VU-LEGO Real Time Target: User's Guide

Version 1.02 by James Peyton Jones, Connor McArthur, Tyler Young, Michael Ciavarella
Center for Nonlinear Dynamics & Control, Villanova University, Villanova PA 19085, USA

Contents

VU-LEGO Real Time Target 2
1. Overview 2
2. Legal Disclaimers and Licenses 2
3. System Requirements 3
4. Download and Installation 4
4.1 Step 0: Install and 'Select' a C Mex Compiler for Matlab and Simulink 4
4.2 Step 1: Download the VU-LEGO Real Time target distribution (VU-LRT) 4
4.3 Step 2: Automated download and installation of 3rd party tools on the Host PC 4
4.4 Step 3: Automated download and installation of firmware on the NXT brick 5
4.5 Step 4: Renaming the NXT brick (optional) 5
5. Getting Started 6
5.1 Checking the installation works: A first example 6
5.2 Constructing a Complete Example 7
5.3 A Test Exercise: Sound frequency and volume modulation 10
5.4 Using USB communication: a first example 10
5.5 Reconstructing the USB example 11
5.6 Using Bluetooth communications 13
5.7 A Real Time Control Example: Motor Speed Control 14
5.8 Project1: A simple line-following robot with proportional feedback 15
5.9 Project2: A simple line-following robot with state machine relay feedback 16
5.10 Other Examples 17
6. Ongoing work 18
References 18
Appendix: Release Notes 19

VU-LEGO Real Time Target

1. Overview

The Villanova University LEGO Real Time Target(VU-LRT) enables rapid implementation of Simulink-based designs on the LEGO Mindstorms NXT platform. The target was developed at Villanova University with support from the National Science Foundation (grant # DUE No. 0837637), the MathWorksInc, and Villanova University's Center for Nonlinear Dynamics & Control (CENDAC) within the College of Engineering. It is similar in aim to the Embedded Coder Robot (ECRobot) package by Takashi Chikamasa [1], but is more consistent with the Real Time Workshop development pathway, and is less constrained by ECRobot's function call / data store architecture. Further details may be found in reference [2].

Please contact the authors directly at james.peyton-jones@villanova.eduto resolve any technical issues that may arise. We would also be interested in hearing from other university professors who might be interested in collaborating on a Phase II National Science Foundation proposal which uses the VU-LRT for innovative undergraduate education.

2. Legal Disclaimers and Licenses

- LEGO® is a trademark of the LEGO Group of companies which do not sponsor, authorize orendorse this project. LEGO® and Mindstorms® are registered trademarks of The LEGO Group.
- According to LEGO Mindstorms NXT Hardware Developer Kit, “When the NXT isdisassembled or when third party firmware is used with the NXT, all warranties are renderedinvalid”. The authors do not take any responsibility for any loss or damage of any kind incurredas a result of the use or the download of the VU-LRT toolbox and related third party tools.
- The terms of use of the MathWorksMATLAB Central website where the VU-LRT toolbox is posted may be found here: http://www.mathworks.com/matlabcentral/disclaimer.html
- The VU-LEGO Real Time target is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
- The VU-LEGO Real Time target is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public Licensefor more details.
- Use of the VU-LEGO Real Time target requires installation of various related third party tools as listed in Section 3. The majority of these are open source projects, distributed under the GNU General Public License, the Mozilla Public License or the TOPPERS License. An exception is the Official Mindstorms NXT Fantom Driver. Installation of these third party tools can be largely automated though the use of the installer program included in the VU-LRT distribution. However, it remains the user’s responsibility to inspect, agree and abide by the license agreements associated with each of these tools. Further details may be found in the Section 3.

3. System Requirements

- Operating system: Windows (tested under Windows XP and Windows 7)
- MATLAB, Simulink and the Real Time Workshop (tested under MATLAB R2010a). It is strongly recommended to use the **32-bit version** of Matlab since a) this is shipped with a mex file compiler and b) the MindStormsFantom Driver is 32-bit, so real-time USB communication is only possible from 32-bit Matlab.
- Bluetooth: Bluetooth 2.0 adapter (tested with D-Link DBT-120 C wireless adaptor)
- LEGO® MindstormsEducation NXT Base Set; Additional construction parts: Education Resource Set
- A collection of related third party tools / products which are installed automatically by the installer program included in the VU-LRT distribution (see section 4 below).A detailed list of these tools / products, their function, and licensing details is as follows:

1. Cygwin – A Windows-based Linux environment. (Required for GNU ARM compiler)
   Distributed primarily under the GNU General Public License see: http://cygwin.com/licensing.html
2. GNU ARM Compiler – Compiles C code for the ARM7 processor inside the LEGO NXT brick
   Distributed primarily under the GNU General Public License and/or the GNU Lesser General Public License.Newlib is covered by several licenses. see: http://www.gnuarm.com/
3. Fantom Driver– A driver for communicating with the LEGO NXT over a USB cable (used by NeXTTool)
   Distributed under the LEGO® MINDSTORMS® NXT Software End User License Agreement
4. NeXTTool – For uploading nxtOSEK application programs from the PC to the LEGO NXT over a USB cable
   Licensing appears to be unspecified.
5. nxtOSEK – A real time operating system for the LEGO NXT brick
   Distributed under the Mozilla Public License and the TOPPERS License see: http://lejos-osek.sourceforge.net/faq.htm
6. sg.exe– An OSEK OIL parser and code generator needed bynxtOSEK
   Distributed under the TOPPERS License
7. 7-Zip – A utility for unzipping the sg.exe download
   Distributed under the GNU Limited General Public Licensesee http://www.7-zip.org/faq.html
8. NXT Enhanced standard firmware – A replacement firmware for the LEGO NXT that allows it to execute ARM binaries in addition to standard NXT programs. Licensing appears to be unspecified.
9. PC-side USB and Bluetooth communication utilities in directories **`VU-LRT\@nxtusb`** and **`VU-LRT\@w32serial`** are derived (with minor modifications) from the Embedded Coder Robot (ECRobot) package, and redistributed as part of VU-LRT. This code is covered by the MathWorks Limited License

4. Download and Installation

4.1 Step 0: Install and'Select' a C Mex Compiler for Matlab and Simulink

Simulink requires a C compiler in order to compile Matlab Executable (Mex) files. This compiler must be installed and 'selected' prior to installing VU-LRT.

1. Check whether you are running the 32-bit (PCWIN) or 64-bit (PCWIN64) 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 mex compiler. Download and install a supported compiler. There are free versions available but installation can be tricky. Once installation is complete, 'select' the compiler by typing **mex -setup** at the Matlab prompt and following the on-screen prompts.

4.2 Step 1: Download the VU-LEGO Real Time target distribution (VU-LRT)

You have probably already completed this step, since this document is itself part of the distribution.

1. Download the VU-LEGO Real Time Target distribution from http://www.mathworks.com/matlabcentral/fileexchange/
2. Extract the zipped `VU-LRT` directory to any location on your hard drive **providing the pathname contains no spaces**. A good choice, consistent with installation defaults is **`C:\RTtools\VU-LRT.`**

4.3 Step 2: Automated downloadand installation of 3rd party tools on the Host PC

Like its predecessor,ECRobot, the VU-LRT target is built on top of thenxtOSEKECRobot package which in turn requires a series of other public domain tools and utilities. A list of these tools and utilities is provided in Section 3. However, the entire toolchain can be installed automatically using the **`Installer`** tool which is part of the VU-LRT target distribution.

1. Start MATLAB, and within MATLAB, navigate to the `VU-LRT`directory. [On Windows 7 platforms, start MATLAB in Administrator mode: Right-click on the MATLAB icon, and select "Run as Administrator"].
2. Typing **`installer`** on its own brings up the menu of options shown in Fig. 1. Alternatively, the user can type **`installer(x)`** where x represents the desired option as listed in Fig.1, and the corresponding action is then executed directly. For example, typing **`installer(1)`** performs a complete download and installation on the host PCin the default toolbox root directory`C:\RTtools.`
3. Reboot the PC to ensure the FantomUSB Driver is picked up properly by the operating system.

Notes / Troubleshooting:

- The **`Compile Mex Files`** menu option (Fig.1) will issue a warning if it cannot find a C compiler with which to compile the mex files associated with the VU-LRT Simulink blocks. Check that installation step 0 was properly performed, and then Select installer menu option #4, or type **`installer(4)`** to compile the required mex files.

- On Windows 7 machines, the mex compiler can be correctly installed but fail to find a file called **`simstruc.h`**. This is a mex compiler issue, unrelated to VU-LRT. Further details, and a workaround can be found here: http://www.mathworks.com/support/bugreports/661855.
- The **`Check Installed tools`** menu option (Fig.1) checks only those tools that are installed below the **`rootInstallDir`**. If previous versions of these tools have been installed elsewhere they will not be detected. It is advisable to remove / delete any old versions of the tool chain prior to installing VU-LRT
- The toolbox root file locations for the program downloads and installation directories can be specified using the extended syntax **`installer(x,rootInstallDir,downloadDir)`**. Further customization is possible by editing the initialsection of the installer.m file, but note that the **`rootInstallDir`** must **`not containanyspaces`**anywhere in the full filepath.
- If ATMEL SAMBA (a tool for communicating with the LEGO NXT) was previously installed on your system, it needs to be uninstalled before running `installer`

Options:
1. Download, Install and Compile Mex files
2. Download tools (only)
3. Install tools (only)
4. Compile Mex files (only)
5. Check Installed tools
6. Download NXT firmware to NXT brick
7. Rename NXT brick
8. Exit

Enter choice:

Fig. 1: VU-LRT Automated Installer menu options

4.4 Step 3: Automated download and installation of firmware on the NXT brick

1. Within MATLAB, type: **`installer(6)`** or select option 6 from the installer menu and follow the on-screen instructions.
2. Cycle the NXT brick power off and on again (if necessary by removing its battery) to reboot the NXT.

4.5 Step 4: Renaming the NXT brick (optional)

The name of the NXT brick is displayed at the top of the NXT's LCD screenwhenever it is powered up, and the same name is generally associated with the BlueTooth COM port when using BlueTooth communications. By default, this name is 'NXT'. When dealing with a classroom full of NXT bricks it is convenient to rename the bricks in order to be able to distinguish between them. The procedure is as follows:

1. Within MATLAB, type: **`installer(7)`** or select option 7 from the installer menu and follow the on-screen instructions.

5. Getting Started

5.1 Checking the installation works: A first example

1. Copy the **VU-LRT\Samples\VU_TestSoundTone1.mdl**model into a working directory, and within MATLAB navigate to that directory.
2. Open**VU_TestSoundTone1.mdl**(Fig. 2) by double-clicking on the file in the *Current Directory* browser
3. Check the NXT brick is powered on and connected by USB to the host PC.
4. In the **VU_TestSoundTone1.mdl**window, select the **Tools/Real-TimeWorkshop/BuildModel** menu item or use the **ctrl+B** shortcut to initiate the build procedure.The compilation process is reported in the MATLAB Command window (see background to Fig.2), and should finish with '**### Finished**'.
5. The NXT brick can now be disconnected (if desired). Using the NXT button/menu interface, navigate to **MyFiles/VU_testSoundTone1** and select **Run**. This invokes the OSEK operating system, as seen by the nxtOSEK splash screen (Fig 3), after which the nxtOSEK main screen appears (Fig 4).
6. As indicated by the nxtOSEK main screen (Fig 4), use the right button to start / **Run**the model executable. The left button will **Stop** execution and return to the nxtOSEK main screen.
7. During execution of the model, the tone is sounded whenever the '**Enter Button**' is pressed (Fig. 2).
8. To terminate both nxtOSEK and any programs that may be running on the NXT press the middle (grey) '**Exit**' button. The NXT will automatically power off. On power-on, the NXT brick will revert to the normal LEGO operating system and GUI.

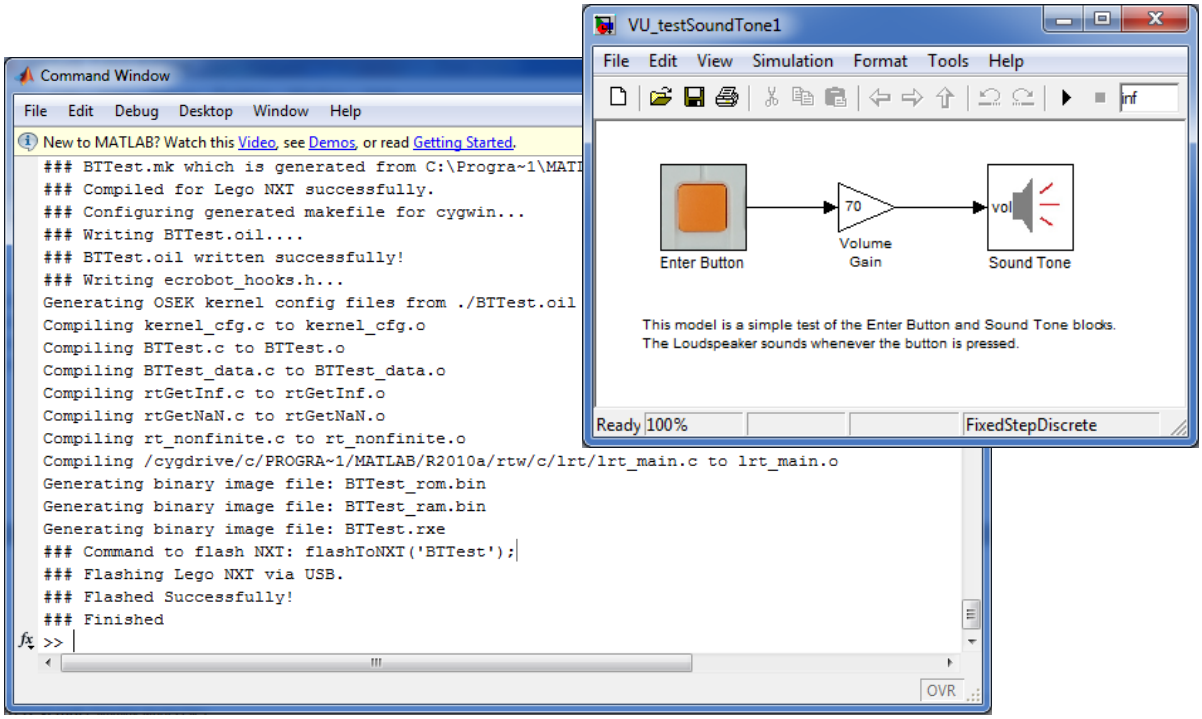


Figure 2: The VU_testSoundTone1 example system, and a screen capture of part of a successful build process

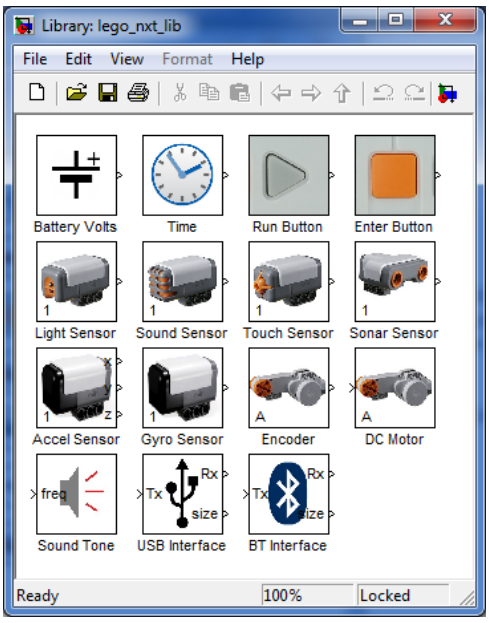
Figure 3: The nxtOSEK splash screen

Figure 4: The main nxtOSEK screen

5.2 Constructing a Complete Example

The aim of this example is to reconstruct the **VU_testSoundTone1** model from scratch in order to learn the design procedure.

1. Within MATLAB, either type **simulink** or click on the Simulink icon to invoke the Simulink Library browser.
2. Select the **File/New/Model** option from the Simulink Library browser menu. **File/Save** the new 'untitled' model as **my_testSoundTone1.mdl**in a working directory.
3. The VU-LRT Blockset, within the Simulink Library browser is shown in Fig.5. Drag and drop the '**Enter Button**' and '**Sound Tone**' blocks into the **my_testSoundTone1.mdl**window. Also drag and drop the '**Gain**' block from the 'Commonly Used Blocks' or 'Math Operations' libraries and double click on the block to set its gain value to 70 (or whatever volume level you choose).



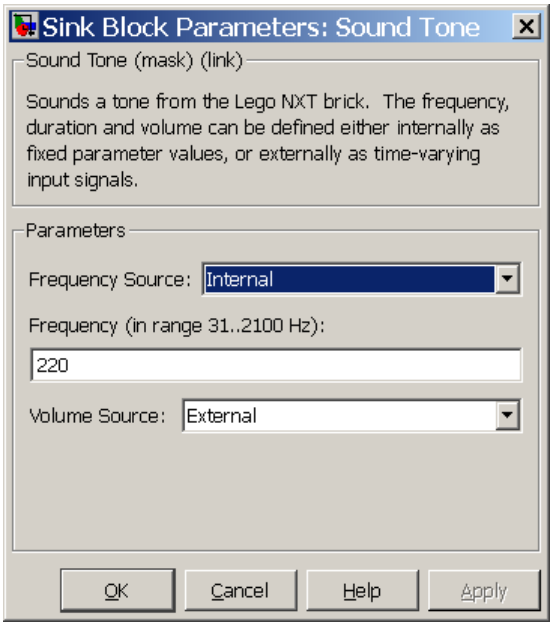


Fig. 5: The VU-LRT Blockset Fig. 6: The Sound Tone block configuration dialog

4. Construct the system shown in Fig. 2. The '**Sound Tone**' block should be configured as shown in Fig. 6, so that the frequency is defined *internally* and set at 220 Hz, while the volume is driven *externally*(ultimately by the button press).
5. The input and output ports of VU-LRT blocks expect or return data of the type specified in the block documentation. This type sensitivity reflects the requirements of the underlying hardware. A common error in VU-LRT applications is caused by type mismatch issues.
6. Select the **Edit/UpdateDiagram** menu option (or **ctrl+D** shortcut) to highlight signal types and potential mismatch issues. It is also good practice to work with the **Format/PortSignalDisplays/PortDataTypes** option turned on.
7. In this example, there is a type mismatch between the output of the Gain block and the Volume input of the Sound Tone block. In many cases, type mismatches can be resolved by defining the output type of a Simulink block using the block's **Signal Attributes** tab within the block dialog. In this case, for example, set the **Gainblock/SignalAttributestab/outputtype** to be **uint8**. Type mismatches can also be resolved using an explicit **Simulink/SignalAttributes/DataTypeConversion** block.
8. Having resolved these issues, the model must be configured for real-time implementation on the NXT brick. Select the **Simulation/ConfigurationParameters** option (or **ctrl+E** short cut), and configure the **Solver** pane as shown in Fig. 7. In particular...
   - Set the **StopTime** to *inf* (so the model will execute continuously until physically stopped)
   - Set the **SolverType**to *Fixed-step*, and the **Solver**to *Discrete*
   - Set the **Fixed-stepsize** (the fundamental sample time) as appropriate for the application. In this case a sample period of *0.1 sec* is acceptable.
   - Set the **TaskingMode** to singleTasking.
9. Still within the Configuration Parameters dialog, configure the **Real-timeWorkshop** pane as shown in Fig.8. In particular...
   - Click the Browse button and select **lrt**, the VU-LEGO Real Time target as the **SystemTargetFile**. The other options should be set automatically and appear as shown in Fig 8.
10. Still within the Configuration Parameters dialog, configure the **Real-TimeWorkshop /LRTCodeGeneration** pane as shown in Fig. 9. In particular...
    - Select whether the model should be compiled for *integer* or floating point arithmetic. Fixed point integer arithmetic is faster and more compact, but can be less accurate.
    - Select whether the build process should automatically *download* the resulting executable to the NXT
    - Select whether the build process should only recompile those files which have changed since the last compilation, or re-build all files regardless of their status. This option is normally *unchecked*.
11. To complete the configuration, click **Apply** to apply the changes and **ok** to dismiss the dialog.
12. Build the model (**ctrl+B**) as before, - ensuring the NXT is powered-on and connected so that the compiled code can be downloaded automatically to the NXT target.
13. On the NXT, run the **my_testSoundTone1** model and verify its correct operation as before.

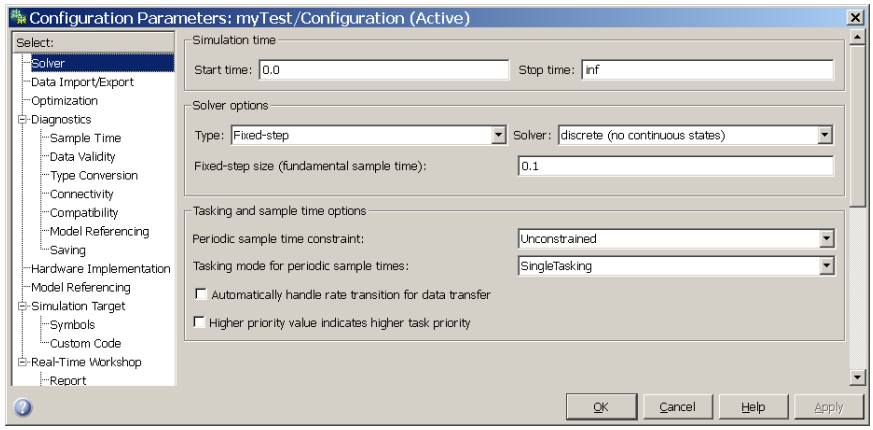


Fig. 7: Configuration parameters, Solver pane

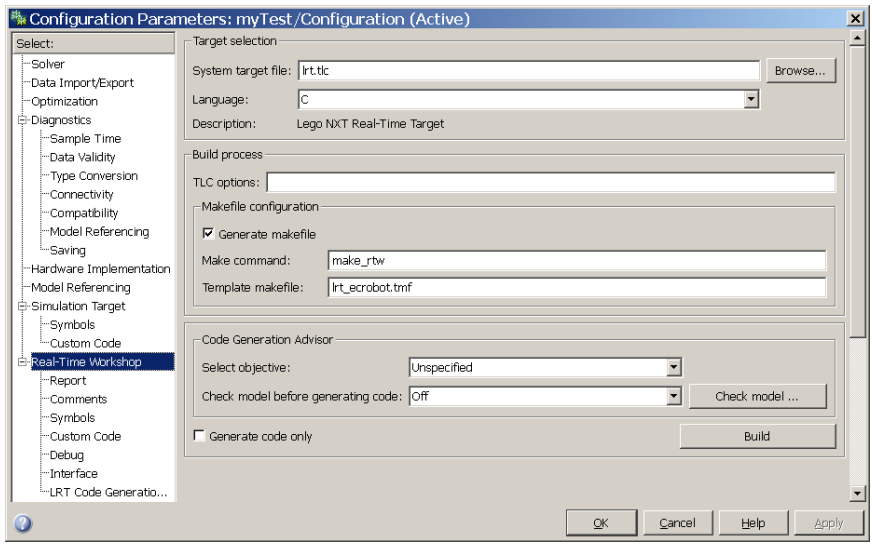


Fig. 8: Configuration Parameters, Real-Time Workshop pane

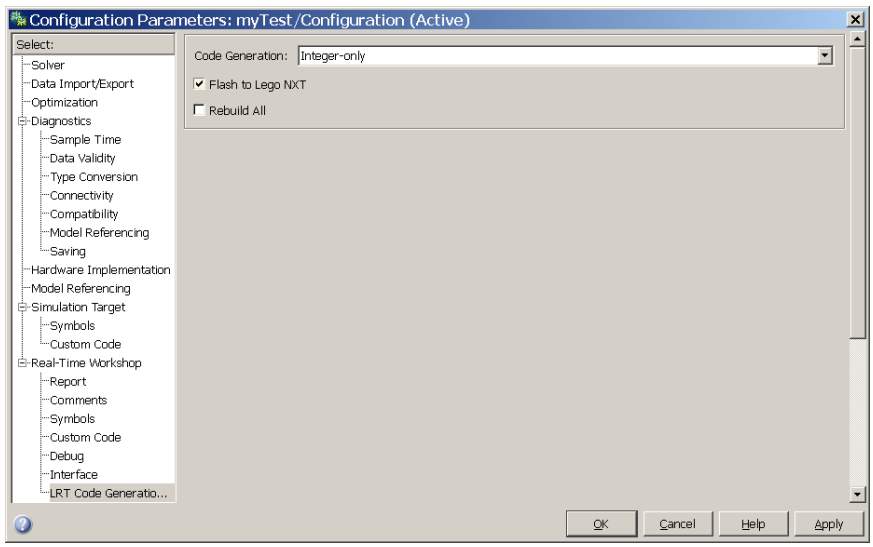


Fig. 9: Configuration Parameters, VU-LRT Code Generation pane

5.3 A Test Exercise: Sound frequency and volume modulation

Apply the same procedure to construct, build and test the model shown in Fig. 10. This is similar to the previous example, but the Sound Tone frequency is modulated about a center frequency of 220 Hz by the encoder input attached to Port B, and the Sound Tone volume is modulated about a center volume of 50 by the encoder input attached to Port C. If needed, a working example of this system can be found in the VU-LRT\Samples directory.

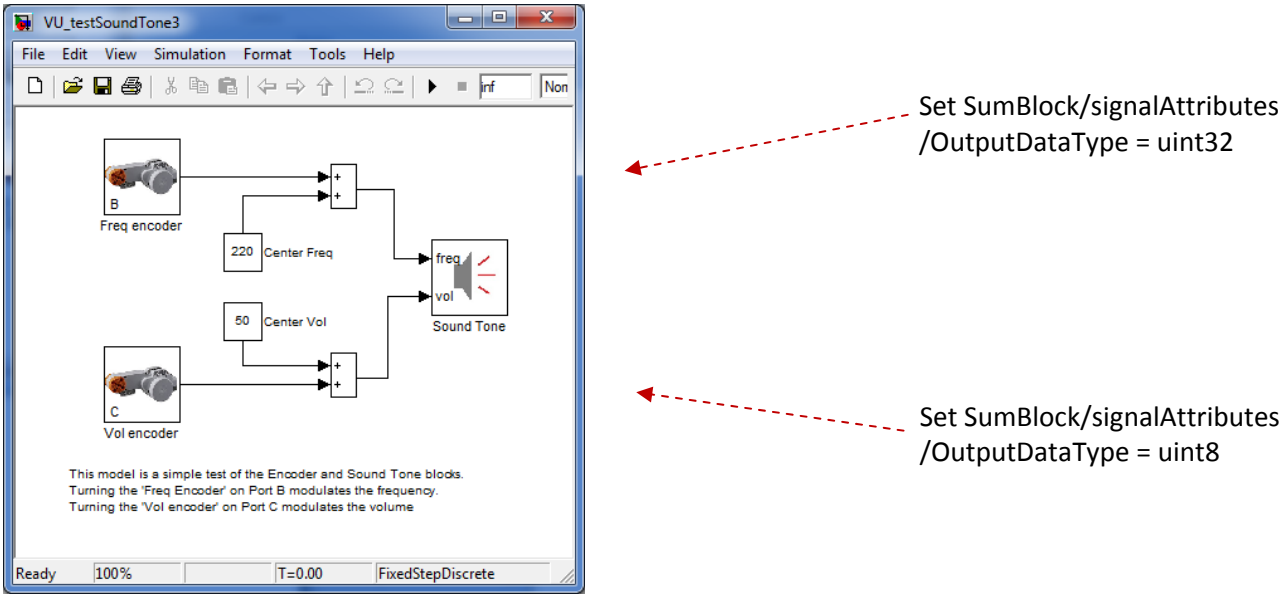


Fig. 10: The VU_testSoundTone3 example system

5.4 Using USB communication: a first example

The USB and BlueTooth Transmit (Tx) and Receive (Rx) blocks can be used to transmit data to/from another system. Typically the 'other' system is the host PC, but it can be any system capable of transmitting or receiving data over USB / BlueTooth, and this includes GameBoy controllers, cell phones and (for blueTooth) even other NXT bricks. The code running on the NXT is designed using VU-LRT blocks. The code on the other / host system must be written separately. Utilities for host-side communication are provided in the VU-LRT distribution (courtesy of the ECRobot project with some modification), and simple examples of their use are given below:

1. Copy the **`VU-LRT\Samples\VU_TestUsbTxRx.mdl`**and **`VU_TestUsbRxTx.m`** files into a working directory, and within MATLAB navigate to that directory.
2. Open **`VU_TestUsbTxRx.mdl`**(Fig.11). The model is identical to the Sound Frequency and Modulation test exercise of section 5.3 and Fig.10, only instead of connecting the final frequency and volume signals directly to the Sound Tone block, the data is piped via the host PC and displayed to the user before being echoed back to the NXT and the Sound Tone block. The 'Pack' and 'Unpack' embedded Matlab functions serve to translate between the data-types used by the model, and the **uint8** data stream of bytes that is transmitted over USB.
3. **Build** (**`ctrl-B`**) and download the model to the NXT.

4. Start / **Run** the model on the NXT. The model has to be running ***before*** the host PC attempts to connect to the NXT target (there has to be a program running to connect to!).
5. Within MATLAB, open the **`VU_TestUsbRxTx.m`**file and observe how the program is designed to establish connectivity, read and display data from the NXT, echo the data back to the NXT, and finally terminate.
6. Assuming the model is already executing on the NXT, start the host/PC side of the communications by typing **`VU_TestUsbRxTx`**at the MATLAB prompt. At this point the NXT loudspeaker should sound, and it should be possible to adjust the frequency and volume, as before, by turning the wheel encoder inputs. The values of the frequency and volume signals should also be displayed in the MATLAB command window.
7. **Exit** the program on the NXT, and the host PC program will also terminate automatically.

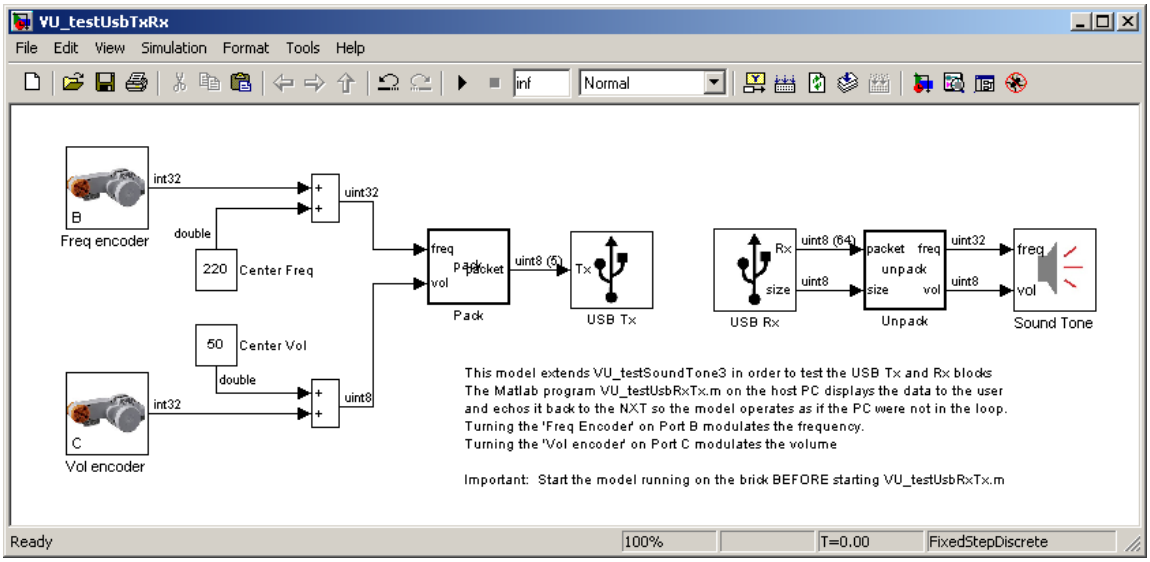


Fig. 11: The VU_testUsbTxRx example system

5.5 Reconstructing the USB example

The aim of this section is to reconstruct the **`VU_testUsbTxRx.mdl`** model in order to learn some points of detail.

1. Copy the **`VU_testSoundTone3.mdl`** model into a working directory, and within MATLAB navigate to that directory.
2. Open the **VU_testSoundTone3.mdl** model and **File/SaveAsmy_testUsbTxRx.mdl**.
3. Delete the signal lines connecting the summation block outputs to the SoundTone block inputs, and in their place add:
   - Two **embedded Matlab** blocks (found under User Defined Functions in the Simulink Library browser)
   - A **USB Tx**block, and a **USB Rx** block (found under the VU-LRT Blockset).
4. Click on the name beneath each embedded MATLAB block and rename the first one ‘Pack’ and the second ‘Unpack’.
5. Double-click the **Pack** block, and edit the code as shown below in order to convert the freq and vol inputs into a **uint8** byte stream for transmission. Note:

- The embedded MATLAB compiler needs to know how much memory to allocate for the packet. The use of the eml.nullcopy function defines this, but does not otherwise generate any code.
- the**typecast** function (unlike the **cast** function) does not alter the data in any way, but simply re-interprets the stored bit pattern as a sequence of (in this case) uint8 bytes.
- The maximum number of bytes in a USB transmission is 64 bytes.

```
function packet = pack(freq,vol)
%First define the size of the packet:
%uint32(freq)=4Bytes + uint8(vol)=1Byte = 5Bytes Total
%Use nullcopy to avoid generating unnecessary code
packet= eml.nullcopy(zeros(5,1,'uint8'));
%Now construct the packet
packet= [typecast(freq,'uint8') typecast(vol,'uint8')]';
```

6. Double-click the **Unpack** block, and edit the code as shown below in order to convert the received **uint8** byte stream into meaningful freq and vol data. Note:
   - The receive buffer is always 64 bytes long. The **Size** output of the **USB Rx** block indicates how many bytes of this buffer contain valid data.
   - the**typecast** function re-interprets the received uint8 packet as a uint32 freq, and a uint8 vol.

```
function [freq,vol] = unpack(packet,size)
%Simply deconstruct the packet
freq = typecast(packet(1:4),'uint32');
vol = uint8(packet(5));
```

7. It is currently necessary to define the **Unpack** block input data types for the packet and size variables manually.Within the Embedded MATLAB Editor for the **Unpack** block, select **Tools/ModelExplorer**. In the Model Explorer window click on the **packet**variable, and within the right-hand pane set the corresponding data type to be **uint8**. Do the same for the **size** variable.
8. Connect the new blocks as shown in Fig. 11.
9. Double-click the **USB Rx** block, and set the sample time to 0.1 sec (ie. the same as the rest of the model). The dialog also allows the user to set the initial value of the packet to be used until fresh data is received from the host. In this example, the packet is set to zeros which means the loudspeaker will not sound until new values are received from the host.
10. Save the model. Then build, download, and test the model as described in Section 5.4

5.6 Using Bluetooth communications

The Bluetooth Bt Rx and BtTx blocks are very similar to their USB counterparts. The aim of this section is to adapt the previous example so that it works wirelessly over Bluetooth rather than over a wired USB connection.

a) Designing and Building the Model

1. Replace the **USB Rx** and **USB Tx** blocks in the previous example with **BT Rx** and **BT Tx**blocks.Ensure that the **Master/Slave**property in both BT blocks is set to **Slave** since in this application the PC host is acting as the Master. Also set the sample time of the **BT Rx**block to **0.1** sec (ie. the same as the rest of the model).
2. Save the modified model as **VU_TestBtTxRx.mdl.** A completed model may also be found in **VU-LRT\Samples\VU_TestBtTxRx.mdl.**
3. **Build** (**ctrl-B**) and download the model to the NXT.

b) Configuring Bluetooth at the System level

The NXT brick must first be 'paired' with the PC Bluetooth host. This is a one off operation that occurs at the Windows operating system level. Once configured, the two devices will pair automatically whenever within range of each other, so these steps should only need performing once.

4. Using the NXT button/menu interface, navigate to **Bluetooth/On|off** and turn Bluetooth **On**. A small ***<** icon will appear in the top left corner of the screen to indicate that Bluetooth is enabled.
5. On Windows7 platforms, from the **Start Menu**, open the system **ControlPanel** and select **HardwareAndSound / ViewDevicesAndPrinters**. A window opens as shown in Fig.12a).
6. Check the NXT is powered on, and then Click **Add a Device** in the menu bar of this window. The NXT should appear in the list of available devices to add. Select this device and click **next**.
7. On the NXT's screen the default **Passkey** of **1234** is displayed. Press the orange **Enter** button to accept.
8. Enter the same **1234** pairing code when requested on the Windows system. The pairing then complete, and the NXT appears in the list of devices as shown in Fig. 12b).
9. **Double click** on the newly added NXT device to examine its properties, select the **hardware** tab, and note the **COM port number** that has been associated with the device. [For NXT to NXT communications (untested as yet) you will probably need the Unique Identifier found on the Bluetooth tab].
10. The NXT and/or PC may now be powered off, or re-programmed as many times as you like, but whenever the two are in range of each other, pairing should happen automatically and communication is possible through the assigned **COM port number** defined in step 10.

c) Running the example application

11. Start +**Run** the model on the NXT. The model has to be running***before*** the host PC attempts to open the BT COM port for communicating with the NXT (there has to be a program running to connect to!).
12. The Matlab program for handling the host/PC end of the communication may be copied from **VU-LRT\Samples\VU_TestBtRxTx.m** into a working directory. Open the file, and observe the close similarity with the corresponding USB program **VU_TestUsbRxTx.m.** The primary difference is that Bluetooth transmissions always include a 2-byte header representing the number of bytes transmitted. This header must be handled by the user program.

8. Assuming the model is already executing on the NXT, start the host/PC side of the communications by typing **`VU_TestBtRxTx('COMxx')`** at the MATLAB prompt, where COMxx represents the COM port number identified in step 10 above.The NXT loudspeaker should now sound, and it should be possible to adjust the frequency and volume, as before, by turning the wheel encoder inputs.The values of the frequency and volume signals should also be displayed in the Matlab command window.
9. **Exit** the program on the NXT, and the host PC program will also terminate automatically.

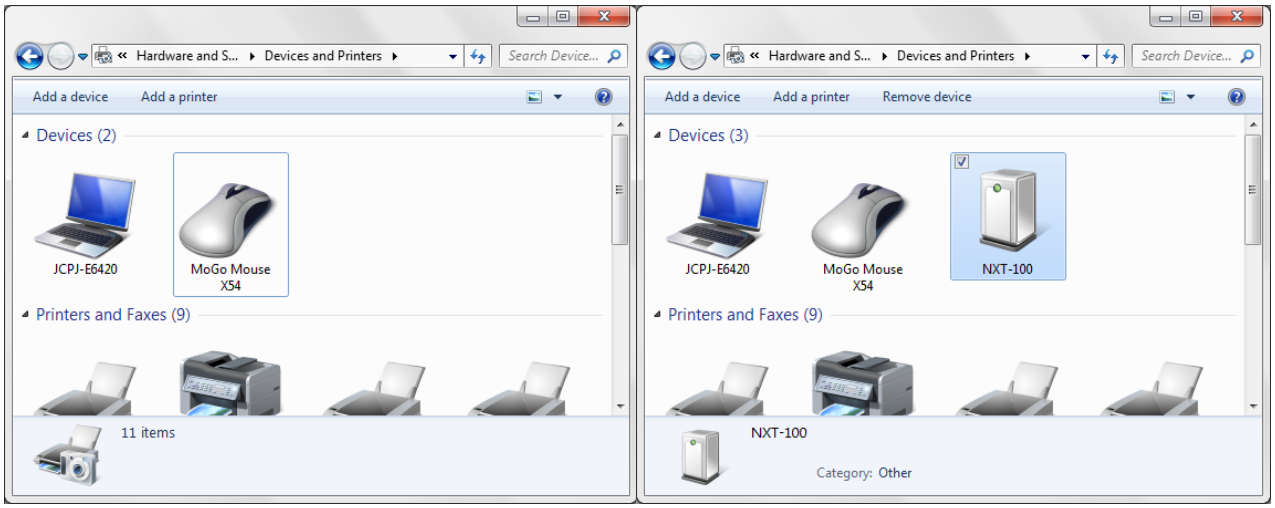


Fig. 12 a) Initial 'Devices and Printers' window

Fig 12 b)'Devices and Printers' window after pairing

5.7 A Real Time Control Example: Motor Speed Control

One of the advantages of embedded implementation is for the implementation of real-time controller designs. A common example is in motor speed control. Simple applications sometimes assume that motor speed is simply proportional to the applied voltage, but motor speed also depends on the load experienced by the motor. In this example, encoder feedback is used to monitor actual speed, and the result compared to a desired speed setpoint. Any error is fed through a Proportional, Integral, Derivative(PID) controller in order to adjust the applied motor voltage so as to maintain the setpoint speed.

1. Copy the **`VU-LRT\Samples\VU_motorSpdCtrl.mdl`** model into a working directory. Within MATLAB navigate to that directory, and open the model (see Fig. 11).
2. The PID controller is implemented as a 'masked' block in order to present a simple user interface for the user when the block is double-clicked (try it!). To see how it is implemented, select the PID block, and choose the **`Edit/LookUnderMask`** option (**`ctrl-V`**).
3. The masked subsystem contains a single 'embedded m-file' block. Embedded m-files are a very powerful and concise way to implement sections of an algorithm. Double-clicking the block brings up the embedded m-file editor, and you can see how the PID algorithm has been implemented.
4. Close the subsystem, build (**`ctrl-B`**) and automatically download the model to the NXT
5. Run the model and observe how the motor rotates at a constant speed even when a load is applied (eg. using your finger as a brake).

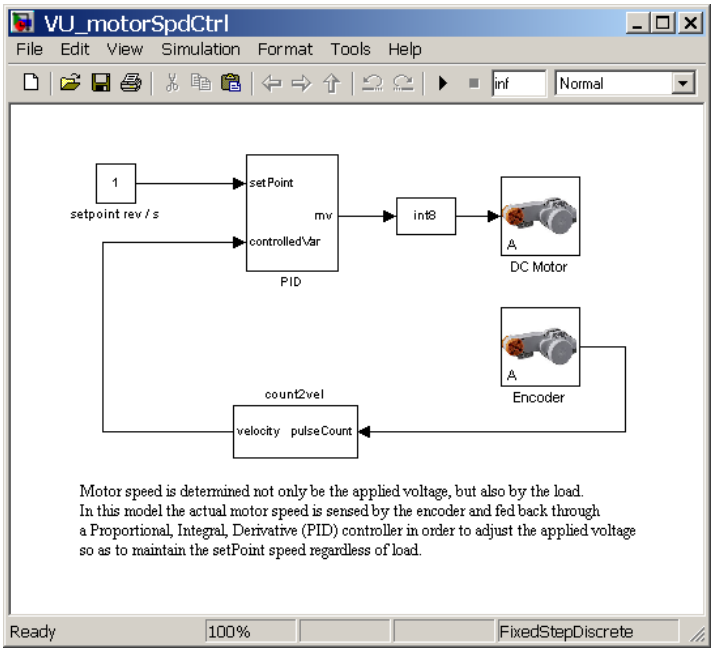


Fig. 11: The VU_motorSpdCtrl.mdl example system

5.8 Project1: A simple line-following robot with proportional feedback

The aim of this project is to implement a simple robot controller which uses feedback from the light sensor to track a black line marked on the floor. If the received light is greater than some pre-definedconstant 'setpoint' value, the robot should turn leftwards. Conversely, if the received light is less than the setpoint the robot should turn to the right. The robot should therefore track the edge of the line, with black to the left, and white to the right.

1. Construct the standard three-wheeled LEGO MindStorms robot with left and right motors on output ports B and C respectively, and with the feedback light sensor attached to input port #3.
2. In the control strategy, **convert** the light signal data type into a double, and then scale it with a **gain** block to obtain the light as a percentage of the maximum possible output (where maxOutput=1023). Then compute the error between the received light percentage and the desired setpointpercentage level.
3. Feed this error through a **gain**, and use it to modulate the speed of the motors, speeding up one and slowing down the other in order to make the robot turn according to the sign and magnitude of the error. The base motor speed should be some pre-defined **constant** value. The motors accept **int8**'s
4. Try your design initially with the light setpoint = 45, the base motor speed = 70, and the controller gain = 3. Build and test the model on the track.
5. Based on the observed results, tune your design and re-test until a fast, smooth, line-following behavior is achieved.This is harder than it sounds(!) because motor characteristics vary considerably from device to device, and because their behavior is highly nonlinear: The motor hardly turns for inputs in the range -40...40. Outside of this range, as the input is increased, the torque increases rapidly but then saturates.

6. In future versions of the VU-LRT toolbox, it is hoped that any tuning adjustments can be made while the program is executing using the Simulink 'External Mode' feature. For the present it is necessary to edit the model, re-build and download each new variant.

5.9 Project2: A simple line-following robot with state machine relay feedback

The aim of this project is to replace the proportional controller in Project1 (section 5.8) with a finite state machine implemented using an embedded MATLAB function**[leftMotor, rightMotor] = fsm(clock, sensedLight)**. The inputs to the state machine are the percentage **sensedLight** intensity, and the system **clock**. The outputs are the**leftMotor**, **rightMotor** signals that are fed to the motors.

1. The states of the robot are to be defined as follows:
   - **State 0**: Calibrating the light sensor while turning left for 2 seconds. The program stores (in persistent variables) the **maxLight** and **minLight**intensities observed during this period.
   - **State 1**: Calibrating the light sensor while turning back right for 2 seconds, and still recording maximum and minimum light intensities. At the end of the 2 second period, the **threshold** light intensity is computed as the average of **maxLight** and **minLight**
   - **State2**: (sensedLight>threshold)=>set motor speeds to move forward while turning left
   - **State 3**: (sensedLight<threshold) =>set motor speeds to move forward while turning right
2. A rough template of the embedded MATLAB function is as follows:

```
function [leftMotor, rightMotor] = fsm(clock, sensedLight)

%Define persistent variables
persistentcurrentStatemaxLightminLight threshold

%Initialize (only for the first time the function is called)
ifisempty(currentState),
currentState= uint8(0);  %Use type casts to make clear what data types
maxLight= ...;            %are to used for each variable.
minLight= ...;
threshold= ...;
end;

leftMotor= int8(0);
rightMotor= int8(0);

switch...
case 0 %Calibration - turning left
leftMotor= ...;
rightMotor= ...;
ifsensedLight>maxLight, ... end;
ifsensedLight<minLight, ...end;
if (clock>2000), currentState= ...; end;
case 1 %Calibration - turning right
leftMotor= ...;
rightMotor= ...;
ifsensedLight>maxLight, ... end;
ifsensedLight<minLight, ...; end;
if (clock>...),
threshold = ...
```

```
currentState= ...;
end;
case 2 %Too much light => turn Left
leftMotor= ...;
rightMotor= ...;
if..., currentState= ...; end;
case 3 %Too little light => turn Right
leftMotor= ...;
rightMotor= ...;
if..., currentState= ...; end;
end
```

3. Build, download and test your model on the track.
4. Based on the observed results, tune your design, re-build and re-test until a fast, smooth, line-following behavior is achieved. As before, this is harder than it sounds(!).

5.10 Other Examples

The VU-LRT toolbox has a number of other examples which may be found in the VU-LRT\Samples directory. Copy these examples to a working directory before building or modifying them. A current list of these examples is as follows:

- **VU_testSoundTone1.mdl** : A simple example to test the correct operation of the VU-LRT toolbox, using the Enter Button to cause the inbuilt loudspeaker to sound whenever the button is pressed
- **VU_testSoundTone2.mdl** : A modification of the prior example using the Enter Button to trigger an enabled sub-system when pressed. In this case the sub-system outputs a different tone each time it is enabled.
- **VU_testSoundTone3.mdl** : In this example the center frequency and volume amplitudes of the Sound Tone block are modulated by inputs from two encoder blocks, so the user can adjust these quantities by turning the motor/encoder wheels by hand.
- **VU_testUsbTxRx.mdl** : In this example **VU_testSoundTone3.mdl** is extended so that the volume and frequency signals are piped over USB via the host machine instead of being directly connected to the loudspeaker. The model executes in conjunction with the MATLAB program **VU_testUsbTxRx.m**
- **VU_testBtTxRx.mdl** : This is identical to the **VU_testUsbTxRx.mdl** except that wireless Bluetooth communications are used in place of the wired USB connection. The model executes in conjunction with the MATLAB program **VU_testBtTxRx.m**
- **VU_motorSpdCtrl.mdl:** This example demonstrates a servo motor which maintains its current speed regardless of external forces using a simple control structure.
- **VU_lineFollow.mdl:** This model implements a simple line-following robot using proportional feedback.
- **VU_lineFollow1.mdl :** This model implements a simple line-following robot using relay feedback which is calibrated and controlled using a finite state machine.

- **`VU_lineFollow1b.mdl`** : This model is similar to **`VU_lineFollow1.mdl`** only the state machine outputs do not drive the motors directly. Instead the state machine outputs speed setpoints to a motor speed controller.
- **`VU_NxtWay.mdl`**: This example is a re-implementation of the Segway-like mobile inverted pendulum controller originally designed by Yorihisa Yamamoto using the ECRobot toolbox. The system uses feedback from a HiTechnic Gyro Sensor to stabilize the system, and feedback from the Sonar Sensor to avoid obstacles.

6. Ongoing work

The VU-LRT toolbox is still under development. Current work includes:

- Implement 'External Mode' communications over USB or Bluetooth.
- Set 'Simulation' mode behavior of blocks to access the NXT remotely over USB or Bluetooth.
- Give each block an 'Attributes' tab to allow users to define input and output data types.
- Design a universal NXT block which encapsulates all possible I/O in a single block.
- Develop a second .tlc file for building more efficient executables using the Embedded Coder toolbox.

Immediate to-do list:

- Test new blocks
- Test NXT-to-NXT Bluetooth communication
- Test automatic type conversion on DC Motor2, and propagate to other blocks if good
- Investigate (with oscilloscope) the apparent nonlinear behavior of the DC motor drives.

References

1. T.Chikamasa, "*Embedded Coder Robot NXT Demo*", http://www.mathworks.com/matlabcentral/fileexchange/13399-embedded-coder-robot-nxt-demo
2. J.C. Peyton Jones, C. McArthur, T.Young, "*From Design to Implementation with Simulink and LEGO NXT*", Proceedings of the Mid-Atlantic ASEE Fall 2010 conference:pdf
3. J.C. Peyton Jones, C. McArthur, T.Young, "*The VU-LEGO Real Time Target: Taking Student Designs to Implementation*", Proceedings of the 2011 ASEE AnnualConference and Exposition, Vancouver, Canada.

Appendix: Release Notes

Version 1.01

- Minor corrections to this document
- Added ADC and PWM blocks
- Added nxtWay example
- Installer no longer assumes installation on the C: drive
- Installer Check Tools now returns the correct result for the GNU ARM Compiler
- Installer warns if savepath does not work, and suggests running in Administrator Mode
- Installer downloads 64-bit version of 7zip on 64-bit machines
- Installer performs more checking on installed C compilers for mex files

Version 1.02

- Requirements section explicitly recommends 32-bit MATLAB versus 64-bit MATLAB.
- Installation instructions include instructions for installing / selecting a C Mex compiler
- Installer includes an option for renaming the NXT brick
- USB Tx and Rx split into separate blocks + host-side USB utility functions added
- Bluetooth Tx and Rx split into separate blocks + host-side USB utility functions added
- Added raw I2C Tx and Rx blocks
- Added I2C-based Compass and Color Sensor blocks
- Replaced Raw ADC block with a more flexible Generic ADC block
- Added a generic block for the family of Vernier sensors