in an attempt to increase or decrease the traction of specific wheels. As always, document any changes you make and record performance changes in your standard table.

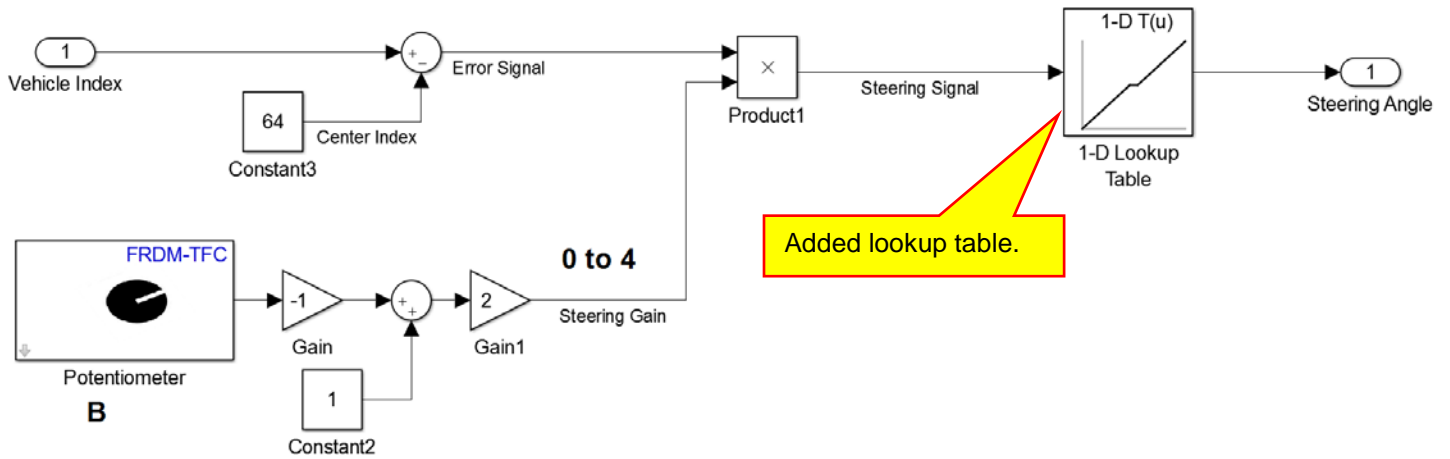
Exercise XV-13: Before adding weights, measure your vehicle's performance to obtain a baseline.

Exercise XV-14: Add weights to specific locations on your vehicle. Document where you place the weights, why you placed them in their specific locations, and what you expect to happen. Measure your vehicle's performance and compare to the baseline and any previous weight trials you may have made. It may take several trials to see an effect or to achieve the desired result. Make separate trials and document each trial separately so that you can compare all of the trials and have a history of all of the variations that you made.

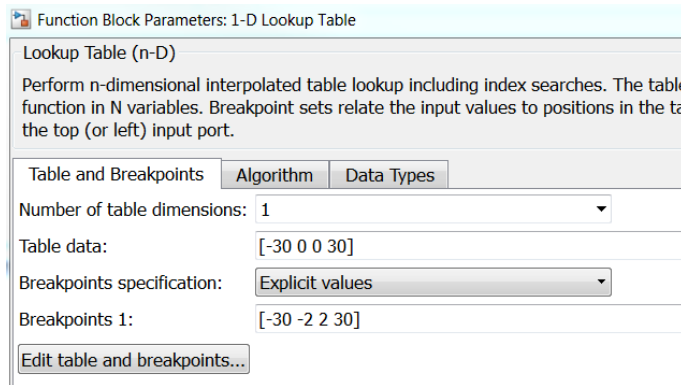
G. Dead Spot Steering

You may have noticed that when you increase the steering gain to a high value, the vehicle might weave back and forth down a straightaway even though it should travel straight. This is because our steering system only outputs a zero steering signal when the camera returns an index of 64. If the camera returns a 63 or 65, a non-zero steering signal will occur causing the vehicle to move to the right or left. In control systems, this weaving is referred to as an oscillation. If the oscillations get to big, the system can go out of control. Thus, for high gain, our method might be too sensitive. Fix this, we will add a dead spot to the steering where the steering signal will be zero unless the steering signal is greater than some threshold. We can easily implement this dead spot with a lookup table.

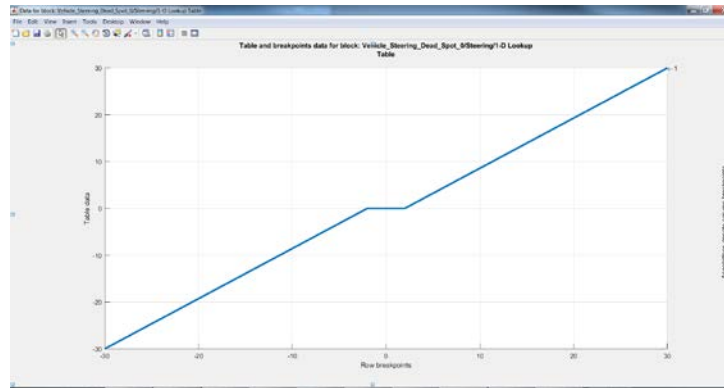
An example is shown below. Note that before we make the changes, our steering signal was originally limited to +/- 30. We will keep these same limits but add a dead spot to the steering. We will modify the blocks inside the steering subsystem:



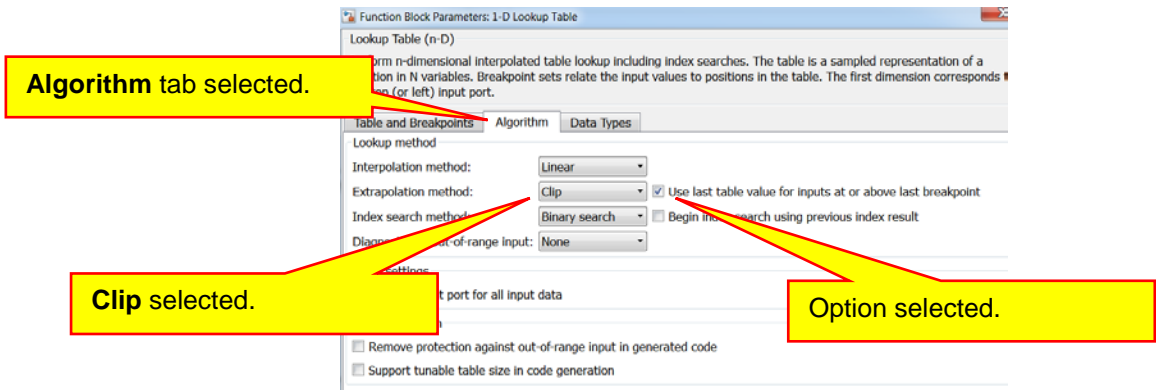
Opening my lookup table, we see that it has a dead spot between -2 and +2:



Remember that the **Breakpoints** are the values of the input signal and the **Table data** are the values of the output. For these settings, when the input is between -2 and +2, the output steering signal is zero. A more detailed plot of this function is show below:



You can choose a different width dead spot if you wish by changing the breakpoints. Also not that in the **Algorithm** tab of the dialog box, you need to specify the options below:

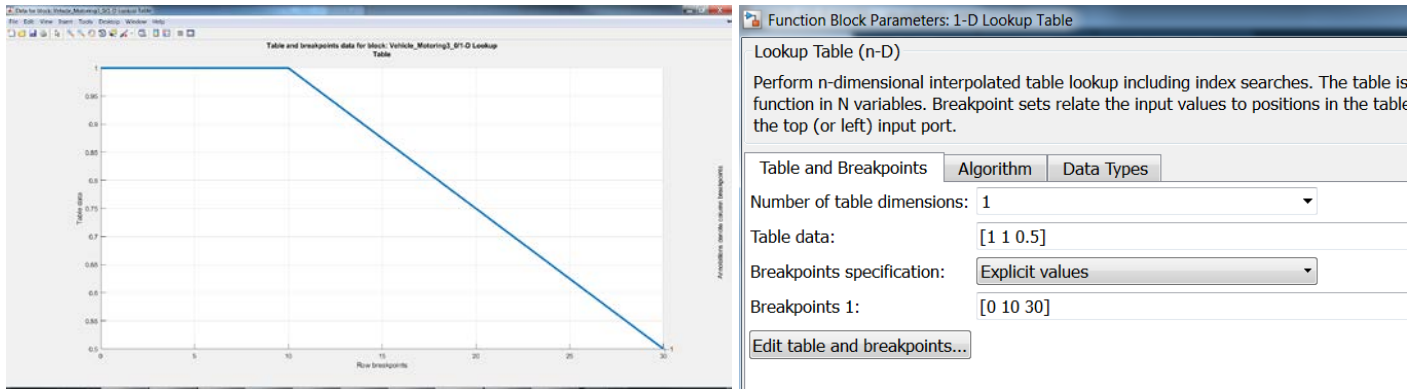


Exercise XV-15: Before adding the dead spot lookup table, measure your vehicle’s performance to obtain a baseline performance.

Exercise XV-16: Add the dead spot lookup table to your control model. Document your changes and what you expect to happen. Measure your vehicle’s performance and compare to the baseline and any other trials that you have made. It may take several trials with different variations in the dead spot to achieve the desired effect. Make separate trials and document each trial separately so that you can compare all of the trials and have a history of all of the variations that you made.

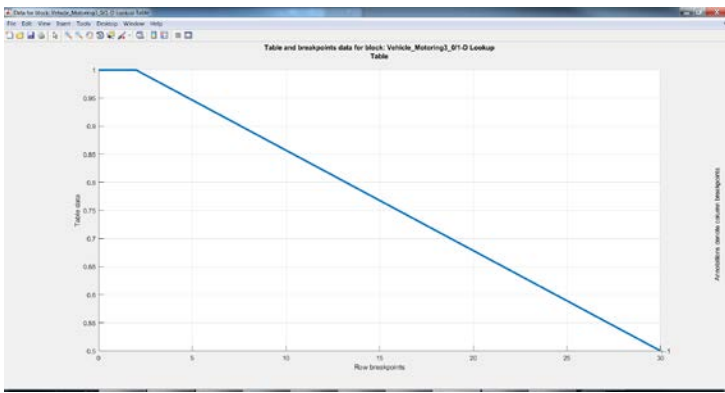
H. Steering Angle Based Speed Table Adjustment

In Section XIV.F we added a lookup table that made the vehicle speed a function of the steering angle. The basic idea was that when the vehicle is turning hard, we should slow down. The plot and data for the lookup table that we used is shown below:



Adding this feature greatly improved the performance of our vehicle. However, we can do better. As shown, the table has no affect for the rather large steering signals of up to 10. Only after a steering signal of 10 is the vehicle speed affected. This has the result that the vehicle goes deep into a curve before the speed is modified. Then, once it is modified, the vehicle speed is drastically reduced. We can probably do better. It would be good to slow down once we detect the slightest hint of a curve, and we probably want to go faster while executing a continuous curve. The present method does not work well in a continuous curve or a curve after a long straightaway where we have built up a lot of speed. Below are some suggestions, but feel free to try your own.

The first method is the same as before, but the speed reduction starts at a steering angle of 2. This will cause the speed reduction to start earlier in a curve:



Function Block Parameters: 1-D Lookup Table

Lookup Table (n-D)
Perform n-dimensional interpolated table lookup including index searches. The table function in N variables. Breakpoint sets relate the input values to positions in the table at the top (or left) input port.

Table and Breakpoints Algorithm Data Types

Number of table dimensions: 1

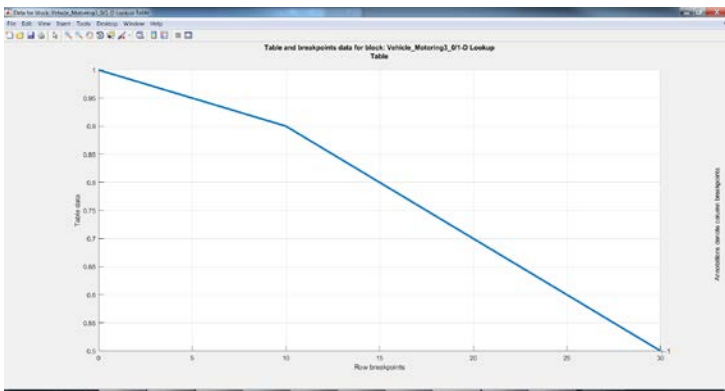
Table data: [1 1 0.5]

Breakpoints specification: Explicit values

Breakpoints 1: [0 2 30]

Edit table and breakpoints...

Next, we will try reducing the speed gradually at the start, then sharper as the steering angle increases:



Function Block Parameters: 1-D Lookup Table

Lookup Table (n-D)
Perform n-dimensional interpolated table lookup including index searches. The table function in N variables. Breakpoint sets relate the input values to positions in the table at the top (or left) input port.

Table and Breakpoints Algorithm Data Types

Number of table dimensions: 1

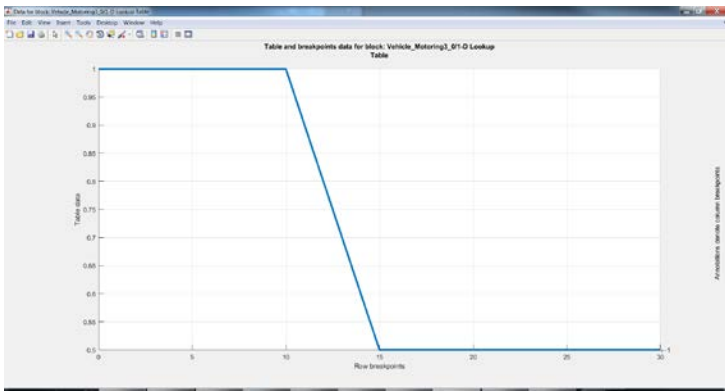
Table data: [1 0.9 0.5]

Breakpoints specification: Explicit values

Breakpoints 1: [0 10 30]

Edit table and breakpoints...

Another possibility would be a two speed control with a sloped change:



Function Block Parameters: 1-D Lookup Table

Lookup Table (n-D)
Perform n-dimensional interpolated table lookup including index searches. The table function in N variables. Breakpoint sets relate the input values to positions in the table at the top (or left) input port.

Table and Breakpoints Algorithm Data Types

Number of table dimensions: 1

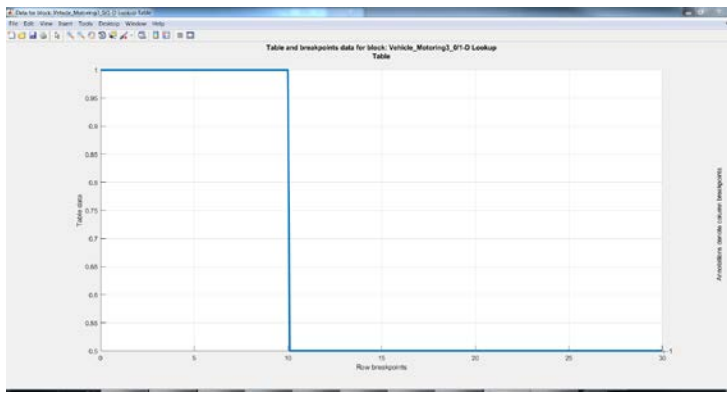
Table data: [1 1 0.5 0.5]

Breakpoints specification: Explicit values

Breakpoints 1: [0 10 15 30]

Edit table and breakpoints...

A variation of the method above would be a two speed stepped control:



Function Block Parameters: 1-D Lookup Table

Lookup Table (n-D)
Perform n-dimensional interpolated table lookup including index searches. The function in N variables. Breakpoint sets relate the input values to positions in the top (or left) input port.

Table and Breakpoints Algorithm Data Types

Number of table dimensions: 1

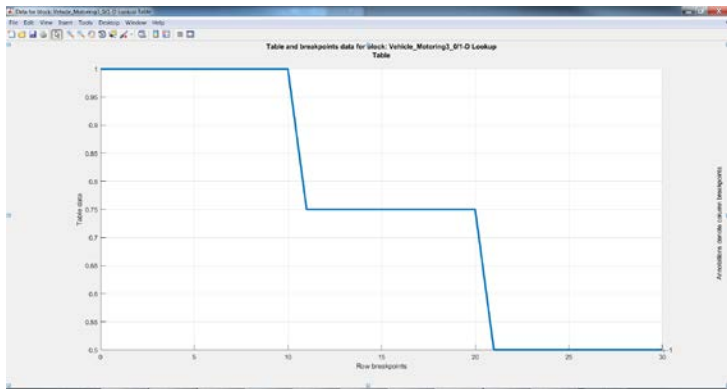
Table data: [1 1 0.5 0.5]

Breakpoints specification: Explicit values

Breakpoints 1: [0 10 10.1 30]

Edit table and breakpoints...

The above method seems a bit drastic. Maybe a 3-step speed control with a slope:



Function Block Parameters: 1-D Lookup Table

Lookup Table (n-D)
Perform n-dimensional interpolated table lookup including index searches. The function in N variables. Breakpoint sets relate the input values to positions in the top (or left) input port.

Table and Breakpoints Algorithm Data Types

Number of table dimensions: 1

Table data: [1 1 0.75 0.75 0.5 .5]

Breakpoints specification: Explicit values

Breakpoints 1: [0 10 11 20 21 30]

Edit table and breakpoints...

The above examples are only suggestions. You may want to try some of them and then come up with your own method. As always, propose ideas, document those ideas, and then record performance results. For all of the lookup tables, make sure that the options on the **Algorithm** tab are selected as shown below:

Function Block Parameters: 1-D Lookup Table

Lookup Table (n-D)
Perform n-dimensional interpolated table lookup including index searches. The table is a sampled representation of a function in N variables. Breakpoint sets relate the input values to positions in the table. The first dimension corresponds to the top (or left) input port.

Table and Breakpoints **Algorithm** Data Types

Lookup method

Interpolation method: Linear

Extrapolation method: Clip Use last table value for inputs at or above last breakpoint

Index search method: Binary search Begin interpolation search using previous index result

Discontinuity at out-of-range input: None

Settings

Input port for all input data

Remove protection against out-of-range input in generated code

Support tunable table size in code generation

Algorithm tab selected.

Clip selected.

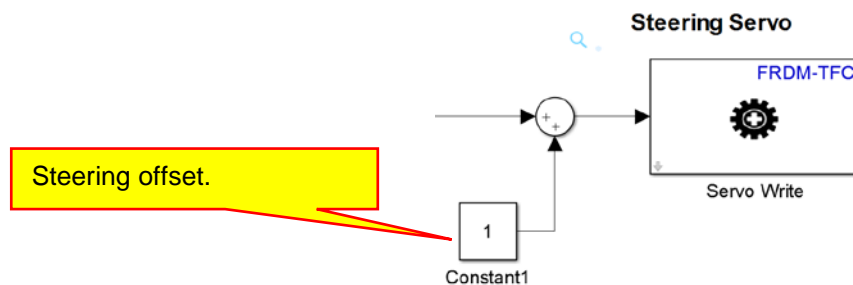
Option selected.

Exercise XV-17: Before implementing the steering angle based vehicle speed lookup table, measure your vehicle’s performance to obtain a baseline.

Exercise XV-18: Modify the steering angle-vehicle speed lookup table in your control model. Document your changes and what you expect to happen. Measure your vehicle’s performance and compare to the baseline and any other trials that you have made. It may take several trials with different variations in the table to achieve the desired effect. Make separate trials and document each trial separately so that you can compare all of the trials and have a history of all of the variations that you have made.

I. Steering Offset

One thing that you may have noticed is that the front wheels don't quite point straight even though you zeroed the servo motor before building the steering. Luckily this is a constant error and is the same every time you use your vehicle. (Unless you have smashed the front end of your vehicle several times and damaged the servo motor.) The result is that the vehicle doesn't drive straight with zero steering angle and a right turn is not quite the same as a left turn. We can correct this problem by adding a constant offset to the steering. Before you test the offset, disable the steering and the motor with the DIP switches. Next, add a constant to the steering signal as shown:



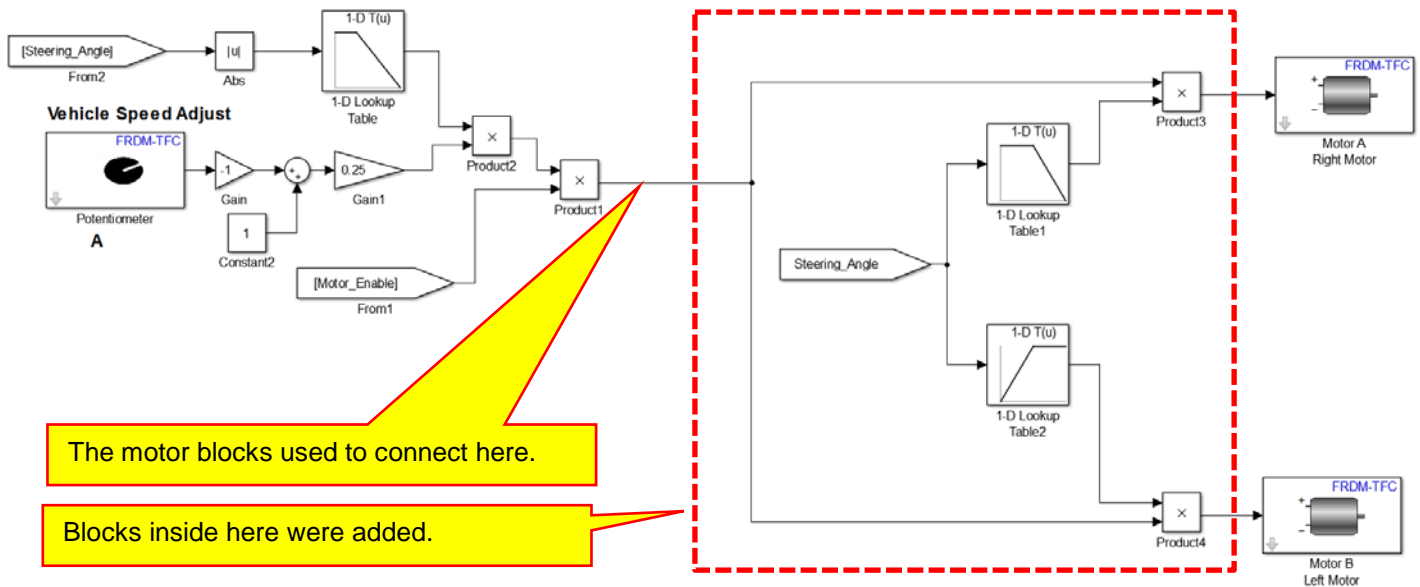
With the steering disabled, this offset is the only input to the servo motor. Build and download the model to your vehicle. Make sure that the steering is disabled. When you turn on the power to your vehicle, the steering will respond to the constant only. Find a value for the constant so that the wheels point straight when the steering is disabled. Every time you change the constant, you will need to build and download the model to the vehicle and test it again. The constant may need to be positive or negative, and is usually small. (Unless you forgot to zero your servo before building the steering.)

Exercise XV-19: Determine the value of the steering offset constant necessary to make your wheels point straight when the steering is disabled and the vehicle is powered.

J. Rear-Wheel Steering

One of the greatest improvements that we can make is rear-wheel steering control. If we apply the same torque to each of the rear wheels, the vehicle will tend to drive straight. However, if we apply full torque to the right rear wheel and no torque to the left rear tire, the vehicle will want to turn to the left. And, if we apply full torque to the left rear wheel and no torque to the right rear tire, the vehicle will want to turn to the right. We can use this property to help the vehicle turn with less steering angle and more control.

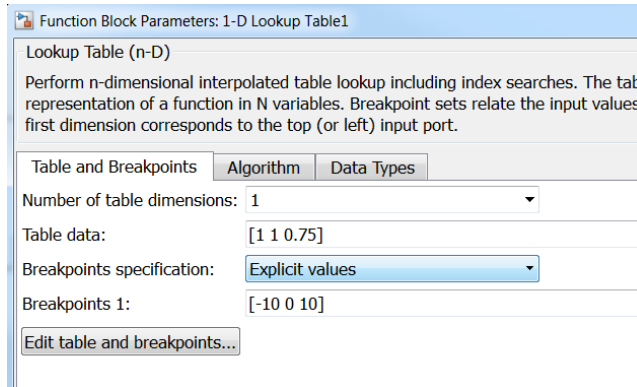
Right now the torque to each wheel is the same no matter what the vehicle is doing. We will modify the control so that the torque to each wheel is based on the steering angle. When the vehicle is turning, we can make the vehicle turn faster and with more control by applying more torque to the outside wheel or less torque to the inside wheel. The implementation below removes torque from the inside wheel:



The motor blocks used to connect here.

Blocks inside here were added.

Only the motor portion of the model is shown. The motors were directly connected to the output of the block named **Product1**. The portion inside of the dashed red line was added. Looking inside **Lookup Table1** we see:

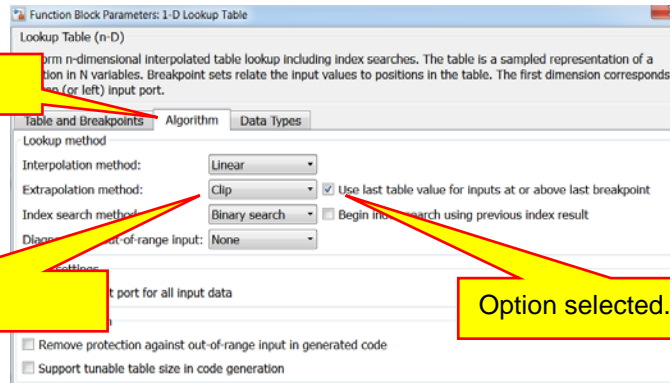


For negative steering angles the right motor gets full power. Hopefully a negative steering angle turns the vehicle to the left. (We will need to verify this.) For positive steering angles, the torque for the right motor is reduced, down to a minimum of 75% of full torque. Make sure that the options below on the **Algorithm** tab are selected:

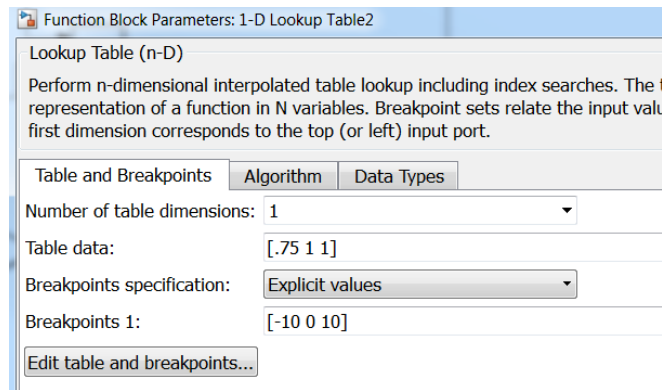
Algorithm tab selected.

Clip selected.

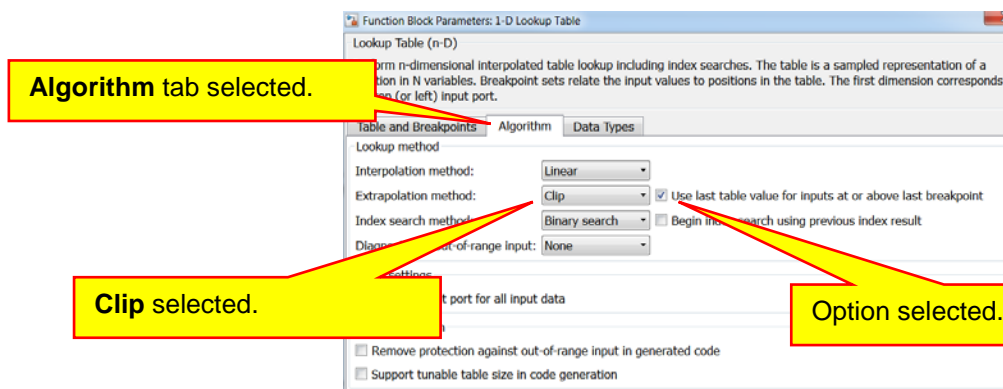
Option selected.



Looking inside **Lookup Table2** we see:



For positive steering angles the left motor gets full power. Hopefully a positive steering angle turns the vehicle to the right. (We will need to verify this.) For negative steering angles, the torque for the right motor is reduced, down to a minimum of 75% of full torque. Make sure that the options below on the **Algorithm** tab are selected:



Exercise XV-20: Before implementing rear-wheel steering, measure your vehicle's performance to obtain a baseline.

Demo XV-1: When you first test this method, hold the vehicle above a straight piece of track. Turn on and enable your vehicle, both the steering and the motors. When you move the vehicle near the line on the right side, the right tire should spin more than the left tire and the front wheels should turn to the left. When you move the vehicle near the line on the left side, the left tire should spin more than the right tire and the front wheels should turn to the right.

Exercise XV-21: Implement rear-wheel steering. Document your changes and what you expect to happen. Measure your vehicle's performance and compare to the baseline and any other trials that you have made. It may take several trials with different variations in the tables to achieve the desired effect. Make separate trials and document each trial separately so that you can compare all of the trials and have a history of all of the variations that you have made.

K. Camera Index Memory

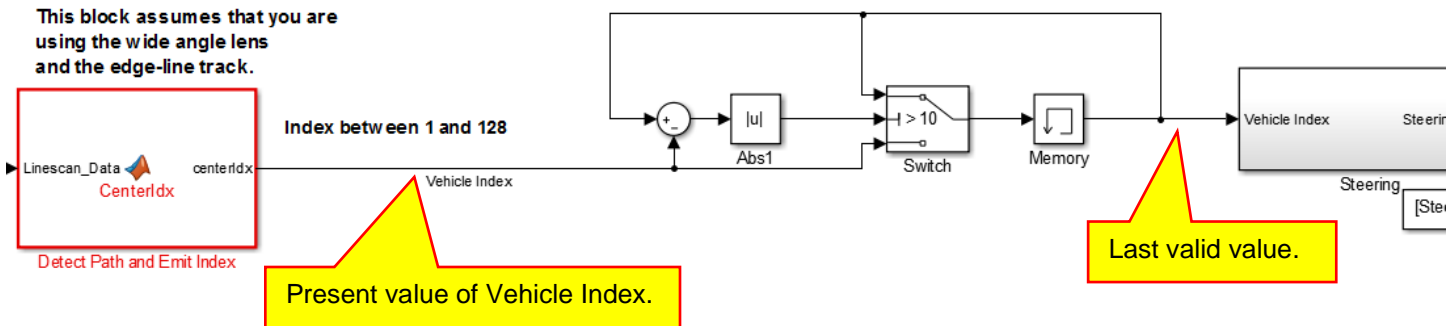
You may have noticed that even with rear-wheel steering and steering-based speed control, both of which help us improve vehicle speed, we still have an upper speed limit after a long straight-away heading into a sharp curve. When we gain a lot of speed in a straightaway, the car travels deep into the curve and comes close to the outside line. As you increase the speed, the car comes closer to the outside line. If the car comes close enough to the outside line, the camera loses the line and the resulting error causes the steering to go straight. This results in the vehicle driving straight rather than follow the curve. (And, if you are using steering based speed control, the car accelerates to top speed as it is driving straight.) The result can be some spectacular crashes.

Luckily, we have a way of determining when this error occurs. When our system is working correctly and following the lines, whether on a curve or on a straight-away, the **Vehicle Index** output by the block **Detect Path and Emit Index** is near 64 and is fairly consistent. No matter how fast the car is moving, the values output by this block do not change much from one reading to the next. (Remember, we get a new reading every 0.02 seconds.) So, the difference between two readings is small when everything is working well. When the camera loses the line, we will have one valid value followed by a value that is very different and is invalid. If this happens, we should ignore the invalid value and use the old value. We will use the old value until a new valid value is obtained.

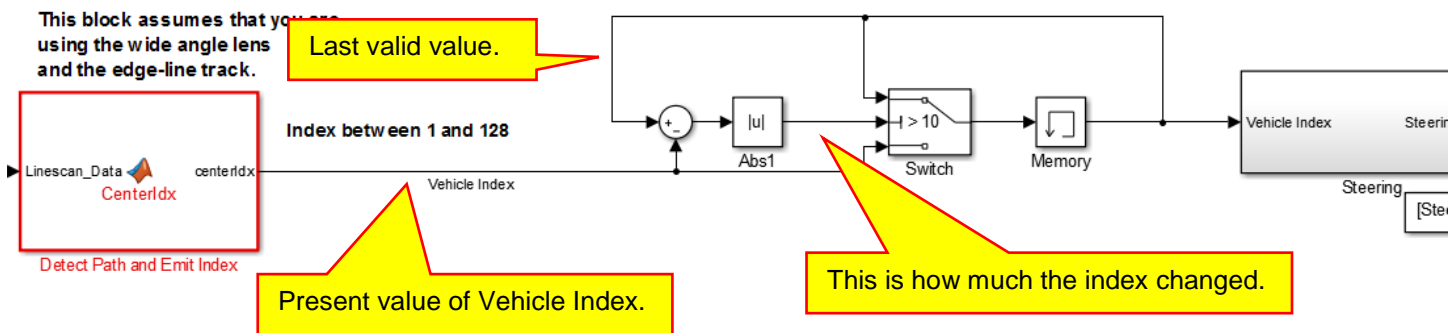
We can implement this algorithm with memory. We can save the last value of the **Vehicle Index** and compare it to the present value. If the difference between these two values is small, then we will assume that the present value is valid and use it for steering. However, if the difference between the present value and the last value is large, we will assume that there is an error and we will ignore the present value and use the value saved in memory.

Imagine that we are travelling fast around a curve, the wheels are turned as much as possible so the vehicle is trying to make the turn. However, the car is travelling too fast and comes so close to the outside line that the camera loses the line. When the camera loses the line, the value of the signal **Vehicle Index** makes a big jump. The algorithm we are implementing will ignore this erroneous value and use the last value. The result will be that the vehicle will continue to turn until it reacquires the line, at which point the **Steering Index** will once again change slowly and produce valid values. The car may come close or go over the outside line, but it will continue to turn and stay on the track.

We can implement this algorithm with a memory block. The output of a memory block is the input from the last time step. Thus, the memory block can be used to remember the last valid value. The memory block is located in library **Simulink / Discrete**. Create the system below:

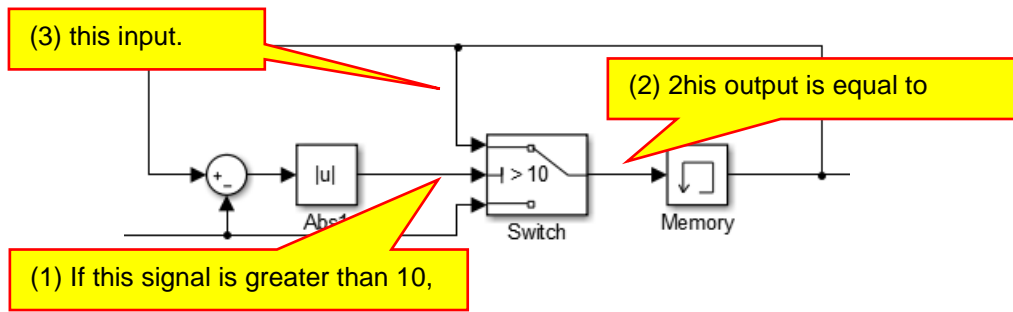


The output of the memory block is the value of the last valid value of the signal **Vehicle Index**. Note that this value is fed back to the difference block. Therefore, the output of the **Abs** block is the difference between the present value of the **Vehicle Index** and the last valid value of the **Vehicle Index**:

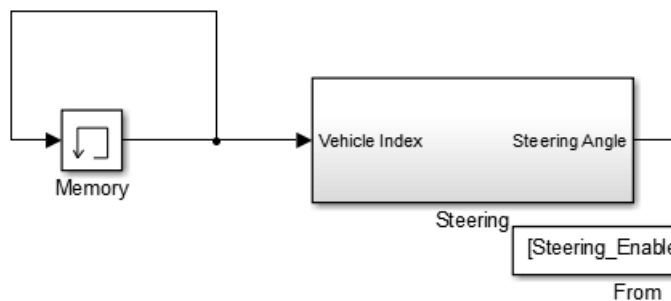


We see that the output of the **Abs** block measures how much the index changes between two consecutive samples. If the index changes by more than 10, we will say that the most recent index is invalid and we will keep the old value. If the difference is less than or equal to 10, we will say that the most recent index is valid and pass it on to the memory block. We will use this to control a switch.

Looking at the switch, if the difference greater than 10, the output of the switch will be equal to the top input. This is shown below. Read the call outs in order of (1), (2) and then (3):

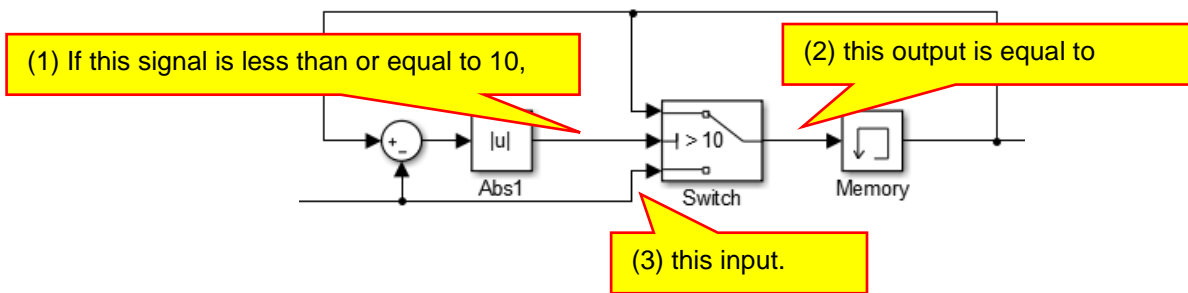


Another way to look at it, when the difference is greater than 10, the above system is equivalent to the one below:

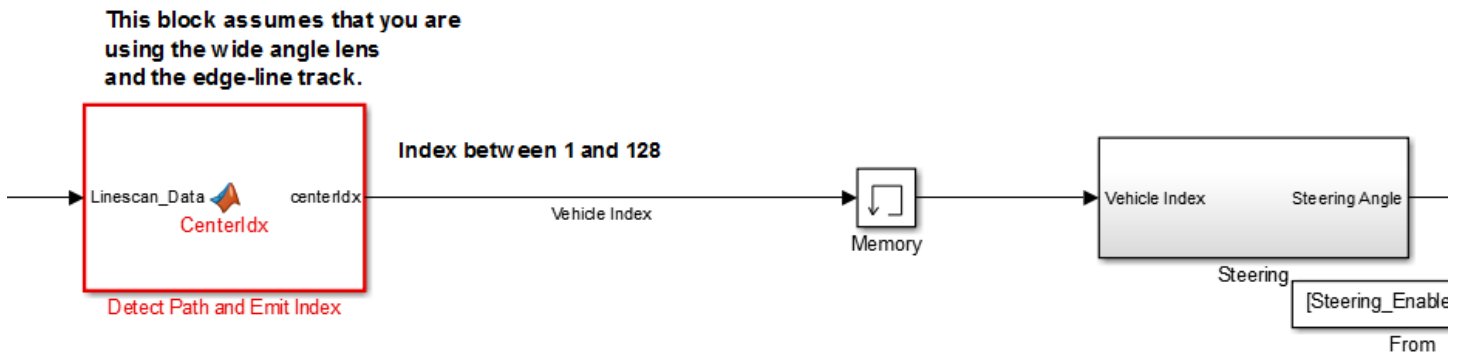


The input of the memory block is equal to its output. This means that the **Memory** block just holds its value. In this case, it holds the last value that it had. For us, this is the last valid value of the Vehicle Index signal.

Looking at the **Switch** again, if the difference less than or equal to 10, the output of the switch will be equal to the bottom input. This is shown below. Read the call outs in order of (1), (2) and then (3):

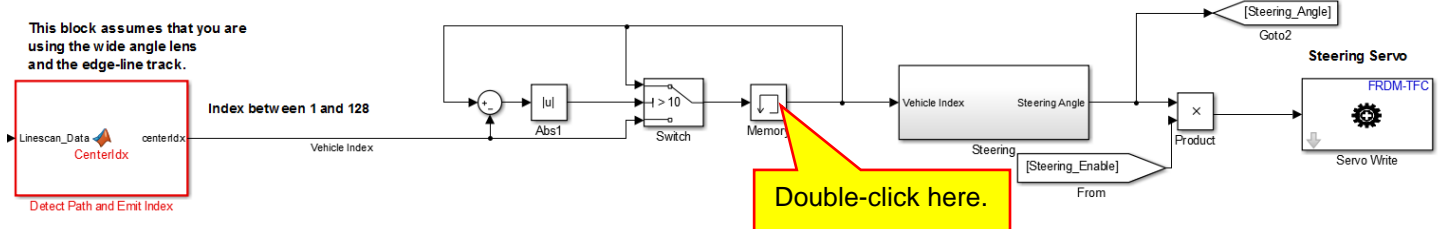


Another way to look at it, when the difference is less than or equal to 10, the above system is equivalent to the one below:

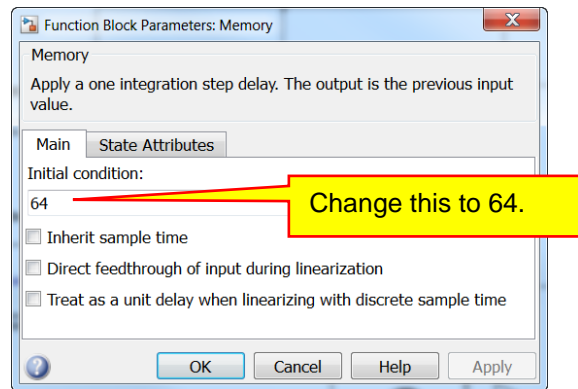


The input of the **Memory** block is now the **Vehicle Index** which means that the memory block stores the present value of the index. Thus, when the difference between the last value of the index and the present value of the index is less than or equal to 10, we consider it to be a valid value and that value is saved and passed along to our steering subsystem. In this way, we can compare the last value of the index to the present value. If they are not that different then we use the present value. If they are very different, we ignore the most recent value and use the one saved in memory. Note that our definition of “very different” is 10. You can change this threshold if you wish.

The complete steering system is repeated below:



One last thing we need to do is to set the initial condition. When the vehicle starts on a straight track, the **Vehicle Index** is typically close to 64. By default, when we power up the vehicle, the initial value of the memory block is zero. Since 64-0 is greater than 10, the above system will treat 0 as the last valid value of the index and remember that value forever. An index of 0 will lock the steering maximum right or maximum left. To avoid this problem, we need to change the initial condition of the memory block. Double-click on the memory block as shown above and then change the initial condition to 64:



To use this model, turn the car off, place the car on a straight section of track, and then turn on the power. You always have to cycle the power because the memory block remembers the last “valid” value of the **Vehicle Index**. If you leave the vehicle on and carry it around, you have no idea what the last “valid” value was. Thus, cycle the power to reinitialize the memory block to 64.

Exercise XV-22: Before implementing camera index memory, measure your vehicle’s performance to obtain a baseline.

Demo XV-2: When you first test this method, start at slow speed and make sure everything works. Then slowly increase the speed and see how it does. You should be able to achieve much higher speeds around the corners.

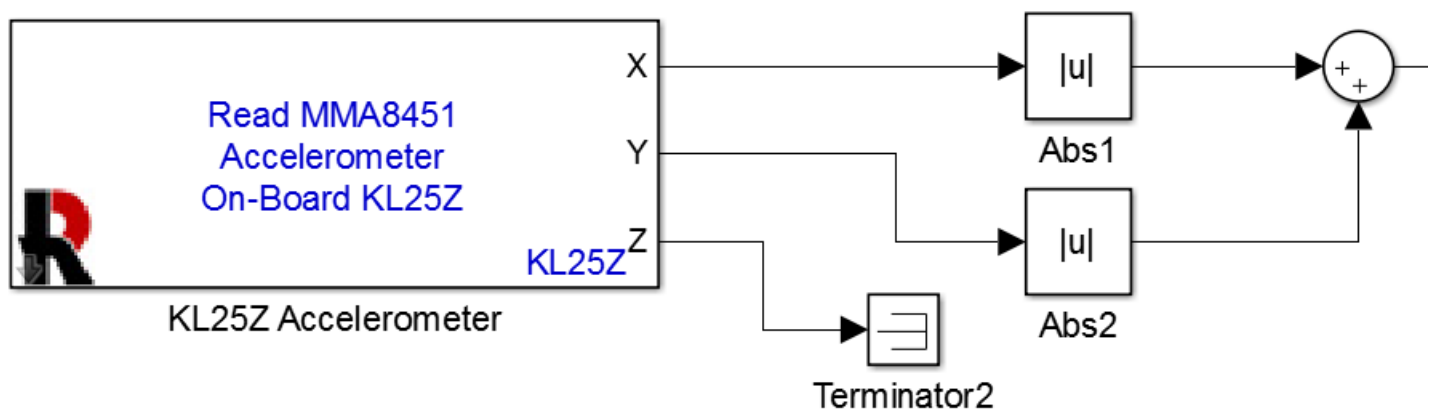
Exercise XV-23: Test this method for different values of the switch threshold. (We set it to 10 in the above example.) Lowering the value means that it will throw out many more values of the index that would otherwise be valid. Making it higher means that more signals that would otherwise be invalid are used. (Lowering the threshold means that it holds more. Raising the threshold makes the system act more like it did before we added the memory.)

L. Crash Detection

The KL25Z has an on-board accelerometer. We can use this sensor to measure changes in acceleration and detect a crash. When your vehicle crashes it will have a large negative acceleration. This is easy to detect. One problem is that the accelerometer will measure large accelerations around a curve (centrifugal forces) and when the vehicle bumps up and down on the track as it is moving.

The accelerometer measures acceleration in three dimensions. The z-axis is up and down and only measures vibration in our case. The x- and y-axes are in the same plane as the vehicle's motion. To verify this, rerun the accelerometer model that we tested in Section II.B starting on page 29. We are interested in large decelerations in the x- and y-directions.

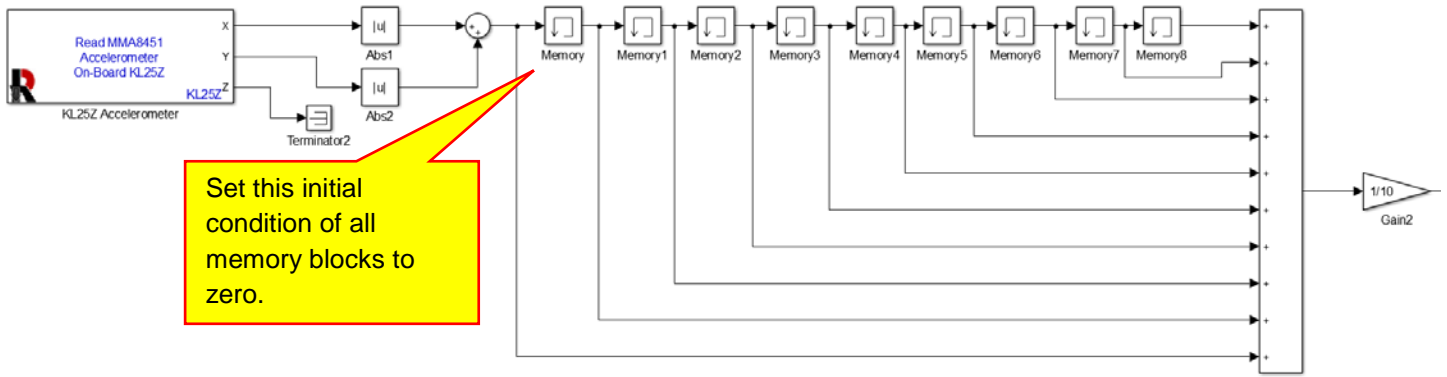
The outputs of the accelerometer is a signal between ± 2 corresponding to $\pm 2g$ of acceleration. If the car was not moving, the output of the x- and y-signals would be zero. (The z output would either be $+1$ or -1 depending on the orientation of the board.) When the vehicle crashes, the vehicle will decelerate rapidly. Since we do not know the orientation of the crash nor the orientation of the accelerometer (we could figure it out if we tried!), we do not know exactly what to expect from the accelerometer. However, we do know that we will read a large negative signal or a large positive signal on the x-axis accelerometer signal and/or a large negative signal or a large positive signal on the y-axis accelerometer signal. Since we don't know which signal will see the largest acceleration (we don't know the direction of the crash) and we don't know if the signal will be positive or negative (we don't know the orientation of the accelerometer), we will take the absolute value of the signals and add them together:



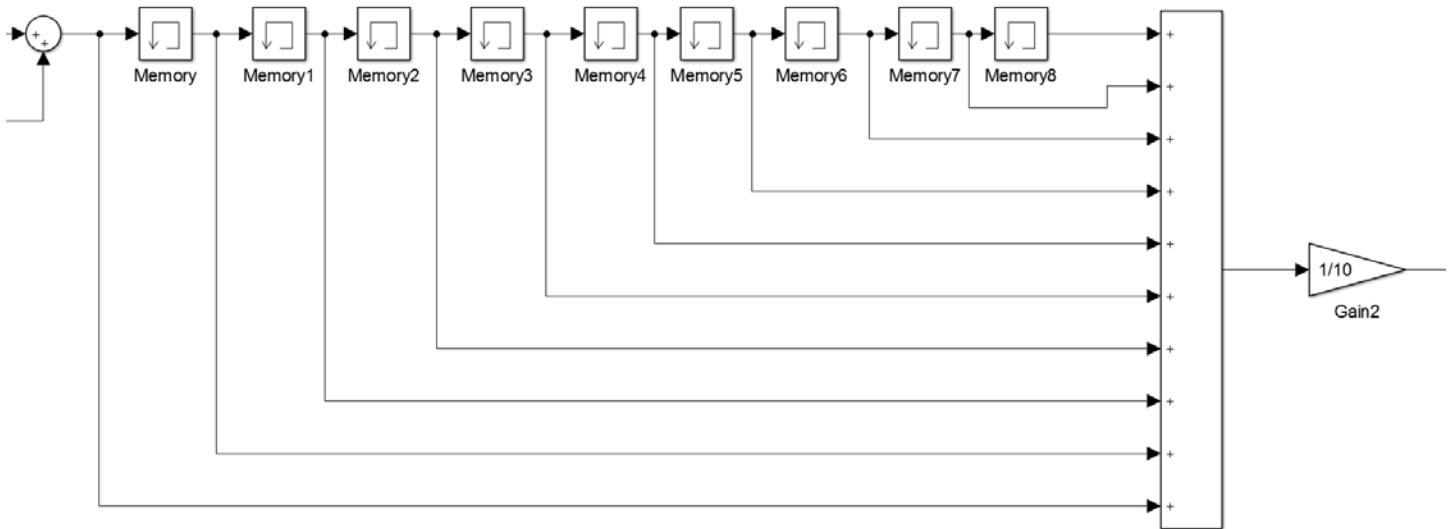
The output of the sum block will be non-zero if we experience a large deceleration in x-y plane.

In the normal operation of the car, it bounces around a lot and we expect the x-axis and y-axis signals to be non-zero just due to bumps on the track. To reduce the risk of a false trigger due to a bump on the track, we will take the average of the last 10 values output by the accelerometer. This way we can say that we have a crash if the last 10 values were high rather than a single value was high. A single value being high could be an error. 10 values in a row being high is probably not an error.

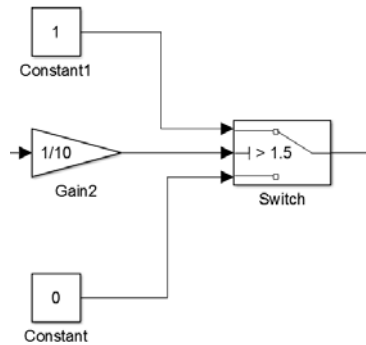
The **Memory** block is a one-step memory. The output of the block is the value of the input from the previous time step. Thus, the memory block remembers the last value of a signal. In the block diagram below, we have 10 **Memory** blocks which remember the last 10 values of output signal of the sum block. If we add these values up and divide by 10, we have a signal that is the average of the last 10 values:



We will zoom in a little further for clarity:

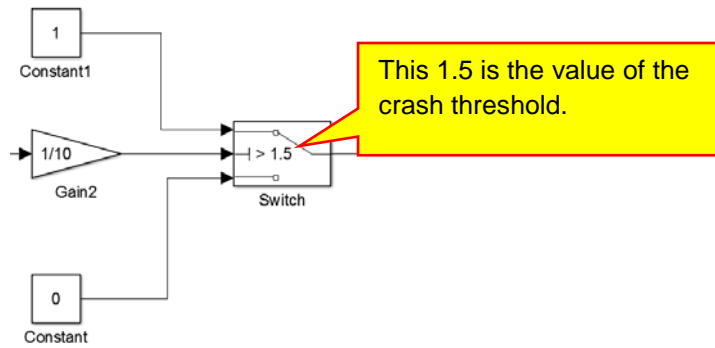


Next, we need to detect if the deceleration is above a threshold. If the output signal of the gain block goes above a specific threshold, we want the vehicle to stop moving, and we want it to remain stopped until we reset the vehicle. This requires us to detect that the signal is above a certain threshold and remember that we went above that threshold. The **Switch** block below detects when the signal goes above a threshold of 1.5.



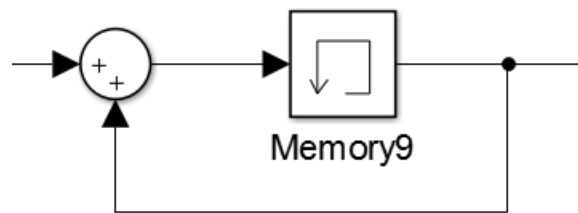
When the output signal of the **Gain** block goes above a threshold of 1.5, the switch will output a 1, indicating a crash. When the output signal is less than 1.5, the switch block will output a zero.

Note that the threshold in this example is 1.5. You can change the value of the threshold.

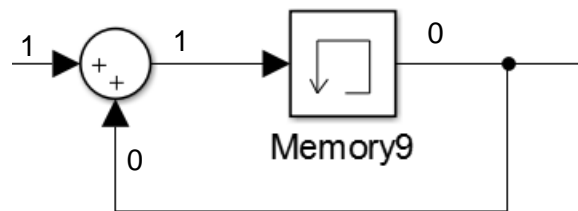


If you make the value of the threshold larger than 1.5, a more severe crash is needed to trigger the crash detection algorithm. If you make the threshold too small, the algorithm will erroneously trigger due to bumps and vehicle turns, and the vehicle may stop when it should not. You will need to experiment with the threshold.

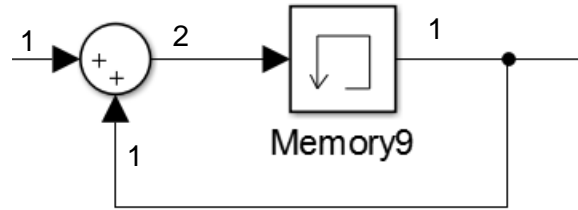
Next, once we detect a crash, we need to remember that a crash occurred. The vehicle should stop and stay stopped. Thus we need a memory. We will do this with the memory block and feedback below:



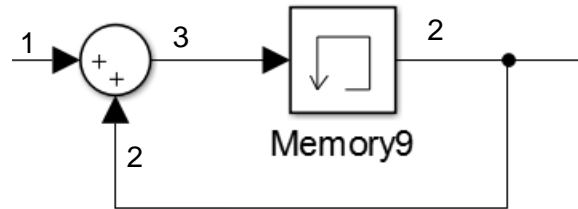
The output of a memory block is the input from the previous time step. When the input to the **Sum** block is a 1, this set of blocks counts up. This is easiest to see with some numbers shown. Suppose the memory block starts at zero (which it does because the default initial condition is 0) and the input to the **Sum** block is a 1. We have the situation below:



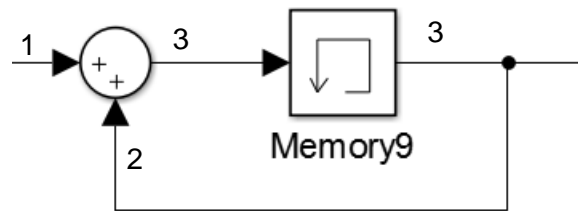
The input to the Memory block will be a 1. So, after the next time step, the output of the memory block will be a 1:



Now, the input to the memory block is a 2. After the next time step, the output of the memory block will be a 2:

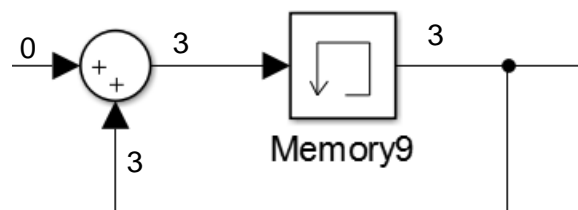


Now, the input to the memory block is a 3. After the next time step, the output of the memory block will be a 3:

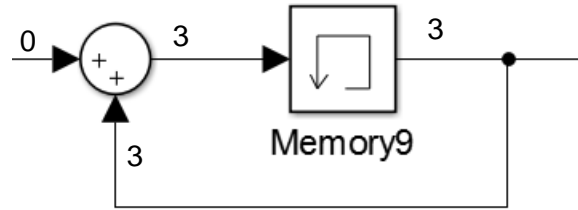


Thus, we see that if the input to the Sum block is a 1, this set of blocks counts up.

Next, suppose the memory block is at 3 and then the input to the **Sum** block changes to 0. We have the situation below:

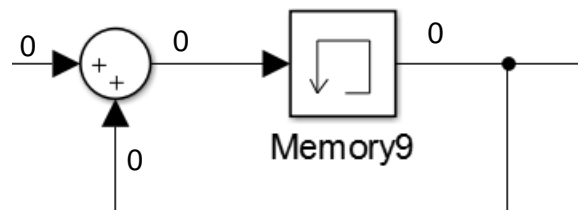


The input to the Memory block will be a 3. So, after the next time step, the output of the memory block remain at 3:

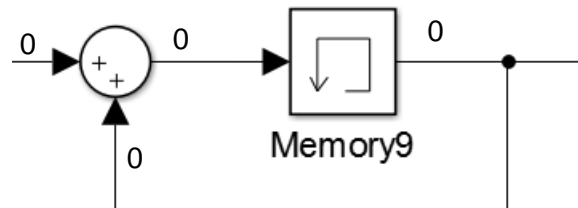


We see that when the input to the **Sum** block is zero, the count holds at its previous value.

Finally, suppose the memory block starts at zero (which it does when we cycle the power) and the input to the **Sum** block is a 0. We have the situation below:

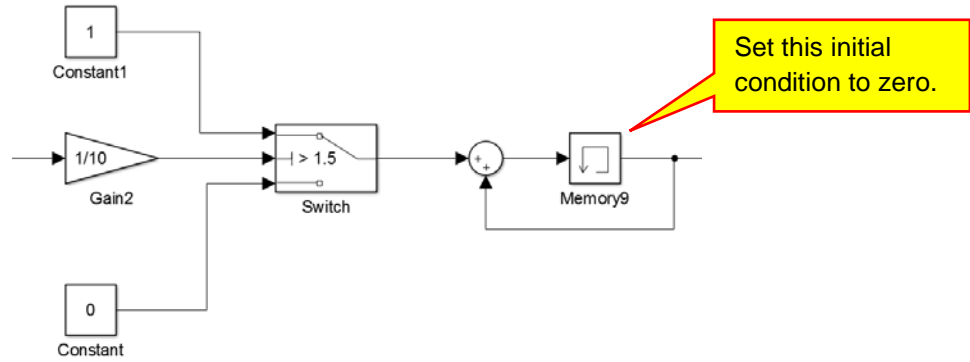


The input to the Memory block will be a 0. So, after the next time step, the output of the memory block remains at zero:



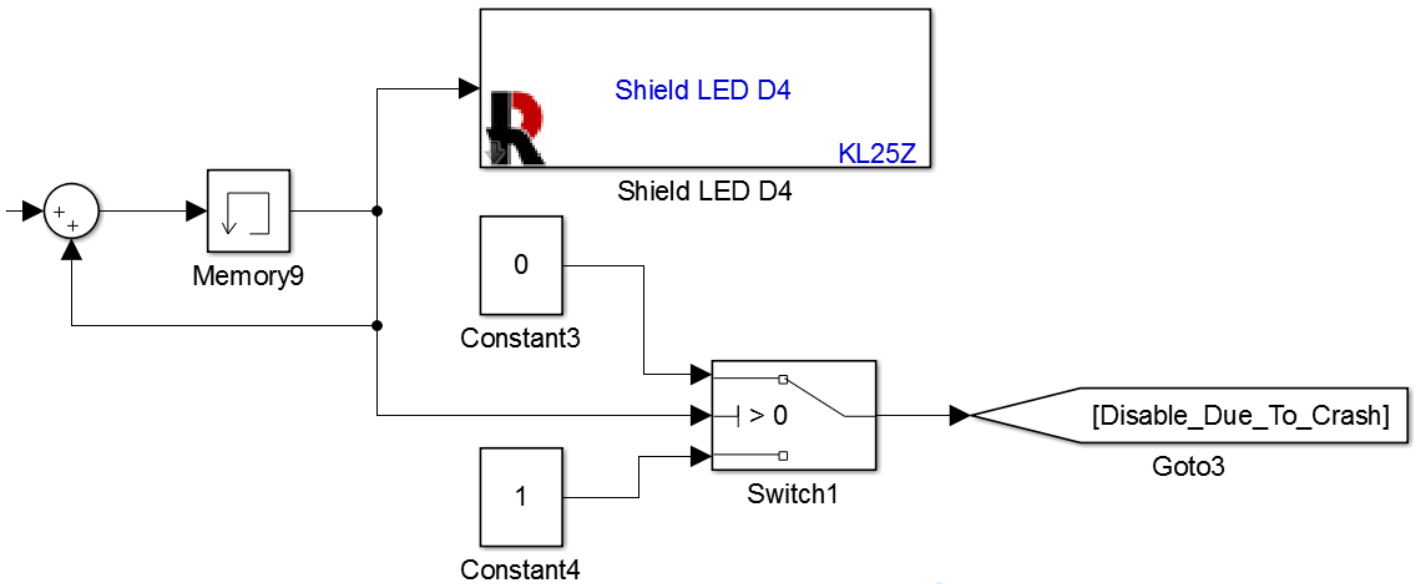
So we see that if the input is a 0 and the Memory block output is zero, the counter holds at zero.

The input to the **Sum** block comes from switch that detects a crash. When we detect a crash, the switch output is a 1. When there is no crash, the switch output is a zero:

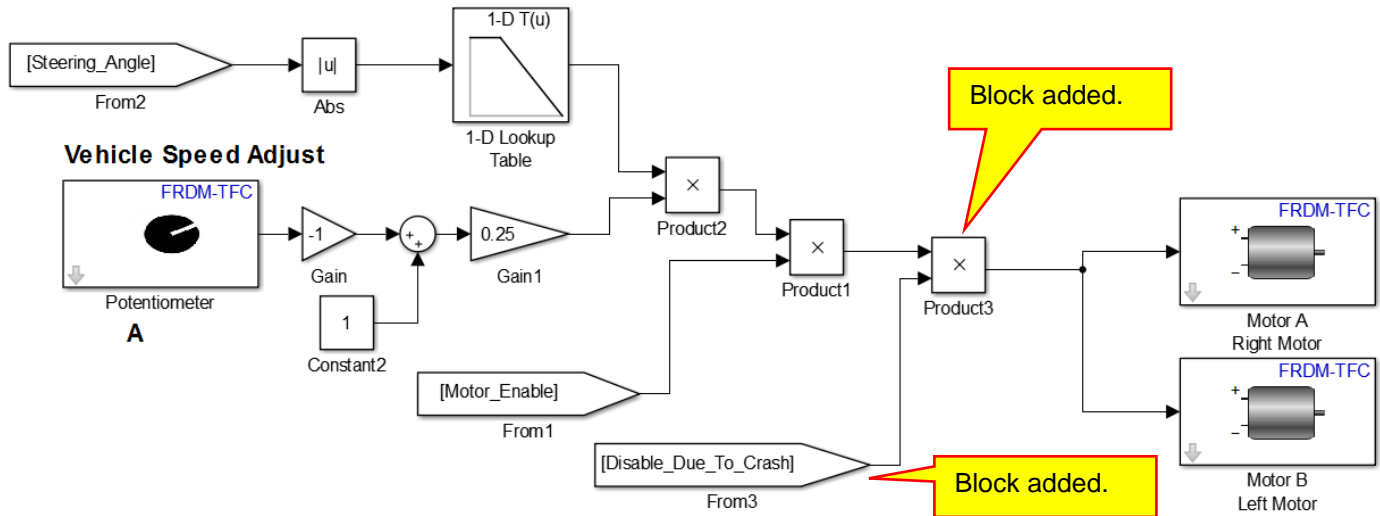


Altogether, the blocks function as follows. After power up and when there is no crash, the memory output is 0 indicating no crash. When a crash occurs, the counter counts up for as long as the crash takes to occur. When the vehicle finally stops, the counter holds at the last value of the count. The memory holds this value until we cycle the power. Thus, the counter indicates no crash when the count is zero. The memory indicates a crash when the count is greater than or equal to 1.

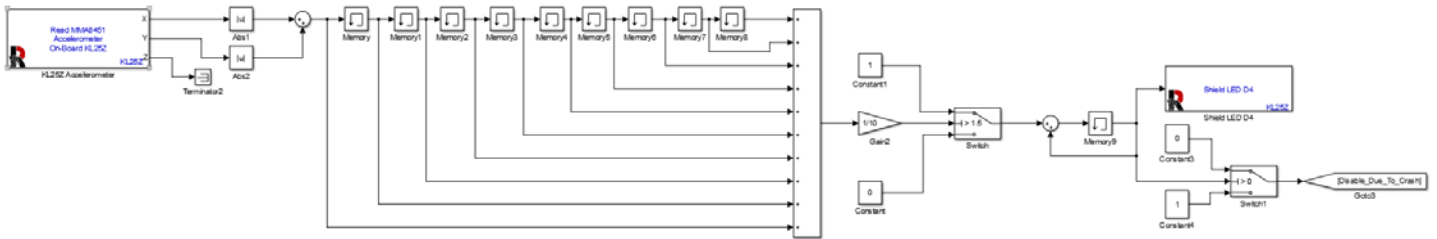
When we have a crash we want to do two things. (1) Light up an LED when a crash is detected so that we know the vehicle is not moving due to a perceived crash. (2) We want the drive wheels to stop. In the blocks below, LED D4 will illuminate when the count is greater than 0:



We have also used a switch to create a signal to disable the motors. When the count is zero (indicating no crash) the output of the switch is 1 which will enable the motors. When the count is greater than 0, the switch output will be 0 which will disable the drive motors. We used a **Goto** tag to rout the switch signal to the motors as shown below:



The entire crash detect logic is shown below, though it is hard to see:

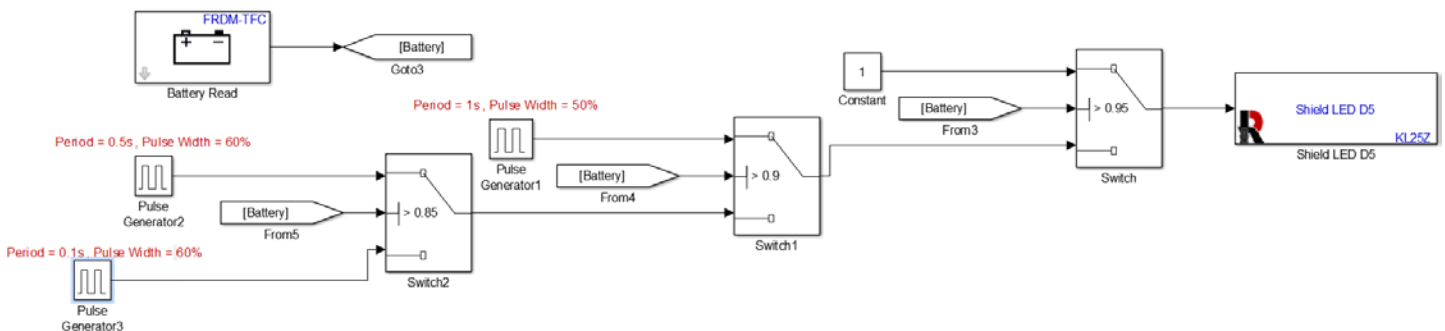


Demo XV-3: Test the crash detection system. First test it by shaking the vehicle. If you shake it violently enough, LED D4 should turn on and the drive wheels should stop spinning. The vehicle will not start again until you cycle the power.

Exercise XV-24: Determine an appropriate value for the crash detection threshold. It should not disable the vehicle for bumps and high speed curves. It should stop the vehicle after it crashes into something.

M. Battery Monitor

The set of blocks below is a battery monitor:



LED D4 is used to indicate the level of charge left in the battery. When near full charge, LED 4 is constantly illuminated. As the battery discharges, LED D4 begins to flash. The more discharged the battery becomes, the faster LED D4 flashes. Three thresholds are used to change the flashing rate.

Demo XV-4: Demo the operation of your battery monitor. Since the battery level does not change very much and it will be hard to test at different levels of the battery state of charge, we will test it at full charge and no

charge. Make sure your battery is fully charged. When you turn on the power to your vehicle, LED D4 should be illuminated constantly. Next, turn off the power to your car and plug in your KL25Z to the USB port. With the power off, the battery will appear to be fully discharged and LED D4 will flash at the fastest rate. We cannot test the other blocks in the battery level indicator until we use the vehicle for a while. However, once LED D4 starts flashing, you know that you should charge the battery soon.

Exercise XV-25: Discuss with your teacher how this set of blocks works. Include the signal output of the **Battery Read** block, the voltages this block reads, the thresholds of the switches, and the signal routing provided by the switches. What is the frequency of each of the pulse generators? (Remember that frequency is the reciprocal of the period.)