Simulink® and LEGO® MINDSTORMS® EV3

A brief workshop on Simulink support for Project Based Learning with LEGO MINDSTORMS EV3

Contents

Motivation for this workshop

As the industry, technology and society make rapid progress, there is a growing need to teach the next generation of engineers ever more complex concepts and build intuition quickly, so that they can apply their knowledge to develop the technology of the future. This calls for hands-on and project-based learning via low-cost, easy to use hardware and software platforms, to make it easier and fun to teach, learn and test the engineering ideas.

Simulink has long been the tool of choice of the industry (automotive, aerospace, electronics, etc.), since it provides a very user-friendly, collaborative and flexible environment for designing, simulating, testing and eventually implementing, complex, multi-domain systems and their control logic.

Starting 2012, Simulink includes the capability to program low-cost hardware like LEGO® MINDSTORMS® NXT and EV3, Arduino, Raspberry Pi, and BeagleBoard. This new capability enables students to develop and test a variety of controls, signal, image and video processing related applications, from within Simulink.

This workshop is based on Simulink Support Package for LEGO MINDSTORMS EV3. The participants will have a chance to work through lab modules with examples of obstruction detecting and line following robots. They will gain practical hands-on experience in building high-level examples themselves. Additionally, participating faculty members would have a chance to understand the potential for use in classrooms with students.

At the end of this workshop, the participant will be able to:

- design, simulate and test custom algorithms in Simulink
- implement these algorithms on low-cost embedded hardware such as LEGO MINDSTORMS EV3 without writing any C-code
- see how easy it is to program low-cost hardware with Simulink

Getting started

If you are new to MATLAB®, Simulink or Simulink Support Packages, take a look at the following introductory material to get started:

- To learn more about MATLAB and Simulink, check out interactive tutorials at: http://www.mathworks.com/academia/student_center/tutorials/
- For the latest information about LEGO MINDSTORMS NXT Support from Simulink, see: http://www.mathworks.com/hardware-support/lego-mindstorms-ev3-simulink.html
- Supported hardware for project based learning: http://www.mathworks.com/academia/hardware-resources/

How to use and work with this manual

I1. Workshop run-through options

While working through this manual, there are several entry points that you can choose from.

- To understand manual notation, we recommend reviewing intro sections I1 and I2.
- Depending on your Simulink skills level, you might start with example problem P0.1, or with the first self-constructed model P1.2.
- Reference material for frequently used tasks is listed in Appendices 1 and 2 for your convenience.

* Start at the section relevant to your Simulink experience level:

Section	Focus	Level	Recommend Experience
Project 0	Introduction to Simulink	Beginner	None
Project 1.1, 1.2	Intro to programming EV3 with Simulink	Beginner	None
Project 1.3, 1.4	Designing and implementing intelligent behavior with Simulink	Intermediate	Simulink
Project 1.4	Remote control of EV3 from Simulink	Intermediate	Simulink, SSP* for EV3
Project 2.1, 2.2	Designing complex behavior	Intermediate	Simulink, SSP for EV3, knowledge of controls
Project 2.3	Using MATLAB code in Simulink models	Intermediate	Simulink, SSP for EV3, Basic Control Theory
Project 2.4	State based system design with Simulink	Advanced	Simulink, SSP for EV3, Basic State Machines
Optional Project 1	Plant Modeling and Parameter Tuning	Advanced	Physical Modeling
Optional Project 2	Controller Design and Optimization	Advanced	Feedback controller design
Optional Project 3	Event-driven System Design	Advanced	State-machines

* SSP = Simulink Support Package

I2. Notation and formatting

- All required information is formatted as standard text on white background.
- Important information is highlighted with a grey background.
- Buttons on the brick and in Simulink are displayed as such or bracketed, e.g., **[OK]** for the OK button or **[Deploy to Hardware]**.
- Menu navigation in Simulink is shown as a sequence,
  e.g., **Tools > Run on Target Hardware > Prepare to Run**
- MATLAB code to be run / copied, is included in the document as below:

```
xlabel('Time [sec]');
```

Project 0: Getting Familiar with Simulink

Simulink

Simulink® is a block diagram environment for multi-domain simulation and Model-Based Design. It supports system-level design, simulation, automatic code generation, continuous test and verification of embedded systems. Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB®, enabling you to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis.

Key Features:

- Graphical editor for building and managing hierarchical block diagrams
- Libraries of predefined blocks for modeling continuous-time and discrete-time systems
- Simulation engine with fixed-step and variable-step ODE solvers
- Scopes and data displays for viewing simulation results
- Project and data management tools for managing model files and data
- Model analysis tools for refining model architecture and increasing simulation speed
- MATLAB Function block for importing MATLAB algorithms into models
- Legacy Code Tool for importing C and C++ code into models

With the help of code generation products like Simulink Coder and Embedded Coder, design logic in Simulink models can also be converted to C-code optimized for specific embedded platforms.

Simulink Library Browser and Simulink Models

To get started with Simulink, open the Simulink Library Browser. To do this, click on the Simulink button on the Home Tab of MATLAB Desktop, or type **`simulink`** at the MATLAB command prompt. Then, create a Blank Model and click on the Library Browser button. You may also open the Library Browser by entering **`slLibraryBrowser`** on the MATLAB Command Window.

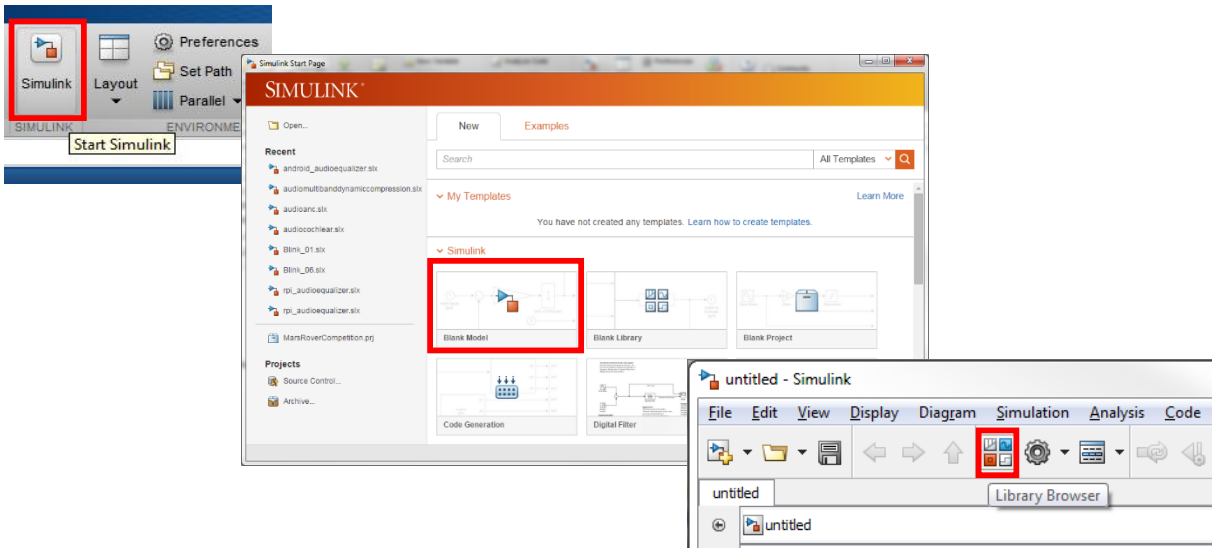

Figure 1: Starting Simulink and opening the Simulink Library Browser

The Simulink Library Browser is a collection of high level blocks that you can use to create a block-diagram representation of the system you are trying to design. From a different perspective, these blocks allow you to access or generate, apply algorithms and visualize or save the processed data or information, which flows through the system.

Once the Simulink Library Browser launches, you will see a window like Figure 2 below. Depending on the products included in the MATLAB installation, you will see some or all of the block libraries. In particular, we will work with the Simulink Support Package for LEGO MINDSTORMS EV3 Hardware, which is highlighted below (Figure 3).

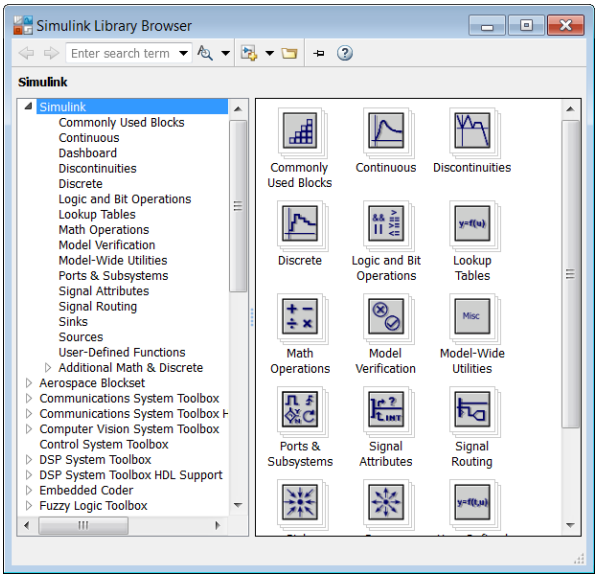

Figure 2: Simulink Library Browser

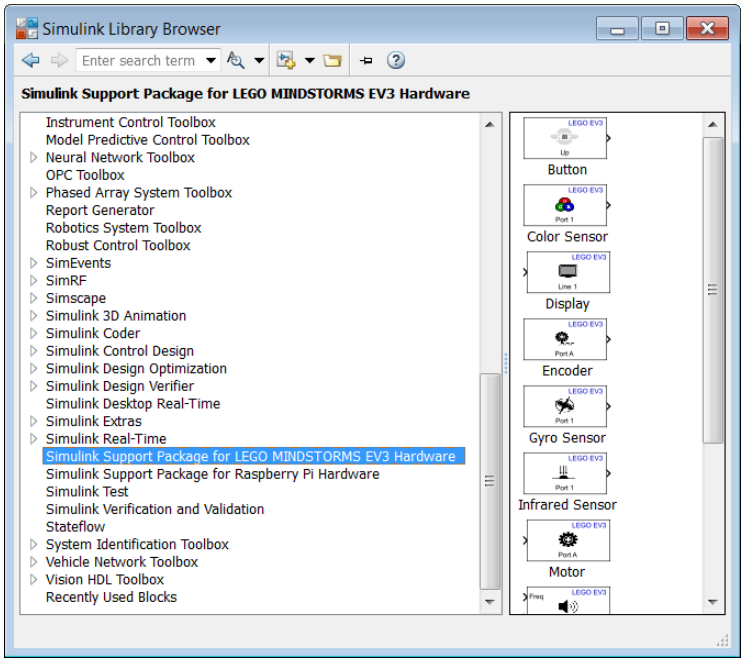

Figure 3: Simulink Support Package for LEGO MINDSTORMS EV3

A Simulink Model (Figure 4) represents the high level design of a system or an algorithm. You can create models by dropping blocks from the Simulink Library. After that, you can run the simulation or deploy it to the hardware. Figure 4 shows the simulation model of a self-balancing LEGO robot. You can explore this model from 'OptPrj2\selfBalancing\EV3_Sim_Balance_Move.slx' in the workshop folder. Instructions on how to run this model are specified at the end of the Optional Project 2 section, page 53.

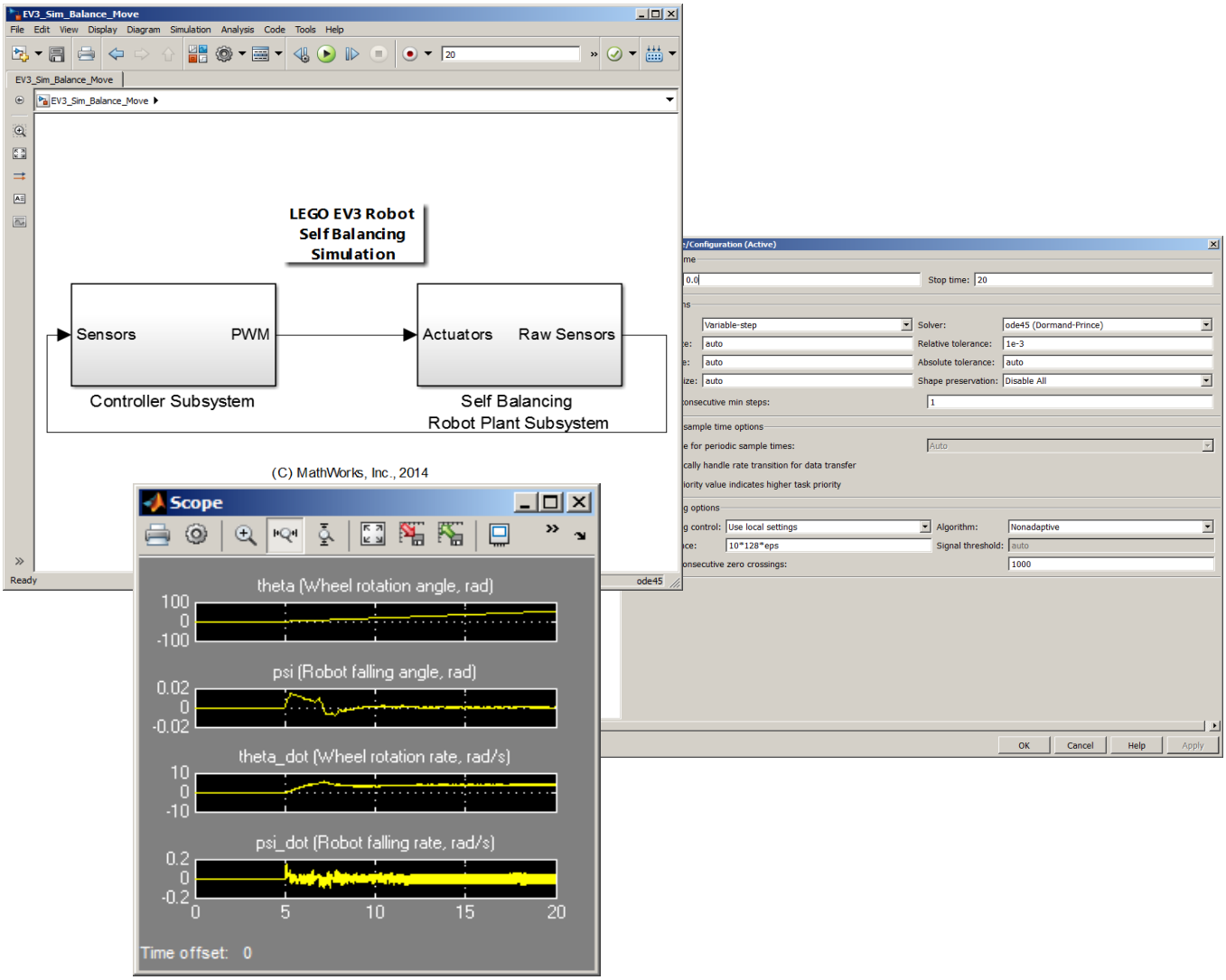

Figure 4: Different parts and simulation of a Simulink model

More on Simulink

If you want to try out some more examples, go to Simulink documentation under:

MATLAB Desktop > Help (Product Documentation) > Simulink > Examples

Project 1: Explore Simulink and LEGO MINDSTORMS EV3

P1.1 Get Started: Program EV3 Status Light with Simulink

Motivation

At the end of this project you will be able to program an EV3 brick from Simulink.

Objective

- Create first model in Simulink
- Check hardware and software installation

Tasks/Challenge:

- Make Green Status Light on the EV3 brick blink.

Solution

1. Create a Simulink model like the one in Figure 5
2. Deploy it to the EV3 brick over Wi-Fi.

When this Simulink model is deployed to EV3, a constant input value will be applied to the Status Light, which will controls its color and blinking.

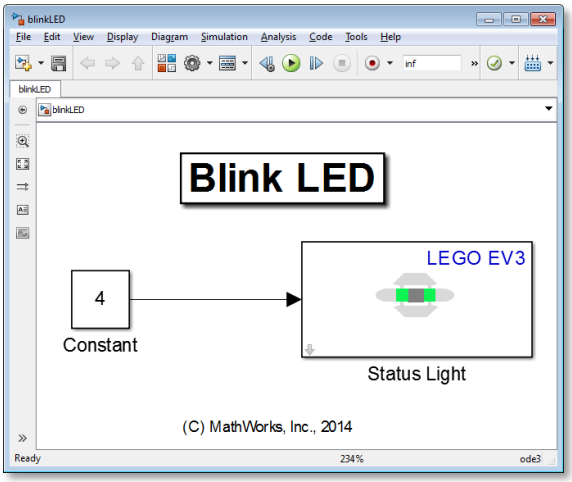

Figure 5: Project 1.1: Blink LED

Steps/Approach:

Create Simulink model

1. Open the Simulink Library Browser
2. Create a new model and save it as blinkLED.slx in the Working directory
3. Find the blocks listed in the table below
4. Drag and drop them into blinkLED.slx window
5. Double click on a block to open its Block Dialog Box
6. Replace/set block properties in the dialog box as listed in the table below

LIBRARY	BLOCK	PROPERTY	SETTING/VALUE
Simulink > Sources	Constant	**Constant value:**	4
Simulink Support Package for LEGO MINDSTORMS EV3 Hardware	Status Light	NA	NA

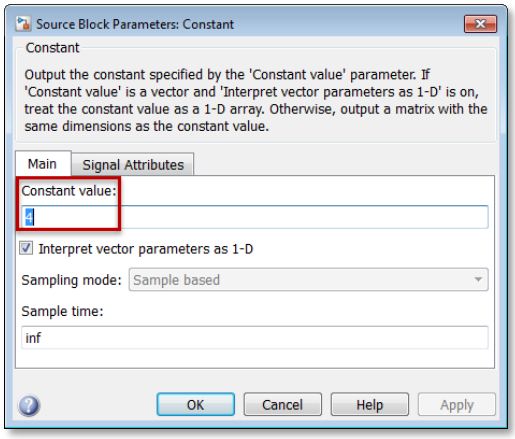

Figure 6: Constant Block Parameter

Prepare Model for Deployment to EV3 Brick

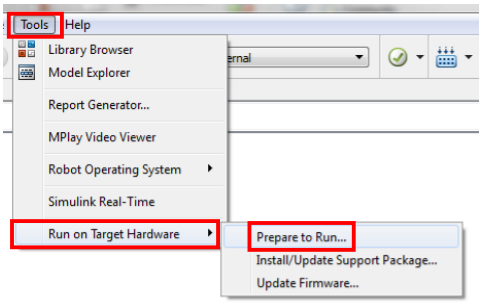

Figure 7: Prepare to run

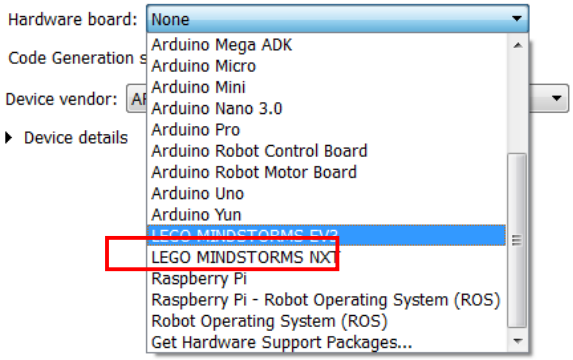

Figure 8: Select target hardware

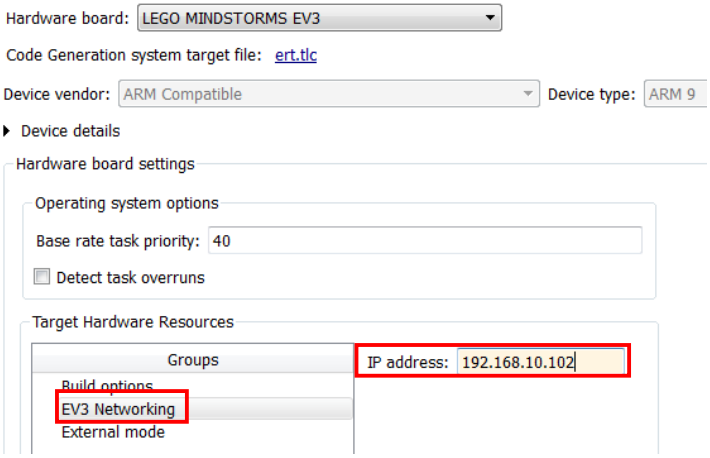

Figure 9: Set EV3 IP Address

To find the IP Address of the EV3 Brick, on the brick, go to:

a. Settings Tab > Brick Info > IP Address **OR**

b. Settings Tab > WiFi > [Click] on the network to which the brick is connected

**** Note that the IP Address of the EV3 Brick might change every time it is connected to the network***

Deploy Simulink Model to EV3 Brick

To deploy the Simulink model to EV3 Brick, just click on the 'Deploy to Hardware' button. Figure below shows what it looks like.

Figure 10: Deploy Simulink model to EV3

Observe

- The green status lights should start blinking

Next Steps

- Use different integer values in the Constant block and see how the Status Lights change color or behavior
- For detailed steps on getting started with Simulink Support Package for LEGO EV3, refer to the following tutorials on MathWorks website:
    - Getting Started with LEGO® MINDSTORMS® EV3™ Hardware: http://www.mathworks.com/help/legomindstormsev3/examples/getting-started-with-lego-mindstorms-ev3-hardware.html
    - Communicating with LEGO® MINDSTORMS® EV3™ Hardware: http://www.mathworks.com/help/legomindstormsev3/examples/communicating-with-lego-mindstorms-ev3-hardware.html

If you are curious…

Q1. How do I test Wi-Fi connection of my PC with the EV3 Brick? ***(See Appendix 2 for more details)***

A1. To test Wi-Fi connection, run the following command at the MATLAB Command Prompt ***with current IP Address noted from Brick Info.***

```
>> h = legoev3('xxx.xxx.xxx.xxx')
```

If an EV3 Brick is found, you will get a valid handle H to the EV3 Brick, which will allow you to interact with the Brick programmatically.

```
h =
  legoev3 with properties:
    ipAddress: 'xxx.xxx.xxx.xxx'
```

Q2. What happens when the 'Deploy to Hardware' button is clicked?

A2. Simulink does the following for you:

- Check algorithm and model integrity – connections, block-settings, settings, data-types, etc.
- Automatically convert model to C-code and use appropriate drivers to create connections to sensors and actuators
- Cross-compile the code to run on EV3 Brick – which is running Linux
- Download compiled code to the brick
- Start the model on the brick

If you observe the bottom left corner of the Simulink model, you will be able to see the following messages being displayed:

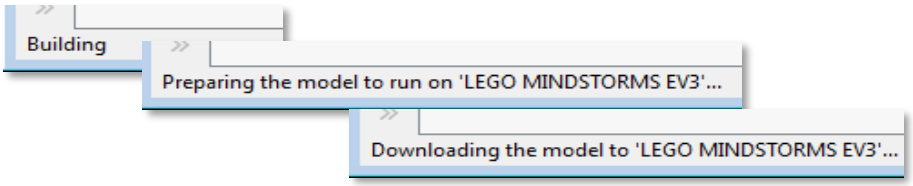

Figure 11: Steps that a Simulink model goes through when deploying to hardware

Q3. How do I stop a model running on the brick?

A3. To stop a deployed model running on the brick, you can do one of the following:

1. **On the brick**: Press the 'Back' button on the brick
2. **From MATLAB**: Call the `stopModel` method of the LEGOEV3 object, and specify the name of the model that is running. E.g.,

```
>> h.stopModel('blinkLED')
ans =
     0
```

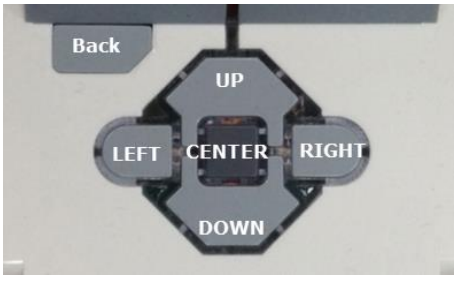

Figure 12: EV3 Buttons

Q4: How do I start a previously deployed model?

A4. On the EV3 Brick:

1. Navigate to Folders (second tab) > mw folder
2. Use UP and DOWN buttons to select the model that you want to start
3. Click on the center button to start the model

P1.2 Work with EV3 I/O: Access and Display EV3 Sensor Value

Motivation

At the end of this project you will be able to access and use EV3 sensor readings. In the next few projects, readings from the Ultrasonic and Light Sensors will be used to develop a line-following, collision-avoiding robot.

Objective

- Calibrate Ultrasonic and Color Sensor

Task/Challenge

- Display Ultrasonic/Color Sensor value on the brick's LCD

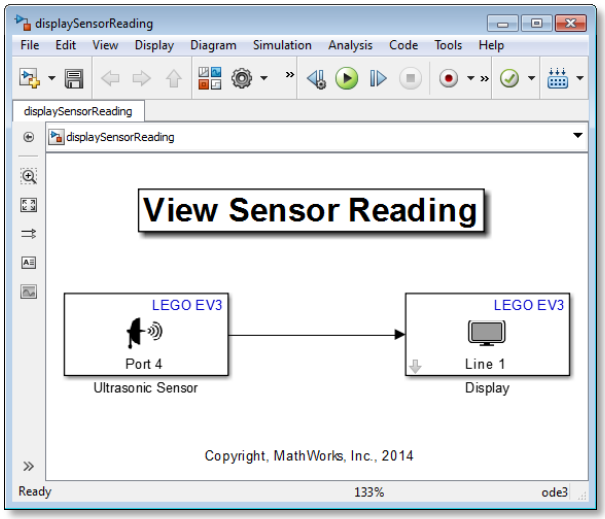

Figure 13: Working with EV3 I/O

Solution

- Build the following Simulink model
- Deploy model to Brick
- Observe sensor values under different distance and lighting conditions

When this Simulink model is deployed to EV3, the reading of the Ultrasonic distance sensor will be displayed on the LCD of EV3 Brick.

Steps/Approach

Create a Simulink Model

1. Create a new Simulink model and save it as displaySensorReading.slx in Prj1/Work folder.
2. Use the blocks, and appropriate settings, shown in the table below to fill in the empty model. The finished model should look like the one in the figure above.

LIBRARY	BLOCK	PROPERTY	SETTING/VALUE
Simulink Support Package for LEGO MINDSTORMS EV3 Hardware	Ultrasonic Sensor	**EV3 brick input port**	Check hardware
	Display	**Label** **Row**	As necessary As necessary

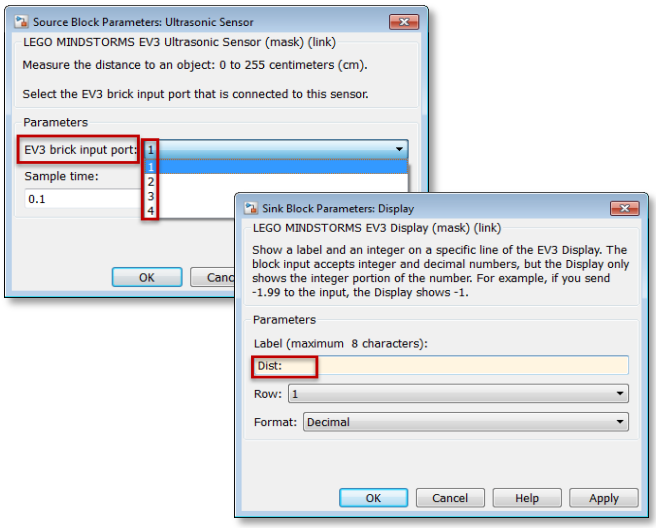

Figure 14: Simulink model block parameters for P1.2

Prepare model for deployment to EV3 brick

Follow the steps in P1.1

Deploy Simulink Model to EV3 Brick

Follow the steps in P1.1

Observe

- Observe distance value displayed on EV3 LCD when an object is placed at different distances in front of it.

Next Steps

a. Replace the Ultrasonic block with the Color Sensor block
b. Change 'Mode' to 'Reflected Light Intensity'
c. Observe and note down sensor reading when held over light and dark objects – these readings will be needed for the line-following robot.

If you are curious…

Q1. How do I add comments to a Simulink model?
A1. To add a comment to a Simulink model, just double-click anywhere inside the model. This will create a text annotation where you can type in and format text.

Q2. How do I change the name of a block in a Simulink model?
A2. When you drag a new block into a Simulink model, it will have the default name displayed just under the block. Click or double-click on that name to make it editable.

P1.3 Model, Test and Implement Collision Avoidance Behavior

Motivation

At the end of this project you will be able to model, test and simulate collision-avoidance logic in Simulink, all before deploying it on the EV3 brick. This will be used be used as part of a subsequent project.

Objective

- Make the robot intelligent so it avoids colliding with obstructions.

Tasks/Challenge

- Make the robot stop if it's too close to an object in front

Solution

1. Design intelligent behavior
2. Simulate and test the behavior
3. Prepare model for implementing on the robot. Final model should look like the one in the figure below.
4. Deploy model to robot

When this Simulink model is deployed to the EV3 Brick, the reading of Ultrasonic distance sensor will be compared to a threshold value. If the value is above the set threshold, the robot will move, otherwise the robot stops, and avoids colliding with objects in front of it.

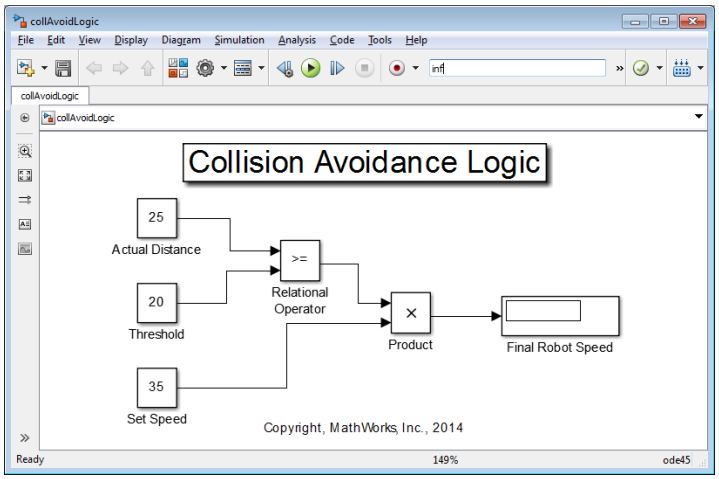

Figure 15: Collision avoidance logic

Steps/Approach

Design and Simulate Intelligent Behavior in Simulink

1. Open the model Prj1/Work/collAvoidLogic.slx
2. Click on the green RUN button and observe the Final Robot Speed
3. Fill in the following table, and confirm that the logic will work:

Actual Distance	Threshold	Set Speed	Final Robot Speed
10	20	35	
25	20	35	
30	50	35	
65	50	35	

Prepare model for deployment to EV3 Brick

1. Replace the 'Actual Distance' and 'Final Robot Speed' blocks by the 'Ultrasonic Sensor' and two 'Motor' blocks respectively. See details in the table below.
2. Save the modified model as collAvoidDeploy.slx. It should look like the model in the figure 15 above.

LIBRARY	BLOCK	PROPERTY	SETTING/VALUE
Simulink Support Package for LEGO MINDSTORMS EV3 Hardware	Ultrasonic Sensor	**EV3 brick input port**	Check hardware
	Motor	**EV3 brick output port**	Check hardware

3. Following the same steps as in P1.1, change the Simulink model settings to choose EV3 as the target hardware, and supply its IP Address.

Deploy Simulink Model to EV3 Brick

Follow the steps in P1.1

Observe

1. Let the robot run on the floor
2. Put an obstruction in front of it and make sure it stops
3. If the obstruction is removed, the robot starts moving again

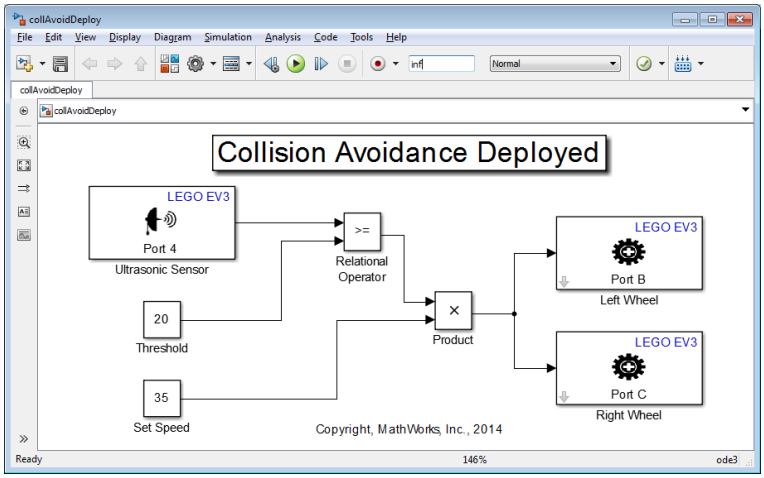

Figure 16: Collision avoidance deployed model

Next Steps

1. Change threshold value and re-deploy the model
2. Repeat step 1 above, till the robot stops within 10-15 cm of the obstruction. Try the following values:

Actual Distance	Threshold	Set Speed	Robot Behavior (Static/Moving)
10	**20**	**35**	
25	**20**	**35**	
30	**50**	**35**	
65	**50**	**35**	

If you are curious…

Q1. Why do I need to change 'Threshold' and 'Set Speed' values?

A1. For a desired performance and response from the robot, say if the robot needs to stop between 10-15 cm of the obstruction, there will be an optimal combination of the values of 'Threshold' and 'Set Speed'.

Q2. Does this have any practical use?

A2. Yes. To achieve a desired system response, the parameters governing the behavior of the system have to be "tuned". This kind of parameter tuning is needed for the optimal performance of every system or product being designed, from automobile and plane engines, to washing machines and toasters. Depending on the product and its complexity, this can be a manual or automated process.

P1.4 Control interactively: Speed and Steering Control

Motivation

At the end of this project, you will be able to change control parameters, while the application is running on the robot. The ability to interactively tune and optimize system design while working with hardware is used by engineers in industry. Finally, in Project 2, you will use this technique to optimize the performance of the line-following robot.

Objective

- Interactively explore steering capabilities of the robot

Tasks/Challenge

While the robot is running,

- Change robot's speed
- Steer the robot

Solution

- Open the supplied Prj1/Work/speedSteerControl.slx model
- Run speedSteerControl.slx model in External Mode
- Change Nominal Speed and Differential values to change speed, steer and explore turning radius for robot

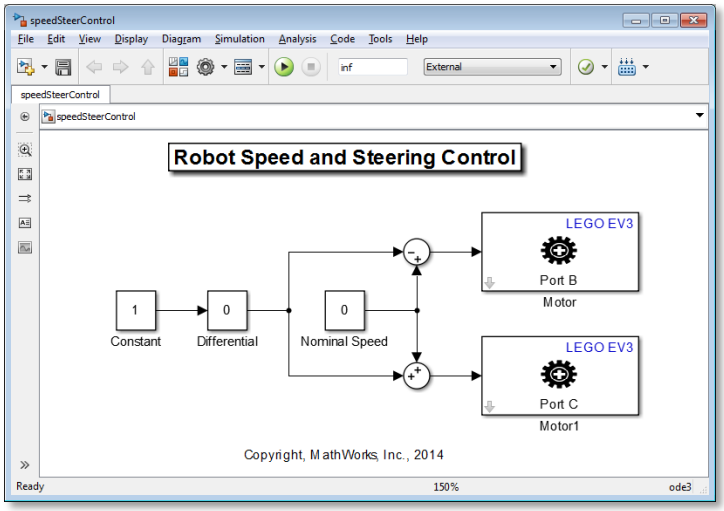

Figure 17: Robot speed and steering control

When this Simulink model is deployed to EV3 Brick, you will be able to change the robot's linear speed by changing the 'Nominal Speed' value, and make it turn by applying a non-zero 'Differential'.

Steps/Approach

Create a Simulink model

Open the supplied speedSteerControl.slx model:

Prepare model for running on EV3

1. Ensure that EV3 is selected and correct IP address is provided under **Tools > Run on Target Hardware > Options**
2. In the main model window, set **Simulation stop time** to **inf** in the edit box, and **Simulation Mode** to **External**, from the drop-down menu

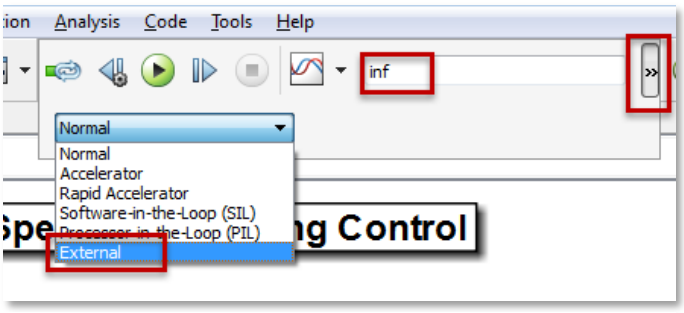

Figure 18: Setup for External Mode

* External Mode (**Figure 18**) allows you to run your model on the EV3, while still using Simulink on your host computer as an interface to interact with your model. This allows for live parameter tuning – you do not need to stop, edit

and restart the model. You just change the parameter (e.g. value in Nominal Speed block in **Figure 17**) and the parameter is automatically passed to the EV3 which uses the new parameter without pausing its operation. External Mode also allows you to log and view data while using Simulink window.

Note: Remember when using External Mode, you need to use the **RUN** button to deploy the model, and the **STOP** button to stop the model (**Figure 19**).

Note: If your model stops working after 10 seconds, you may have forgotten to adjust the Stop time in the model window (**Figure 18**).

Run model in External Mode

- Click on the **Run** button to deploy the model to EV3 Brick
- Click on the **Stop** button to stop the model execution on the brick

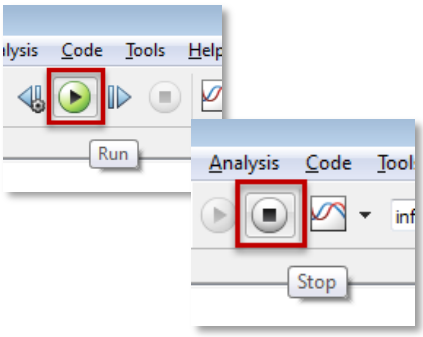

Figure 19: Running and stopping model in External Mode

Observe

1. In the bottom left corner of the model, observe the following messages:

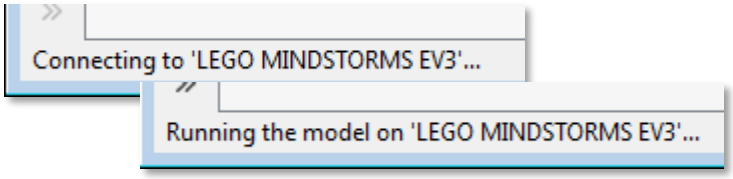

Figure 20: Simulink messages for External Mode

2. Also observe that a Windows System Prompt opens up – this indicates that a connection has been established:

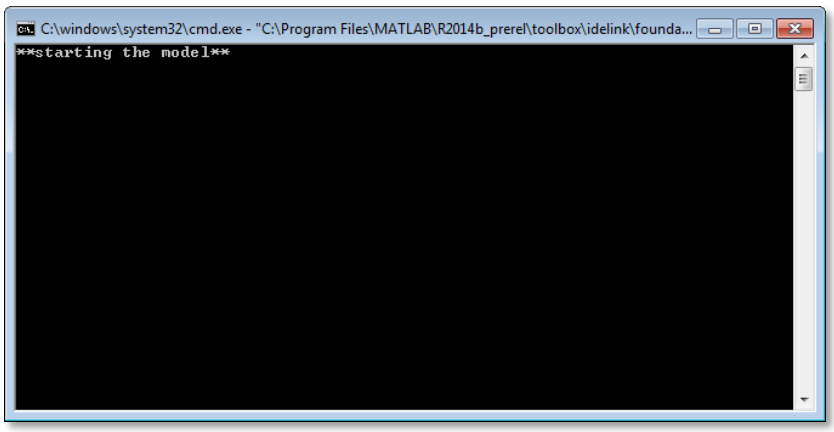

Figure 21: Message on system prompt when initiating External Mode

3. Once the model starts on the robot, a message is displayed:

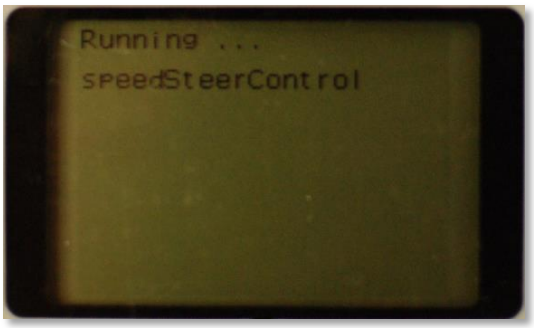

Figure 22: Message on EV3 LCD when running a model

Next Steps

1. Make the robot go straight
2. Make the robot turn left or right (or right)
3. Change turning radius of robot
4. Try to make the robot follow the supplied track

If you are curious…

Q1. What is External Mode?

A1. When you are developing algorithms, it is often necessary to determine appropriate values of critical algorithm parameters in iterative fashion. Simulink's External mode feature enables you to accelerate the process by letting you change certain parameter values while the model is running on target hardware, without stopping the model. When you change parameter values from within Simulink, the modified parameter values are communicated to the target hardware immediately. The effects of the parameters tuning activity may be monitored by viewing algorithm signals on scopes or displays in Simulink.

Project 2: Complex System Design with Simulink

P2.1 Line Following Robot

Motivation

At the end of this project you will be able to implement a simple line-following robot, using a feedback controller based on readings from the EV3 Light Sensor. With the help of External Mode operation, you will also be able to change controller parameters in order to tune the robot's motion. The controller parameter values will be needed in the next project for the line-following, collision-avoiding robot.

Objective:

- Make the robot automatically follow a line

Tasks/Challenge:

1. Create a feedback loop based on light sensor value
2. Automatically steer robot based on light sensor readings

Solution:

1. Open the supplied Prj2/Work/basicLineFollowing.slx model
2. Interactively tune controller parameters to make the robot follow the line

When this Simulink model is deployed on the EV3 Brick, robot runs in a proportional feedback control loop. Any deviation in light intensity value from the threshold creates a speed differential for the wheels, and makes the robot turn. Thus enabling it to follow a curved line.

Steps/Approach

Start from a supplied Simulink model

Open the supplied Prj2/Work/basicLineFollowing.slx model

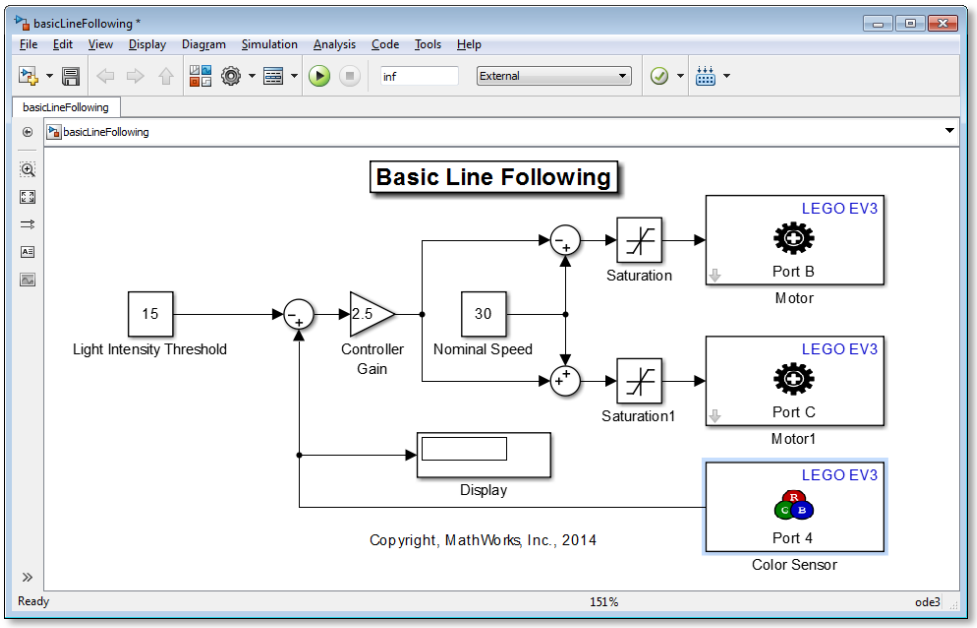

Figure 23: Basic Line Following

Prepare and run model in External Mode

Follow steps outlined in P1.4

Tune controller parameters

While running the Simulink model in External Mode, tune the following parameters

- Light Intensity Threshold
- Controller Gain
- Nominal Speed

A good line following robot:

- Should be able to go around the track in 20 seconds
- Should not oscillate too much

Next Steps

Note down the following values needed for a satisfactory line-following behavior. These will be needed in the projects to follow:

Parameter	Value
Light Intensity Threshold	
Controller Gain	
Nominal Speed	

P2.2 Line-Following and Collision-Avoiding Robot – With Base Simulink Blocks

Motivation

At the end of this project, you will be able to easily combine designs from multiple models into a single Simulink model. This makes it easy to distribute system design tasks, and collaborate in a team environment. The Simulink model at the end of this project will contain the design for Line-Following and Collision-Avoiding Robot.

Objective:

- Make the robot follow a line and also avoid collision with objects on the track

Tasks/Challenge

- Combine the line-following and collision-avoidance logic developed earlier

Solution

- Put both algorithms in the same Simulink model

When the completed model is deployed to the EV3 robot, it will follow a line, as well as avoid collisions with objects in front of it.

Steps/Approach

There are several ways to approach this problem. Here is a simple approach to start with.

- The logic for both line-following and collision-avoidance can be put in the same Simulink model
- Based on the readings of the Ultrasonic Sensor, a SWITCH block can be used to determine which logic is in action

Develop and test behavior with simulation

Open the supplied skeleton model followAndAvoidLogicBlocks_Start.slx and make necessary connections to support the following logic:

a. If there is no obstruction in front of the robot, the robot follows a line
b. If there is an obstruction in front of the robot, the robot stops

For more information on how to use the SWITCH block:

- Read the information in the block's dialog box
- Click on 'Help' button in the dialog box to open up documentation on the block

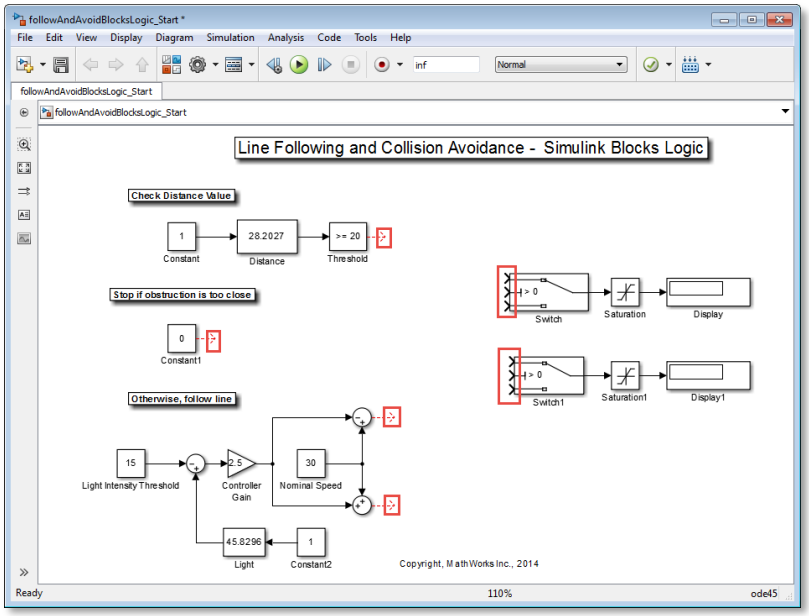

Figure 24: Skeleton model for Line Following with Collision Avoidance behavior

Prepare model for deployment to EV3 brick

Make appropriate block substitutions, and changes to model parameters in order to implement the new logic on the EV3 brick.

The final model should look like this:

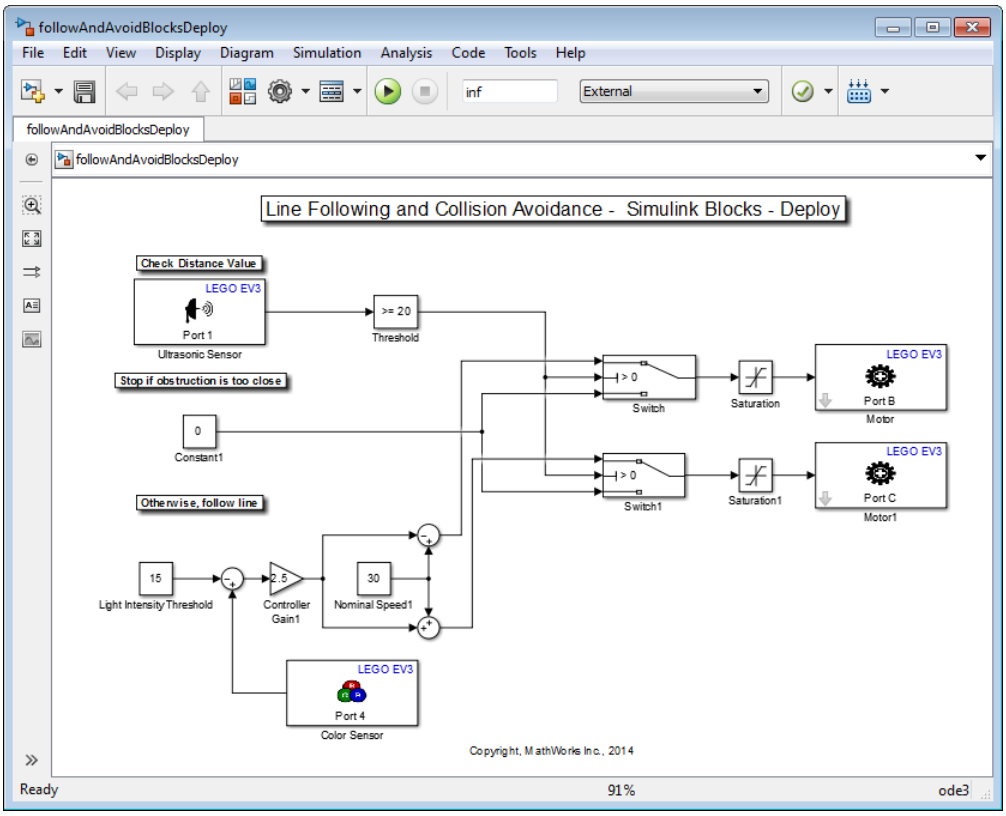

Figure 25: Completed line following, collision avoidance behavior

P2.3 Line-Following and Collision-Avoiding Robot – With User-Defined MATLAB Function Block

Motivation

At the end of this project, you will know how to use the MATLAB Function Block to include MATLAB code inside a Simulink model. Using the MATLAB Function Block is a programmatic approach which complements the block-diagram based approach in Simulink.

Objective:

- Use programmatic approach to design and implement intelligent behavior

Task

- Use MATLAB code to implement the line-following, collision-avoidance

Solution

- Use MATLAB Function block in a Simulink model

Steps/Approach

Create Simulink model

1. Open the supplied Prj2/Work/followAndAvoidMLFcnLogic_Start.slx model

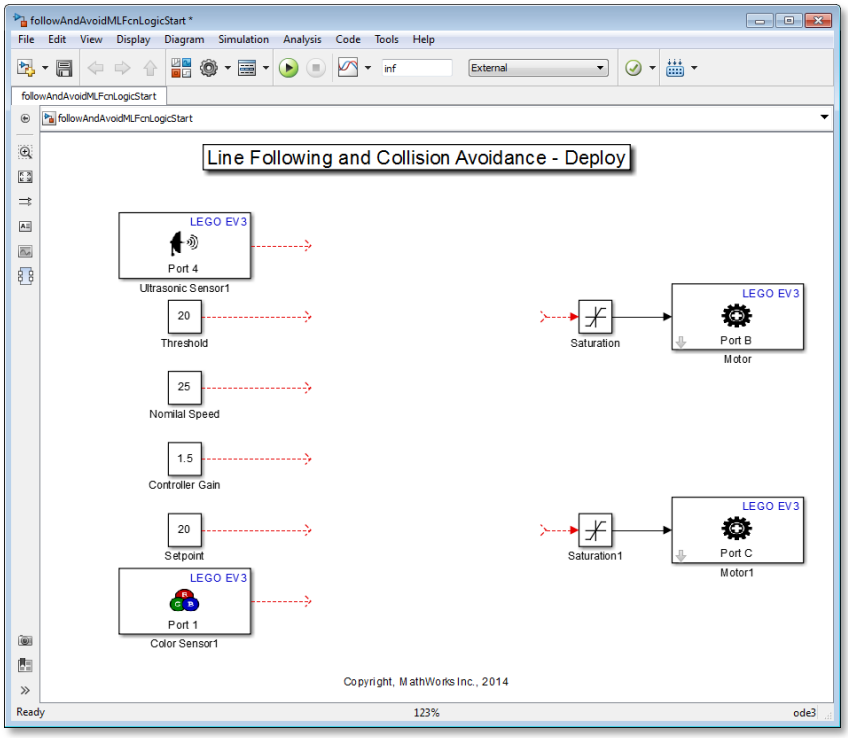

Figure 26: Using MATLAB code in Simulink model

2. Add the MATLAB Function block from **Simulink > User Defined Function** library to the model

3. Double click on the block to open the code editor, and type in the following MATLAB Code

```
function [leftMotor, rightMotor] = control(distVal, distThresh,
baseSpeed, gain, setpoint, sensedLight)
%#codegen

if distVal > distThresh % Compare obstruction distance to threshold
    err = sensedLight - setpoint; % Line following logic
    leftMotor = baseSpeed - gain*err;
    rightMotor = baseSpeed + gain*err;
else
    leftMotor = uint8(0); % Collision avoidance logic
    rightMotor = uint8(0);
end
```

4. Choose appropriate values for Threshold, Nominal Speed, Controller Gain and Setpoint.
5. After saving the model and making appropriate connections, the final model should look like the one below:

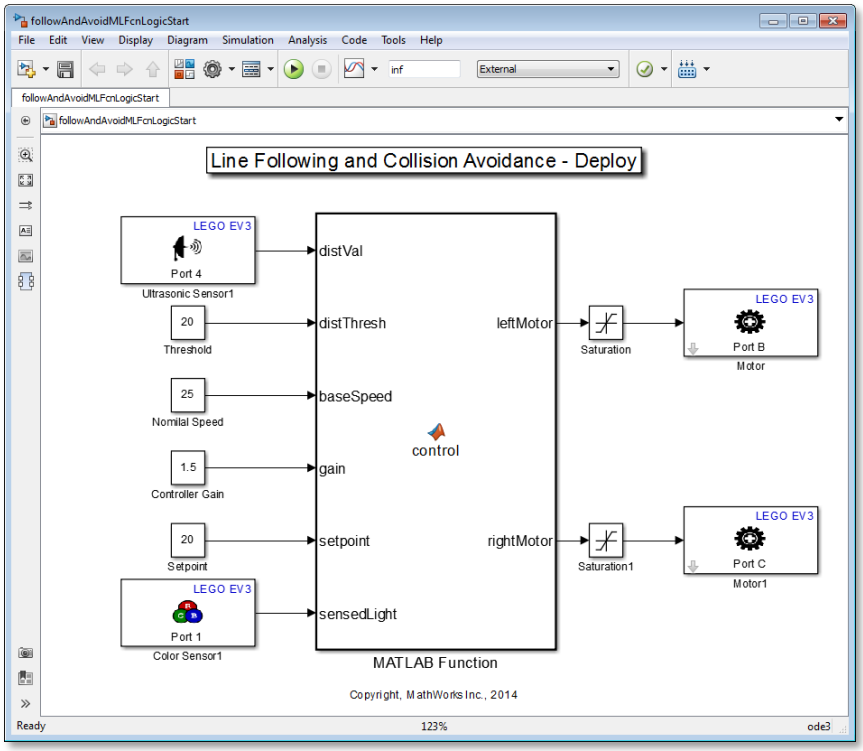

Figure 27: MATLAB Function block in Simulink model

Prepare the model for deployment to the EV3 Brick

- Make necessary changes so that the model can run on the EV3 brick in External Mode

Observe

- Confirm that the behavior of the robot is exactly same as before.

If you are curious…

Q1. What is the meaning of **%#codegen** in the code inside MATLAB Function block
A1. The presence of this keyword tells MATLAB that the code in the current file will be translated to C-code. The presence of this keyword alerts the MATLAB Code Analyzer about the intended use of the file, and initiates automatic checks to ensure integrity of the automatically generated C-code.

Q2. Which MATLAB functions can be translated to C-code?
A2. For a list of functions that can be used in MATLAB Function block, see: http://www.mathworks.com/help/simulink/ug/functions-supported-for-code-generation--alphabetical-list.html

Q3. Can I see the C-code generated for a Simulink model?
A3. Yes. If you have access to the Simulink Coder and Embedded Coder products, you can see the C-code generated from a Simulink model.

P2.4 Line-Following and Collision-Avoiding Robot – Event-driven modeling

Motivation

At the end of this project you will be able to use the MATLAB Function Block in a Simulink model to develop control logic for an event-driven system.

Many systems operate in different modes of operation, or states that require different parts of the system to react. For example, a car with automatic transmission gets data from the shift lever, and uses logic to select the mode of operation, which then controls the motion – forward, reverse or parking.

For this event-driven system, we will use a state machine which is a model of a reactive system. This model defines a finite set of states and behaviors and how the system transitions from one state to another when certain conditions are true.

Objective:

- Use a programmatic approach to building control logic using finite-state machines and allowing for event-driven state transitions.

Task

- Use MATLAB code to implement line-following, collision-avoidance and programming a touch sensor to act as an on-off switch, allowing the robot to transitions between the following states based on specific events:
    - State 0: Stop - action triggered by a touch sensor.
    - State 1: Forward motion with line-following in the absence of an obstacle – action triggered by the touch sensor and the absence of an obstacle.
    - State 2: Reverse motion – triggered by the presence of an obstacle.

Solution

- Use MATLAB Function block in a Simulink model and persistent variables to define and allow for state transitions based on sensor inputs.

Steps/Approach

Create Simulink model

1. Open the supplied Prj2/Work/followAndAvoidBlocksDeployMLFcnEvent_Start.slx. See Figure 27.
2. Add the MATLAB Function block from Simulink > User Defined Function library to the model
3. Double click on the block to open the code editor, and type in the MATLAB Code on Page 30.
4. Make desired changes to parameters governing the controller behavior – distance thresholds for obstacle avoidance and response to obstacle – if necessary.
5. Choose appropriate values for Threshold, Average Speed, Controller Gain and Setpoint.
6. After saving the model and making appropriate connections, the final model should look like the one in Figure 29.

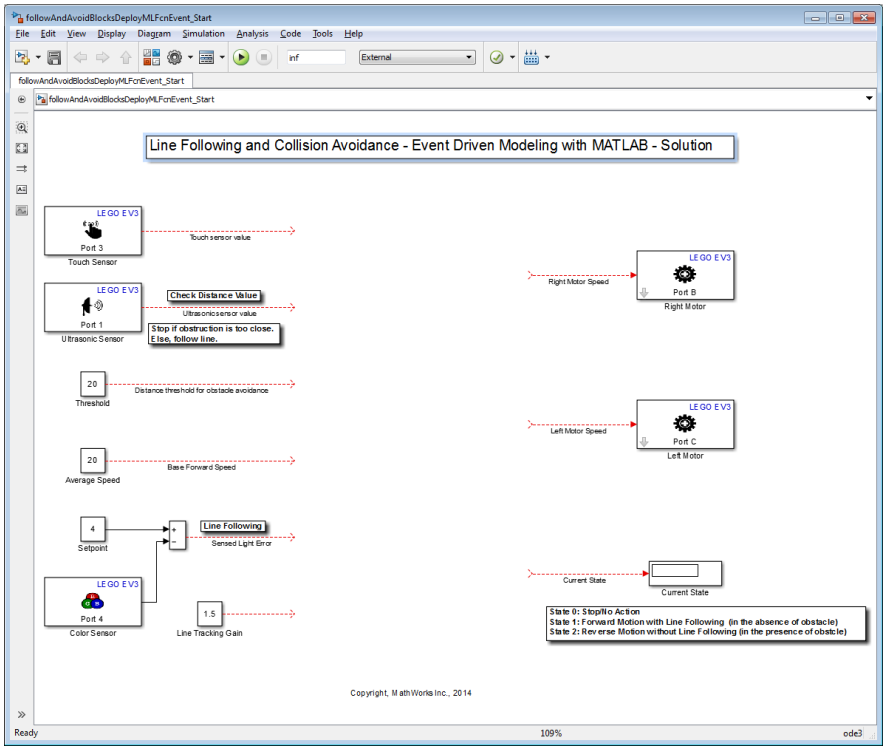

Figure 28: Using MATLAB code in Simulink model

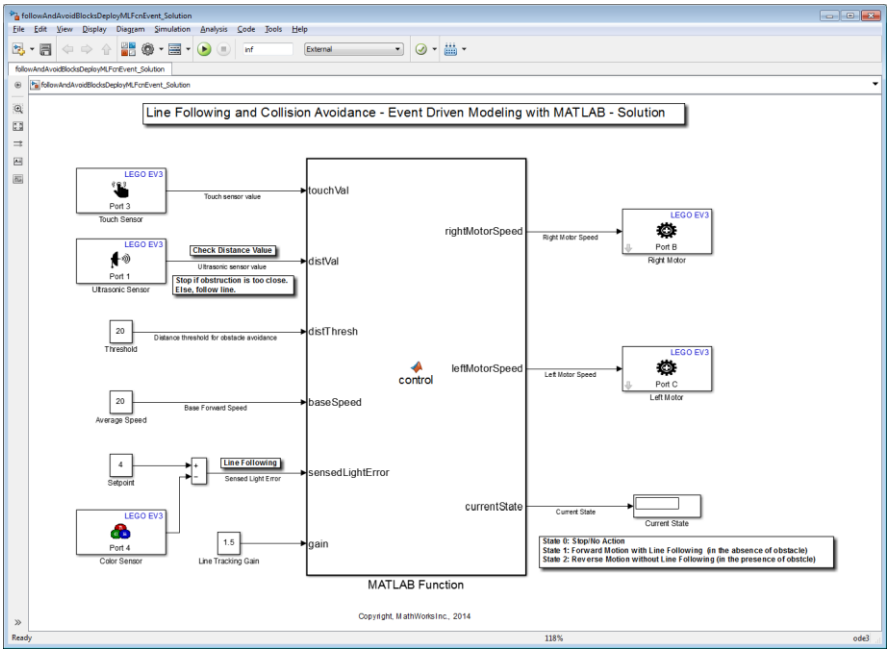

Figure 29: MATLAB Function block in Simulink model

```
function [rightMotorSpeed, leftMotorSpeed, currentState] =
control(touchVal, distVal, distThresh, baseSpeed, sensedLightError,
gain)
%#codegen

% Initialize variables and set default values
persistent state
persistent status
if isempty(state), state = 0; end;
if isempty(status), status = 0; end;

% Convert datatype for calculations
sensedLightError = double(sensedLightError);

if status == 0 && state == 0 && touchVal == 1
    status = 1;
elseif status == 1 && touchVal == 1
    status = 0;
end

% Use the Touch sensor to trigger action

if status == 1 && touchVal == 0 && distVal >= 2*distThresh
    state = 1; % Start/Move Forward
elseif status == 1 && touchVal == 0 && distVal < distThresh
    state = 2; % Move Reverse
else
    state = 0;
end

currentState = state;

% Choose appropriate system state
switch state
    case 0 % Stop
        desiredSpeed = 0;
        desiredRotationRate = 0;
    case 1 % Move Forward
        desiredSpeed = baseSpeed;
        desiredRotationRate = gain*sensedLightError;
    case 2 % Move reverse
        desiredSpeed = -baseSpeed/2;
        desiredRotationRate = 0;
    otherwise % Do nothing
        desiredSpeed = 0;
        desiredRotationRate = 0;
end;

% Calculate motor speeds
rightMotorSpeed = desiredSpeed + desiredRotationRate;
leftMotorSpeed = desiredSpeed - desiredRotationRate;
```

Prepare the model for deployment to the EV3 Brick

- Make necessary changes so that the model can run on the EV3 brick in External Mode.

Observe

- Confirm that the behavior of the robot is similar to the earlier models, with the modified response to obstacles and the added functionality of a touch sensor acting as a power switch.
- Place an obstacle in the path of the robot (observable by the Ultrasonic Sensor) and observe the state transitions as indicated by the 'Current Status' display block and the corresponding system behavior.

If you are curious…

Q1. What are persistent variables?
A1. Persistent variables are variables that are local to the function in which they are declared; yet their values are retained in memory between successive time steps. Persistent variables are similar to global variables because the MATLAB® software creates permanent storage for both. They differ from global variables in that persistent variables are known only to the function in which they are declared. This prevents persistent variables from being changed by other functions or from the MATLAB command line. Persistent variables retain their values between simulation steps, and thus the same memory location is accessible throughout the simulation.

Q2. Is there a better way to program state transitions than a MATLAB function block?
A2. Developing complex decision logic triggered by specific events, or event-driven modeling, can be, and often is, accomplished by writing conventional code. This works well for simple controller behavior, but as the complexity of control action scales up, writing code in the traditional sense can get very tedious very quickly, and there are clear advantages that a more visual approach for representing and programming event-driven transitions provides.

Event-driven modeling, under such a visual framework, is facilitated by Stateflow®, an environment for defining events and state transitions towards representing complex control action. Stateflow allows for modeling and simulating combinatorial and sequential decision logic based on state machines and flow charts. Stateflow lets you combine graphical and tabular representations, including state transition diagrams, flow charts, state transition tables, and truth tables, to model how your system reacts to events, time-based conditions, and external input signals. The control logic and dynamic event-driven state transitions presented in this exercise could be replicated with greater ease and improved controller behavior visualization using Stateflow, as discussed and demonstrated in Appendix 3.

Optional Project 1: Plant Modeling for LEGO EV3 Robot

OP1.1 Simulation Model of EV3 Robot

Motivation

At the end of this section you will be able to understand robot motion behavior. For e.g. when does the robot move straight, when does it turn, by what radius etc.

Objective

- Develop a simulation plant model (mathematical model) of EV3 robot assembly

Tasks/Challenge

- Import existing CAD model into Simulink
- Add actuation to the robot wheels and observe robot motion

Solution

1. Import 'EV3_CAD.xml' into Simulink
2. Explore supplied model 'EV3sim_Openloop.slx', give it various inputs to see robot move around

Steps/Approach

The files required for this section are in the 'OptPrj1\plantModeling' folder.

Import the XML file into Simulink

1. Execute the following MATLAB command to import the XML file into Simulink.

```
>> smimport('EV3_CAD.xml')
```

'EV3_CAD.slx' will automatically be generated and looks similar to Figure 30.

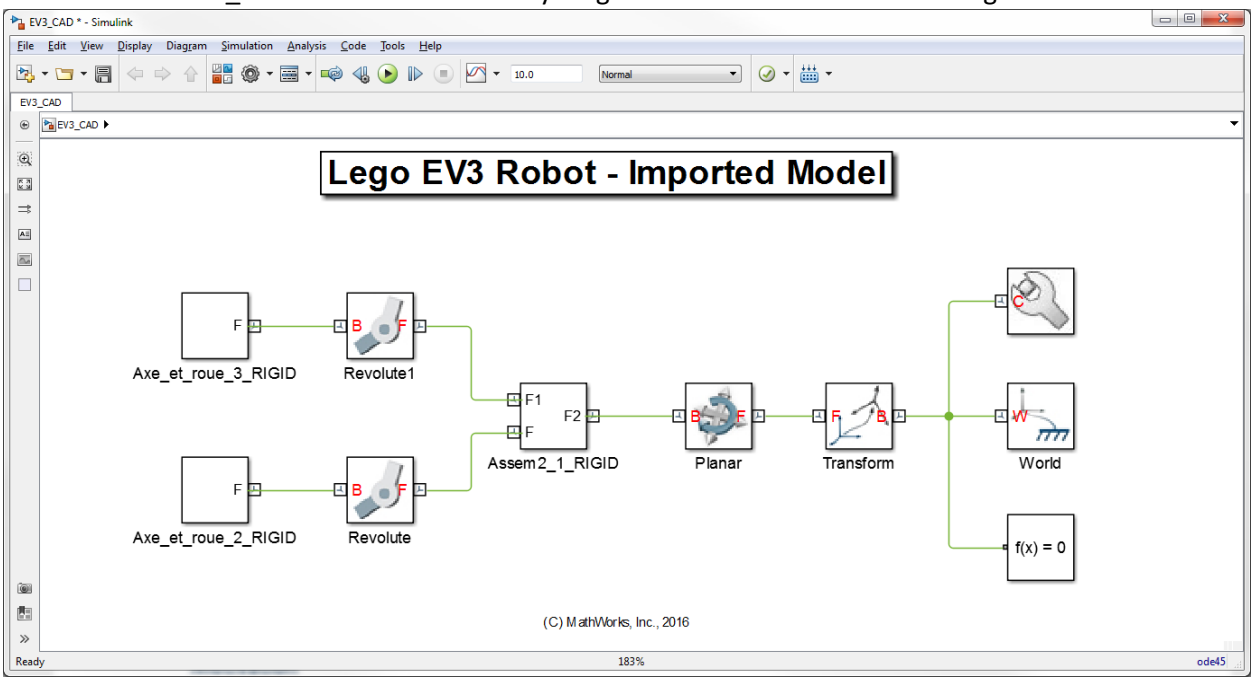

Figure 30 – Imported Simscape Multibody Model from the CAD model

Importing CAD model into Simulink is a 2 step process.

- An add-on software called **Simscape Multibody Link** automatically exports CAD models from supported CAD software to the XML format.
- We then use the SMIMPORT command to bring it into Simulink. The generated Simulink model uses blocks from the Simscape Multibody Library (as we did above).

Find more information about these at the following links:

- **Simscape Multibody :** http://www.mathworks.com/products/simmechanics/
- **Simscape Multibody Link:** http://www.mathworks.com/help/physmod/smlink/ug/installing-and-linking-simmechanics-link-software.html

2. Select **Simulation > Update Diagram**.

The updated diagram provides the simulation with the results of the latest changes that you have made to a model. This also brings up the **Mechanics Explorer** window that shows the robot configuration. To show the robot in an appropriate view convention, make note of the options in the **Mechanics Explorer** window (Figure 31 highlighted in a red box):

3. Choose Y up (XY Front) for **'View Convention'**
4. Click on the **'Isometric View'** button to show the robot in an appropriate view

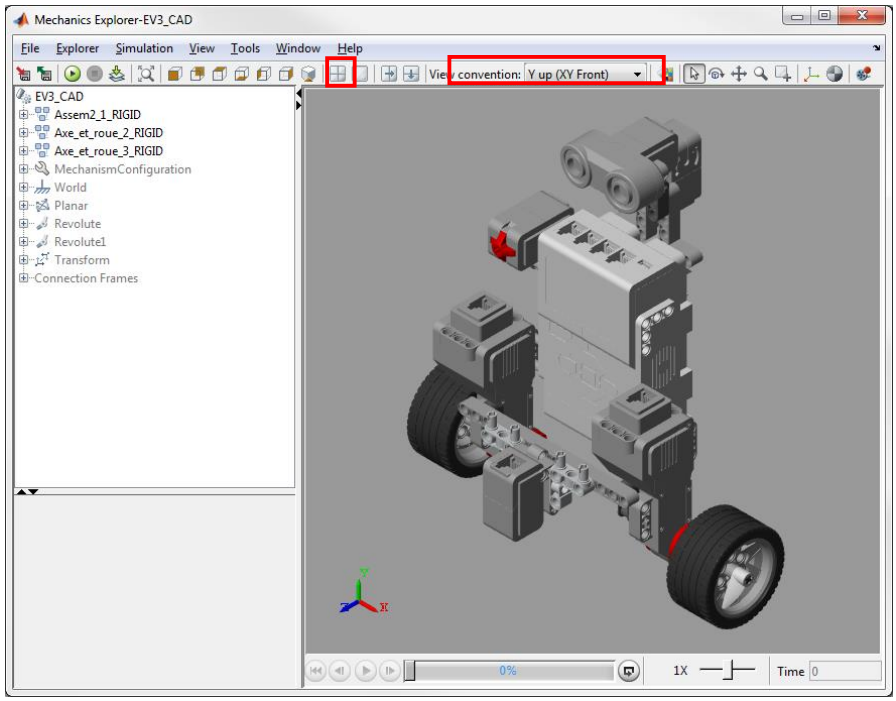

Figure 31: Mechanics Explorer - Visualization

This model has all the details about the dynamics of our robot's mechanical system. We can use this simulation model to better understand the robot motion behavior without going through the exercise of deriving detailed equations of motion for the robot. However, this model by itself does not do anything. We need to add some actuation to the robot which is the next step.

Actuate the robot

1. Open the supplied model 'EV3sim_Openloop.slx' as shown in Figure 32.

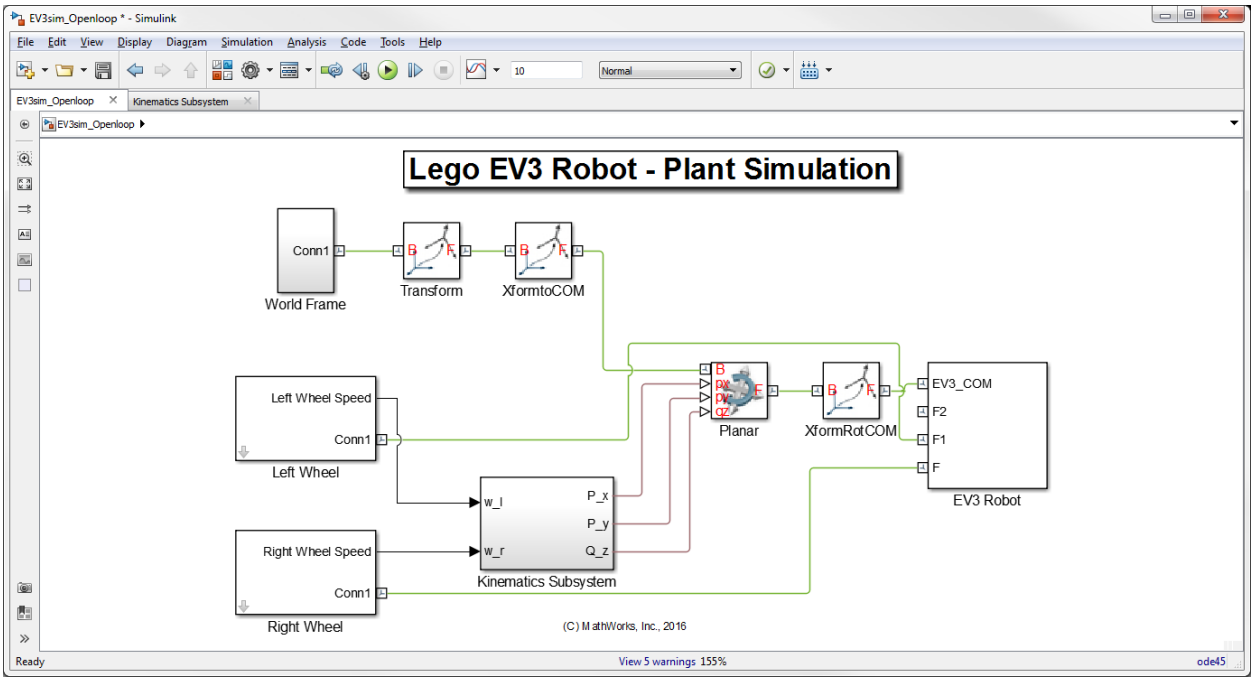

Figure 32 – Open Loop EV3 Robot Simulation Model

2. Explore the 'Kinematics Subsystem' by double clicking it. This is shown in Figure 33.

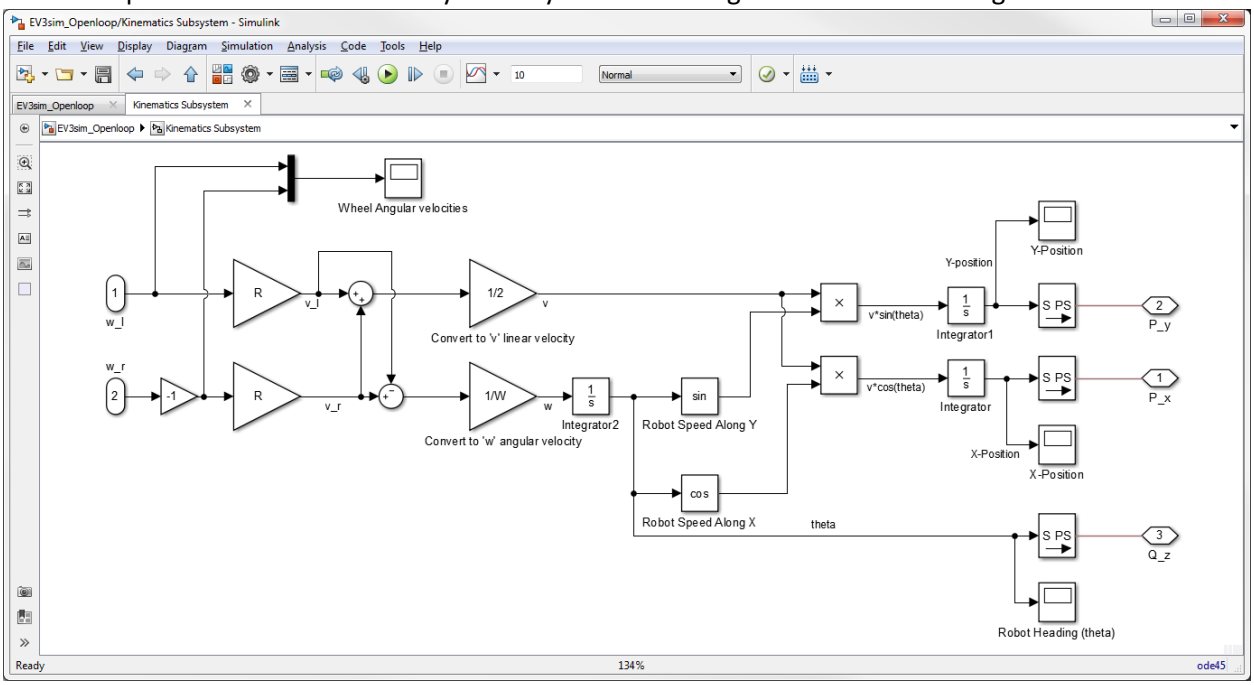

Figure 33 – Kinematics of a Differential Drive Robot

This subsystem is implemented to actuate the robot center of mass based on the individual wheel angular velocities. The equations implemented here are the general equations that govern a differential drive planar robot's position and heading. Please take a look at the following link for more information on this topic: http://planning.cs.uiuc.edu/node659.html

3. Double click on the 'Left Wheel' and 'Right Wheel' subsystems to see what the supplied angular velocities to the wheels are.
4. Run a simulation and observe the robot moving around in the Mechanics explorer window. Fill in the following table with wheel angular velocity values so that you can verify robot behavior:

Desired Motion	Left Wheel Angular Velocity	Right Wheel Angular Velocity
Straight Forward		
Rotate counter clockwise with ZERO turning radius		
Rotate clockwise with constant turning radius (non-zero)		

5. Click on the badge as shown in Figure 34 to **'Look inside mask'** of one of the wheel subsystems and navigate to either the 'Left' or 'Right' motor subsystems.

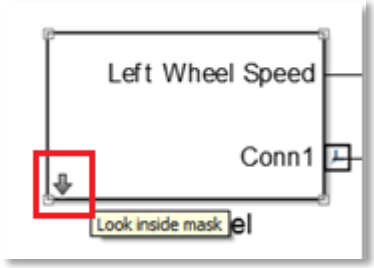

Figure 34 - Look inside mask

You can see that the motor is idealized and motor dynamics are not present. The next section is about how to develop this motor model in order to obtain a more detailed simulation plant model that can better capture the real dynamics of the system.

OP1.2 Motor Modeling for EV3 Plant

Motivation

At the end of this section you will be able to better understand the motor system behavior. This can also be used later for applications like motor speed control design (refer Optional Project 2).

Objective

- Develop motor simulation model

Tasks/Challenge

- Include motor dynamics based on first principles

Solution

1. Explore 'Motor Subsystem' in the supplied model 'MotorOpenLoop.slx'.
2. Optimize for simulation motor parameters from experimental data

Steps/Approach

The files required for this section are in the 'OptPrj1\motorModeling' folder.

Include Motor Dynamics

1. Open the model 'MotorOpenLoop.slx'. Open the subsystem 'MotorOpenLoop/Motor Subsystem' (Figure 35) to take a look at the motor model.

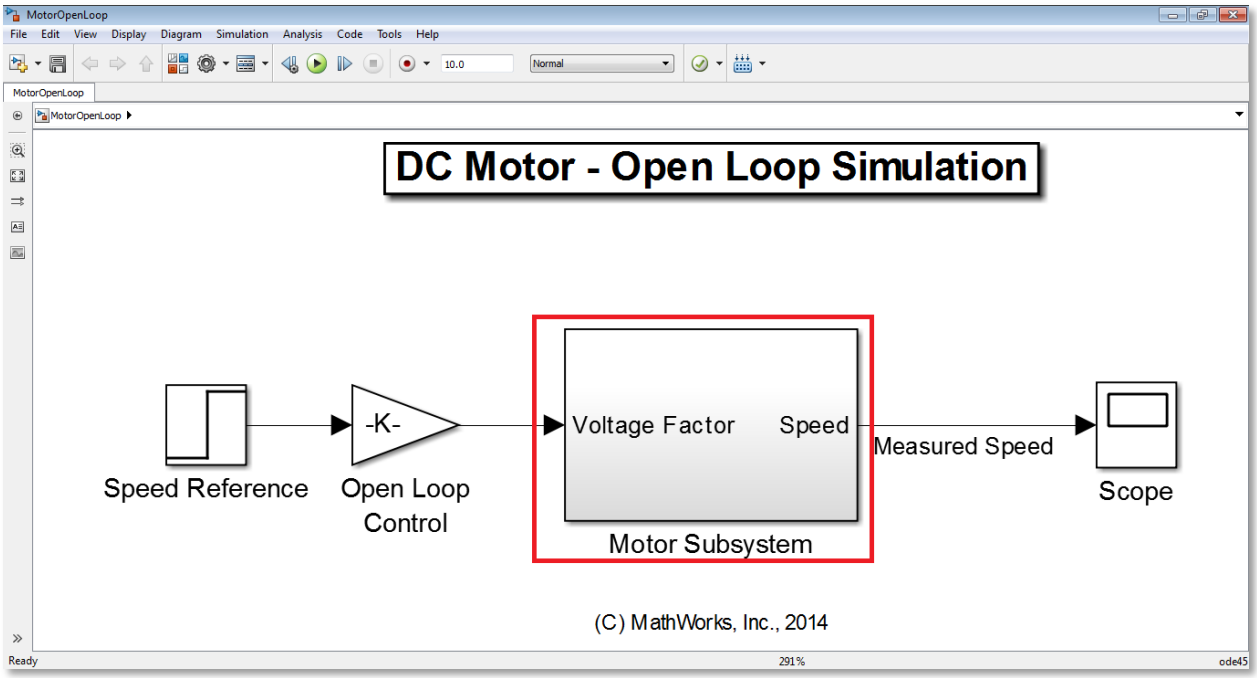

Figure 35 – Motor Simulation Model

Here, we are modeling the motor using first principles and implementing it using physical equations. These are as follows:

$$L\frac{di(t)}{dt} = v_{in}(t) - v_{emf}(t) - Ri(t) \text{ - Kirchoff's law}$$

$$J\frac{d\omega(t)}{dt} = T_m(t) - b\omega(t) \text{ - Newton's law}$$

Where,

R, L = Motor equivalent circuit resistance and inductance respectively
J = Moment of Inertia of the rotor
b = Damping coefficient of the rotor
Vin = Input voltage
Vemf = Back emf
Tm = Motor torque

$$T_m(t) = K_m i(t)$$
$$v_{emf}(t) = K_{emf}\omega(t)$$

2. Simulate the model and examine the **'Scope'** window to see the measured speed of the motor.
3. Change the different motor parameters in the **MATLAB base workspace** (J, b, K, R and L) to understand how these parameters affect the motor behavior physically. Based on this, fill the following table varying the parameter 'J' and all the other parameters set to a value of 1:

Parameter value	Steady State achieved in 10 seconds? Yes/No
J = 1	
J = 10	

Think about why this happens to better understand motor behavior.

Now, we have an initial motor model. However, we still are yet to confirm if this model represents the real EV3 motor. In the next few sections we are going to obtain experimental data from the EV3 motor, and then estimate motor parameters by comparing experimental data with the simulation results.

Acquire data

1. Open the model 'EV3_DataAcquisition.slx'.

This model has sample reference signal in a **'Signal Builder'** block that actuates the EV3 motor. The encoder values are then measured and differentiated to calculate the respective angular velocity.

2. Click on Code and then on External Mode Control Panel, then on Signal & Triggering and change the trigger duration to 2500 samples.

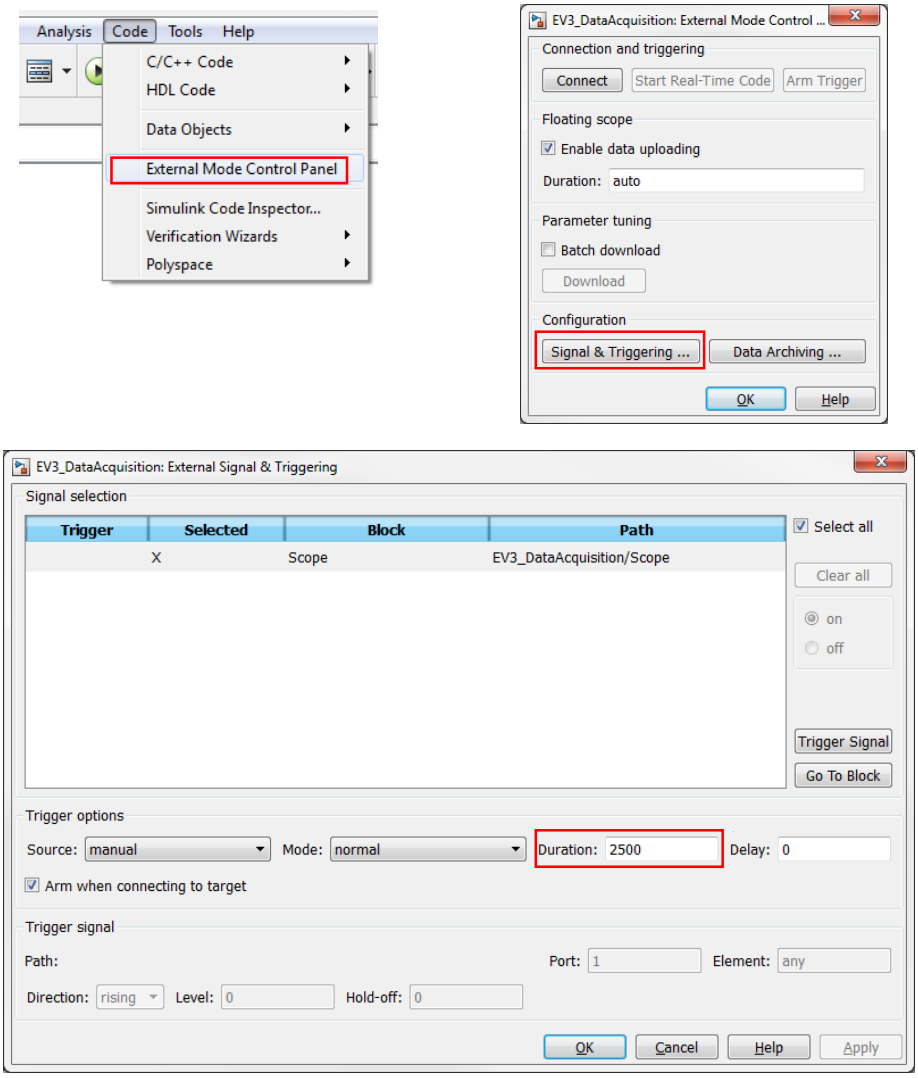

Figure 36 – Set up External Mode parameters

3. Run this model on the EV3 hardware in **'External Mode'** and acquire data needed for motor parameters estimation.

4. Once the model has finished running, click on the **'Process Data'** button (Figure 37) in the Simulink model and you will see two MATLAB figures with the acquired data that shows the estimation and validation data sets. This is also available in the MATLAB workspace as variables as shown in Figure 36.

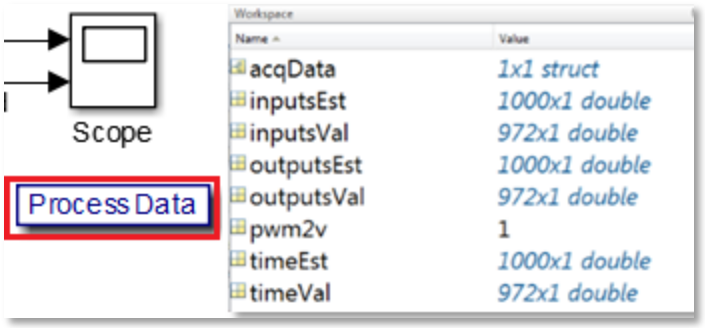

Figure 37 - Process Data, MATLAB Workspace – Acquired Data

Clicking on **'Process Data'** runs a MATLAB script and filters the acquired data for future use.
Now that we have the experimental data, let us compare this with our simulation results and estimate a good set of parameters that represents our DC motor well.

Estimate Parameters

1. Open the 'MotorParamModel.slx' Simulink model.
2. Run the MATLAB script 'initMotorParams.m' to initialize all motor parameters to '1'.
3. Click on **'Load Parameter Estimation GUI'** as shown in Figure 38.

Figure 38 - Launch Parameter Estimation

This will open up the graphical interface for parameter estimation.

4. Navigate to the **'Parameter Estimation'** tab as shown in Figure 39 and click on **'Start'** to begin the parameter estimation.

   The Parameter Estimation tool automatically runs a non-linear least squares optimization routine in the background. This routine iterates over different values of the model parameters (J,K,L,R,b), such that the response of the mathematical model of the motor matches that of the real EV3 motor. The cost function used is the sum of squared errors between the actual and simulated responses. Explore the tool to check out the other optimization algorithms and settings provided.

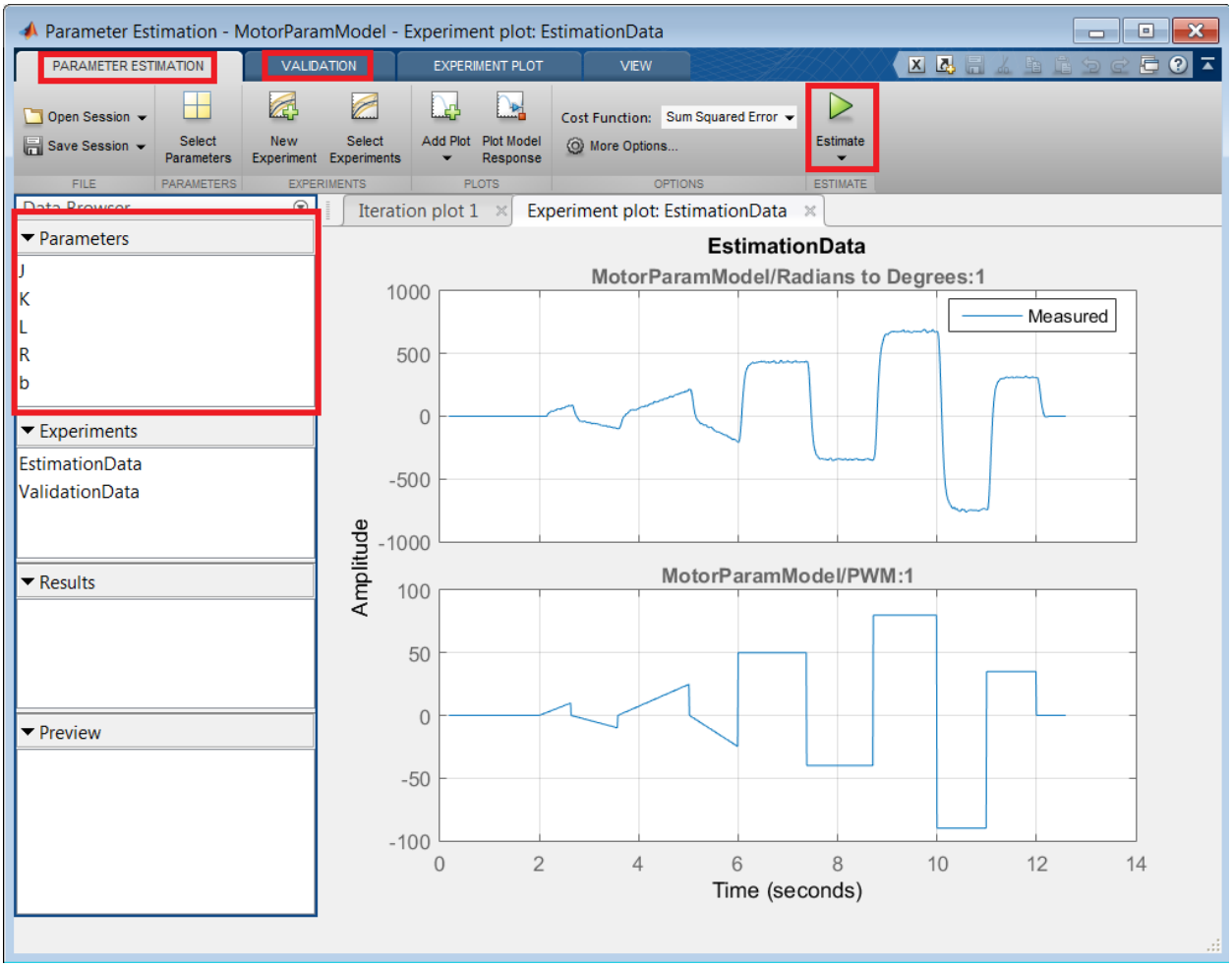

Figure 39 - Parameter Estimation Tool – Estimation

5. Once the estimation is finished, navigate to the **'Parameters'** pane (Figure 39).

Here you can view the estimated parameter values. The final optimized values of these parameters are now available in the MATLAB workspace as well.

After estimating parameters, it's a good practice to validate these parameters against a new experimental dataset.

6. Next, navigate to the **'Validation'** tab. Click 'Select Experiments', check 'ValidationData' check box under 'Validation' and click on **'Validate'** as shown in Figure 40.

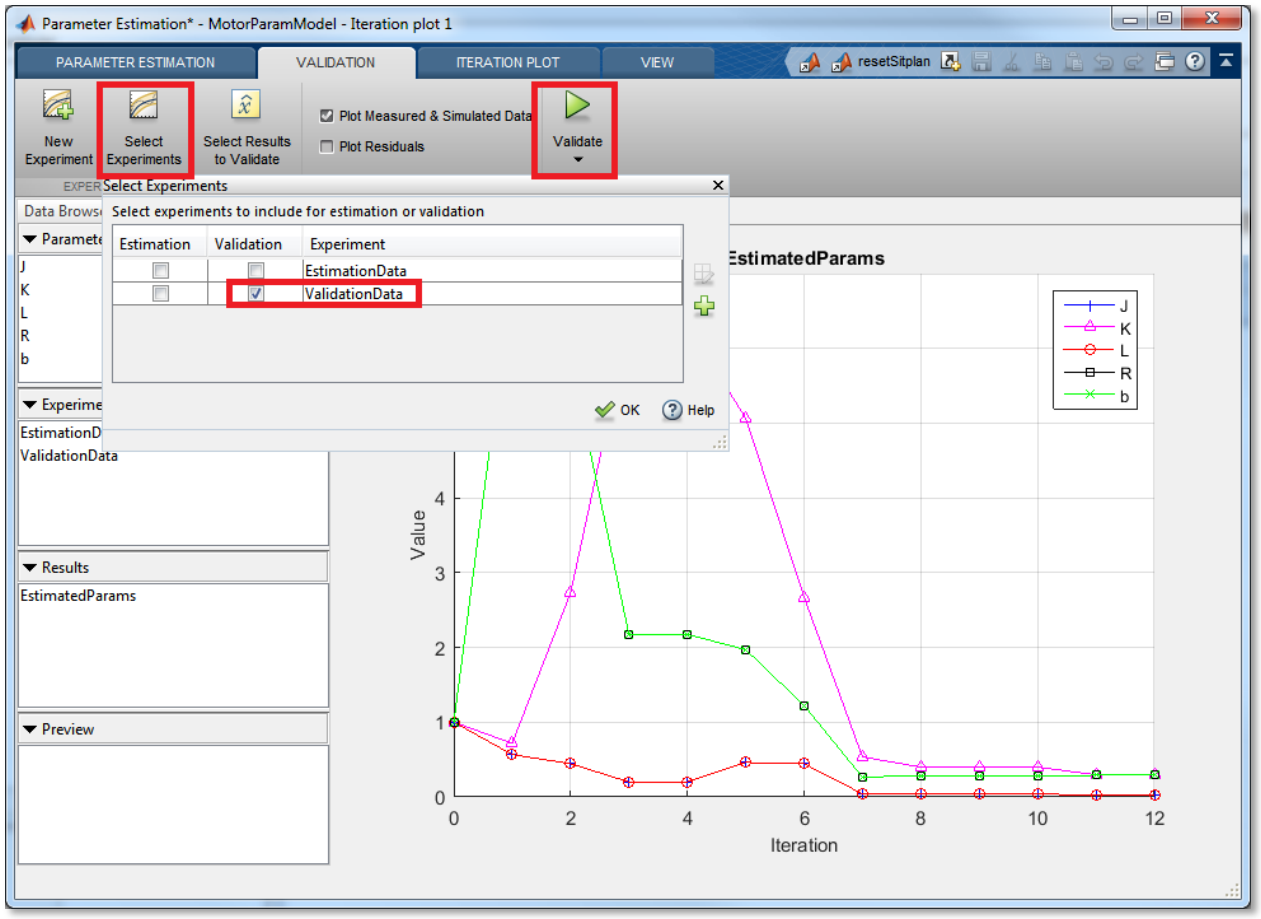

Figure 40 - Parameter Estimation Tool – Validation

So far, we have developed the simulation model of the mechanical system in P3.1 and the motor system in P3.2. Let us integrate these two systems to get our final simulation model.

Simulate Final Model

1. Open the model 'EV3sim_FullModel.slx'.

This model has the motor model already implemented with the EV3 mechanical model as shown in Figure 41.

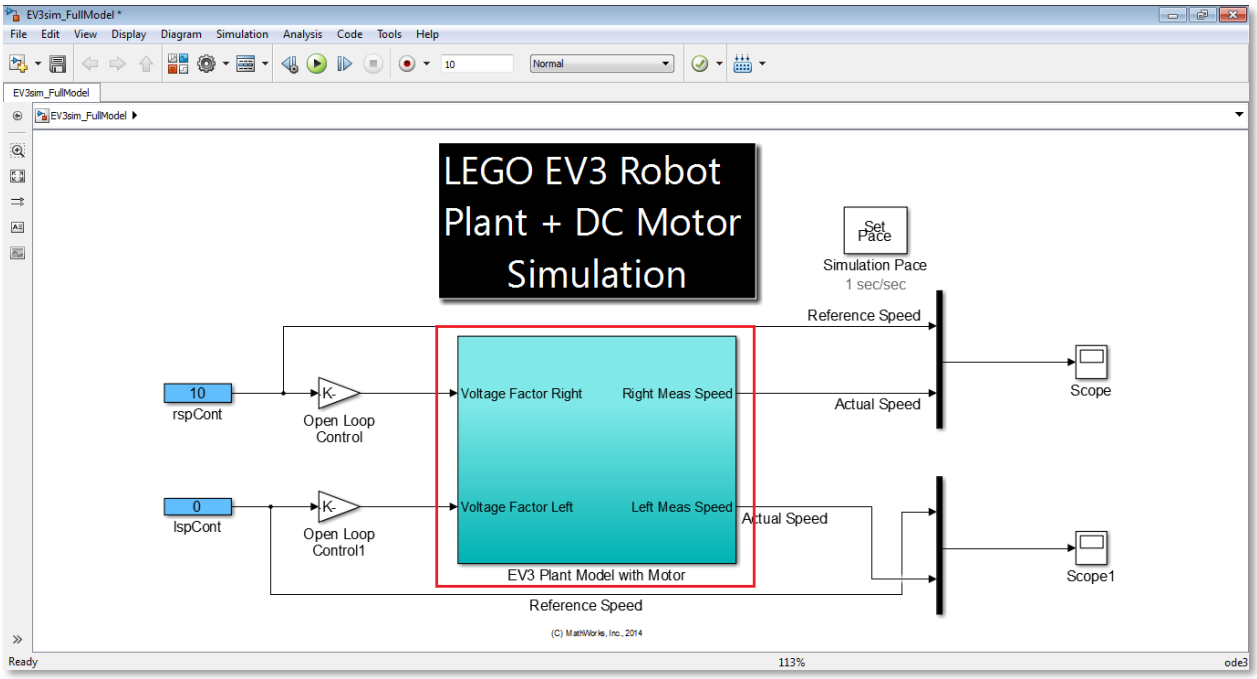

Figure 41 - Fully integrated simulation model

2. Note that in 'EV3sim_FullModel/EV3 Plant Model with Motor' the motor feeds the speed and hence position to a mechanical assembly (Figure 42). Also, navigate to 'EV3sim_FullModel/EV3 Plant Model with Motor/Left Motor Subsystem' subsystem. This is the same model that we developed in OP1.2.

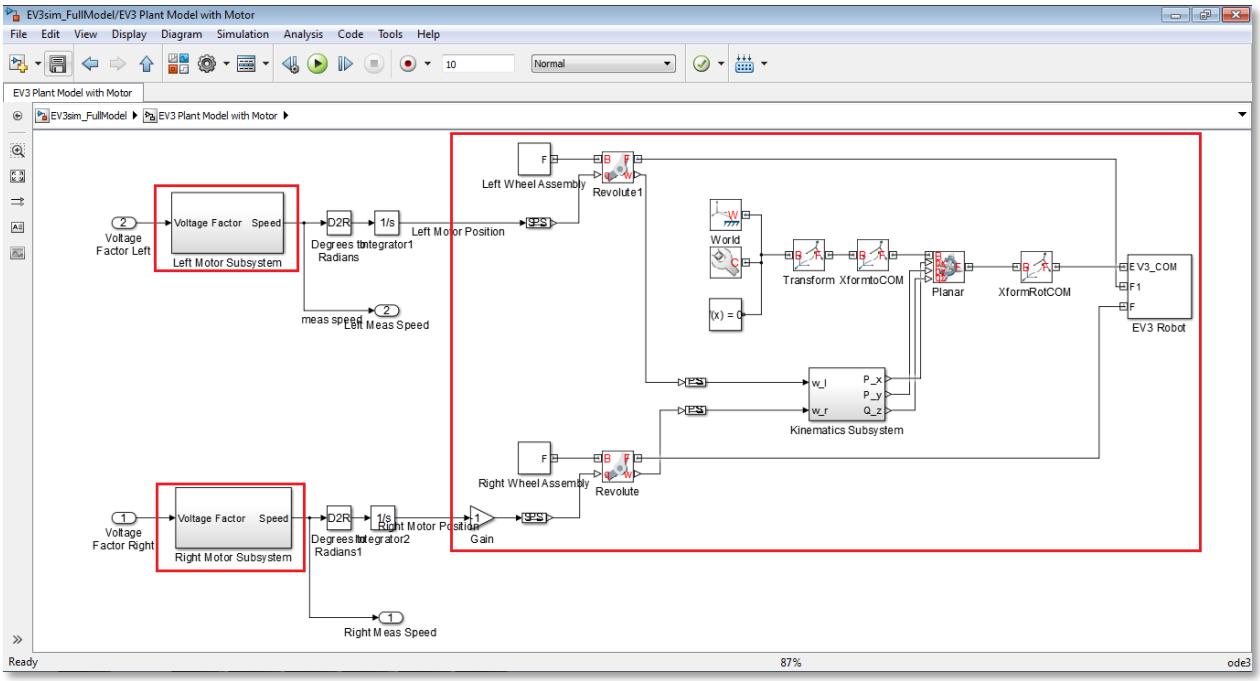

Figure 42 - Motor Subsystems feeding to the Mechanical Assembly

3. Hit the Simulink 'Run' button and see the robot move in the Mechanics Explorer window.
4. Also check the 'Scope' window to see that the 'Reference' speed and the 'Measured' speed are not exactly same. The system in its current state is not able to faithfully track the reference commands.

Next Steps

Design a controller for the EV3 Robot

If you are curious…

Q1: Are there other ways to model a motor?
A1: There are several other ways to model a motor. In general any system can be modeled using first principles or a data driven approach. In the first approach, there is a necessity to have knowledge about the physical nature of the system to put it into equations. This is what we did in the project above fitting parameters in an equation to real world data. In the second approach, we get input/output data and obtain a simulation model of a generic structure. For an example of data driven approach execute the following command in MATLAB to see how you can estimate an 'n' order transfer function based on input/output data. You can see in this case, there is no necessity to know about the physical nature of the system. Based on data, one can just get the I/O relationship of a system.

```
>>doc tfest
```

Q2: Why doesn't the measured speed and reference speed match for the motor model?
A2: The motor model is a dynamic system. In order to faithfully track a reference signal, we need to implement a closed loop control system. If you are interested in control design, move on to the next project!

Optional Project 2: Controller Design for LEGO Robot

OP2.1 Design PID Controller for EV3 Robot

Motivation:

At the end of this section you will be able to make the EV3 motors and the robot, accurately track a desired reference speed.

Objective:

- Design closed-loop controller for EV3

Tasks/Challenge:

- Design and tune a PID controller to track the EV3 desired motor speed.

Solution:

- Tune the controller gains either manually or automatically to design an initial PID controller for the EV3.

Steps/Approach

The files required for this section are in the 'OptPrj2\controllerDesign' folder.

Observe Open-loop Plant behavior

1. Open the 'EV3_without_control.slx' model.

   This is an open-loop simulation model of the EV3 motor dynamics that you just developed in the previous chapter.

2. Simulate the model and observe that the actual (measured) motor speed (purple) that drives the EV3 wheels is different from the desired motor speed (yellow) as seen in the Scope on Figure 43.

With just open loop control, a constant steady-state error can be observed. Therefore, in order to accurately track the reference step input, we need to implement a closed loop control system. Also, having a closed loop control system will help us tackle disturbances better whereas the open loop will easily fail in that scenario.

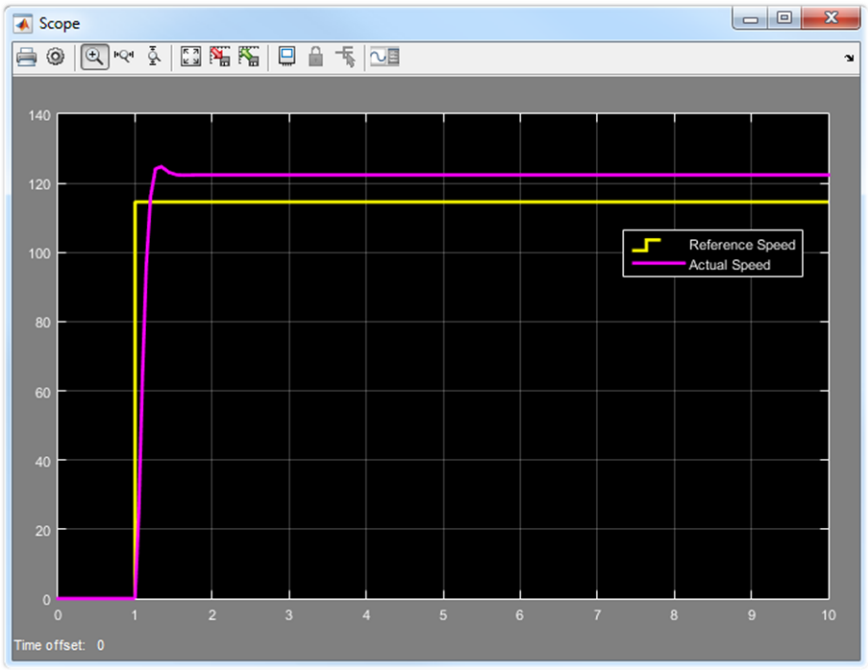

Figure 43 - Reference speed v/s measured speed for the open Loop motor

Design PID Controller

1. Open the 'EV3_with_PIDcontrol_untuned.slx'.

In this model, observe that we are designing a closed-loop (feedback) PID controller that compares the actual (measured) speed against the reference (desired) speed. The controller takes the error as input and commands the motor with appropriate voltage depending on the controller gains (P, I and D).

2. Double-click the PID Controller block.

Looking at the default P, I and D gain values, we observe that it is currently just a Proportional Integral (PI) controller with gains set to 1.

3. Simulate the model to observe that the system is unstable.
4. Tune the controller gains using one of the following methods:
   a) Trial and Error: Manually changing the controller gains to 0.1 yields a stable system. "Trial and Error" method can be cumbersome if you do not know the exact values of the controller gains. Therefore, the second method of automatic tuning is easier and preferred over approach 1.
   b) Select the 'Tune' button in the PID controller block to automatically tune the controller gains and design an initial controller for the plant (EV3).

When the "Tune" button is selected, it brings up the PID Tuner interface (Figure 44) and the following actions occur:

- The PID Tuner automatically linearizes the plant at the operating point specified by the model initial conditions. The linearized plant can be of any order and can include any time delays.
- The PID Tuner computes an initial compensator design for the linearized plant model
- The PID Tuner displays the original and the tuned system step responses.

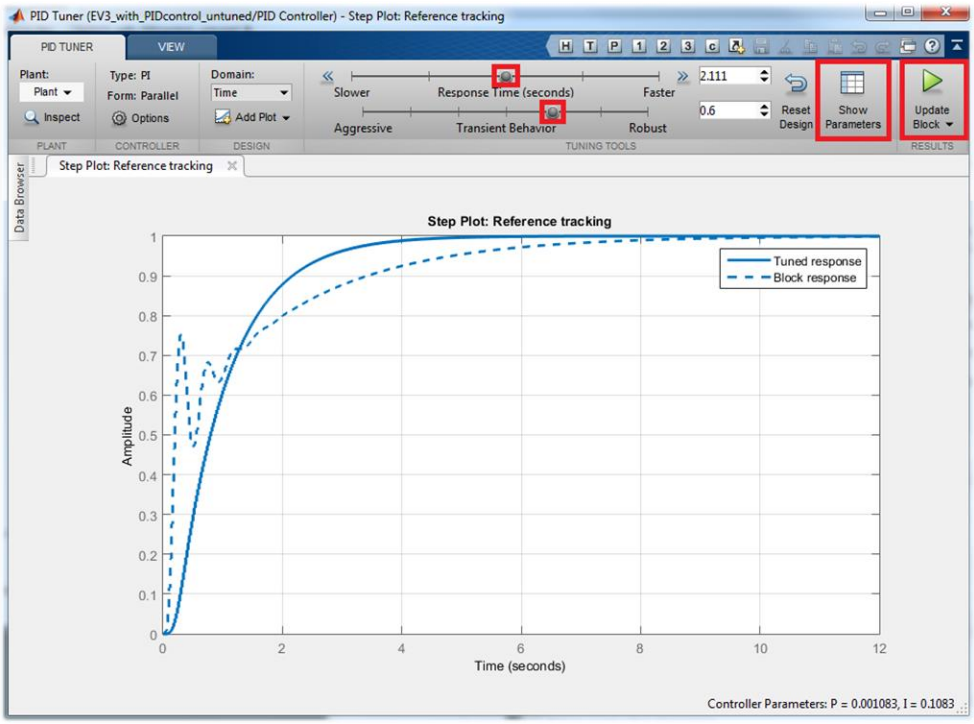

Figure 44 – PID Tuner interface

The PID Tuner interface gives the user the ability to view/update the controller gains by interactively changing the controller response time/robustness. One can interactively change the slider values for the response time and transient response in the time domain or loop bandwidth and phase margin in the frequency domain.

5. Move the slider values to change the gain and observe that the tuned controller gains get updated every time you change the slider values by selecting the "Show Parameters" button.

Controls engineers typically derive the controller design requirements based on hardware experiments, prior system knowledge etc.

6. Interactively change the slider values to obtain the following controller requirements for EV3 if necessary.

   Rise time < 0.4 sec
   Setting Time ~ 1 sec
   Overshoot < 5 %
   Gain Margin > 6dB
   Phase Margin > 60 deg

7. Select the 'Show Parameters' button on top-right of the PID Tuner (Figure 44) to observe how the tuned controller parameters and response characteristics (like rise time, overshoot etc.) compare against the original block parameters. (Figure 45)

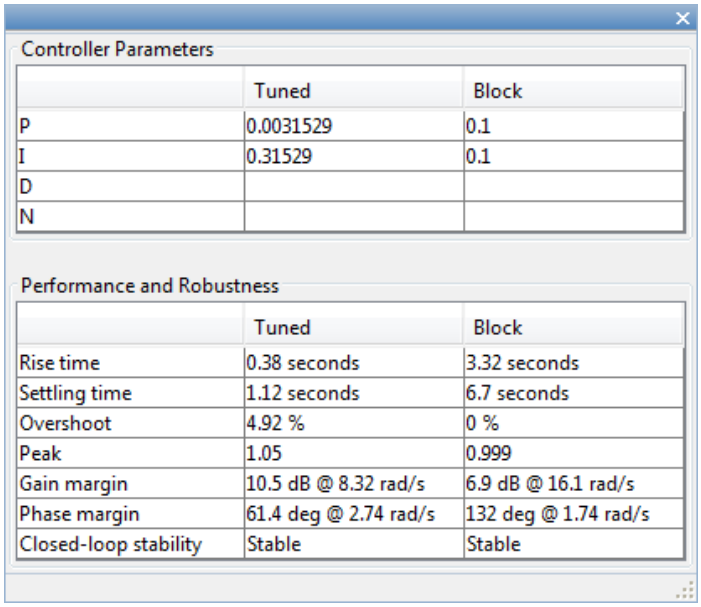

Figure 45 – Comparison between tuned parameters and original block parameters

8. Select the "Update Block" button on the top-right of the PID Tuner interface (see Figure 44) to update the PID Controller block with the tuned controller gains.
9. Fill in the following table with the corresponding controller gains.

	Controller Gains
Proportional (P)	
Integral (I)	
Derivative (D)	

Simulate EV3 Robot Model with Initial Controller

1. Save the previous model as 'EV3_with_PIDcontrol_myGains.slx', and simulate the model to see the effect of the updated control gains.
2. Look at the measured response for the controller designed in the above step. (If the results are different, refer to 'EV3_with_PIDcontrol_tuned.slx' to compare with your model)

The motor tracks the reference step input pretty well as can be seen in the scope (Figure 46).

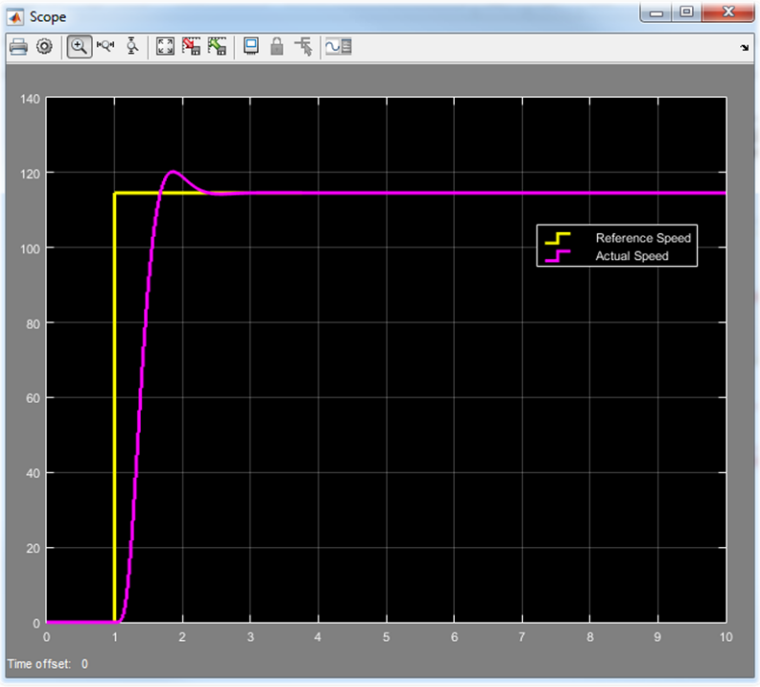

Figure 46 – Reference speed v/s measured speed with the tuned controller - Simulation

Test the Controller on the EV3 Hardware

1. Open the model 'EV3_Controller.slx' from the current directory.
2. Simulate the model in "External" mode to test the controller algorithm on the EV3 hardware.

Observe that the actual EV3 robot hardware with the controller tracks the step input pretty well (Figure 47). The difference between the performance on the real hardware and our simulation, can be attributed to our simplistic DC motor simulation model. For e.g. we have not modeled the friction in the motor. The next logical step would be to add this detail and use our simulation model to get a better controller.

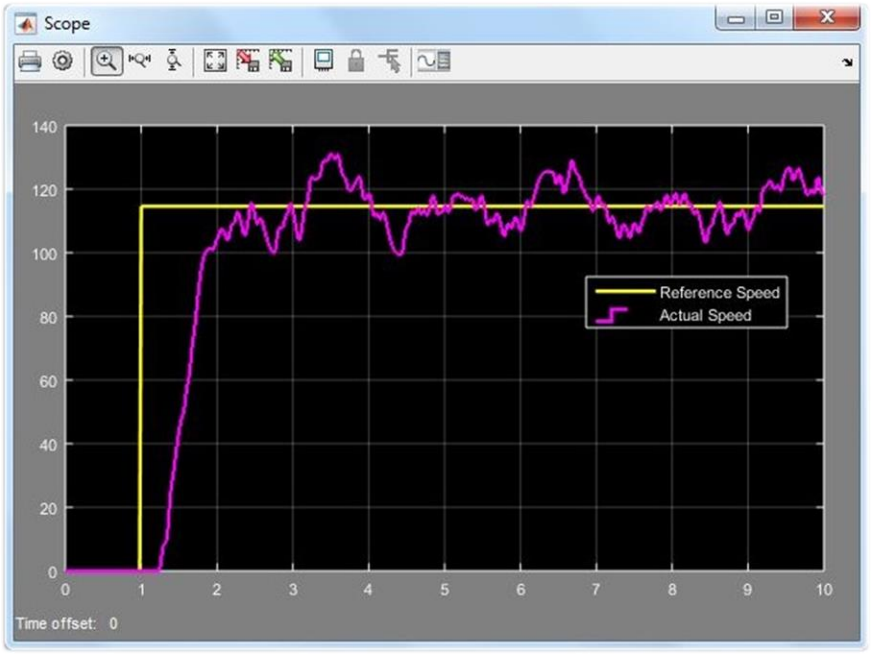

Figure 47 – Reference speed v/s measured speed with the tuned controller - Hardware

If you are curious…

Q1: How does the PID Tuner decide the controller gains that it computes automatically?
A1: PID Tuner takes the following objectives into consideration:
Closed-loop stability — Closed-loop system output remains bounded for bounded input.
Adequate performance — Closed-loop system tracks reference changes and suppresses disturbances as rapidly as possible. The larger the loop bandwidth (the frequency of unity open-loop gain), the faster the controller responds to changes in the reference or disturbances in the loop.
Adequate robustness — Controller loop design has enough gain margin and phase margin to allow for modeling errors or variations in system dynamics.

MathWorks® algorithm for tuning PID controllers (based on Åström, K. J. and Hägglund, T. Advanced PID Control, Research Triangle Park, NC: Instrumentation, Systems, and Automation Society, 2006.) meets these objectives by tuning the PID gains to achieve a good balance between performance and robustness. The algorithm designs an initial controller by choosing a bandwidth to achieve that balance, based upon the open-loop frequency response of your linearized model.

Q2: Is there a way in which I can re-tune the controller at a different operating point?
A2: To set a new operating point for the PID Tuner, there are several methods. For more information on these methods, please refer the following online documentation page.
http://www.mathworks.com/help/releases/R2015a/slcontrol/ug/tune-at-a-different-operating-point.html

Q3: Is there a way to optimize the controller gains for a desired system step response?
A3: Yes, we can use response optimization techniques to optimize the controller gains for a desired step response. If you are interested in controller design using response optimization, refer project OP2.2.

OP2.2 Fine-tune PID Controller for desired step response

Motivation:

At the end of this section you will be able to design controllers that satisfy specified controller objectives like desired step response characteristics.

Objective:

- Fine tune the closed-loop controller for EV3 robot using optimization techniques.

Tasks/Challenge:

- Understand design constraints and requirements for the controller performance
- Setting up an optimization routine to determine PID gains needed to meet requirements.
- Explore the different options within the optimization technique

Solution:

1. Run the model 'EV3_SDO_PID_untuned.slx' to see the initial response of the system in simulation
2. Make use of Simulink Design Optimization to specify a controller objective and tune the PID controller accordingly.

Steps/Approach

The files required for this section are in the 'OptPrj2\controllerDesign' folder.

Observe un-optimized closed-loop plant behavior

In our first step, we are going to examine the current performance of the PID controller with default gains.

1. Open the 'EV3_SDO_PID_untuned.slx' model that consists of the closed-loop motor dynamics.
2. Load the .MAT file 'SDO_Start_Session_part1.mat'.
   This MAT file will set-up parameters for the P-I-D controller and plant model.
3. Simulate the model and observe that it takes more than 5 seconds for the wheels to reach the desired wheel speed. This is showcased in the Figure 48.

   Examine the display blocks highlighted in green which show the current P and I gain values. These values were hand-picked arbitrarily and are not optimal in terms of fast trajectory tracking.

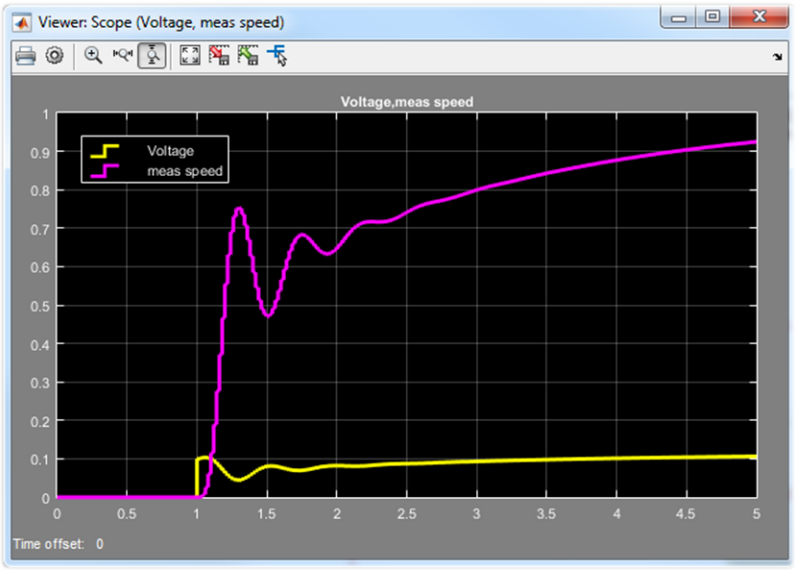

Figure 48 - Voltage Output and Measured Speed

Examine Desired Response Characteristics

The 'Check Step Response Characteristics' block is a way to specify how we want our system response to behave to ensure it stays within a specified bound.

1. Open up the block 'Check Step Response Characteristics' and click the play button. This will run the simulation again, and examine the plot of the feedback output (Figure 49).

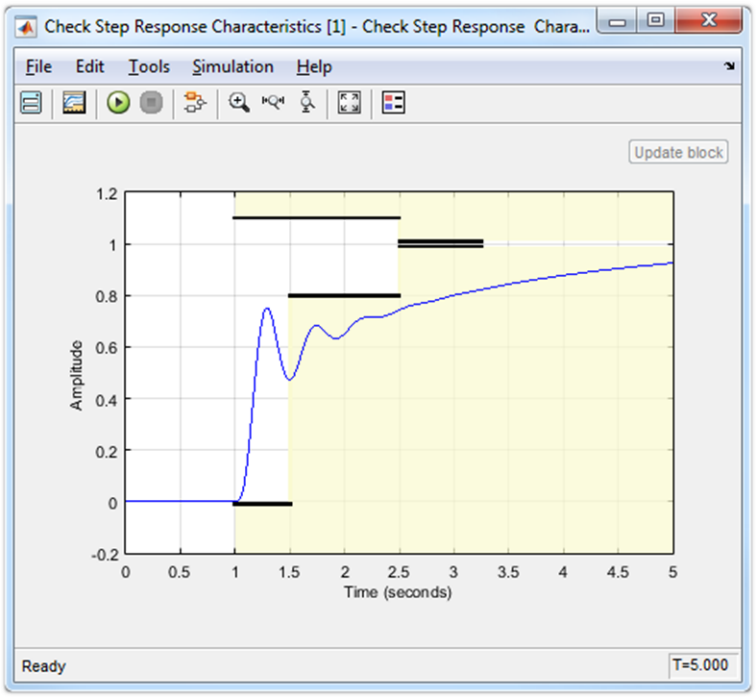

Figure 49 - Examining controller performance versus desired response

Our desired response behavior is depicted by the white back-ground in Figure 47. Our objective is to tune our controller to ensure that it reaches our desired set-point faster by staying in the white region. If the response of the system falls outside the white region it may over-shoot too much or track the desired reference point too slowly. The darkened area in the plot depict "forbidden" zones. Our desired

response must not enter these areas in order to achieve fast settling times and reasonable over-shoot values.

2. Examine that with the current PID values have clearly failed to meet our performance objectives as shown in Figure 47. The actual output passes through the darkened region and is out of bounds.

Using Simulink Design Optimization to find optimal PID Gains

In order to meet our desired response characteristics, we can employ Simulink Design Optimization (SDO) to find the set of gains which will meet our objectives outlined in the 'Check Step Response Characteristics' and 'Constrain Voltage' block. The 'Constrain Voltage' block is used to limit voltage to be within +-5 Volts. Using SDO offers a much more automated method of fine tuning a control system than tuning values by hand manually.

1. With the 'EV3_SDO_PID_untuned.slx' model opened, go to **Analysis > Response Optimization**
2. Click on **Open Session** and then select **Open from file**. Load the 'SDO_Start_Session_part2' MAT file. This will load an SDO session which will set up the variables to tune in the model for the optimization process.
3. Examine the current PI values which will act as the independent variables in the optimization. This can be viewed by first clicking on the **Response Optimization** tab. Click on the Pencil Icon to bring up the design variables menu. This is shown in Figure 50.

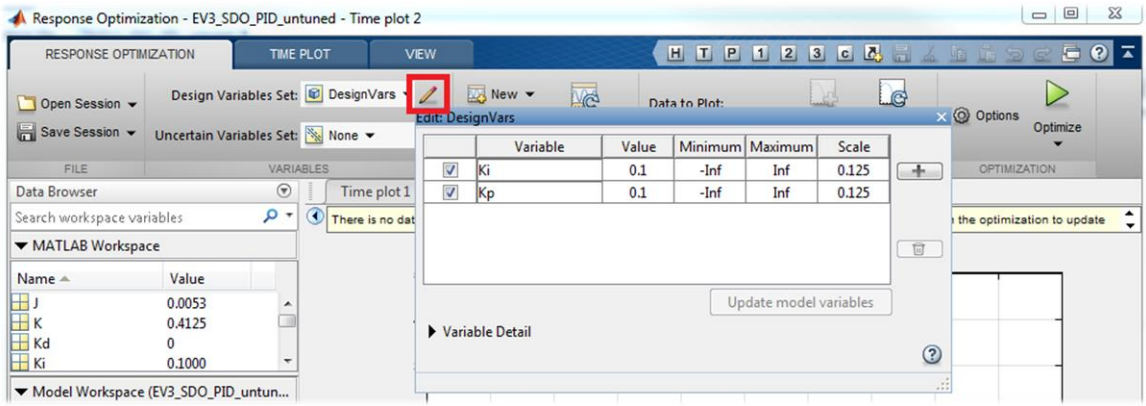

Figure 50 - PID values as design variables

Before starting the optimization, we want to visualize the output when the automated tuning is in progress.

4. Click on the 'Time plot 1' tab to choose the step response characteristics to view. This is shown in Figure 51.

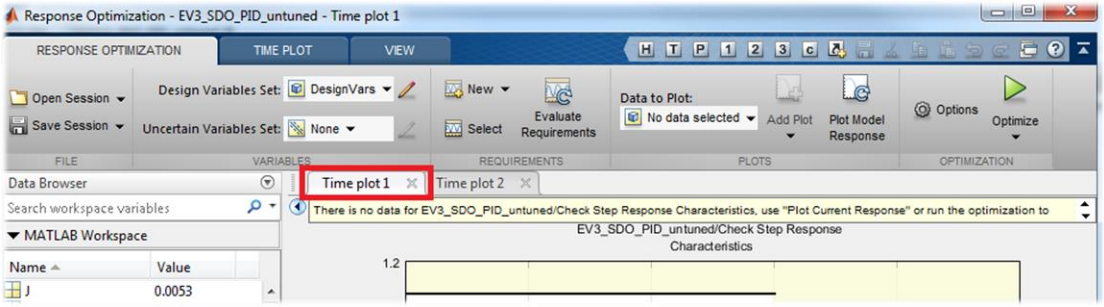

Figure 51 – Choose time plot

5. Now, click on the green arrow Optimize button to begin the optimization process. Examine how the controller response changes in each iteration of the optimization. Also, examine the model as the Kp and Ki values in the display blocks change.

The optimization routine is running the model over many iterations and tuning each PID gain while being constrained to the specifications outlined in the constrain blocks.

The final response should look like the bold line depicted in Figure 52. The lighter shades belong to the responses during the various iterations of the optimization routine.

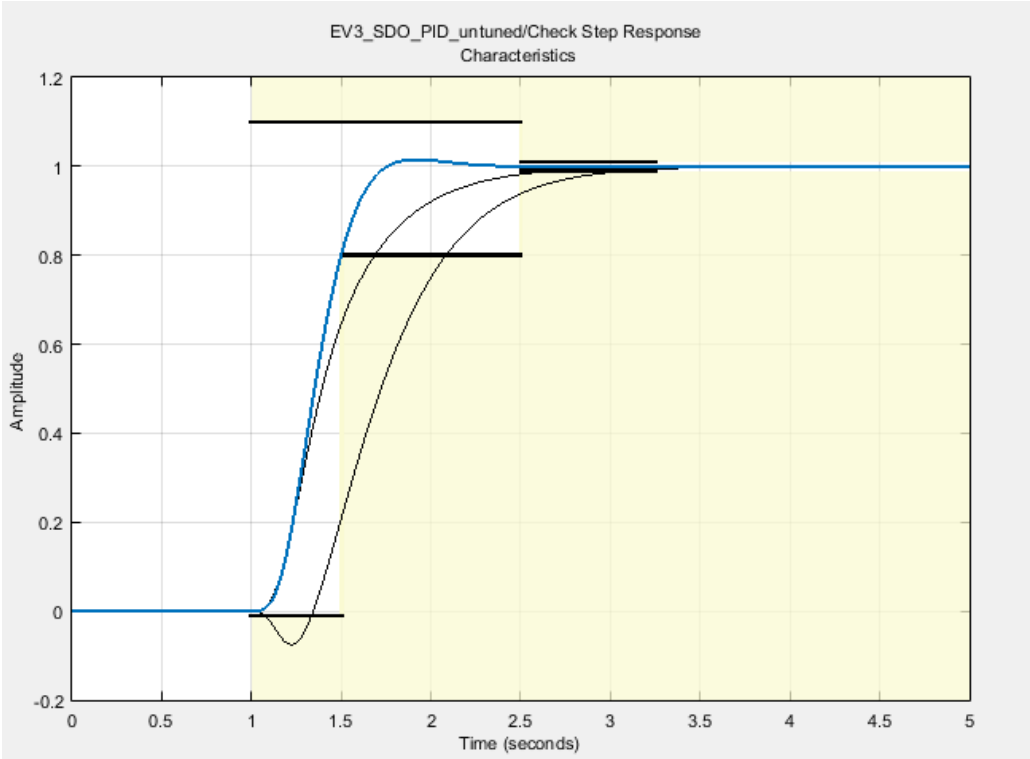

Figure 52 - Result of Simulink Design Optimization

Using Simulink Design Optimization to find optimal PID Gains

Providing that you have completed the SDO steps above, now try running the same control system on the hardware.

1. Open the model 'EV3_Controller.slx'.

2. Replace the PID controller block in the model with the PID controller block from 'EV3_SDO_PID_untuned.slx'. Follow the same procedure as before to run the model on the hardware to see the effect of the new gains.

If you are curious…

Q1: What kind optimization routine is being used?
A1: By default, the **Gradient Descent** method is selected and the algorithm is set to **Sequential Quadratic Programming.** This is specified under the **Optimization Options** tab in the SDO options. To learn more about what each method and algorithm does, please consult this documentation.
http://www.mathworks.com/help/sldo/ug/optimization-options.html#brztydr-1

Q2: How can I speed up the optimization technique?
A2: The speed of the optimization routine will depend on the complexity of the model since each iteration of the optimization will execute the model to examine the output. Simulink Design Optimization offers the ability to use the Parallel Computing Toolbox to run multiple iterations amongst different CPU cores (called "Workers"). This option can be found under **Options** and clicking on the **Parallel Options** tab. To read more about this process, please consult this documentation.

http://www.mathworks.com/help/sldo/ug/how-to-use-parallel-computing-gui_bs2zebr-1.html#brztxje-10

Q3: Is there a more advanced control problem that can be experienced with LEGO EV3, MATLAB and Simulink?
A3: Given that the LEGO EV3 has so many different sensors, the possibilities are potentially limitless. For e.g. explore the self-balancing robot with the following steps:
With the help of a workshop facilitator, change the robot to a simpler self-balancing configuration
Navigate to the 'selfBalancing' folder and run the following scripts:
`EV3_Plant`: This script generates the state space model of the EV3 robot in a self-balancing configuration. This plant is based on the NXTGWay model as described in **'NXTway plant model.pdf'**
`EV3_Controller`: This script generates an LQR controller for self-balancing.
Simulate the model 'EV3_Sim_Balance_Move.slx' and take a look at the scope window output. Understand what the simulation output means physically with respect to the robot stability.
'Deploy' the model 'EV3_HW_Balance_Move.slx' to see the LEGO EV3 hardware self-balance with the same controller!
NOTE: Be sure to keep the robot in a relatively upright position before the model starts running. The controller works only around this vertical equilibrium because of the way in which we have modeled the simulation model and hence also designed the controller with that assumption.

Optional Project 3: Event-driven modeling using Stateflow

Motivation

At the end of this project you will be able to use a Stateflow Chart in a Simulink model to develop control logic for an event-driven system. This project provides an alternative workflow to developing event-driven control algorithms than the one discussed in Section 2.4.

Vehicle transmission systems, flight controllers for airplanes and space vehicles, and control algorithms for robotic and autonomous systems are some examples of dynamic systems that transition between a finite set of states, with transitions being triggered by events. MATLAB allows for defining and developing an event-driven controller, as is demonstrated in Section 2.4, but as the complexity of the states and event-triggered transitions increases, Stateflow provides a more natural and intuitive environment for developing such controllers while abstracting away much of the rigors of manually programming event triggers and state transitions. In the 'External Mode' in Simulink, Stateflow also allows for visualizing event triggers and state transitions in near real-time as the controller is implemented on the embedded platform, providing clearer insights into controller performance for optimized design changes.

Similar to Section 2.4, we will use a state machine which is a model of a reactive system. This model defines a finite set of states and behaviors and how the system transitions from one state to another when certain conditions are true. The states and transitions are defined using a Stateflow chart within the Simulink model.

Objective:

- Use Stateflow® to building control logic using finite-state machines and allowing for event-driven state transitions, similar to Section 2.4

Task

- Use Stateflow to implement line-following, collision-avoidance and programming a touch sensor to act as an on-off switch, allowing the robot to transitions between the following states based on specific events:
    - State 0: Stop - action triggered by a touch sensor.
    - State 1: Forward motion with line-following in the absence of an obstacle – action triggered by the touch sensor and the absence of an obstacle.
    - State 2: Reverse motion – triggered by the presence of an obstacle.

Solution

As we start to build more complex controller logic and define certain actions triggered in response to specific events as identified using sensor systems, programming event logic and defining state transitions in MATLAB can very quickly get very daunting. Stateflow®, a visual environment designed specifically for event-driven modeling, abstracts away much of the complexity involved in programmatically developing complex logic design for finite-state machines, and can significantly help speed up the prototyping and deployment of controllers, especially when the complexity of the control logic scales up significantly.

Stateflow® is an environment for modeling and simulating combinatorial and sequential decision logic based on state machines and flow charts. Stateflow lets you combine graphical and tabular representations, including state transition diagrams, flow charts, state transition tables, and truth tables, to model how your system reacts to events, time-based conditions, and external input signals. Further details on Stateflow can be found at the following URL:
http://www.mathworks.com/products/stateflow/

A webinar on 'Control Logic Made Easy with Stateflow' can be accessed here:
http://www.mathworks.com/videos/control-logic-made-easy-with-stateflow-81914.html

Steps/Approach

Create Simulink model

6. Open the supplied OptPrj3/followAndAvoidBlocksDeployStateFlowEvent.slx

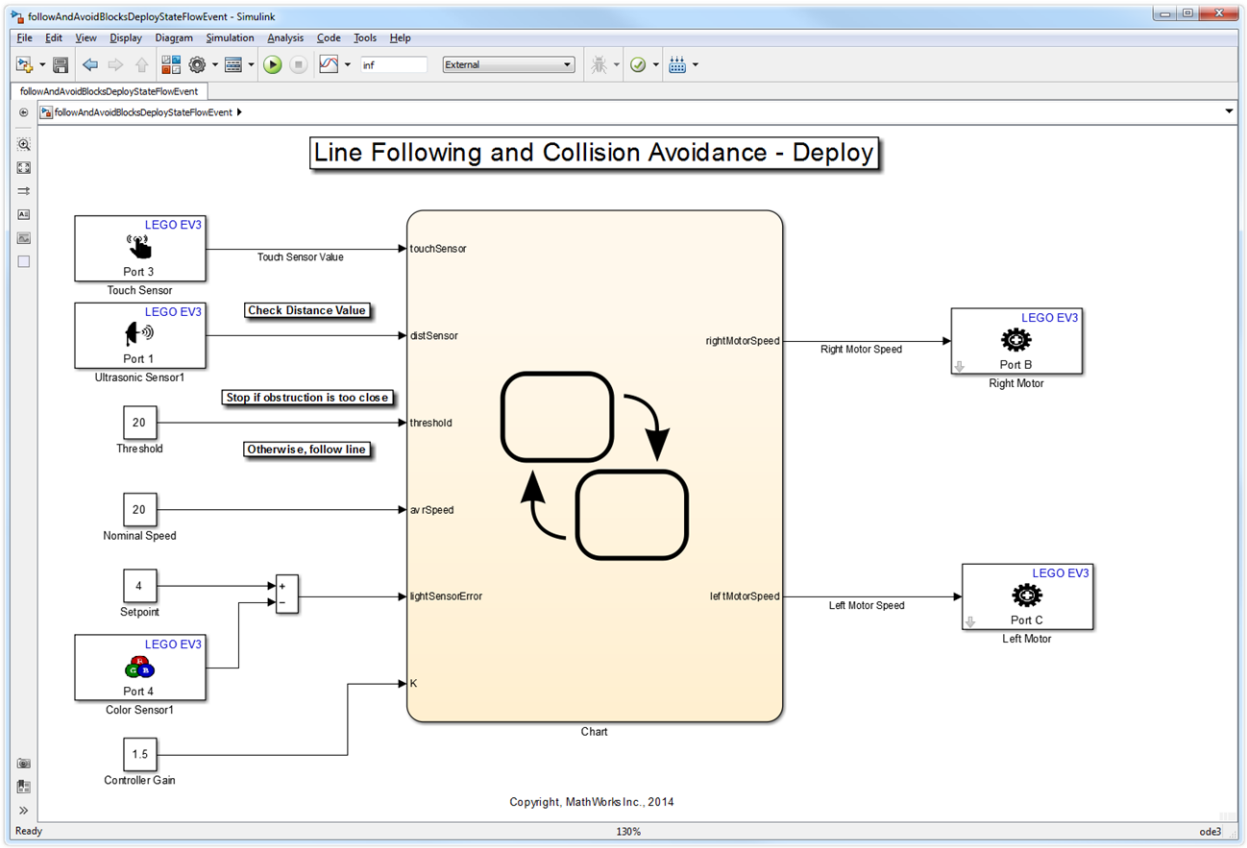

Figure 53: Using Stateflow in Simulink model

7. The model should be very similar to the one used in Section P2.4 in functionality. Observe the state transitions and control logic within the Stateflow block and develop familiarity with Stateflow features, state and state transition definitions.
8. Double click on the block to open the Stateflow model. You should see something similar to Figure 28.

9. Observe the states and events that trigger state transitions, and make necessary adjustments to the control logic.
10. Make desired changes to parameters governing the controller behavior – distance thresholds for obstacle avoidance and response to obstacle, if necessary.
11. Choose appropriate values for Threshold, Average Speed, Controller Gain and Setpoint.

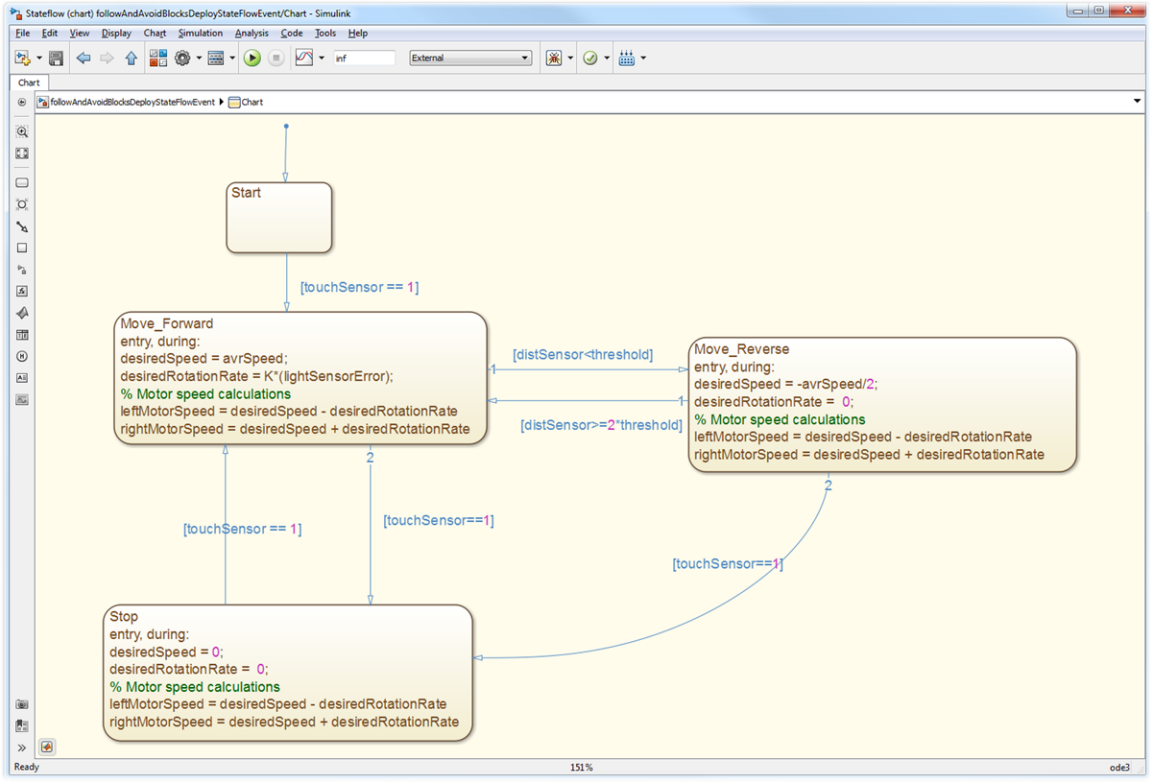

Figure 54: MATLAB Function block in Simulink model

Prepare the model for deployment to the EV3 Brick

- Make necessary changes so that the model can run on the EV3 brick in External Mode

Observe

- Confirm that the behavior of the robot is similar to the earlier models, with the modified response to obstacles and the added functionality of a touch sensor acting as a power switch.
- Place an obstacle in the path of the robot (observable by the Ultrasonic Sensor) and observe the Stateflow model within the Stateflow Chart block.
- Trigger the events that cause state transitions (introduction of an obstacle within the field of view of the Ultrasonic Sensor, for instance). Notice how the current state is highlighted with a blue outline, and ensure that specific events trigger transitions to corresponding states and that the controller behaves as observed in Section 2.4.

If you are curious…

Q1. What else can I do with Stateflow?
A1. In addition to developing algorithms with complex control logic graphically, Stateflow provides a tabular interface too, for modeling system logic using state machines. In Stateflow you can represent combinatorial logic graphically with flow charts and in tabular format with truth tables.

Designing logic involves defining conditions to be checked and subsequent actions to be performed. Stateflow enables you to define conditions and actions in C or in MATLAB®. You can manage the data used in conditions and actions from the Simulink® Model Explorer. Before executing your design, Stateflow notifies you of possible state inconsistencies, unused data and events, and invalid transitions.

More details on Stateflow and its key features can be found at:
http://www.mathworks.com/products/stateflow/features.html#key-features

Domain-specific examples on using Stateflow can also be found at:
http://www.mathworks.com/products/stateflow/model-examples.html

Appendix 1: Background Information

Simulink

Simulink® is a block diagram environment for multidomain simulation and Model-Based Design. It supports simulation, automatic code generation, and continuous test and verification of embedded systems.

Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB®, enabling you to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis.

Hardware Support Packages

With the help of free add-ons called Hardware Support Packages, Simulink provides built-in support for prototyping, testing, and running models on low-cost target hardware, such as Arduino®, LEGO® MINDSTORMS® robots, and Raspberry Pi. You can design algorithms in Simulink for control systems, robotics, audio processing, and computer vision applications and see them perform with hardware. This hardware support is also available in MATLAB and Simulink Student Version.

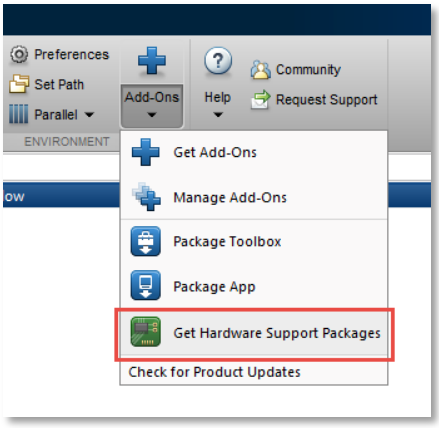

Figure 55: Get Hardware Support Packages

Simulink built-in support for hardware includes:

- Automated installation and configuration
- Target hardware device libraries of Simulink blocks that connect to I/O ports, sensors, and actuators
- Streamlined workflows for designing, building, and executing algorithms on supported target hardware
- Direct communication between Simulink and the target hardware
- Interactive parameter tuning and signal monitoring of your application as it runs
- Model deployment for autonomous execution

LEGO MINDSTORMS EV3

The LEGO MINDSTORMS EV3 provides a low-cost, yet remarkably powerful embedded target which includes a powerful ARM9 processor, USB port for WiFi and Internet connectivity, Micro SD card reader, back-lit buttons and 4 motor ports. In order to harness this capability, however, it is necessary to have an appropriate programming pathway. In this context, the Simulink enables high-level Simulink designs to be automatically cross-compiled for execution on the NXT Intelligent Brick without users having to engage in low-level programming. Furthermore, Simulink's 'External Mode' capability allows users to interact with, monitor and tune the code as it executes fully autonomously on the EV3 Intelligent Brick.

Appendix 2: Software and Hardware Setup

Software

In order to run through projects 1&2 in this workshop, the following software is needed:

1. Window or Linux. For supported operating systems see: http://www.mathworks.com/support/sysreq/current_release/index.html
2. MATLAB and Simulink R2014a or higher
3. A C-compiler that works with Stateflow: http://www.mathworks.com/support/compilers
4. Simulink Support Package for LEGO MINDSTORMS EV3

Hardware

1. 45544 LEGO® MINDSTORMS® Education EV3 Core Set: https://education.lego.com/en-us/lego-education-product-database/mindstorms-ev3/45544-lego-mindstorms-education-ev3-core-set

Figure 56: LEGO EV3 Education Base Set

2. Wi-Fi dongle for EV3. See recommended dongle here: http://www.lego.com/en-us/mindstorms/support/faq

Configuration

Wi-Fi

1. Setup a local Wi-Fi network or use an existing one that allows devices on the network to communicate with each other
2. Connect EV3 Intelligent Brick by following instructions on Pages 22-23 in this document from LEGO: http://cache.lego.com/r/education/-/media/lego%20education/home/downloads/user%20guides/global/ev3/ev3-user-guide-en.pdf
3. Note down the IP Address of the EV3 Intelligent Brick from:
   a. Settings Tab > Brick Info > IP Address OR
   b. Setting Tab > WiFi > [Click] on the network to which the brick is connected

   **** Note that the IP Address of the EV3 Brick might change every time it is connected to the network***
4. Connect the computer to the same Wi-Fi network as the EV3 Intelligent Brick
5. Run the following command at the MATLAB Command Prompt with IP Address noted from Brick Info.

   ```
   >> h = legoev3('xxx.xxx.xxx.xxx')
   ```

   If an EV3 Brick is found, you will get a valid handle **h** to the EV3 Brick, which will allow you to interact with the Brick programmatically.

   ```
   h =
     legoev3 with properties:
       ipAddress: 'xxx.xxx.xxx.xxx'
   ```