Simulink and Raspberry Pi Workshop Manual

A brief workshop on Simulink Support for Project Based Learning with Raspberry Pi

MathWorks, Inc.

4/14/2015

Version 2.2: R2013b, R2014a, R2014b, R2015a

Simulink and Raspberry Pi Workshop Manual

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.

Simulink includes the capability to program low-cost hardware like LEGO® Mindstorms® robots, Arduino®, Raspberry Pi®, and many others. This new capability enables students to develop and test a variety of controls, signal, image and video processing related applications, from within Simulink and Stateflow®.

This workshop is based on Simulink Support Package for Raspberry Pi. You will have a chance to work through lab modules with examples of image and video processing. 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 the classroom with students.

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

1. design, simulate and test custom algorithms in Simulink
2. implement these algorithms on low-cost embedded hardware such as Raspberry Pi without writing any C-code
3. see how easy it is to program low-cost hardware with Simulink
4. incorporate user-written libraries and functions in C into their Simulink models and embedded applications

Getting Started

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

a. To learn more about MATLAB and Simulink, check out interactive tutorials at: mathworks.com/academia/student_center/tutorials/
b. For the latest information about Raspberry Pi Support from Simulink, see: mathworks.com/hardware-support/raspberry-pi.html
c. Supported hardware for project based learning: mathworks.com/academia/hardware-resources/

Introduction: 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 sections R1-R3 for your convenience.

Start at the section relevant to your Simulink experience level:

Section	Topics Covered	Level	Recommended Experience
I1, I2	Run-through options, Notation and Formatting	Introductory	None
P0.1, P0.2, P0.3	Getting Familiar with Simulink	Beginner	None
P1.1	Checking Product Installation	Beginner	Simulink
P1.2	System Design with Simulink	Experienced	Simulink, SSP* for RPi
P2.1 – P2.4	Image Inversion/ Object Detection	Experienced	Simulink, SSP* for RPi
P3.1 – 3.6	Edge Detection	Advanced	Simulink, SSP* for RPi
P4.1 – 4.3	Optional: GPIO & C Programming	Experienced	Simulink, SSP* for RPi

*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]** (**Figure 1**).
- Menu navigation in Simulink is shown as a sequence, e.g., **MyFiles > SoftwareFiles > SoundTone1 > Run**.
- MATLAB code to be run / copied, is included in the document as below:

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

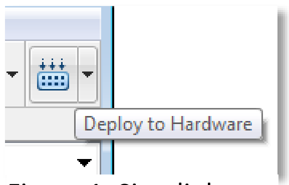


Figure 1: Simulink button example

Getting Familiar with Simulink

P0.1 Simulink

Simulink® is a block diagram environment for multi-domain simulation and Model-Based Design. It supports system-level design, 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.

Key capabilities

- 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 for differential equations.
- 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, Simulink models can also be converted to C-code optimized for specific embedded platforms.

P0.2 Simulink Library Browser and Simulink Model

To get started with Simulink, launch the Simulink Library Browser. To do this, click on the Simulink Library Browser button on the Home Tab of MATLAB Desktop, or type **`simulink`** at the MATLAB command prompt.

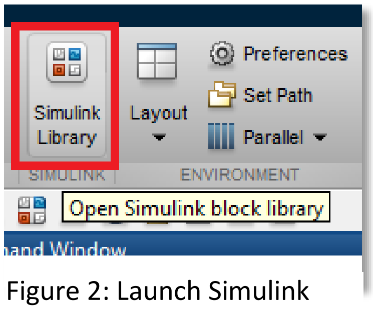


Figure 2: Launch Simulink Library Browser

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 (**Figure 2**), you will see a window like **Figure 3** 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 Raspberry Pi Hardware, which is highlighted below.

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.

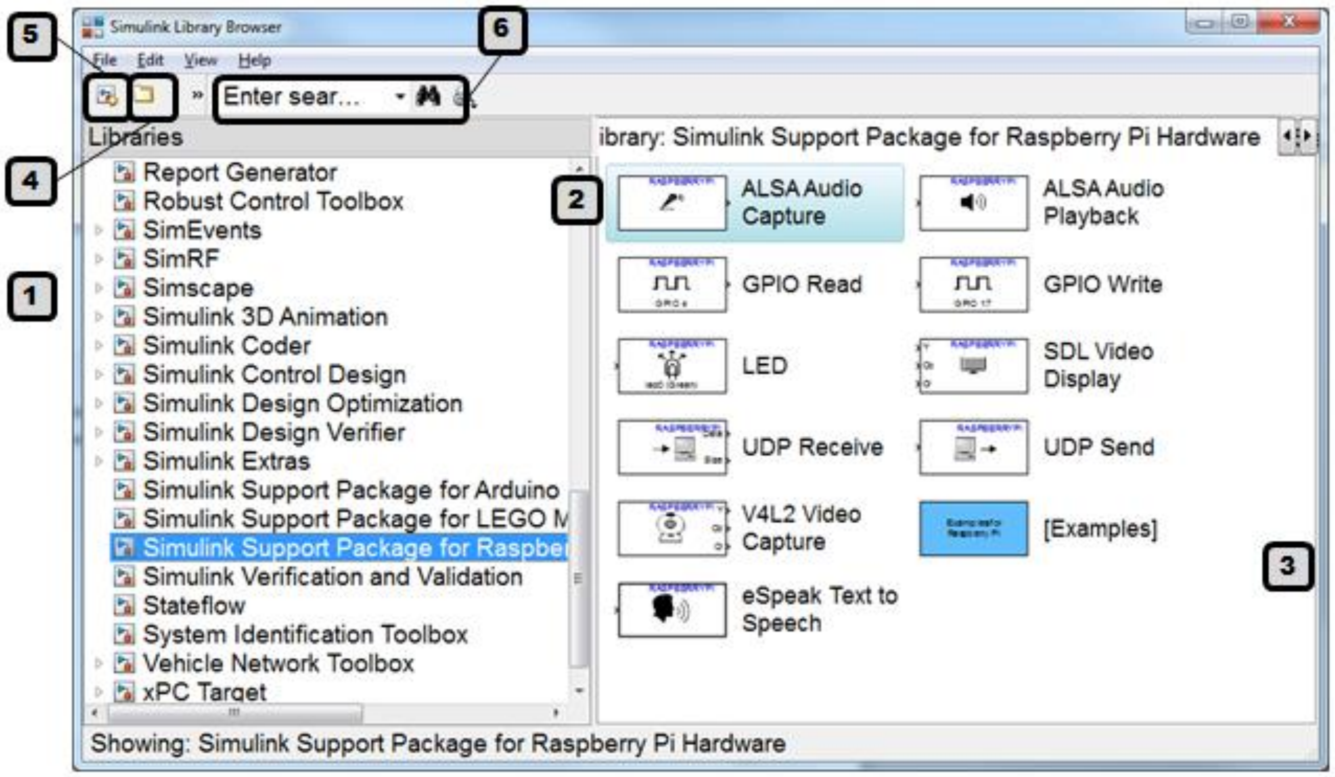


Figure 3: Simulink Library Browser

1	Simulink Library Browser	4	**[Open]** an existing Simulink model
2	Simulink block libraries	5	**[Create]** a new model
3	Library blocks	6	**[Search]** for library blocks

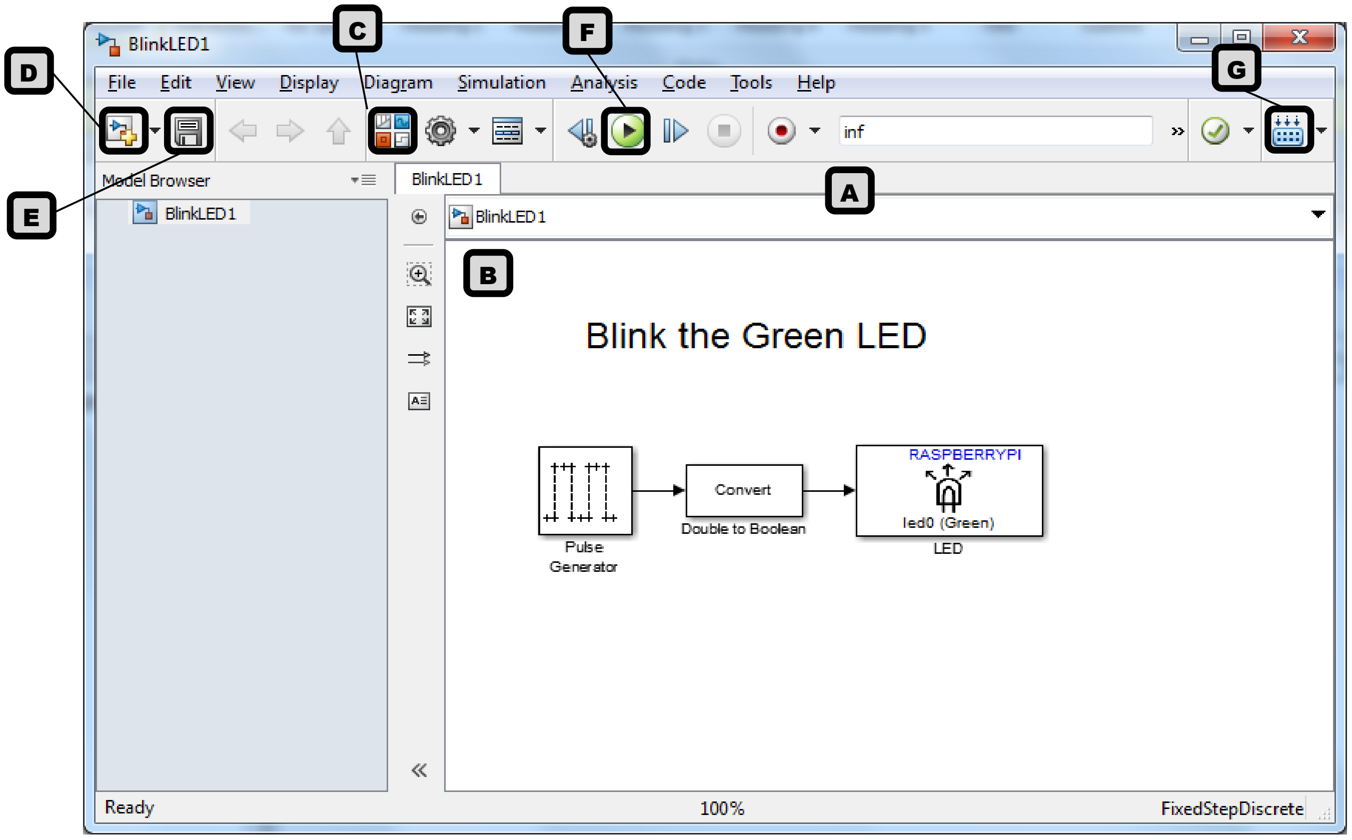


Figure 4: Simulink model

A	Simulink Model Window	E	**[Save]** a Simulink model
B	Simulink model	F	**[Run]** Simulink model
C	**[Open]** the Simulink Library Browser	G	**[Deploy]** Simulink model to hardware
D	**[Create]** a new Simulink model		

P0.3 Simulation with Simulink

A Simulink model allows you to develop a high-level, graphical design of your algorithm or system.

Say for example XYZ Automobiles is trying to design a new car. As you can imagine, a car is a complicated system with different mechanical, electrical and hydraulic components, whose interaction is controlled by intelligent computers running embedded code. Any mistakes in the design process can result in significant revenue loss for XYZ Automobiles. To avoid this, they will actually design a car and test a significant part of its performance in a computer simulation environment, before they even make it.

The industry standard way to do this is by creating mathematical models of different components, and then simulating their interaction based on intelligent algorithms, that make sure that things behave in a nice way. This approach is becoming standard in almost all automotive, aerospace, defense and hi-tech industries now.

A Simulink model allows you to design these intelligent algorithms and test them, before running them in the real world. In addition to that, the intelligent behavior can also be translated to code in different low-level languages – e.g. C, HDL, PLC text – that can be deployed directly to embedded controllers like the Engine Control Unit (ECU) in your car.

We will work with the code generation capability of Simulink for the rest of this workshop. Let's take a look at the ***simulation*** capabilities to start with.

Once you have created a Simulink model (**Figure 6**) representing your system, and set up appropriate simulation parameters, you can just click on the green RUN button, and observe the behavior of your system over time.

If you are curious, this is a simplified version of how Simulink works:

When you click on the green run button (**Figure 5**), the Simulink model that you created with built-in or custom blocks is translated to an equivalent differential equation. Simulink then uses one of the built-in differential equation solvers, to solver this differential equation over time, and consequently, simulate the dynamic behavior of the system.

Given the complex nature of this whole process, there are several pieces of information needed from the system designer – e.g. the nature of data exchanged between blocks, some notion about the dynamic nature of the system, simulation time and step size, interaction preferences, etc. All of these can be controlled by changing the appropriate settings in the 'Model Configuration Parameters' (**Figure 8**) of each Simulink model.

To learn more about key features of Simulink, check out the following webpage: mathworks.com/products/simulink/features.html

If you want to see how simulation works:

a. Open the model for simulating a bouncing ball, by typing **`sldemo_bounce`** at the MATLAB Command Prompt.
b. Check model configuration from **Simulation > Model Configuration Parameters** (**Figure 8**)
c. Run the model by clicking on the green **[Run]** (**Figure 5**) button to run the simulation and observe the behavior (**Figure 7**).

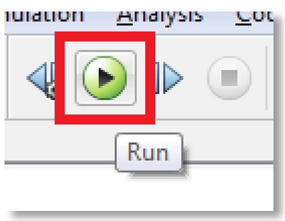


Figure 5: Run Button

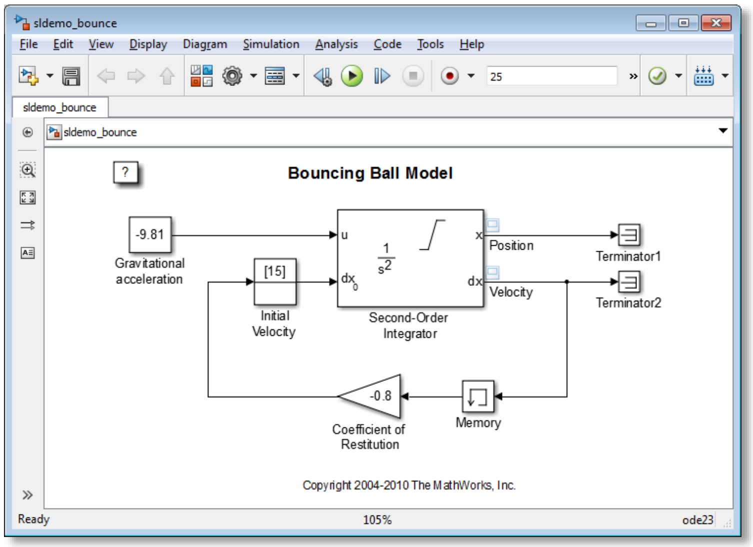

sldemo_bounce
File Edit View Display Diagram Simulation Analysis Code Tools Help
25
Bouncing Ball Model
-9.81
Gravitational acceleration
[15]
Initial Velocity
u
1/s2
x
dx
dx0
Second-Order Integrator
Position
Velocity
Terminator1
Terminator2
-0.8
Coefficient of Restitution
Memory
Copyright 2004-2010 The MathWorks, Inc.
Ready
105%
ode23

Figure 6: Bouncing ball model (sldemo_bounce.slx)

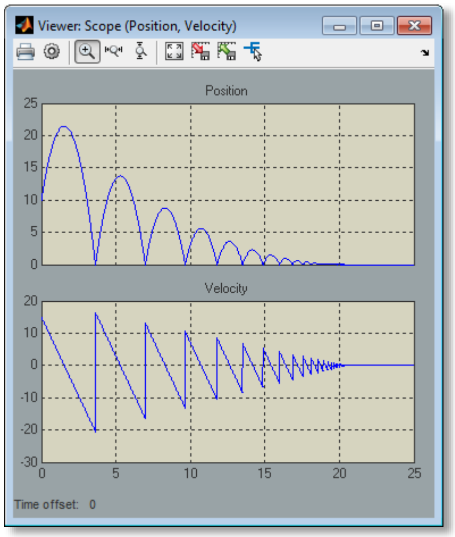
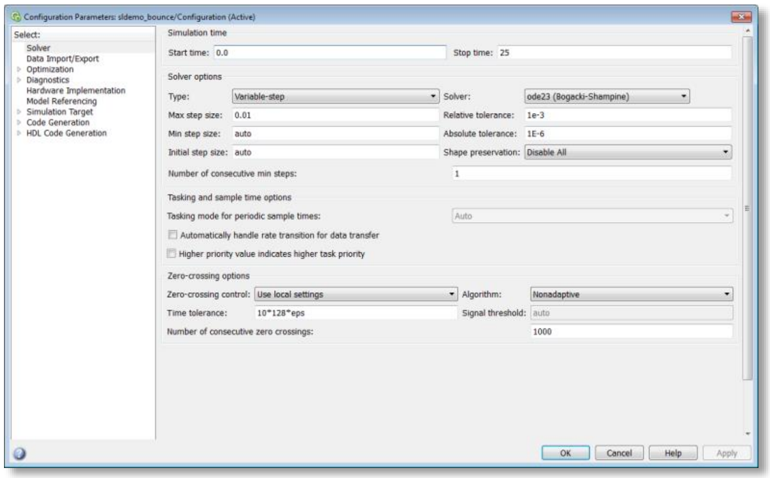


Figure 7: Bouncing ball visualization



Configuration Parameters: sldemo_bounce/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
- HDL Code Generation

Simulation time

Start time: 0.0 Stop time: 25

Solver options

Type: Variable-step Solver: ode23 (Bogacki-Shampine)

Max step size: 0.01 Relative tolerance: 1e-3

Min step size: auto Absolute tolerance: 1E-6

Initial step size: auto Shape preservation: Disable All

Number of consecutive min steps: 1

Tasking and sample time options

Tasking mode for periodic sample times: Auto

☐ Automatically handle rate transition for data transfer

☐ Higher priority value indicates higher task priority

Zero-crossing options

Zero-crossing control: Use local settings Algorithm: Nonadaptive

Time tolerance: 10*128*eps Signal threshold: auto

Number of consecutive zero crossings: 1000

Figure 8: Simulink model configuration parameters

P0.4 More on Simulink

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

MATLAB Desktop > Help > Documentation > Simulink > Examples

Project 1: Blink LED – First Application with Simulink and Raspberry Pi

P1.1 Blink the Small Green LED on the Raspberry Pi

Objective: Check hardware and software installation.

Tasks/Challenge:

- Run an existing model, **BlinkLED1.slx** (**Figure 9**).

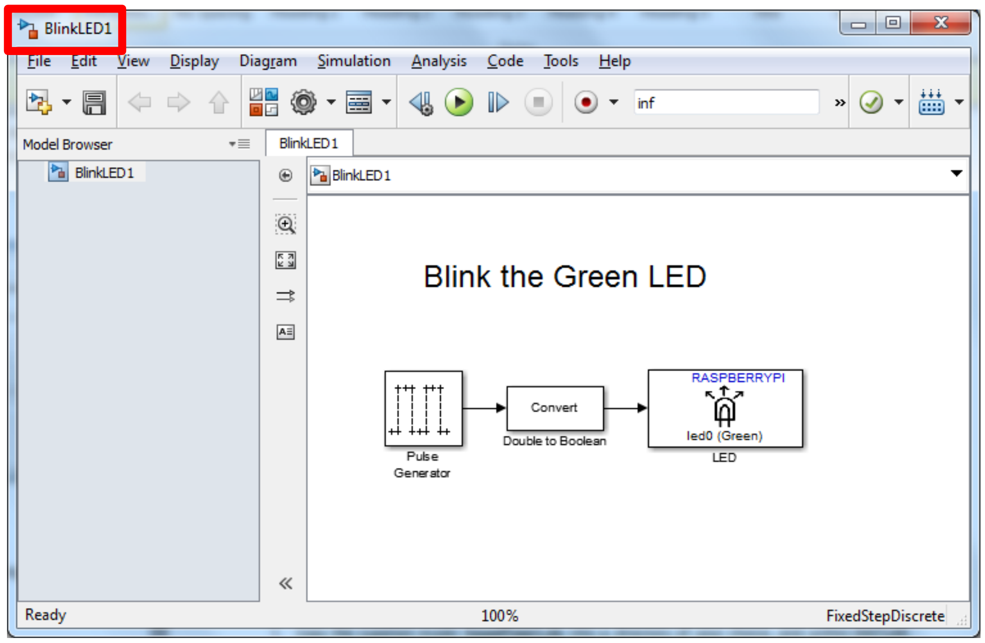


Figure 9: Simulink model BlinkLED1.slx

Steps/Approach:

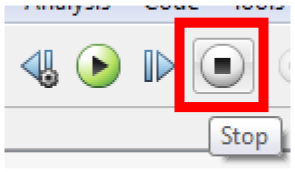
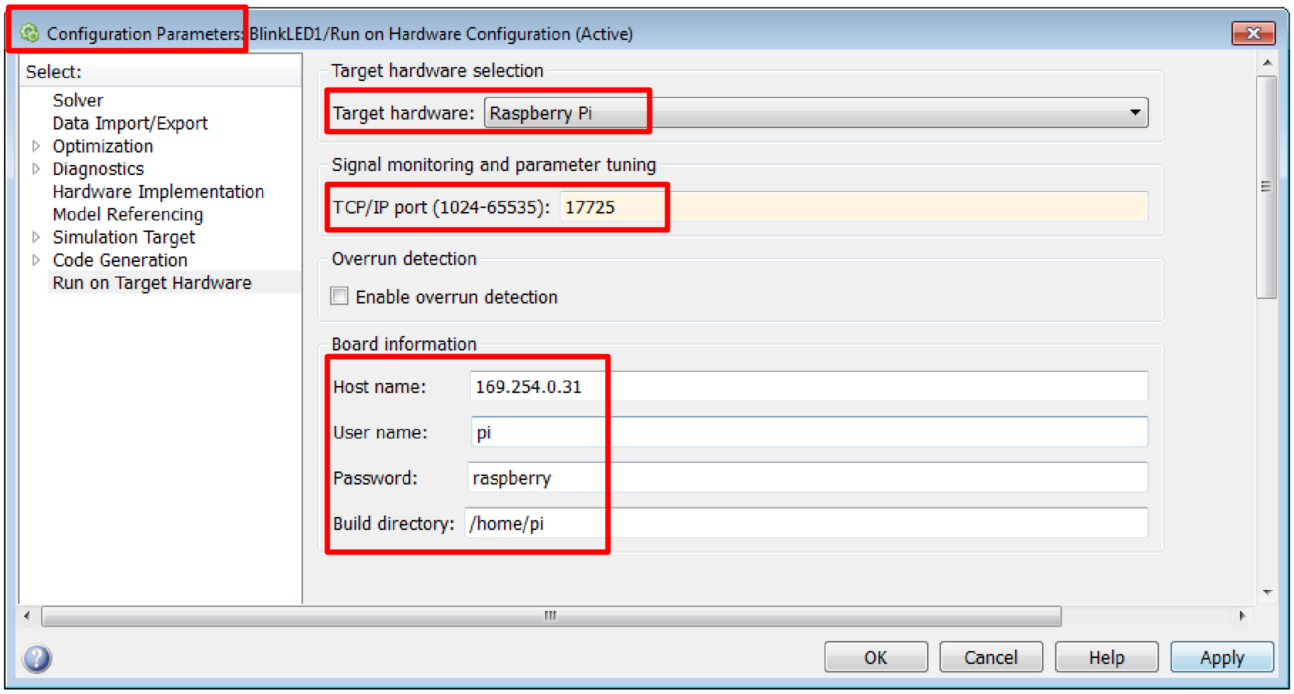


Figure 10: Stop button

1. If a model is already running, click the stop button (**Figure 10**).
2. Within MATLAB, navigate to the Working directory.
3. Open the model **BlinkLED1.slx** by double-clicking on the file in the *Current Directory* browser.
4. Check that the Raspberry Pi is powered on and connected to the host PC with a Ethernet cable Select **BlinkLED1.slx > Tools > Run on Target Hardware > Options** menu item, and check for the following settings (**Figure 11**), note your Host name IP address and Click **[OK]** to close the window.
   Note: Your Host name IP address may be different; common values are: 169.254.0.31, or 169.254.0.2.


Configuration Parameters: BlinkLED1/Run on Hardware Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
- Run on Target Hardware

Target hardware selection

Target hardware: Raspberry Pi

Signal monitoring and parameter tuning

TCP/IP port (1024-65535): 17725

Overrun detection

☐ Enable overrun detection

Board information

Host name: 169.254.0.31

User name: pi

Password: raspberry

Build directory: /home/pi

Figure 11: BlinkLED1 Configuration Parameters

5. To test your connection to the Rapsberry Pi, at the MATLAB prompt type `!ping 169.254.0.31` (or the using the Host name IP address of your Raspberry Pi from step 4). Ensure that you get a positive response and not a "Request timed out" reply.
6. On the **BlinkLED1** model window, **[Deploy]** the model to the Raspberry Pi (**Figure 12**). Note that:
   - This action initiates the build, download and run procedure. Progress is reported in the status bar at the bottom left of the model window, terminating with the message, 'Model successfully downloaded …'
   - The build process automatically starts / runs the executable on the Raspberry Pi as can be observed from the blinking LED (LED0) on the Raspberry Pi (**Figure 13**).

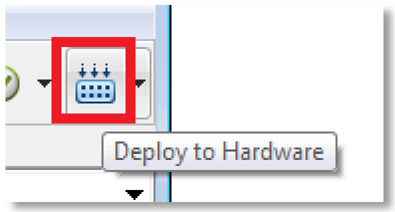


Figure 12: Deploy to Hardware button

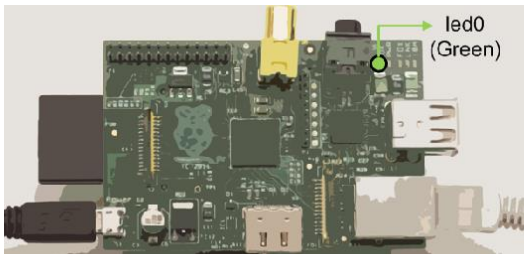


Figure 13: LED0 on Raspberry Pi board

Additional notes:

- It may take a few seconds to open a Simulink model, unless you have already started Simulink.

- It is not possible to compile the model without having a Raspberry Pi connected. Simulink will produce an error message.
- Tip: If you do not have a Raspberry Pi, you can test the syntactic integrity of your model using the menu option **Simulation > Update Diagram**, (or simply **[Ctrl-D]**).
- The program running on the Raspberry Pi can be started and stopped from MATLAB as follows:
  - At the MATLAB prompt assign `h = raspberrypi`. This will create a Raspberry Pi object in MATLAB which can then communicate with the unit. You can note the parameters of the object.
  - Type `h.connect` at the MATLAB prompt which connects the object to the Raspberry Pi.
  - `h.stop('BlinkLED1')` stops the program running and `h.run('BlinkLED1')` runs the application BlinkLED1.
  - It is also possible to log on to the Raspberry Pi via a shell. `h.openShell('ssh')` opens a shell. Username is **pi** and password **raspberry**. The program can also be run and stopped like all other Linux application at the OS level of the Raspberry Pi.

P1.2 Build Your Own BlinkLED1 Model

Objective: Learn how to create models in Simulink

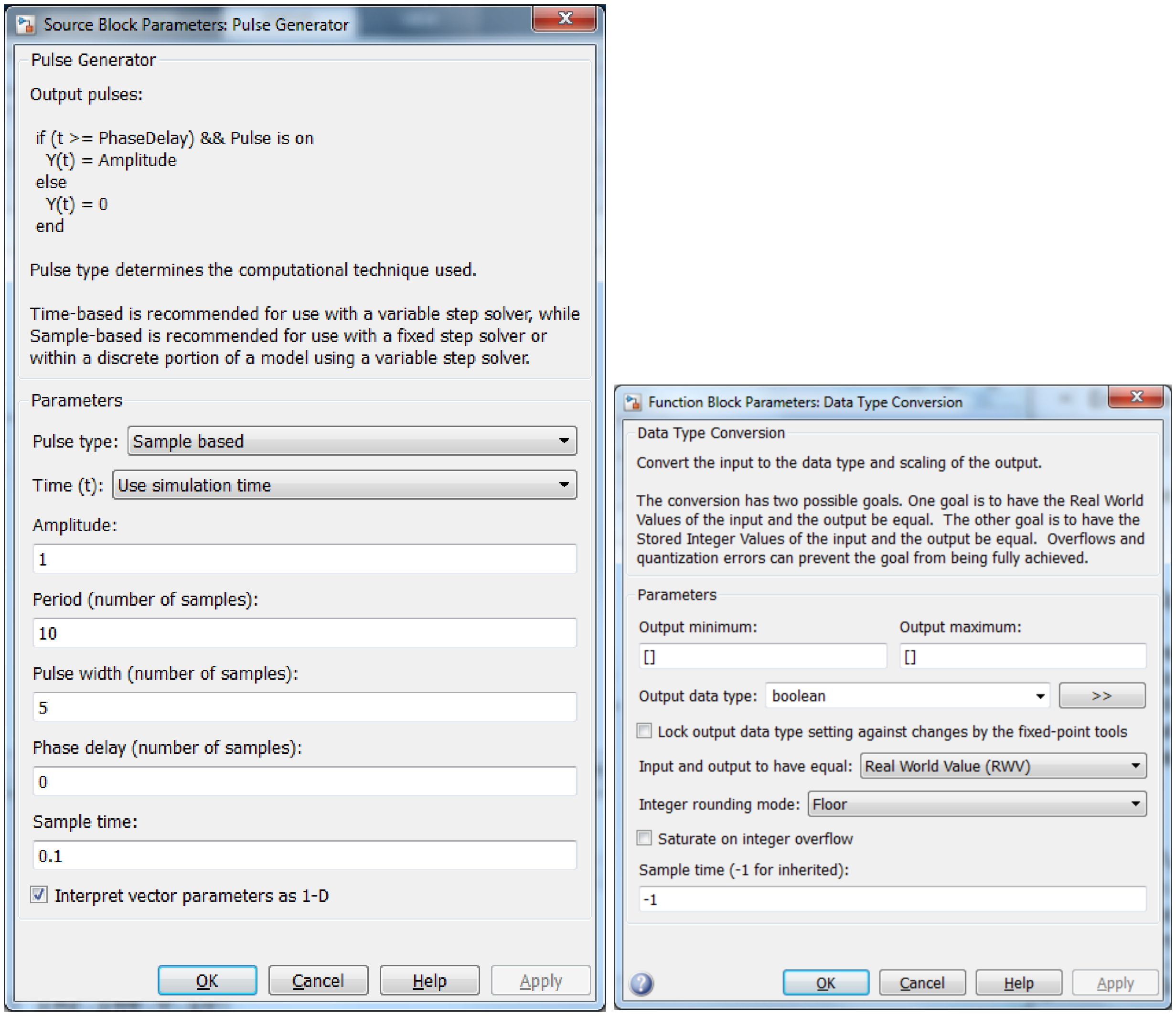
Task/Challenge: Re-construct the **BlinkLED1.slx** (**Figure 9**) model from scratch to get familiar with the design process.

Steps/Approach:

1. **[Open]** the Simulink Library Browser.
2. **[Create]** a new model and **[Save]** the new 'untitled' model as **myBlinkLED1.slx** in the working directory.
3. **[Search]** for the three blocks listed in the table below and **[Drag and drop]** them into the **myBlinkLED1.slx** window.

LIBRARY	BLOCK	PROPERTY	SETTING/VALUE
Simulink Support Package for Raspberry Pi Hardware	LED		
Simulink > Signal Attributes	Convert		
Simulink > Sources	Pulse Generator	**Pulse type:** **Time:** **Amplitude:** **Period:** **Pulse Width:** **SampleTime:**	Sample Based Use Simulation Time 1 20 10 0.1

4. After configuring the Pulse Generator, Convert and LED blocks according to the settings in the table above, their dialog boxes should look like **Figure 14**:



Source Block Parameters: Pulse Generator

Pulse Generator

Output pulses:

```
if (t >= PhaseDelay) && Pulse is on
  Y(t) = Amplitude
else
  Y(t) = 0
end
```

Pulse type determines the computational technique used.

Time-based is recommended for use with a variable step solver, while Sample-based is recommended for use with a fixed step solver or within a discrete portion of a model using a variable step solver.

Parameters

Pulse type: Sample based

Time (t): Use simulation time

Amplitude: 1

Period (number of samples): 10

Pulse width (number of samples): 5

Phase delay (number of samples): 0

Sample time: 0.1

[✓] Interpret vector parameters as 1-D

Function Block Parameters: Data Type Conversion

Data Type Conversion

Convert the input to the data type and scaling of the output.

The conversion has two possible goals. One goal is to have the Real World Values of the input and the output be equal. The other goal is to have the Stored Integer Values of the input and the output be equal. Overflows and quantization errors can prevent the goal from being fully achieved.

Parameters

Output minimum: []

Output maximum: []

Output data type: boolean

[ ] Lock output data type setting against changes by the fixed-point tools

Input and output to have equal: Real World Value (RWV)

Integer rounding mode: Floor

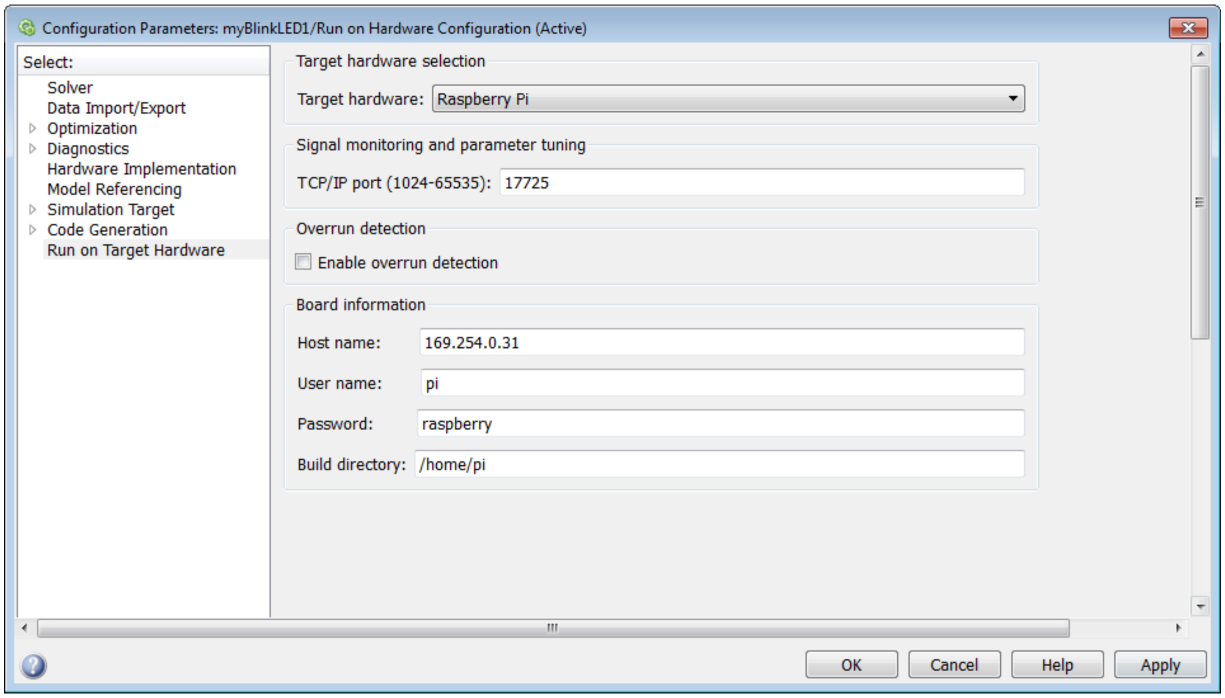
[ ] Saturate on integer overflow

Sample time (-1 for inherited): -1

Figure 14: Block Settings for Pulse Generator & Convert blocks for BlinkLED1.slx model

Note: We use a sample-based pulse generator, since running on hardware requires fixed-step solvers.

5. Use **Display > Signals and Ports > Port Data Types** and select the **Simulation > Update Diagram** menu option (or **Ctrl + D** shortcut) to highlight signal types and potential mismatch/truncation issues.
   - Simulink ensures that the data types of Raspberry Pi block inputs are automatically type cast to the appropriate data type for the underlying hardware, but it is good practice to be aware of the data types used in your model.
6. To configure the model for real-time implementation on the Raspberry Pi select **Tools > Run on Target Hardware > Prepare to Run…** and check for the following settings in the **Run on Target Hardware** and **Solver** panes (see **Figure 15 & 16**):



Configuration Parameters: myBlinkLED1/Run on Hardware Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
- Run on Target Hardware

Target hardware selection

Target hardware: Raspberry Pi

Signal monitoring and parameter tuning

TCP/IP port (1024-65535): 17725

Overrun detection

- [ ] Enable overrun detection

Board information

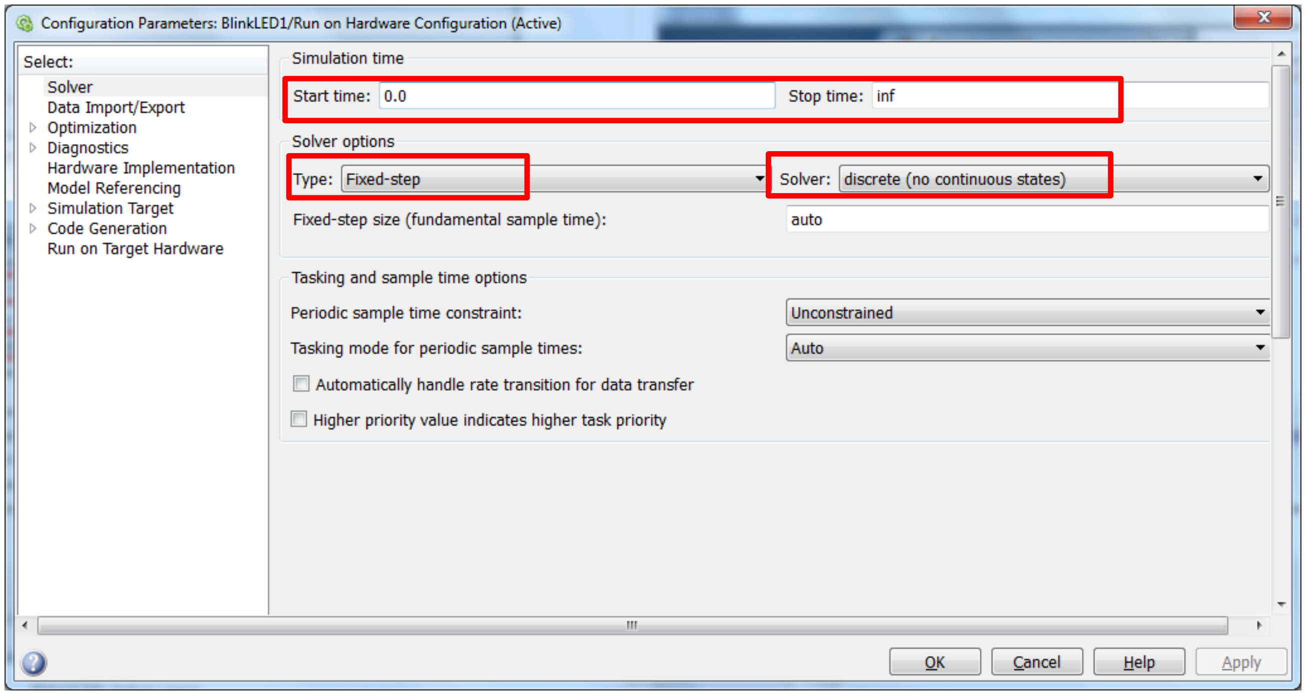
Host name: 169.254.0.31

User name: pi

Password: raspberry

Build directory: /home/pi

Figure 15: Run on Target Hardware pane



Configuration Parameters: BlinkLED1/Run on Hardware Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
- Run on Target Hardware

Simulation time

Start time: 0.0 Stop time: inf

Solver options

Type: Fixed-step Solver: discrete (no continuous states)

Fixed-step size (fundamental sample time): auto

Tasking and sample time options

Periodic sample time constraint: Unconstrained

Tasking mode for periodic sample times: Auto

- [ ] Automatically handle rate transition for data transfer
- [ ] Higher priority value indicates higher task priority

Figure 16: Solver pane

7. Click **[Apply]** to apply the changes and **[OK]** to dismiss the dialog. It is also good practice to **[Save]** your work before proceeding to run the model.
8. Verify that the Raspberry Pi is powered-on and connected.
9. **[Deploy]** the model to the Raspberry Pi (**Figure 12**).
10. Once the program starts running you should see the blinking LED (**Figure 13**).

Additional notes:

- The program can be started and stopped from MATLAB as follows:
  - At the MATLAB prompt assign `h = raspberrypi`. This will create a Raspberry Pi object in MATLAB which can then communicate with the unit.
  - Type `h.connect` at the MATLAB prompt which connects the object to the Raspberry Pi.
  - `h.stop('myBlinkLED1')` stops the program running and `h.run('myBlinkLED1')` runs the application BlinkLED1.

Project 2: Object Detection with Video Processing

P2.1 Image Inversion: Simple Video Processing

Objective: To use the live video input, build a simple video processing application and get familiar with External Mode* operation of Simulink.

Tasks/ Challenge: Design and build an application that takes an video input and inverts the colour values. Also use external mode to change the "inversion" threshold and examine the effect on the resulting video.

Steps / Approach:

1. Start building a new model using the following blocks

LIBRARY	BLOCK	PROPERTY	SETTING/VALUE
Simulink Support Package for Raspberry Pi	V4.2 Video Capture	**Image Size:** **Pixel Format:** **Sample Time:**	160 x 120 RGB 0.1
	SDL Video Display	**Pixel Format:**	RGB
Simulink > Commonly Used Blocks	Constant	**Constant Value:** **Output Data Type:**	255 uint8
	Sum	**Icon Shape:** **List of signs:**	Round or Rectangular -+

2. Apply the same procedure to construct the **ImageInversion** model shown in **Figure 17**. You must configure the model for running on Raspberry Pi. You'll also need to enable External Mode* as shown in **Figure 18**. When using External Mode, you need to use the **RUN** button to deploy the model, and the **STOP** button to stop the model (**Figure 18**). The resulting display is shown in **Figure 19**.

   * External Mode (**Figure 18**) allows you to run your model on the Raspberry Pi, 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 constant block in **Figure 17**) and the parameter is automatically passed to the Raspberry Pi which uses the new parameter without pausing it's operation. External Mode also allows you to log and view data while using Simulink window. With this model we can see the output of the model with the SDL Display block (**Figure 18**) on our host computer.

   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 18**).

   Note: If your model stops working after 10 seconds, you may have forgotten to adjust the Stop time in the Solver pane (**Figure 16**).

Note: To set the Constant Output Data Type to **uint8**, use the Signal Attributes tab. Double click on the Constant block and go to the Signal Attributes tab in the dialogue box.

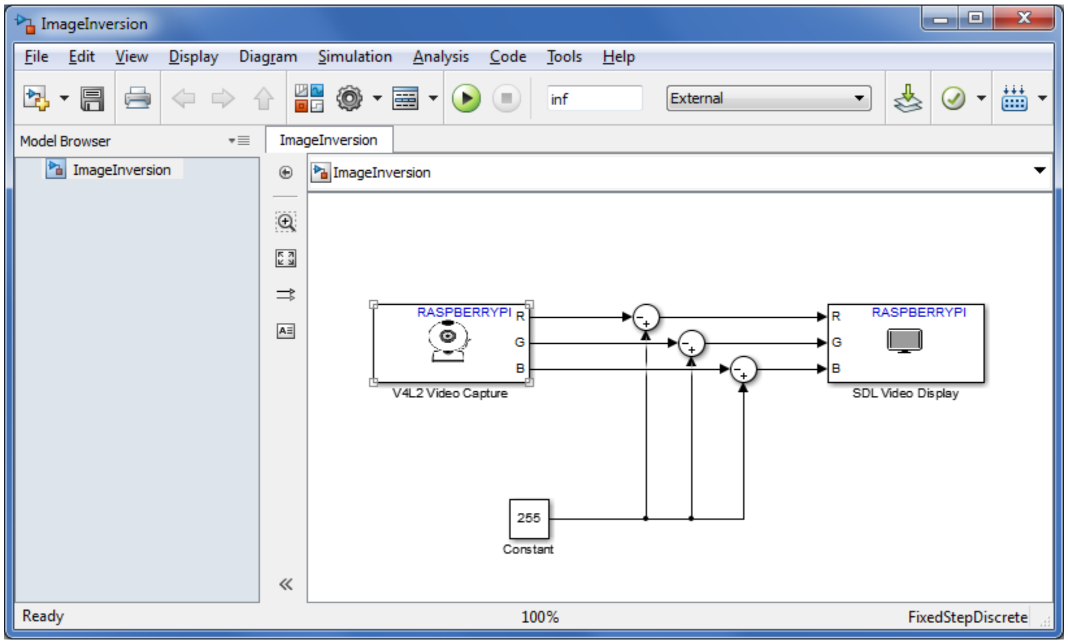
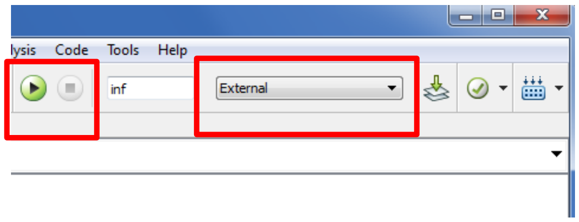


Figure 17: Image Inversion model

Question: (Answers on last page of manual)

1. What happens when you change the constant 255 to other values?



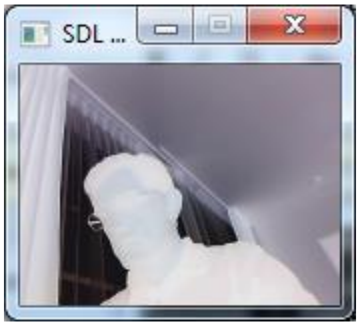


Figure 18: External Mode, Run, and Stop buttons

Figure 19: Image Inversion display

P2.2 Object Detection: Simple Object Detection

Objective: To build an application that detects a green object and is able to detect it around the screen by placing a red square in the centroid of the object. Learn about the following product features: external mode, data logging features of scopes, how to build subsystems, MATLAB functions and use built-in functions.

Task/ Challenge: Build a Simulink model in stages that is capable of detecting a green objects in the input video. The idea is to perform an appropriate arithmetic transformation on the RGB components of the colour image to

- Get an image intensity of green, and then to
- Set a threshold in order to convert the image into binary & compute the centroid
- Mark the centroid of the area with a red dot, and output to the display

Steps/ Approach: There are 3 parts to this project.

P2.2.1 Part 1: Build the Object Detection Model

First we need to build the model, which uses the webcam as input, detects a green object, and outputs to the display.

1. Use the following blocks to build the model

LIBRARY	BLOCK	PROPERTY	SETTING/VALUE
Simulink Support Package for Raspberry Pi	V4.2 Video Capture	**Image Size:** **Pixel Format:** **Sample Time:**	160 x 120 RGB 0.1
	SDL Video Display	**Pixel Format:**	RGB
Commonly Used Blocks	Subsystem		
Inside Subsystem			
Commonly Used Blocks	Constant	**Constant Value:**	255
	Sum	**Icon Shape:** **List of signs:**	Round or Rectangular --+
	Inport		
	Outport		
Math Operations	Product	**No of Inputs:**	*/
	Gain	**Gain:**	255
Logic and Bit Operations	Relational Operator	**Relational Operator:**	>=

2. Build **ObjectDetection.slx** model as in **Figure 20 & 21**. The Subsystem block helps you visually organize your model and does not affect the numeric calculations or operation on the Raspberry Pi. Inports and Outports of the Subsystem block define the interface to the higher level in the hierarchy.

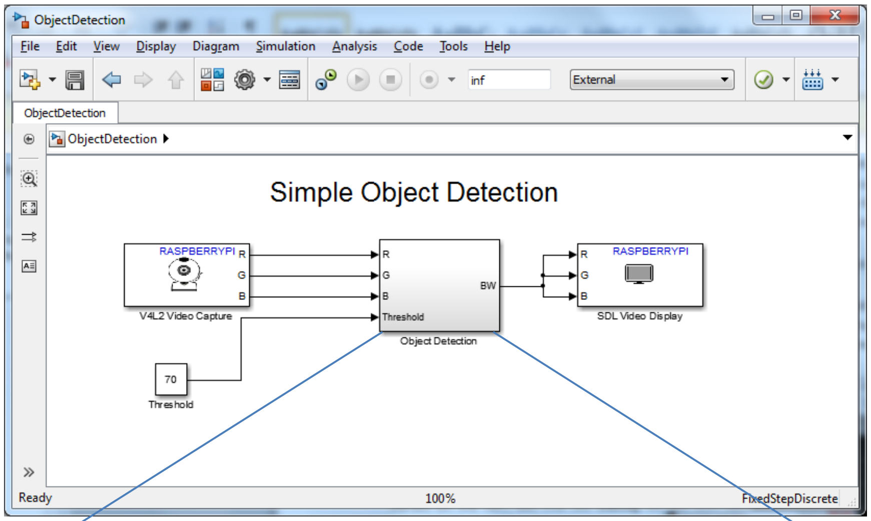


Figure 20: Object Detection Example

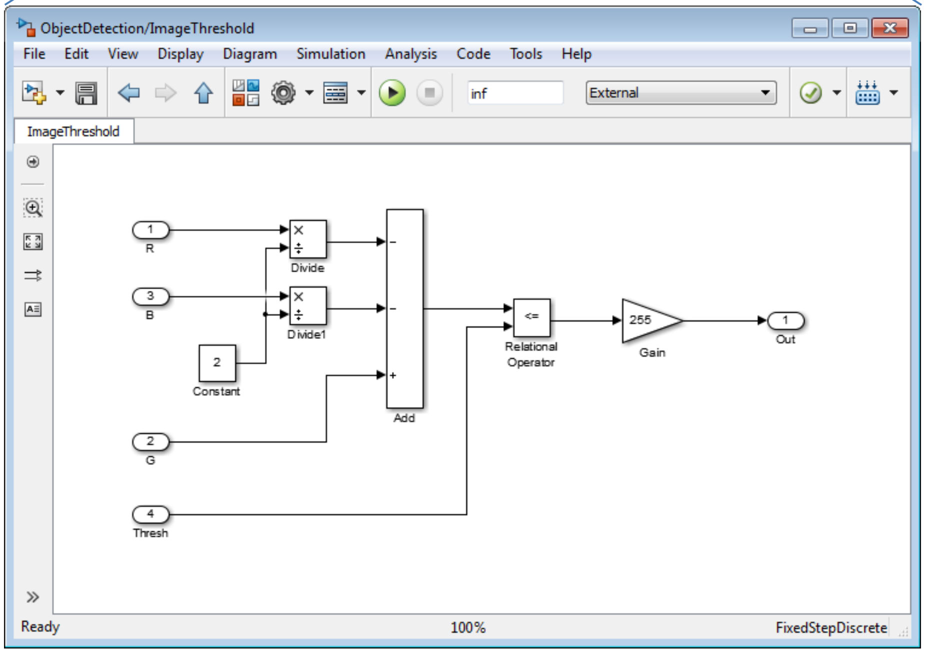


Figure 21: Object Detection subsystem

3. Note that all of the computations are to be performed in **uint8.** (Answers on last page of manual)
   a. How would you ensure that the output of the blocks are of the correct type at the end of each block computation? Hint: **Display > Signals & Ports > Port Data Types**
   b. Why should the value of Gain be 255 to display a binary image?
   c. How would you ensure that the **Add** block does not overflow?

4. Ensure that the external mode is enabled (**Figure 18**).
5. Having ensured the Raspberry Pi is powered-on and connected, click the **RUN** button to deploy the model in external mode.
6. What do you see on the display? Modify the value of the Threshold and see how it affects the size and shape of the binary image.

P2.2.2 Part 2: Find the Centroid

Next, we will find the centroid of this binary image by implementing a custom algorithm using a MATLAB Function block. The MATLAB Function block allows you to write your MATLAB code and have it execute inside of Simulink.

Sometimes it is easier to express algorithms in a textual language compared to a graphical one. In this instance assume we are already familiar with functions **find()** which returns the rows and columns of non-zero elements of a matrix, and **mean()** which returns the arithmetic mean of a vector. To find the centroid of an binary image we first need to obtain the rows and columns of non-zero elemnts and subsequently compute the mean value of the rows and columns, which in turn will give us the coordinates of the centroid.

LIBRARY	BLOCK	PROPERTY	SETTING/VALUE
Simulink > Commonly Used Blocks	Scope		
Simulink > User-Defined Functions	MATLAB Function		

7. Add the Scope block and user defined MATLAB Function block, and then insert the code as shown below in **Figures 22 & 23**.

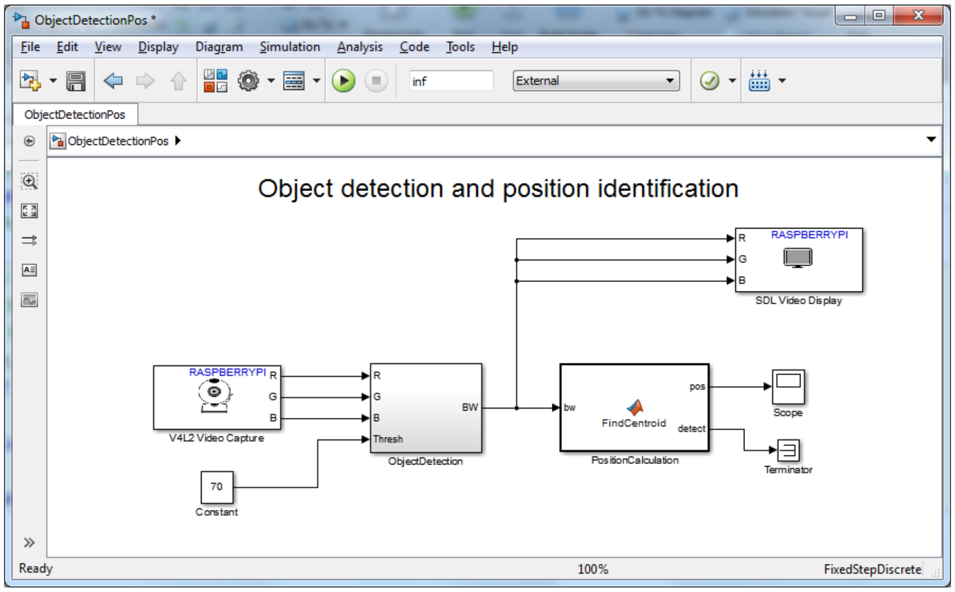
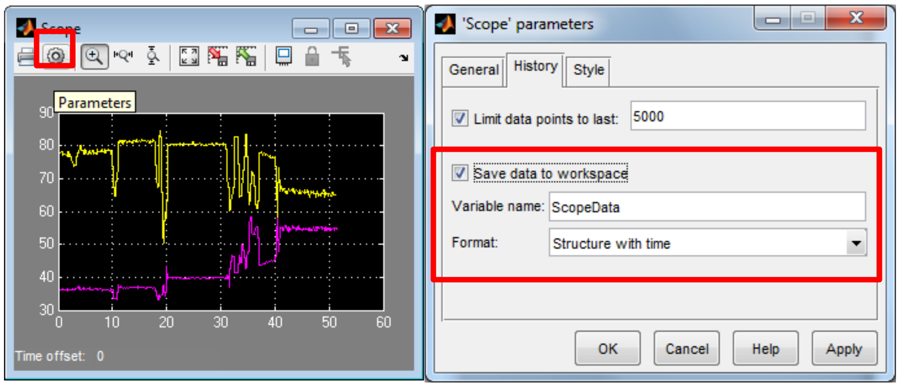


Figure 22: Object Detection model

```
function [pos, detect] = FindCentroid(bw)
%#codegen
[r,c]=find(bw);
if isempty(r)
    pos = [-1; -1];
    detect = false;
else
    pos = [mean(r); mean(c)];
    detect = true;
end
```

Figure 23: Finding the centroid script

8. You can use the logging facility in the Scope (logging the historical information stored by the Scope) to record the time response of the centroid of the object. To do this, open the Scope and click Parameters (gear icon) in upper left of the Scope window and edit the History tab (**Figure 24**).



Limit data points to last: 5000

Save data to workspace

Variable name: ScopeData

Format: Structure with time

Figure 24: Logging data with the scope

9. Log some data and plot the results in MATLAB to look at the trajectory of the object over time. Note that you can do time responses as well as plotting the x, y coordinates against each other.
10. You can plot the data using **plot** or **simplot** commands.

P2.2.3 Part 3: Mark the Centroid on the Image

And finally we now place a red marker on the centroid and output the marker on the display.

LIBRARY	BLOCK	PROPERTY	SETTING/VALUE
User-Defined Functions	MATLAB Function		

11. Create a model **ObjectMarker.slx** as below (**Figure 25 & 26**):

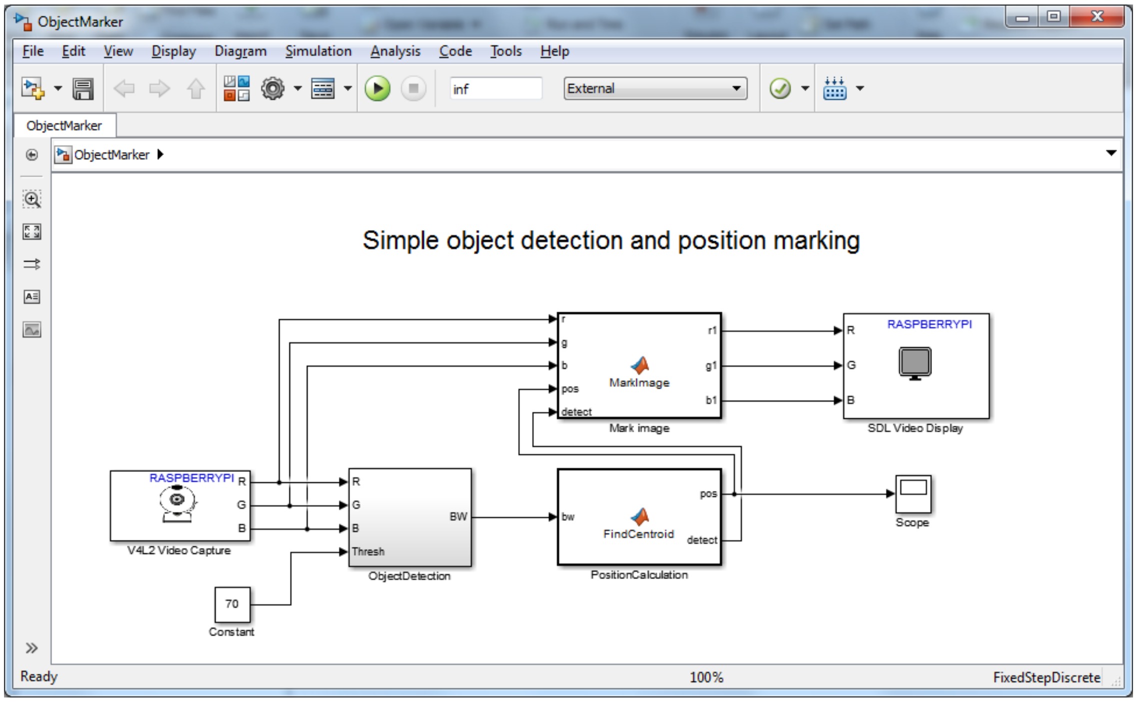


Figure 25: Object Marker model

```
function [r1,g1,b1] = MarkImage(r,g,b, pos, detect)
%#codegen
r1 = r; g1 = g; b1 = b;
if detect,
    r1(pos(1)-3:pos(1)+3, pos(2)-3:pos(2)+3) = 255;
    g1(pos(1)-3:pos(1)+3, pos(2)-3:pos(2)+3) = 0;
    b1(pos(1)-3:pos(1)+3, pos(2)-3:pos(2)+3) = 0;
end
```

Figure 26: Marking the centroid with a red square

12. Having ensured the Raspberry Pi is powered-on and connected, click the **RUN** button to deploy the model in external mode.
13. Test your model with a green object. Do you see a red square tracking the centroid of your green object? What happens if you introduce a second green object?

P2.3 Optional: Using General Purpose IO Pins

If additional hardware is provided to explore use of General Purpose IO pins on the Raspberry Pi, please follow the instructions in Appendix 4.

Project 3: Edge Detection

P3.1 Finding the Slope Local Maxima for a Two Variable Function

Objective: To build an application that computes the slope local maxima for a two variable function. Learn to simulate the system model, explore parameter tuning, and show simulation results.

Tasks/ Challenge: Build a Simulink model that determines the slope local maxima for a two variable function. The idea is to implement a two-step approach that first determines the slope of the two-variable function by computing the **gradient magnitude** of that function, and then determines the slope local maxima coordinates by finding where the slope function exceeds a given **threshold**.

Figure 27 & 28 show the one-dimensional case, while **Figure 29 & 30** depict a two-variable function and its gradient magnitude, respectively.

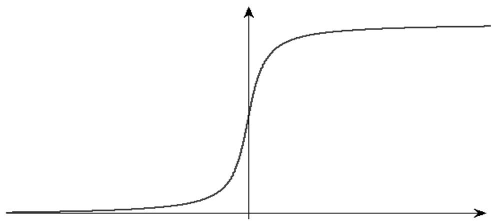


Figure 27: Steep slope of a single-variable function. It may be interpreted as the strong image contrast for an edge (one-dimensional view)

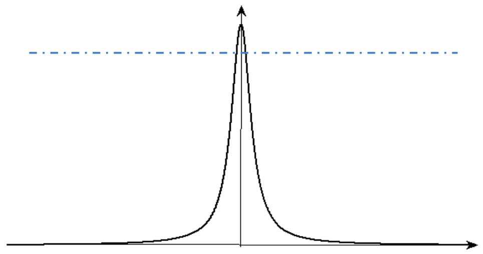


Figure 28: Maximizing the gradient function with thresholding. It may be interpreted as the image edge detection algorithm (one-dimensional view)

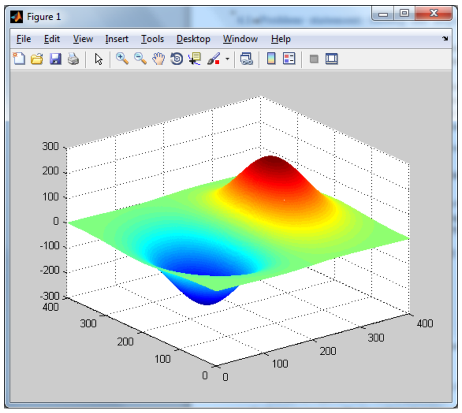


Figure 29: Two-variable function plot

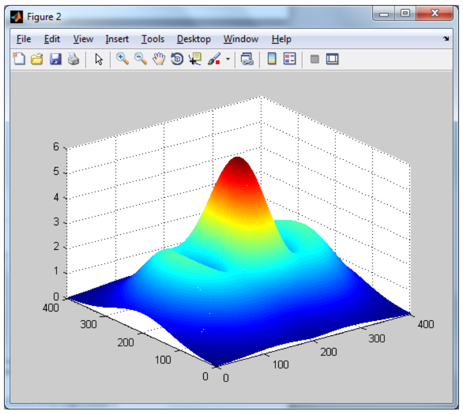


Figure 30: Gradient magnitude function

Steps / Approach:

1. Open the **MaximumSlope_start.slx** by double-clicking on the file in the *Current Directory* browser (**Figure 31**). Note that this model uses model callback functions to set variable z and intitailise the figures. This is a common way to initialise data in a Simulink model.
2. **[Search]** for the blocks listed in the table below and **[Drag and Drop]** them into the **MaximumSlope_start.slx** model.

LIBRARY	BLOCK	PROPERTY	SETTING/VALUE
Ports & Subsystems	Subsystem		
	Inport		
User-Defined Functions	MATLAB Function		
Logic and Bit Operations	Relational Operator	**Relational Operator:**	>=
Math Operations	Gain	**Gain:**	255

3. Rename the Subsystem as **MaximumSlope**, double click it and build the algorithm subsystem with the blocks of the above table, as in **Figure 32**.
4. Double click the MATLAB Function block and edit as follows:

```
function y  = ComputeGradient(u)
%#codegen

% computing gradient components
[gx,gy] = gradient(u);

% computing gradient magnitude
y = sqrt(gx.^2 + gy.^2);
```

5. Double click the Gain block. In the **Signal Attribute** tab, set the **Output data type** as **uint8**.
6. Set the **Simulation Stop Time** to **inf**.
7. **[Run]** the model in “Normal” Simulation mode. While the simulation is running, double click the Threshold block. Modify its value and see its impact on local maxima detection (**Figure 33**).
8. **[Save]** your model as **MaximumSlope.slx** in the working directory.

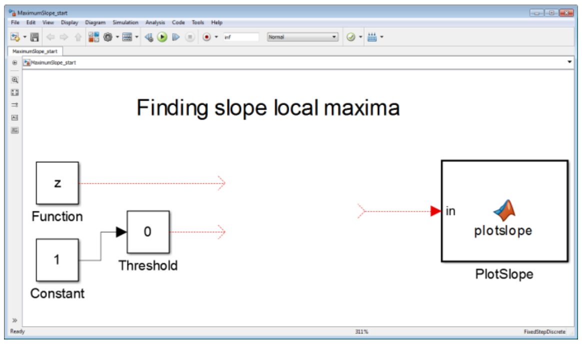


Figure 31: Starting model for slope local maxima

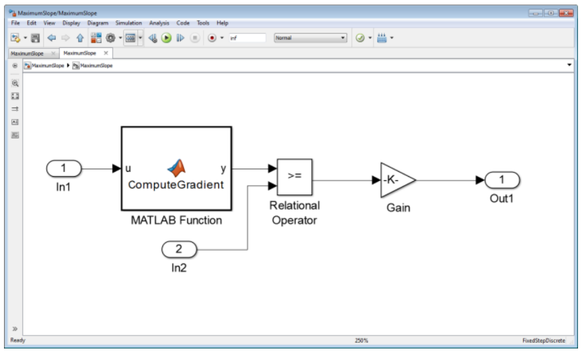


Figure 32: Maximum slope subsystem

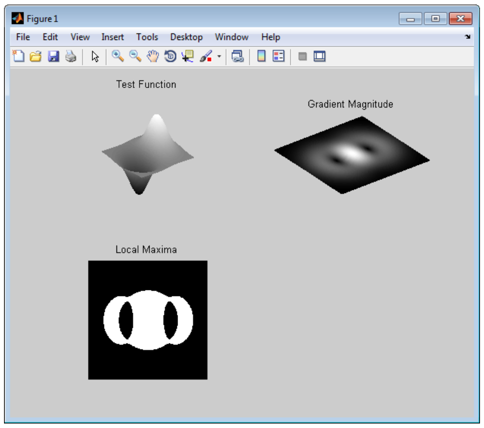


Figure 33: Gradient magnitude thresholding

P3.2 Application Example: Edge Detection Algorithm

Edge detection is a fundamental tool in image processing. It aims at identifying points in a digital image at which the image brightness changes sharply (**Figure 27**). The points at which image brightness changes sharply are typically organized into a set of curved line segments termed *edges*.

Objective: To build an algorithm that can detect the edges on an input image from a file or an image acquisition device.

Tasks/ Challenge: Given that the image intensity may be regarded as a two-variable function, build a Simulink model that detects the edges by computing its slope's local maxima, as explained in the previous Section (Figure 26).

Steps / Approach:

1. Open the **edgeDetectionReference_start.slx** by double-clicking the file in the *Current Directory* browser (**Figure 34**).

2. Open your previously saved **MaximumSlope.slx** model, copy the **MaximumSlope** Subsystem and paste it on **edgeDetectionReference_start.slx**.
3. Connect all the signals properly and save the model as **edgeDetectionReference.slx** in the current directory.
4. **[Run]** the model in "Normal" Simulation mode. While the simulation is running, double click the Threshold block. Find an appropriate value that provides accurate edge detection on the test image (**Figure 35**).

Optional:

5. You can test the algorithm on real-life images, after they are converted to grayscale. To try the model with an example image (which has been converted to grayscale for you), double-click the Constant block called TestImageBitMap, and replace **Constant value** field with **ylena**.

   Note: The constant block may act as an image source by defining an array (ylena in our case) as a constant, which is the pixel-based representation of a given grayscale image.
6. **[Run]** the simulation and tune the threshold to determine proper edge detection on such image.

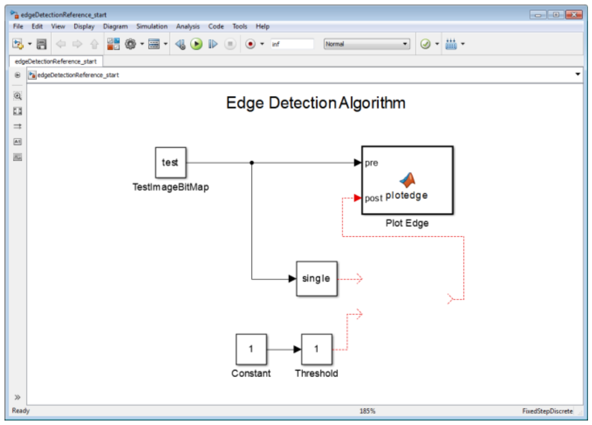


Figure 34: Reference Edge Detection starting model

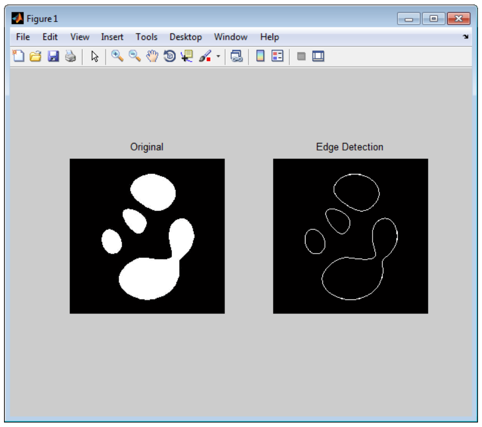


Figure 35: Edge Detection result on test image

P3.3 Implementing the Algorithm on Raspberry Pi

Objective: To implement the edge detection algorithm on Raspberry Pi and test on live video acquired from the webcam

Tasks/ Challenge: Reuse the algorithm that has been verified through simulation, and build the Simulink model to be deployed ion the target hardware.

Steps / Approach:

1. Open the **edgeDetectionReference_RPi_start.slx** by double-clicking the file in the *Current Directory* browser (**Figure 36**).
2. Open your previously saved **EdgeDetectionReference.slx** model, copy the **MaximumSlope** Subsystem and paste it on **edgeDetectionReference_RPi_start.slx**.
3. Connect all the signals properly and save the model as **edgeDetectionReference_RPi.slx** in the current directory.
4. Ensure the Raspberry Pi is powered-on and connected to the computer through the Ethernet cable. **Connect** the webcam to the USB port on the Raspberry Pi.
5. Click the **RUN** button to run the model in external mode.
6. Double click the Threshold block. Tune its value to determine proper edge detection on the video sequence (**Figure 37**).

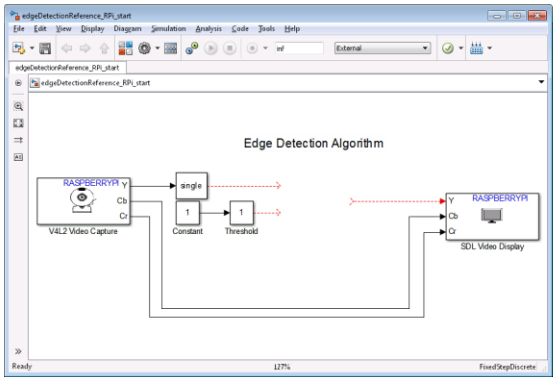


Figure 36: Edge Detection starting model

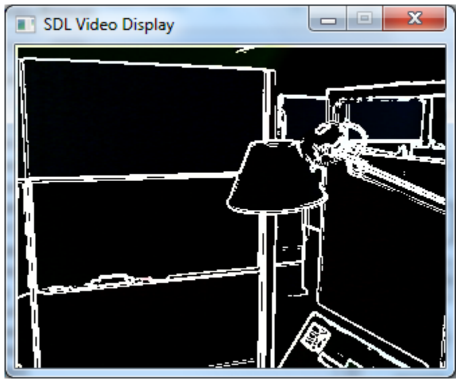


Figure 37: Live Edge Detection on Raspberry Pi

P3.4 Computing the Gradient as Two Dimensional Convolution

Objective: To get an alternative implementation of the gradient computation.

Tasks/ Challenge: Build a Simulink model that implements the gradient computation as a two-dimensional convolution.

As shown in the previous Section, edge detection requires computing the derivative of a two-dimensional image. The **Sobel** edge detector uses intensity values only in a 3×3 region around each image point to approximate the corresponding image gradient. More precisely, it uses a pair of **3x3 convolution masks**, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows). As a result, the mask is slid over the image as a **2-D convolution** with that image, manipulating a square of pixels at a time.

The actual Sobel masks are shown below:

1	0	-1
2	0	-2
1	0	-1

1	2	1
0	0	0
-1	-2	-1

Steps / Approach:

1. On the **edgeDetectionReference_RPi.slx** model, rename the **MaximumSlope** Subsystem as **MaximumSlopeFilter**, and save the model as **edgeDetectionFilter_RPi.slx**
2. Open the **MaximumSlopeFilter** and edit the MATLAB Function block as follows:

```
function y = ComputeGradient(u)
%#codegen

% computing gradient components
kx = [1,  0,  -1;
      2,  0,  -2;
      1,  0,  -1];

ky = [1,  2,  1;
      0,  0,  0;
     -1, -2, -1];

% 2-D convolution
gx = conv2(u,kx,'same');
gy = conv2(u,ky,'same');

% computing gradient magnitude
y = sqrt(gx.^2 + gy.^2);
```

3. Ensure the Raspberry Pi is powered-on and connected to the computer through the Ethernet cable. **Connect** the webcam to the USB port on the Raspberry Pi.
4. Click the **RUN** button to run the model in external mode. Double click the Threshold block. Tune its value to determine proper edge detection on the video sequence

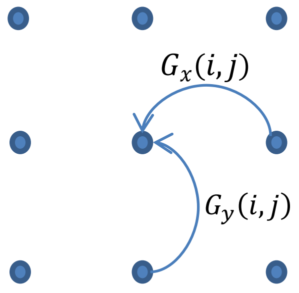
P3.5 Analyzing Alternative Algorithm Implementations

Objective: To simplify the computation of the gradient function. Compare the results to the reference algorithm and evaluate the difference through simulation.

Tasks/ Challenge: Build a Simulink model that implements the gradient computation by means of difference functions. Implement a testbench to measure the difference between the golden reference and the simplified algorithm.

In the previous section, we learnt that, according to the Sobel method, the horizontal and vertical gradient components may be computed as 2-D convolutions of the input image with two specific 3x3 masks.
Let us now consider alternative, simpler ways to implement such algorithm.

In order to estimate the gradient components G_x and G_y for a given image pixel at position (*i, j*), we can make the following assumption: we consider only the contribution of the pixels laying on the *i^th^* row and the *j^th^* column to compute G_x and G_y, respectively. In other words, we can use a simple **difference function** to approximate the computation of the derivative in the horizontal and vertical directions, as depicted in **Figure 38**.



$$\boxed{\begin{aligned} G_x(i,j) &\approx u(i,j+1) - u(i,j) \\ G_y(i,j) &\approx u(i+1,j) - u(i,j) \end{aligned}}$$

Figure 38: Approximate computation of the gradient components

Steps / Approach:

1. Open **EdgeDetectionCompare_start.slx** by double-clicking on the file in the *Current Directory* browser.
2. Copy (from your model in Project 3.4) and paste the **MaximumSlopeFilter** Subsystem, and rename it as **MaximumSlopeSimplified**.

3. Open the Subsystem and edit the MATLAB Function block as follows:

```
function y = ComputeGradient(u)
%#codegen

gx = zeros(size(u),'single');
gy = zeros(size(u),'single');

% difference function as a gradient approximation
gx(1:end - 1,:) = diff(u,1,1);
gy(:,1:end - 1) = diff(u,1,2);

% computing gradient magnitude
y = sqrt(gx.^2 + gy.^2);
```

4. Connect the **MaximumSlopeSimplified** Subsystem as in **Figure 39** shown below.

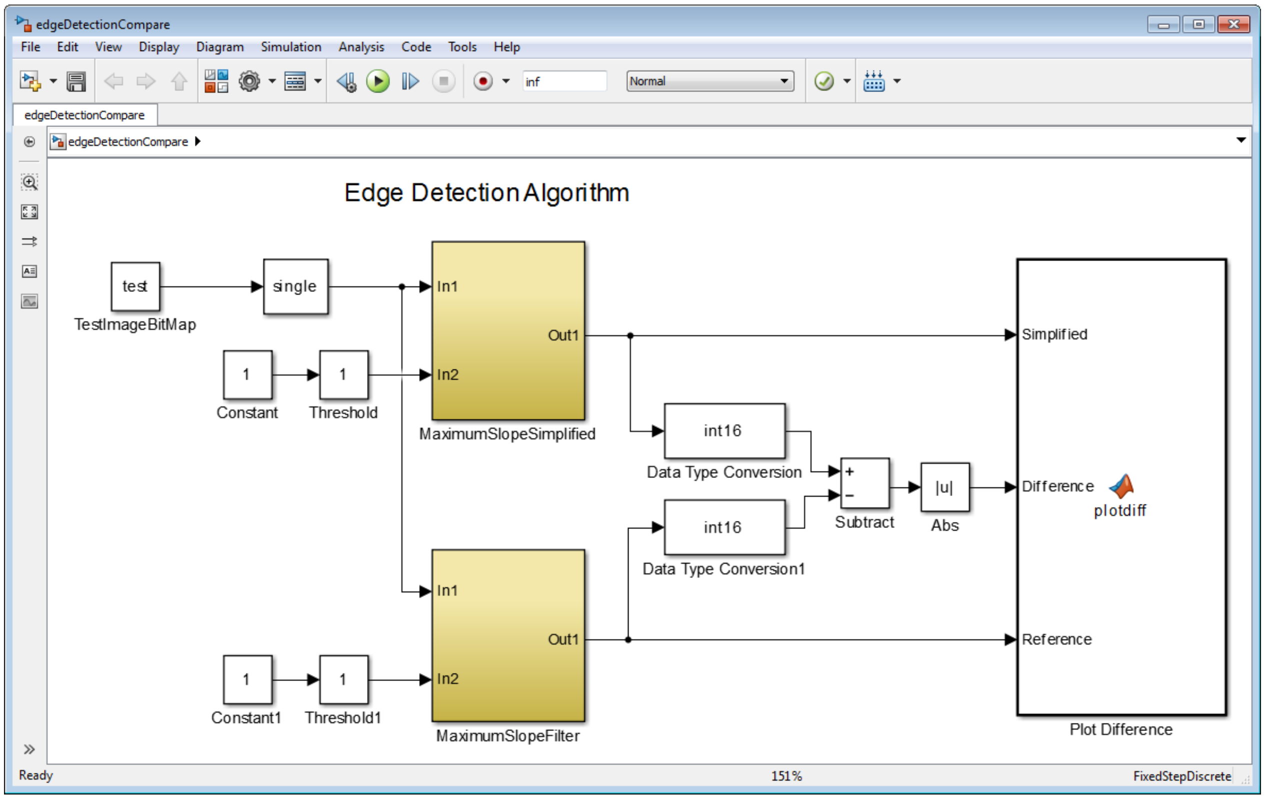


Figure 39: Comparison of two different edge detection algorithm implementations

5. **[Save]** the model as **edgeDetectionCompare.slx** in the working directory.

6. **[Run]** the model in "Normal" Simulation mode. and observe the difference image. Try and minimize the difference by tuning the thresholds.

Optional:

7. Change the input image to **ylena** and observe the results.
8. Open the previously saved **edgeDetectionFilter_RPi.slx** model. Replace **MaximumSlopeFilter** Subsystem with **MaximumSlopeSimplified**. **[Save]** the model as **edgeDetectionSimplified_RPi.slx**.
9. Click the **RUN** model in external mode and validate the simplified edge detection algorithm on the live video from the webcam.

Questions: (Answers on last page of manual)

1. How do the results of step 8 above using the simplified algorithm compare with the reference model?
2. How would you measure the difference? What you could do to minimize it?
3. Would it be possible to simplify further the edge detection algorithm? **Hint**: think about the following equation:

$$|G| = \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y|$$

P3.6 Brief Task Overrun Analysis

Objective: To evaluate the computational complexity of a given algorithm. Get familiar with the task overrun analysis capability provided by the target support package.

Tasks/ Challenge: Run the Simulink model on the target hardware by enabling the overrun detection to see if any task overrun occurs during execution. A **task overrun** occurs when a task is scheduled to execute before a previous instance of the same task has completed

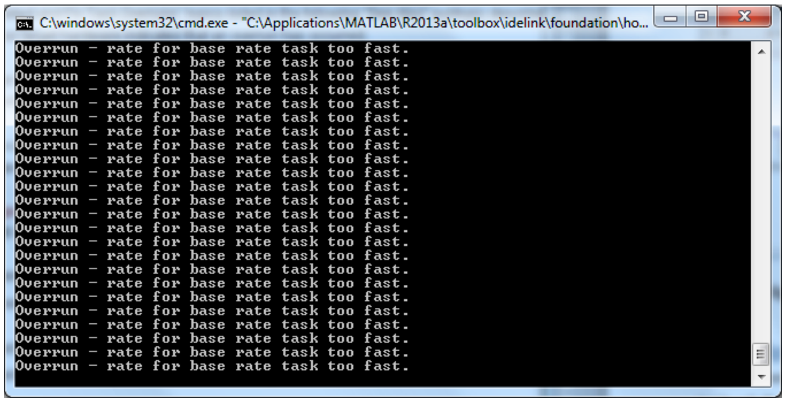
Standard scheduling works well when a processor is moderately loaded. However, if the processor becomes overloaded because of the high amount of processing involved with a specific algorithm, task scheduling may fail, thus affecting system performance.

Frame dropping is a typical effect of task overruns. The Simulink Support Package for Raspberry Pi enables actual task overrun detection for a specific model running on the target hardware.

Steps / Approach:

1. Open the previously saved **edgeDetectionFilter_RPi.slx** model.
2. Remove the Threshold block and connect the Constant block directly to the Threshold port of the **MaximumSlopeFilter** Subsystem.
3. Select **Tools > Run on Target Hardware > Options…**, check **Enable overrun detection** on the **Overrun detection** section. Click **[Ok]**.

4. On the Simulink model window, **[Deploy]** the model to the Raspberry Pi. Wait for the model to run on Raspberry Pi.
5. The windows command prompt will show the detected overruns because of too fast rate for the task, as shown in **Figure 40**.



```
C:\windows\system32\cmd.exe - "C:\Applications\MATLAB\R2013a\toolbox\idelink\foundation\ho...
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
Overrun - rate for base rate task too fast.
```

Figure 40: Overruns at the command prompt

To fix an overrun condition, you can:

1. **Increase the sample times for the model and the blocks in it. One way to do that is to decrease the video input frame rate;**
2. **Reduce the model complexity.**

Questions: (Answers on last page of manual)

a. How can you reduce the video input frame rate for your **edgeDetectionFilter_RPi.slx** model?
b. What you think it would be the maximum video input frame rate avoiding overruns? How would you verify that?
c. Try and see if you can experience overruns differences among **edgeDetectionFilter_RPi.slx** and **edgeDetectionSimplified_RPi.slx** models for any specific input frame rate.

   Hint: please remember to remove the Threshold block and connect the Constant block to the Threshold port of the **MaximumSlopeSimplified** Subsystem on the **edgeDetectionSimplified_RPi.slx** model for fair performance comparison.

Appendix 1: Background and References

Background:

The Raspberry Pi is a credit-card sized, low-cost, single-board computer with audio and video input/output, designed for teaching. The Raspberry Pi features a Broadcom® system-on-a-chip which includes an ARM processor 512 or 1GB MB RAM, and a VideoCore IV GPU. Raspberry Pi provides peripheral connectivity for stereo audio and digital video (1080p) and supports USB and Ethernet.

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 models to be automatically cross-compiled for execution on the Raspberry Pi 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 Raspberry Pi.

In addition, to Raspberry Pi Support from Simulink [4], starting in R2014a, there is also Raspberry Pi Support from MATLAB [3]. This provides a library of MATLAB functions which allow you to acquire data from sensors and imaging devices attached to a Raspberry Pi. However the MATLAB code runs on the host computer and not as a standalone application on the Raspberry Pi. In order to run a model or algorithm on the Raspberry Pi you would use the Raspberry Pi Support from Simulink and the approach detailed in this manual.

MATLAB and Simulink support for low-cost hardware such as LEGO MINDSTORMS robots, Arduino, and Raspberry Pi is a professionally supported feature developed by the MathWorks, [1, 5, 6]. However, other tools with similar aims have previously been available on the MathWorks File Exchange site.

References:

1. LEGO MINDSTORMS NXT Support from Simulink, http://www.mathworks.com/academia/lego-mindstorms-nxt-software/legomindstorms-simulink.html
2. MATLAB Support Package for Arduino (aka ArduinoIO Package), http://www.mathworks.com/matlabcentral/fileexchange/32374-matlab-support-package-for-arduino-aka-arduinoio-package
3. Raspberry Pi Support from MATLAB, http://www.mathworks.com/hardware-support/raspberry-pi-matlab.html
4. Raspberry Pi Support from Simulink, http://www.mathworks.com/hardware-support/raspberry-pi-simulink.html
5. LEGO MINDSTORMS EV3 Support from Simulink, http://www.mathworks.com/hardware-support/lego-mindstorms-ev3-simulink.html
6. Arduino Support from Simulink, http://www.mathworks.com/hardware-support/arduino-simulink.html

Appendix 2: System Requirements and Setup

System Requirements and Information:

- Operating system: Microsoft Windows or Mac OS X. All material tested under Windows 7 64-bit.
- **MATLAB and Simulink**, at Release R2013a or newer (Windows) or Release R2015a or newer (Mac OS X)
  - **MATLAB and Simulink Student Version** / **MATLAB and Simulink Student Suite** also supported
  - OPTIONAL: **Computer Vision System Toolbox** & **DSP System Toolbox** for barCodeRPi example
- If using 64-bit MATLAB it will be necessary to install a compatible C compiler, see below for details.
- Hardware Kit containing:
  - Raspberry Pi: Raspberry Pi 1 Model B, Raspberry Pi 1 Model B+, or Raspberry Pi 2 Model B
    - Note: Raspberry Pi Model A and Model A+ is NOT Supported
  - Memory card:
    - For Raspberry Pi 1 Model B: 4 GB SD Memory Card
    - For Rapsberry Pi 1 Model B+ or Rapsberry Pi 2 Model B: 4 GB Micro SD Memory Card
  - USB cable: USB A/MicroB – 6ft to power the Raspberry Pi
  - Ethernet Cable – 6 ft long
  - USB to Ethernet adapter 10/100 Mbit/s
    - Note: We have tested adapters that use ASIX AX88772 (10/100) and ASIX AX88178 (10/100/1000) chipsets.
  - USB webcam: We have tested with Logitech Webcam C270 and Logitech Webcam C210
  - SD / micro SD card reader to write the firmware onto the SD card
  - Props for workshop: green post-it notes, barcode print outs, etc
- A collection of related third party tools / products which are installed automatically by the **Support Package Installer** tool. A detailed list of these tools / products, their function, and licensing details is provided during the installation process.

Download, Installation, Connection and Testing

Step 1: Install and 'Select' a C Mex Compiler for MATLAB and Simulink

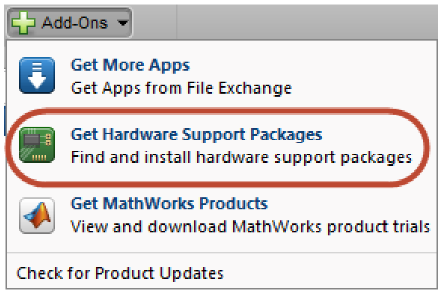
Simulink requires a C compiler in order to compile auto-generated C-code,that would be downloaded on the Raspberry Pi. This compiler must be installed and 'selected' prior to using Support Package for Raspberry Pi Hardware.

1. Check whether you are running the 32-bit (PCWIN) or 64-bit (PCWIN64, MAC64) version of MATLAB by typing **computer** at the MATLAB prompt.
2. 32-bit Versions of MATLAB ship with a C mex compiler, but the installed compiler must also be 'selected'. Type **mex –setup** at the MATLAB prompt and following the on-screen prompts to do this.
3. 64-bit versions of MATLAB do not ship with a C compiler. Download and install a supported compiler. If using the free Microsoft SDK, first install the .NET Framework 4.0, *then* install the Microsoft SDK. Once installed, the compiler must also be 'selected' as described in Step 2 above.

Step 2: Automated Download and Install of the Raspberry Pi Support from Simulink

Brief installation notes are provided below. More detailed installation instructions, including screen shots can be found in Help > Simulink > Target Hardware > Raspberry Pi > Install Support for Raspberry Pi Hardware

1. On the MATLAB Toolstrip click **Add-Ons** and select **Get Hardware Support Packages**

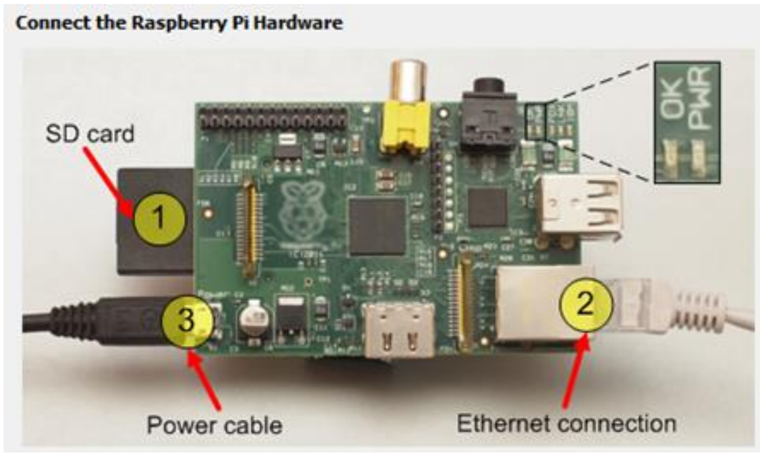

Add-Ons
Get More Apps
Get Apps from File Exchange
Get Hardware Support Packages
Find and install hardware support packages
Get MathWorks Products
View and download MathWorks product trials
Check for Product Updates

   a. Note: You can also type `targetinstaller` at the MATLAB prompt.
2. On Windows 7 platforms, you may be prompted to allow MATLAB to restart in **Administrator Mode** in order to permit installation of new software. Click **OK**.
3. On the Install or Update page of the target installer, select '**Internet (recommended)**', then click Next
4. On the Select Target page, select the **Simulink Raspberry Pi checkbox**, then click Next
5. The following page lists the third party software that will be installed. Click **Install**.
6. At the Installation Complete page (which appears after a few minutes), select **Close** to exit installation. [The screen also gives the option to continue to update the firmware on the Raspberry Pi, but this is not necessary if the Raspberry Pi already has updated firmware.]

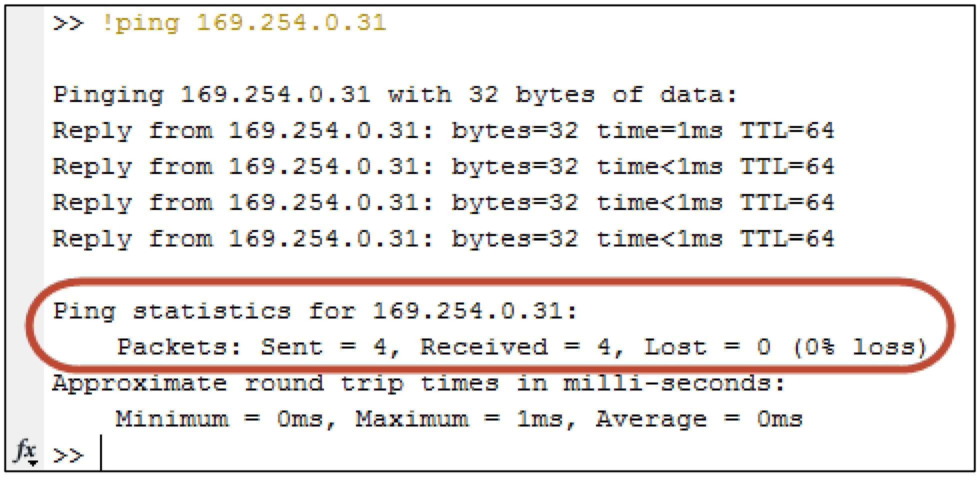
Step 3: Connect Raspberry Pi to Computer

Instructions on how to connect up the Raspberry Pi to your computer:

1. Insert the SD/ micro SD memory card into the Raspberry Pi
2. Connect Ethernet cable:
   a. Connect USB Ethernet dongle to computer (use USB port)
   b. Connect Ethernet cable to USB Ethernet dongle
   c. Connect Ethernet cable to Raspberry Pi
3. Connect USB to micro USB cable to computer (USB port) and to the Raspberry Pi (micro USB port).
   a. Wait a minute for Raspberry Pi to boot
4. Connect webcam to USB port on Raspberry Pi (not shown below)



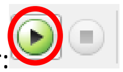
5. Test your connection between computer to Raspberry Pi)
    a. Type at the MATLAB command line: `!ping 169.254.0.31`
    b. You should see the result shown below:

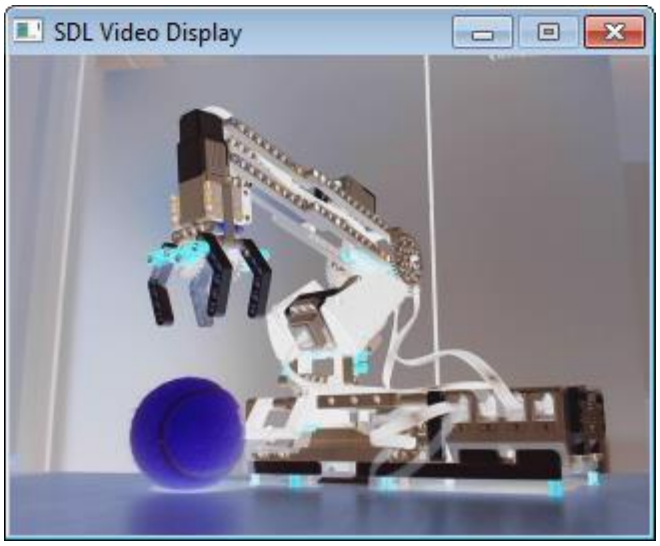


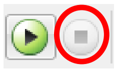
```
>> !ping 169.254.0.31

Pinging 169.254.0.31 with 32 bytes of data:
Reply from 169.254.0.31: bytes=32 time=1ms TTL=64
Reply from 169.254.0.31: bytes=32 time<1ms TTL=64
Reply from 169.254.0.31: bytes=32 time<1ms TTL=64
Reply from 169.254.0.31: bytes=32 time<1ms TTL=64

Ping statistics for 169.254.0.31:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss)
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
fx >>
```

6. Open and configure test model
    a. Open Image Inversion model typing at the MATLAB command line: `raspberrypi_inversion`
    b. Click Tools -> Run on Target Hardware -> Options
    c. Check to make sure you have Target hardware = Raspberry Pi. Also check the IP address. Click OK.
7. Run the test model
    a. Click Run on Simulink toolbar: 
    b. Watch the status bar for progress:
    c. Output should be an SDL Display with an inverted image:



    d. Click Stop on Simulink toolbar: 

You have now successfully setup and tested your Raspberry Pi with webcam.

Appendix 3: Working with Raspberry Pi

Basic Raspberry Pi Information:

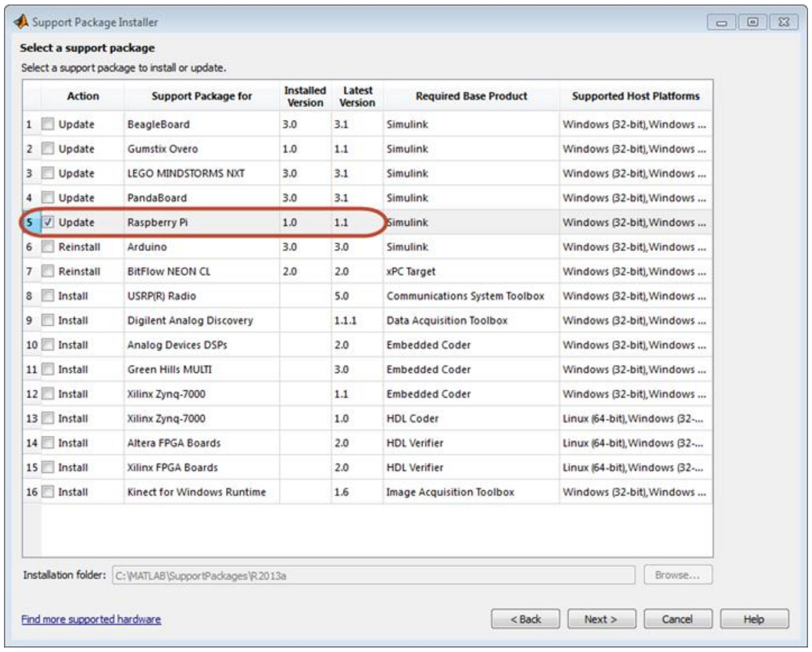
You can get basic information on the Raspberry Pi at http://www.raspberrypi.org/faqs

1. Ports: 2 to 4 USB ports, micro USB port for power, Ethernet port, SD card / micro SD socket, HDMI output port, and a set of GPIO pins (26 or 40).
2. Dimensions: ~85mm x 56mm x 21mm, with overlap for the SD / micro SD card and connectors. Weight is ~45g.
3. Power: 5V 700 mA via the micro USB port
4. Models: Raspberry Pi 1 Model A (retired), Raspberry Pi 1 Model B (retired), Raspberry Pi 1 Model A+, Raspberry PI 1 Model B+, Raspberry Pi 2 Model B

Note: **Simulink currently only supports Raspberry Pi 1 Model B & Model B+, and Raspberry Pi 2 Model B**

Troubleshooting Tips for Common Issues:

1. Are there different versions of the Support packages? How do I know what version of a Support package I have? How do I update my Support Package?
   a. Yes, in some cases there are updates to Support Packages
   b. To check the version of your support package is up to date, start the support package installer and check the version as shown below:



Select a support package

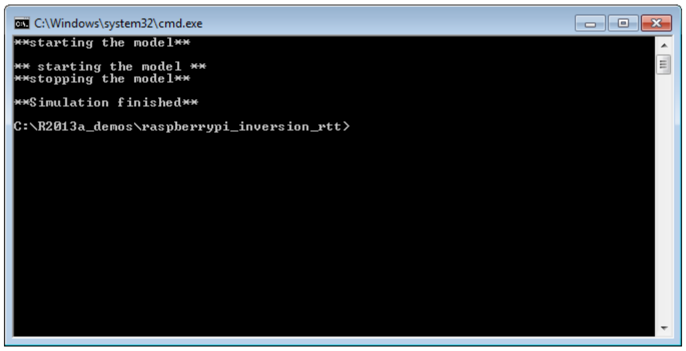
Select a support package to install or update.

	Action	Support Package for	Installed Version	Latest Version	Required Base Product	Supported Host Platforms
1	☐ Update	BeagleBoard	3.0	3.1	Simulink	Windows (32-bit),Windows ...
2	☐ Update	Gumstix Overo	1.0	1.1	Simulink	Windows (32-bit),Windows ...
3	☐ Update	LEGO MINDSTORMS NXT	3.0	3.1	Simulink	Windows (32-bit),Windows ...
4	☐ Update	PandaBoard	3.0	3.1	Simulink	Windows (32-bit),Windows ...
5	✓ Update	Raspberry Pi	1.0	1.1	Simulink	Windows (32-bit),Windows ...
6	☐ Reinstall	Arduino	3.0	3.0	Simulink	Windows (32-bit),Windows ...
7	☐ Reinstall	BitFlow NEON CL	2.0	2.0	xPC Target	Windows (32-bit),Windows ...
8	☐ Install	USRP(R) Radio		5.0	Communications System Toolbox	Windows (32-bit),Windows ...
9	☐ Install	Digilent Analog Discovery		1.1.1	Data Acquisition Toolbox	Windows (32-bit),Windows ...
10	☐ Install	Analog Devices DSPs		2.0	Embedded Coder	Windows (32-bit),Windows ...
11	☐ Install	Green Hills MULTI		3.0	Embedded Coder	Windows (32-bit),Windows ...
12	☐ Install	Xilinx Zynq-7000		1.1	Embedded Coder	Windows (32-bit),Windows ...
13	☐ Install	Xilinx Zynq-7000		1.0	HDL Coder	Linux (64-bit),Windows (32-...
14	☐ Install	Altera FPGA Boards		2.0	HDL Verifier	Linux (64-bit),Windows (32-...
15	☐ Install	Xilinx FPGA Boards		2.0	HDL Verifier	Linux (64-bit),Windows (32-...
16	☐ Install	Kinect for Windows Runtime		1.6	Image Acquisition Toolbox	Windows (32-bit),Windows ...

Installation folder: C:\MATLAB\SupportPackages\R2013a

   c. You can click Update check box and Next to update the support package.

2. What do I do with the cmd.exe windows?



C:\Windows\system32\cmd.exe

```
**starting the model**

** starting the model **
**stopping the model**

**Simulation finished**

C:\R2013a_demos\raspberrypi_inversion_rtt>
```

   a. Close them by click on the red X in upper right once they say `**Simulation finished**`

3. I get a "TCP 17725" error
   a. Usually this is because you are trying to download a model to the Rapsberry Pi that is already running a model (possibly from a prior "Deploy to Hardware" step).
   b. To workaround, try to:
      i. At the MATLAB prompt assign **h = raspberrypi**. This will create a Raspberry Pi object in MATLAB which can then communicate with the unit.
      ii. Type **h.connect** at the MATLAB prompt which connects the object to the Raspberry Pi.
      iii. **h.stop('BlinkLED1')** stops the program running (Use the name of the model that is running
   c. If this doesn't work, or you do not know the name of the model that is running, power cycle the Raspberry Pi by unplugging the Micro-USB connector, waiting 5 seconds, and plugging it back in. It will take the Raspberry Pi about 45 seconds to fully reboot.
   d. Try to ping to verify your connection, and then run your model again.

4. I get intermittent errors where the Raspberry Pi, webcam, or circuits stop working.
   a. This is likely due to being unable to source enough current/voltage to drive everything from the host computer USB port, especially after the voltage drop from the USB-Micro-USB cable.
   b. Workaround: You will need to purchase a USB power adapter (minimum 5V, 1A). These are commonly available, and often used for Android devices.

Appendix 4: Raspberry Pi Driving Hardware using GPIO Pins

See document "Raspberry_Pi_Workshop_Manual_R201**_Appendix_4" for an advanced project on interfacing with the Raspberry PI GPIO pins, calling C Functions from Simulink, and calling C Function Libaries from Simulink (WiringPi).

Answers to Questions in Manual:

Q: P2.1: What happens when you change the constant 255 to other values?
A: P2.1: You see the colours changing as the numbers representing the bitmap change. Sometimes you see massive changes in colour which is because the numbers are **uint8** and you get wrap around in the arithmetic.

Q: P2.2.1: Step 3: Note that all of the computations are to be performed in **uint8.**

a. How would you ensure that the output of the blocks are of the correct type at the end of each block computation? Hint: **Display > Signals & Ports > Port Data Types**
b. Why should the value of Gain be 255 to display a binary image?
c. How would you ensure that the **Add** block does not overflow?

A: P2.2.1: Step 3:

a. In the signal attribute tab of each block examine the Output data type. Where appropriate you can change it to uint8 or select one of the "inherit" options. On input side, if appropriate, you can select the check box require all inputs to have same data type. Simulink allows many degrees of freedom in how the computations are performed.
b. 255
c. You can select Saturate on integer overflow tick box in Signal Attributes tab

Q: P3.5:

1. How do the results of step 8 above using the simplified algorithm compare with the reference model?
2. How would you measure the difference? What you could do to minimize it?
3. Would it be possible to simplify further the edge detection algorithm?

A: P3.5:

1. Generally speaking, the results are different because of the different edge detection implementations.
2. The difference may be computed as the mean value of the absolute difference (mean error) of the two processed images. To minimize the difference, one could run multiple simulations and identify the minimum mean error by storing its value against varying threshold values of the two algorithms.
3. The above formula simplifies the computation of the gradient vector magnitude by approximation, hence reducing the computational load.

Q: P3.6:

a. How can you reduce the video input frame rate for your **edgeDetectionFilter_RPi.slx** model?
b. What you think it would be the maximum video input frame rate avoiding overruns? How would you verify that?

A: P3.6:

a. Double click V4L2 Video Capture block. The sample time parameter from 1/fps sets the input frame rate (fps = 12 in Model Properties -> Callbacks -> PreloadFcn). Change 1/fps to something like 1/5.
b. The maximum video input frame rate may be experimentally determined by running multiple simulations with different video input frame rates and the 'Enable overrun detection' flag checked.