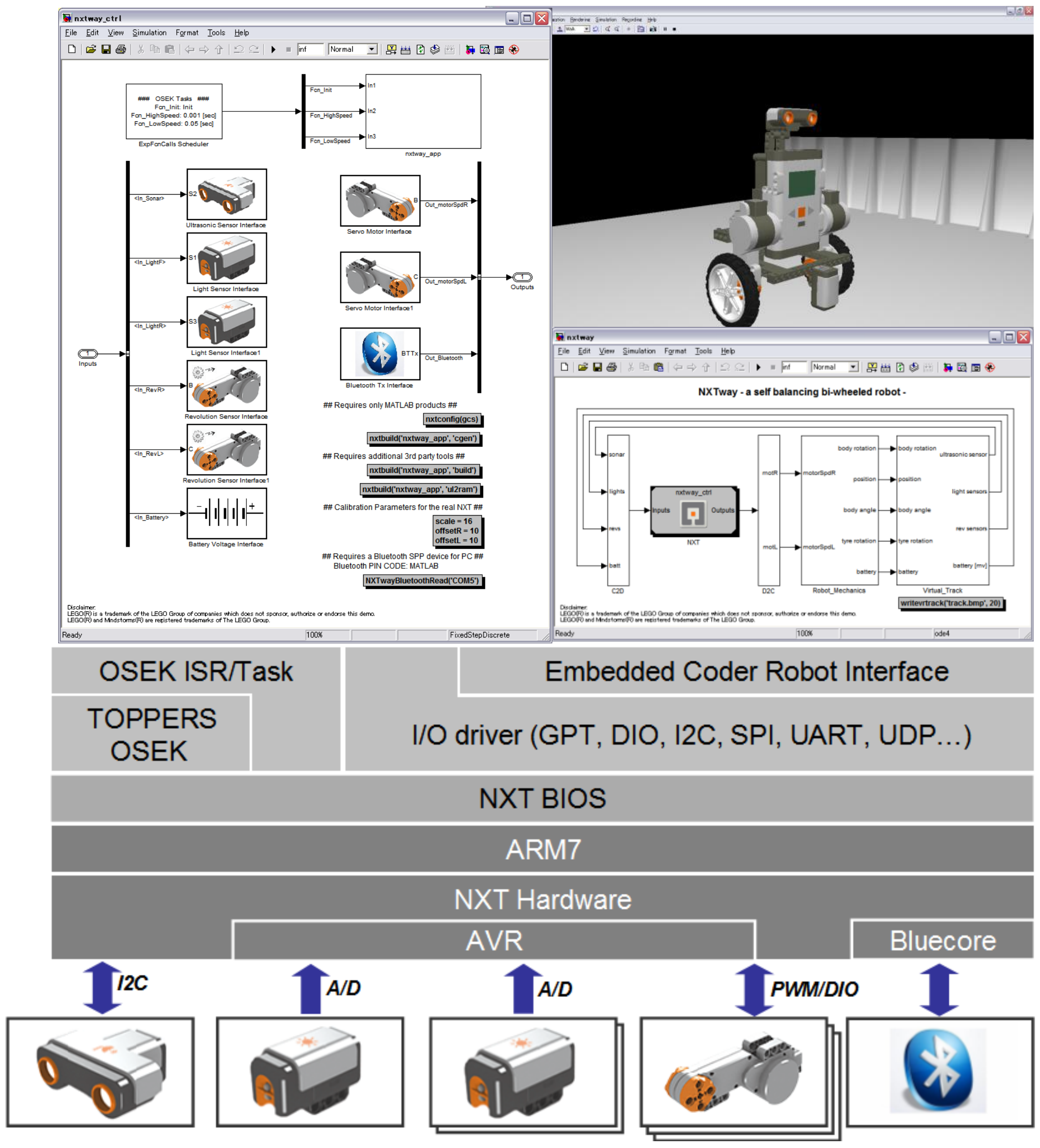
Embedded Coder Robot NXT Modeling Tips


OSEK ISR/Task
TOPPERS OSEK
Embedded Coder Robot Interface
I/O driver (GPT, DIO, I2C, SPI, UART, UDP…)
NXT BIOS
ARM7
NXT Hardware
AVR
Bluecore
I2C
A/D
A/D
PWM/DIO
nxtway_ctrl
nxtway
NXTway - a self balancing bi-wheeled robot -

Revision History

Revision	Description	Author
1.00	10/31/2006 Released as a controller design demo	Takashi Chikamasa
1.01	12/05/2006 Added Ultrasonic Sensor to NXTmouse/NXTway	Takashi Chikamasa
2.00	02/28/2007 Support RTW-EC based target deployment feature	Takashi Chikamasa
2.02	03/26/2007 Used a new LEJOS NXJ distribution	Takashi Chikamasa
3.00	06/01/2007 - Integrate LEJOS OSEK RTOS - Support Bluetooth for target deployment - Obsolete Keyboard Input block	Takashi Chikamasa
3.03	09/07/2007 - Support MATLAB R2007b - Support floating-point code generation - Added NXT GamePad Data Logger block - Added w32serial API for Bluetooth communication	Takashi Chikamasa Tomoki Fukuda
3.04	10/19/2007 - Support HiTechnic Acceleration and Gyro Sensors	Takashi Chikamasa
3.05	02/10/2008 - Support Bluetooth NXT to NXT communication - Added Sound Tone Blocks - Refactored customizations	Takashi Chikamasa
3.06	03/01/2008 - Changed program upload tools (use LEGO MINDSTORMS NXT Driver) - Added NXT Remocon demo -Added brake/float configuration for Servo Motor interface block - Re-organized docs (Separate instruction and modeling tips)	Takashi Chikamasa

Revision	Description	Author
3.10	04/03/2008 - ecrobotnxtsetup.m enables GUI based setup for target deployment - Changed for LEJOS OSEK 2.00 or later version (support application Flash by NXT BIOS) - Obsolete LEJOS C code generation	Takashi Chikamasa
3.12	06/01/2008 - Changed LEJOS OSEK to nxtOSEK - Added Sound WAV blocks	Takashi Chikamasa
3.14	11/01/2008 - Added USB Rx/Tx blocks and samples - Added ENTER button blocks - Added RUN button blocks - Added MATLAB USB API (nxtusb)	Takashi Chikamasa Tomoki Fukuda
3.16	January 2009 -Bundled nxtOSEK -Updated nxtusb API -Exported Function-Call Scheduler block supports TOPPERS/ATK(OSEK) and TOPPERS/JSP(μITRON4.0) -Refactored nxtbuild	Takashi Chikamasa Tomoki Fukuda
3.18	April 2010 - Removed Light Sensor based NXTway demo - Refactored some MATLAB scripts - Support MATLAB R2010a	Takashi Chikamasa

TABLE OF CONTENTS

Introduction

This document describes a MATLAB® and Simulink® based development environment for LEGO® Mindstorms® NXT (hereafter, NXT). The environment is called “Embedded Coder Robot for LEGO Mindstorms NXT (ECRobot NXT)” and it provides a modeling capability for NXT control strategy, plant dynamics, and the ability to simulate and visualize these model components in a 3-D virtual reality graphical environment. ECRobot NXT also provides for Real-Time Workshop® Embedded Coder based target deployment with an open source OSEK RTOS to the real NXT hardware. This means that you can fully experience Model-Based Design including modeling, simulation, code generation, target build, and testing on a real NXT for the LEGO Mindstorms NXT.

Disclaimer

LEGO® is a trademark of the LEGO Group of companies which do not sponsor, authorize or endorse this project. LEGO® and Mindstorms® are registered trademarks of The LEGO Group.

According to LEGO Mindstorms NXT Hardware Developer Kit, “*Important note: When the NXT is disassembled or when third party firmware is used with the NXT, all warranties are rendered invalid*”. It means that if you uploaded an ECRobot NXT generated binary executable to your NXT, your NXT may be out of any warranties provided from LEGO. Therefore, please be sure that the author of this document does not take any responsibility for any loss or damage of any kind incurred as a result of the use or the download of ECRobot NXT and related third party tools.

Disclaimer about the MathWorks and the products which are used for this demo, please check the following URL: http://www.mathworks.com/matlabcentral/disclaimer.html

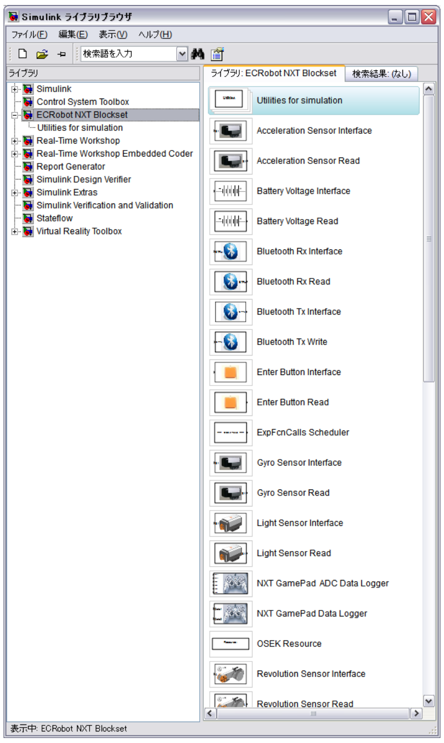
ECRobot NXT is just an example to present features of MATLAB products. Therefore, please be sure that ECRobot NXT is not an official software development environment of LEGO Mindstorms NXT.

Prerequisites

This document assumes readers are experienced with MATLAB, Simulink, Stateflow®, Real-Time Workshop®, and Real-Time Workshop Embedded Coder in general.

ECRobot NXT Blockset

If ECRobot NXT was installed successfully, ECRobot NXT Blockset comes up in Simulink Library Browser.

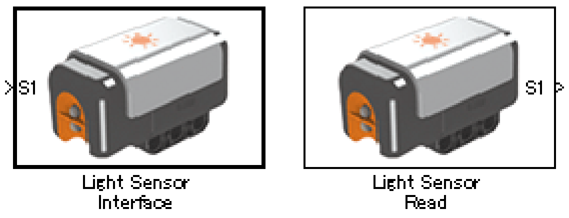


Simulink
Control System Toolbox
ECRobot NXT Blockset
Utilities for simulation
Real-Time Workshop
Real-Time Workshop Embedded Coder
Report Generator
Simulink Design Verifier
Simulink Extras
Simulink Verification and Validation
Stateflow
Virtual Reality Toolbox

Utilities for simulation
Acceleration Sensor Interface
Acceleration Sensor Read
Battery Voltage Interface
Battery Voltage Read
Bluetooth Rx Interface
Bluetooth Rx Read
Bluetooth Tx Interface
Bluetooth Tx Write
Enter Button Interface
Enter Button Read
ExpFcnCalls Scheduler
Gyro Sensor Interface
Gyro Sensor Read
Light Sensor Interface
Light Sensor Read
NXT GamePad ADC Data Logger
NXT GamePad Data Logger
OSEK Resource
Revolution Sensor Interface
Revolution Sensor Read

表示中: ECRobot NXT Blockset

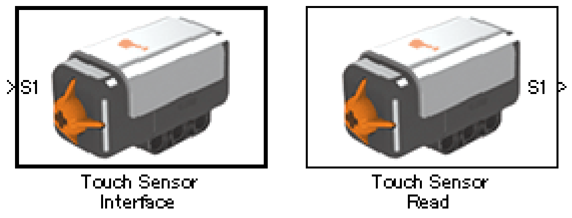
Light Sensor blocks:



Light Sensor blocks are used to measure brightness. Greater value means darker (or lower reflection). Light Sensor consists of two blocks. *Light Sensor Interface* block is an interface block to the outside of a NXT controller model. *Light Sensor Read* block is used to read Light Sensor data. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code.

Data Type	unt16
Dimension	[1 1]
Data Range	0 to 1023 (raw A/D value)
Port ID	S1/S2/S3/S4

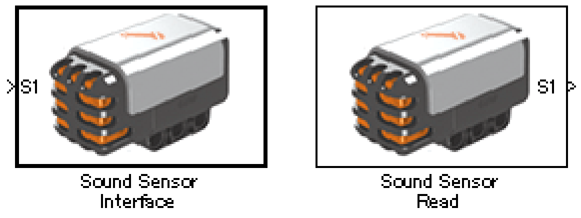
Touch Sensor blocks:



Touch Sensor blocks are used to detect contact with an obstacle. If the Touch Sensor had contact with an obstacle, the sensor returns 1. The Touch Sensor consists of two blocks. *Touch Sensor Interface* block is an interface block to the outside of a NXT controller model. *Touch Sensor Read* block is used to read the Touch Sensor data. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code.

Data Type	unt8
Dimension	[1 1]
Data Range	0(not touched), 1(touched)
Port ID	S1/S2/S3/S4

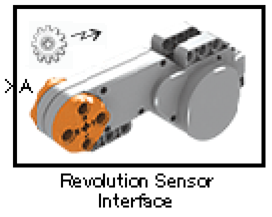
Sound Sensor blocks:

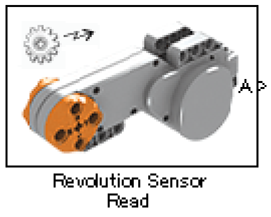


Sound Sensor blocks are used to measure the sound pressure. Smaller value means louder sound. Sound Sensor consists of two blocks. *Sound Sensor Interface* block is an interface block to the outside of a NXT controller model. *Sound Sensor Read* block is used to read Sound Sensor data. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code.

Data Type	unt16
Dimension	[1 1]
Data Range	0 to 1023 (raw A/D value)
Port ID	S1/S2/S3/S4

Revolution Sensor blocks:

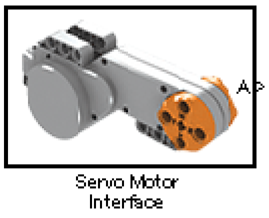


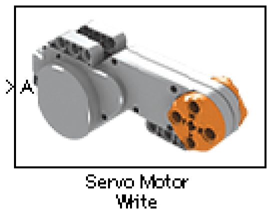


Revolution Sensor blocks are used to measure the revolution of a Servo Motor. Revolution Sensor consists of two blocks. *Revolution Sensor Interface* block is an interface block to the outside of a NXT controller model. *Revolution Sensor Read* block is used to read Servo Motor revolution data. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code.

Data Type	int32
Dimension	[1 1]
Data Range	Range of int32 [deg]
Port ID	A/B/C

Servo Motor blocks:

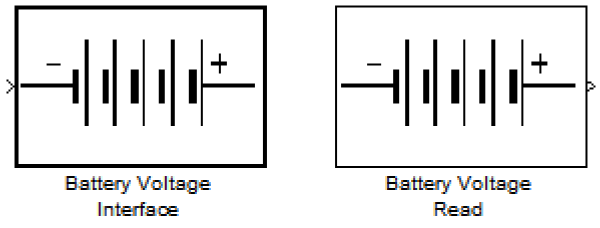




Servo Motor blocks are used to control a Servo Motor. Servo Motor consists of two blocks. *Servo Motor Interface* block is an interface block to the outside of a NXT controller model. *Servo Motor Write* block is used to set Servo Motor control data. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code.

Data Type	int8
Dimension	[1 1]
Data Range	-100 to 100
Port ID	A/B/C
Mode	Brake/Float

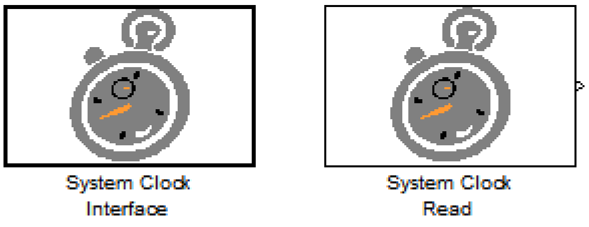
Battery Voltage blocks:



Battery Voltage blocks are used to monitor the battery voltage of NXT. Battery Voltage consists of two blocks. *Battery Voltage Interface* block is an interface block to the outside of a NXT controller model. *Battery Voltage Read* block is used to read battery voltage data. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code.

Data Type	unt16
Dimension	[1 1]
Data Range	0 to possible NXT maximum voltage value in mv (i.e. 9000 = 9.000 V)
Port ID	None

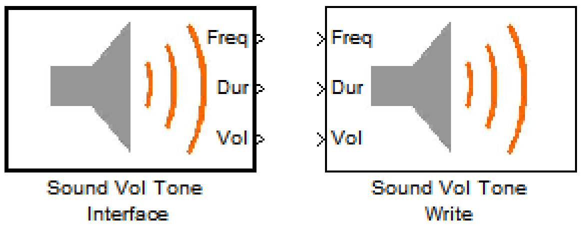
System Clock blocks:



System Clock blocks are used to read the system tick of NXT. System Clock consists of two blocks. *System Clock Interface* block has no input/output, however, it supplies system tick to *System Clock Read* block during simulation. *System Clock Read* block is used to read system tick data. They will be used to implement an appropriate device API in the generated code. In the real NXT, system tick is started from 0 when the NXT was turned on. (It is not started when the ECRobot application program begins)

Data Type	unt32
Dimension	[1 1]
Data Range	0 to maximum of uint32 in mille-second
Port ID	None

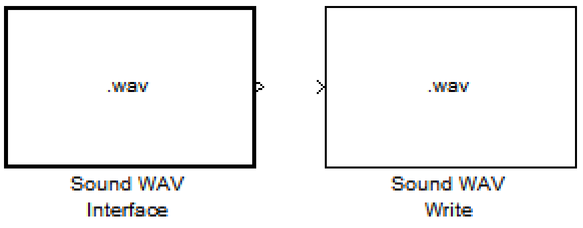
Sound Volume Tone blocks:



Sound Volume Tone blocks are used to generate sound which is determined by frequency and duration. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code. For more detailed information about how to use Sound Tone block, please see samples/TestSoundTone.mdl.

Data Type	Freq: unt32, Dur:uint32
Dimension	[1 1]
Data Range	Freq: 31 [Hz] to 14080 [Hz], Dur: 10 to 2560 [msec], Vol: 0 to 100 (0 is mute)
Port ID	None

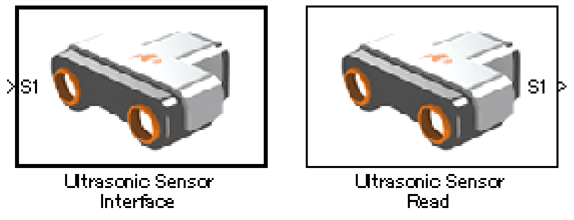
Sound WAV blocks:



Sound WAV Tone blocks are used to generate sound which is saved as a 8bit monoral WAV file. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code. For more detailed information about how to use Sound Tone block, please see samples/TestSoundWAV.mdl.

Data Type	Freq: unt32, Dur:uint32
Dimension	[1 1]
Data Range	-
Port ID	None

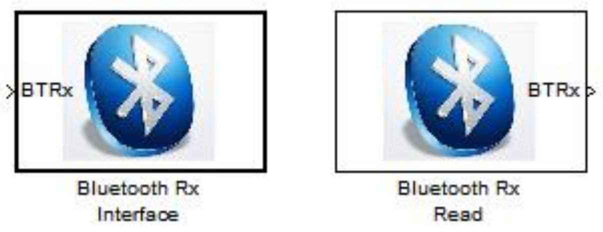
Ultrasonic Sensor blocks:



Ultrasonic Sensor blocks are used to measure the distance from an obstacle or to detect an obstacle without contact. Ultrasonic Sensor blocks consist of two blocks. *Ultrasonic Sensor Interface* block is an interface block to the outside of a NXT controller model. *Ultrasonic Sensor Read* block is used to read Ultrasonic Sensor data. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code.

Data Type	int32
Dimension	[1 1]
Data Range	0 to 255 [cm], -1 (the sensor is not ready for measurement)
Port ID	S1/S2/S3/S4

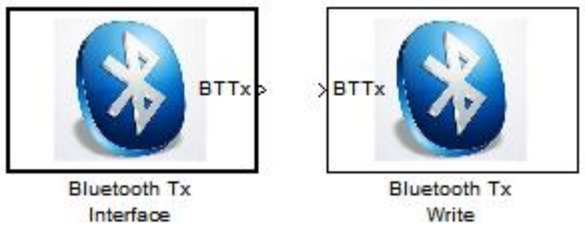
Bluetooth Rx blocks:



Bluetooth Rx blocks represent Bluetooth wireless communication from PC to NXT. Bluetooth Rx blocks consist of two blocks. *Bluetooth Rx Interface* block is an interface block to the outside of the NXT controller model. *Bluetooth Rx Read* block is used to read data. Bluetooth Rx blocks are allowed to allocate only a single instance in a model. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code. Bluetooth connection is supported for only PC – NXT (not supported for NXT - NXT).

Data Type	uint8
Dimension	32
Data Range	0 to 255
Port ID	None

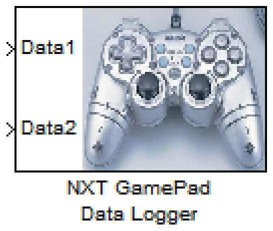
Bluetooth Tx blocks:



Bluetooth Tx blocks represent Bluetooth wireless communication from NXT to PC. Bluetooth Tx blocks consist of two blocks. *Bluetooth Tx Interface* block is an interface block to the outside of the NXT controller model. *Bluetooth Tx Write* block is used to send data. Bluetooth Tx blocks are allowed to allocate only a single instance in a model. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code. Bluetooth connection is supported for both of PC – NXT of NXT - NXT.

Data Type	uint8
Dimension	32
Data Range	0 to 255
Port ID	None

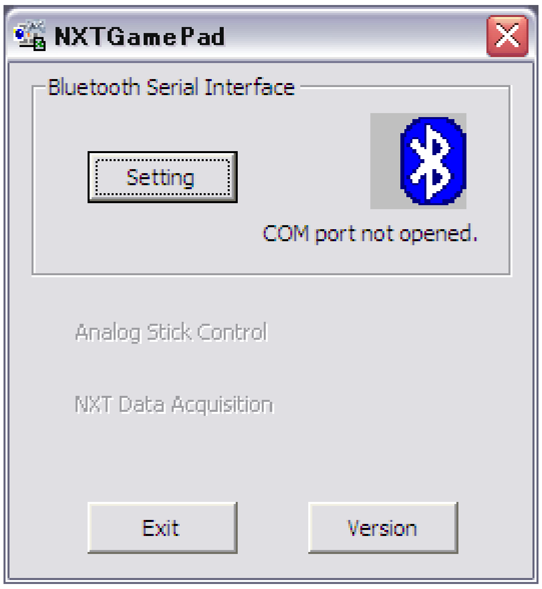
NXT GamePad Data Logger block:



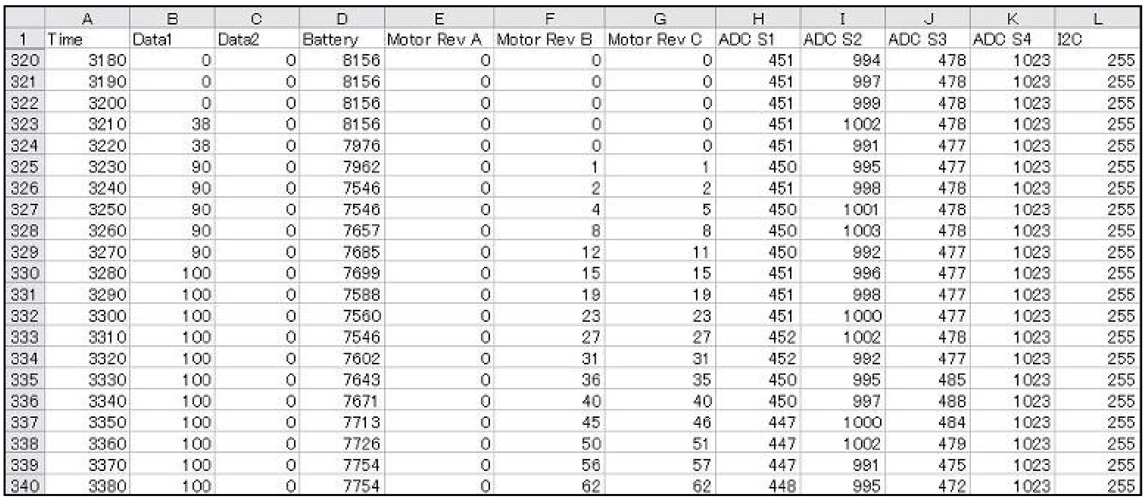
nxtOSEK has provided a utility tool which is called NXT GamePad. NXT GamePad enables user to remotely control a NXT which has nxtOSEK via Bluetooth by using a PC analog game pad controller. Since V1.01, user also has been able to select 'Analog Stick Control' and/or 'NXT Data Acquisition' features depending on purposes. Even if user did not have an analog game pad, NXT GamePad could be used for only 'NXT Data Acquisition' (data logging). For more detailed information about NXT GamePad, please see http://lejos-osek.sourceforge.net/nxtgamepad.htm
NXT GamePad Data Logger block does not affect simulation result, but it will be used to implement an appropriate device API in the generated code. Data1 and Data2 could be used for logging Analog GamePad inputs. The logged data could be stored into a CSV file and it is useful to analyze the behavior of NXT application in MATLAB.

NXT GamePad Data Logger block should not be used with *Bluetooth Tx Write* block/*NXT GamePad ADC Data Logger* block in a model.

Data Type	int8
Dimension	[1 1]
Data Range	-128 to 127
Port ID	None



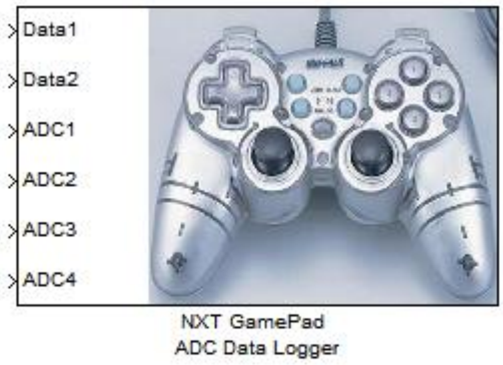
COM port not opened.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Time	Data1	Data2	Battery	Motor Rev A	Motor Rev B	Motor Rev C	ADC S1	ADC S2	ADC S3	ADC S4	I2C
320	3180	0	0	8156	0	0	0	451	994	478	1023	255
321	3190	0	0	8156	0	0	0	451	997	478	1023	255
322	3200	0	0	8156	0	0	0	451	999	478	1023	255
323	3210	38	0	8156	0	0	0	451	1002	478	1023	255
324	3220	38	0	7976	0	0	0	451	991	477	1023	255
325	3230	90	0	7962	0	1	1	450	995	477	1023	255
326	3240	90	0	7546	0	2	2	451	998	478	1023	255
327	3250	90	0	7546	0	4	5	450	1001	478	1023	255
328	3260	90	0	7657	0	8	8	450	1003	478	1023	255
329	3270	90	0	7685	0	12	11	450	992	477	1023	255
330	3280	100	0	7699	0	15	15	451	996	477	1023	255
331	3290	100	0	7588	0	19	19	451	998	477	1023	255
332	3300	100	0	7560	0	23	23	451	1000	477	1023	255
333	3310	100	0	7546	0	27	27	452	1002	478	1023	255
334	3320	100	0	7602	0	31	31	452	992	477	1023	255
335	3330	100	0	7643	0	36	35	450	995	485	1023	255
336	3340	100	0	7671	0	40	40	450	997	488	1023	255
337	3350	100	0	7713	0	45	46	447	1000	484	1023	255
338	3360	100	0	7726	0	50	51	447	1002	479	1023	255
339	3370	100	0	7754	0	56	57	447	991	475	1023	255
340	3380	100	0	7754	0	62	62	448	995	472	1023	255

NXT GamePad main dialog and the logged data in CSV file

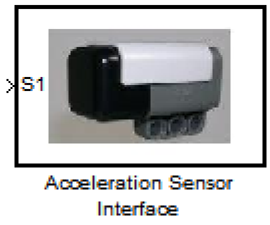
NXT GamePad ADC Data Logger block:

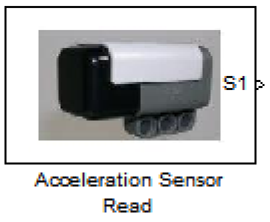


NXT GamePad ADC Data Logger has the similar functionality as *NXT GamePad Data Logger*, but user can configure four signals to be logged. *NXT GamePad ADC Data Logger* block should not be used with *Bluetooth Tx Write* block/*NXT GamePad Data Logger* block in a model. This block requires nxtOSEK platform in *Exported Function-Calls Schedule*r block to display user selected ADC values in the LCD.

Input name	Data type	Data range	Dimension
Data1/Data2	int8	-128 to 127	1
ADC1/ADC2/ADC3/ADC4	int16	-32768 to 32767	1

Acceleration Sensor blocks:

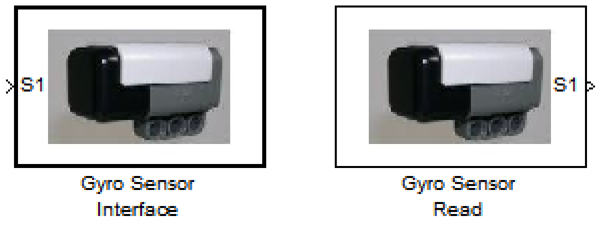




Acceleration Sensor blocks are used to measure acceleration in three axes. Acceleration Sensor is not a standard LEGO product, but made by HiTechnic (http://www.hitechnic.com/)
Acceleration Sensor blocks consist of two blocks. *Acceleration Sensor Interface* block is an interface block to the outside of a NXT controller model. *Acceleration Sensor Read* block is used to read Acceleration Sensor data. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code.

Data Type	int16
Dimension	[1 3]　(index 1: x axis, index 2: y axis, index 3: z axis)
Data Range	-512 to 511
Port ID	S1/S2/S3/S4

Gyro Sensor blocks:

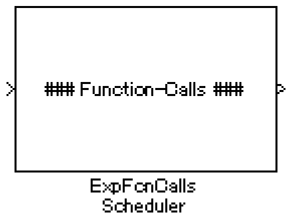


Gyro Sensor blocks are used to measure a value representing the number of degrees per second of rotation. Gyro Sensor is not a standard LEGO product, but made by HiTechnic (http://www.hitechnic.com/)

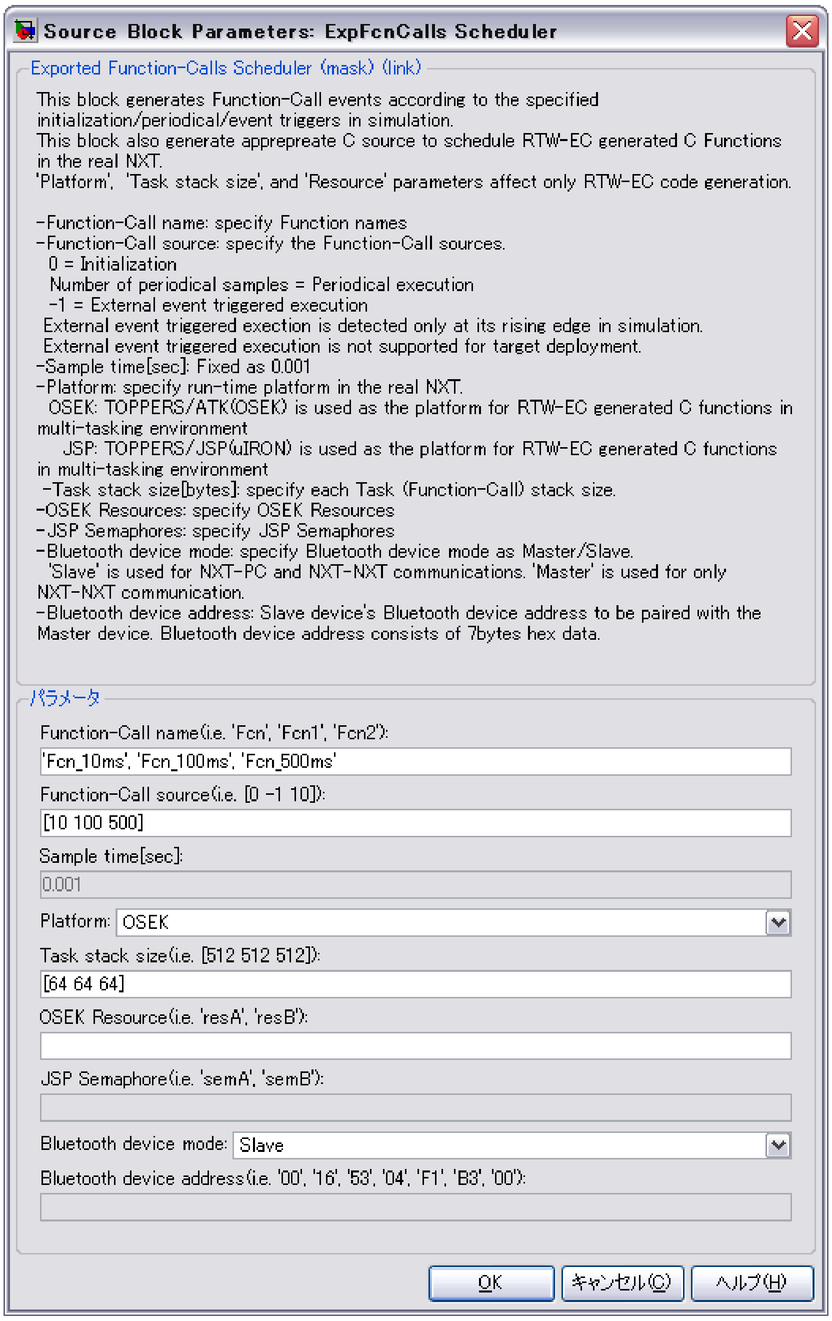
Gyro Sensor consists of two blocks. *Gyro Sensor Interface* block is an interface block to the outside of a NXT controller model. *Gyro Sensor Read* block is used to read Gyro Sensor data. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code.

Data Type	unt16
Dimension	[1 1]
Data Range	0 to 1023 (raw A/D value)
Port ID	S1/S2/S3/S4

Exported Function-Calls Scheduler block:



Typical embedded control system software (i.e. automotive power-train ECU software) is required to execute its control strategy at several different timings (i.e. initialization, asynchronous, periodical). *Exported Function-Calls Scheduler* block controls the execution timing of *Function-Call Subsystems* during simulation according to specified parameters on the Block Parameters dialog. This parameter information will also be used for ecrobot_main.c file generation. Since Embedded Coder Robot NXT v3.16, Platform software can be configured either TOPPERS/OSEK or TOPPERS/JSP. TOPPERS/JSP is complied with Japan original open RTOS specification "μITRON 4.0". μITRON has been widely used in mainly consumer electronics, industrial machinery… for more than 20 years in Japan.



Source Block Parameters: ExpFcnCalls Scheduler

Exported Function-Calls Scheduler (mask) (link)

This block generates Function-Call events according to the specified initialization/periodical/event triggers in simulation.
This block also generate apprepreate C source to schedule RTW-EC generated C Functions in the real NXT.
'Platform', 'Task stack size', and 'Resource' parameters affect only RTW-EC code generation.

-Function-Call name: specify Function names
-Function-Call source: specify the Function-Call sources.
 0 = Initialization
 Number of periodical samples = Periodical execution
 -1 = External event triggered execution
 External event triggered exection is detected only at its rising edge in simulation.
 External event triggered execution is not supported for target deployment.
-Sample time[sec]: Fixed as 0.001
-Platform: specify run-time platform in the real NXT.
 OSEK: TOPPERS/ATK(OSEK) is used as the platform for RTW-EC generated C functions in multi-tasking environment
 JSP: TOPPERS/JSP(uIRON) is used as the platform for RTW-EC generated C functions in multi-tasking environment
-Task stack size[bytes]: specify each Task (Function-Call) stack size.
-OSEK Resources: specify OSEK Resources
-JSP Semaphores: specify JSP Semaphores
-Bluetooth device mode: specify Bluetooth device mode as Master/Slave.
 'Slave' is used for NXT-PC and NXT-NXT communications. 'Master' is used for only NXT-NXT communication.
-Bluetooth device address: Slave device's Bluetooth device address to be paired with the Master device. Bluetooth device address consists of 7bytes hex data.

パラメータ

Function-Call name(i.e. 'Fcn', 'Fcn1', 'Fcn2'): 'Fcn_10ms', 'Fcn_100ms', 'Fcn_500ms'

Function-Call source(i.e. [0 -1 10]): [10 100 500]

Sample time[sec]: 0.001

Platform: OSEK

Task stack size(i.e. [512 512 512]): [64 64 64]

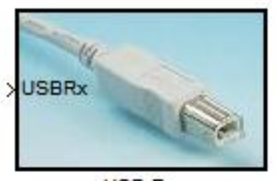
OSEK Resource(i.e. 'resA', 'resB'):

JSP Semaphore(i.e. 'semA', 'semB'):

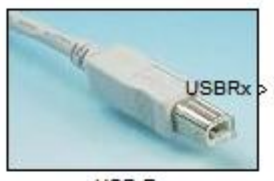
Bluetooth device mode: Slave

Bluetooth device address(i.e. '00', '16', '53', '04', 'F1', 'B3', '00'):

USB Rx blocks:



USB Rx
Interface

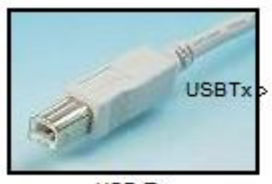


USB Rx
Read

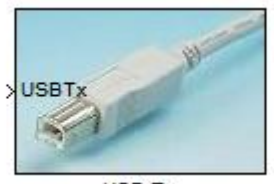
USB Rx blocks represent USB communication from PC to NXT. USB Rx blocks consist of two blocks. *USB Rx Interface* block is an interface block to the outside of the NXT controller model. *USB Rx Read* block is used to read data. USB Rx blocks are allowed to allocate only a single instance in a model. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code.

Data Type	uint8
Dimension	64
Data Range	0 to 255
Port ID	None

USB Tx blocks:



USB Tx
Interface

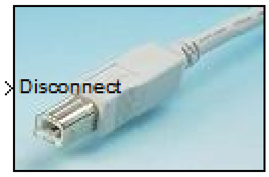


USB Tx
Write

USB Tx blocks represent USB communication from NXT to PC. USB Tx blocks consist of two blocks. *USB Tx Interface* block is an interface block to the outside of the NXT controller model. *USB Tx Write* block is used to send data. USB Tx blocks are allowed to allocate only a single instance in a model. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code.

Data Type	uint8
Dimension	64
Data Range	0 to 255
Port ID	None

USB Disconnect block:

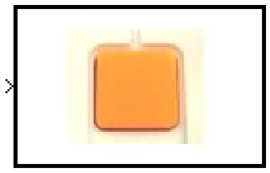


USB Disconnect

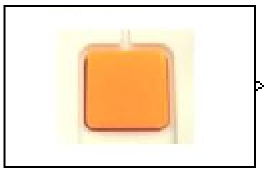
USB Disconnect block represents disconnection of a USB communication from NXT to PC. *USB Disconnect* block is used to disconnect a USB connection between the NXT and the PC. USB Disconnect block is allowed to allocate only a single instance in a model. In simulation, this block is just place holder; however, they will be used to implement an appropriate device API in the generated code.

Data Type	uint8
Dimension	1
Data Range	1(request disconnection) / 0(not disconnect)
Port ID	None

ENTER button blocks:



Enter Button Interface

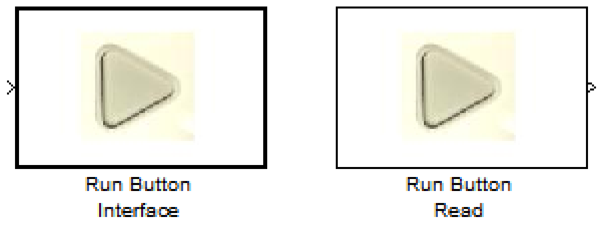


Enter Button Read

ENTER button blocks are used to detect contact with an obstacle. If the ENTER button is pressed, it returns 1. ENTER button blocks consist of two blocks. *ENTER button Interface* block is an interface block to the outside of a NXT controller model. *ENTER button Read* block is used to read the ENTER button status. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code.

Data Type	unt8
Dimension	[1 1]
Data Range	0(not touched), 1(touched)
Port ID	S1/S2/S3/S4

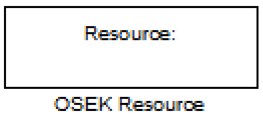
RUN button blocks:



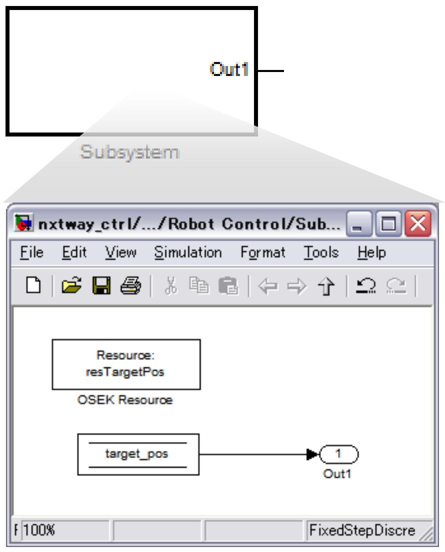
RUN button blocks are used to detect contact with an obstacle. If the RUN button is pressed, it returns 1. RUN button blocks consist of two blocks. *RUN button Interface* block is an interface block to the outside of a NXT controller model. *RUN button Read* block is used to read the RUN button status. In simulation, these blocks are just place holders; however, they will be used to implement an appropriate device API in the generated code.

Data Type	unt8
Dimension	[1 1]
Data Range	0(not touched), 1(touched)
Port ID	S1/S2/S3/S4

OSEK Resource block:



OSEK Resource block is used to implement the OSEK GetResource/ReleaseResource API at the beginning/end of the critical section. In Simulink, the critical section should be packed into an Atomic Subsystem and an OSEK Resource block should be allocated inside of the Atomic Subsystem with a specified Resource. To specify a Resource identifier in the block, you need to define the Resource identifier in the *Exported Function-Call Scheduler* block. The *OSEK Resource* block does not affect simulation and is only valid for code generation.



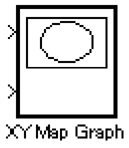
```
#ifdef OSEK
    GetResource( resTargetPos );
#endif

rtb_DataStoreRead4 = rtDWork.target_pos;

#ifdef OSEK
    ReleaseResource( resTargetPos );
#endif
```

Example of *OSEK Resource* block and generated code (nxtway_ctrl.mdl)

XY Map Graph block:



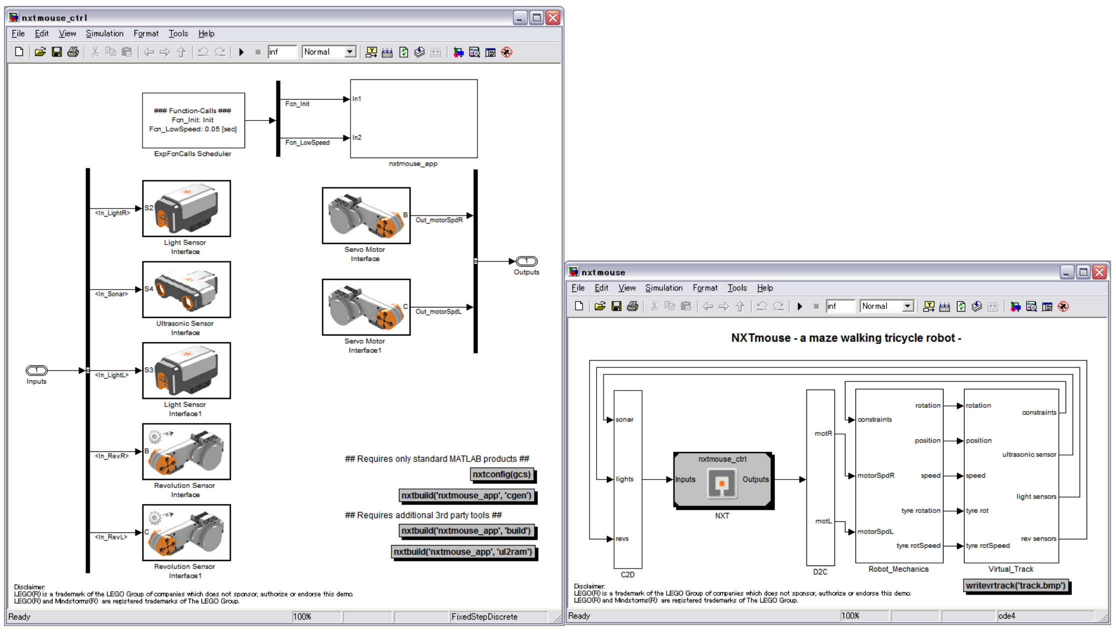
XY Map Graph block is stored in the Utilities for simulation and it is useful to see the footprint of a robot on virtual track during simulation. *XY Map Graph* is based on a Simulink standard block which is called *XY Graph*.

ECRobot NXT modeling guidelines

In the last couple of years, many new features have been added to Simulink, Stateflow, Real-Time Workshop, and Real-Time Workshop Embedded Coder products. These tools are core products for Model-Based Design. This ECRobot NXTBlockset and demo models are designed under modeling guidelines that are based on the new features of the latest MathWorks products.

Separation of plant and controller by Model Reference:

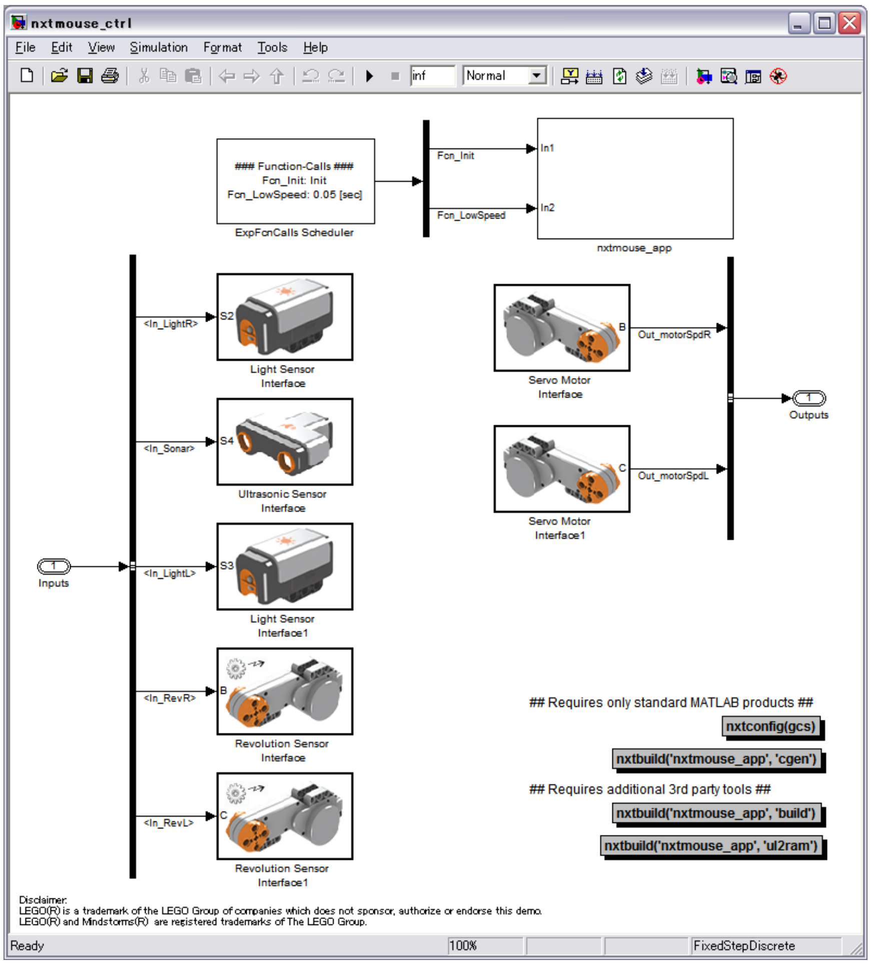
ECRobot NXT demo uses the Model Reference feature to split the controller and plant parts into different Simulink model files. This modeling approach allows a parallel development process for controller design and physical plant design. Simulink Buses and Bus Objects are used to keep the interface between the controller model and plant model.


nxtmouse_ctrl
ExpFcnCalls Scheduler
nxtmouse_app
Light Sensor Interface
Ultrasonic Sensor Interface
Light Sensor Interface1
Revolution Sensor Interface
Revolution Sensor Interface1
Servo Motor Interface
Servo Motor Interface1
Inputs
Outputs
## Requires only standard MATLAB products ##
nxtconfig(gcs)
nxtbuild('nxtmouse_app', 'cgen')
## Requires additional 3rd party tools ##
nxtbuild('nxtmouse_app', 'build')
nxtbuild('nxtmouse_app', 'ul2ram')
nxtmouse
NXTmouse - a maze walking tricycle robot -
C2D
NXT
D2C
Robot_Mechanics
Virtual_Track
writevrtrack('track.bmp')

Controller model (nxtmouse_ctrl.mdl) and plant model (nxtmouse.mdl)

Software environment and application architecture in controller model:

In the software industry, separation of software environment (i.e. RTOS, device API…) and target independent application software is a common technique to achieve reusability of software components and support large-scale team based development. This proven technique also can be used for Simulink controller modeling. In the ECRobot NXT controller model, the software environment (scheduler) and application Subsystem are clearly separated. Additionally, external interfaces (NXT sensors, motors, and communication device) are represented by specific interface blocks. Inside of an application Subsystem, there are *Function-Call Subsystems* that are executed by an *Exported Function-Calls Scheduler* block. The architecture of the ECRobot NXT controller model follows a real embedded software architecture, and is thus a highly readable NXT controller model that serves as an executable specification.



Function-Calls
Fcn_Init: Init
Fcn_LowSpeed: 0.05 [sec]

Requires only standard MATLAB products

```
nxtconfig(gcs)
nxtbuild('nxtmouse_app', 'cgen')
```

Requires additional 3rd party tools

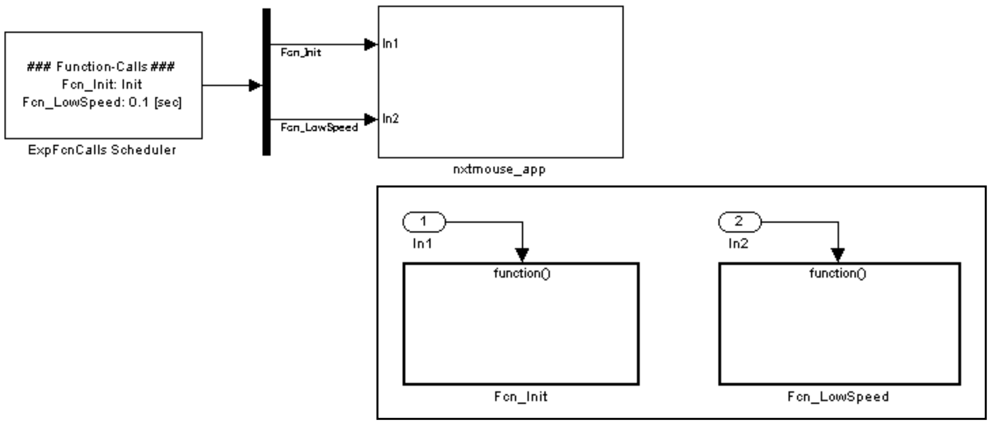
```
nxtbuild('nxtmouse_app', 'build')
nxtbuild('nxtmouse_app', 'ul2ram')
```

Disclaimer:
LEGO(R) is a trademark of the LEGO Group of companies which does not sponsor, authorize or endorse this demo.
LEGO(R) and Mindstorms(R) are registered trademarks of The LEGO Group.

Controller model (nxtmouse_ctrl.mdl)

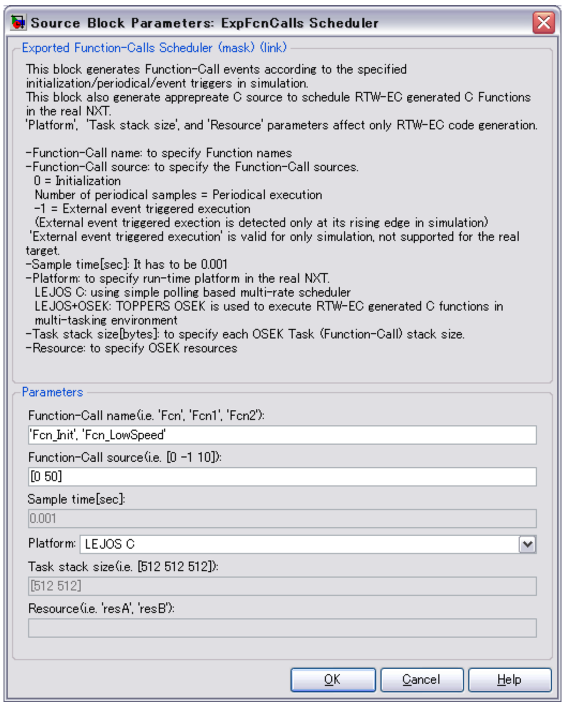
Exported Function-Calls Scheduler controls execution timing of application Subsystems:

In general, execution of a *Function-Call Subsystem* is controlled by a *Function-Call Generator* or a Stateflow Event. Especially, Stateflow allows users to design a complex scheduler system. In ECRobot NXT, an alternative approach is introduced to control the execution timing using *Function-Call Subsystems*.



Exported Function-Calls Scheduler and *Function-Call Subsystems*

Exported Function-Calls Scheduler block is a C MEX S-Function based custom block. It provides a Block Parameters dialog to specify the necessary information for controlling the execution timing of *Function-Call Subsystems*. Since R2006a, Simulink and Real-Time Workshop Embedded Coder have provided an 'Exporting Function-Call Subsystem code generation' feature that enables one to generate highly efficient void-void C functions from *Function-Call Subsystems*.


Source Block Parameters: ExpFcnCalls Scheduler

Exported Function-Calls Scheduler (mask) (link)

This block generates Function-Call events according to the specified initialization/periodical/event triggers in simulation.
This block also generate apprepreate C source to schedule RTW-EC generated C Functions in the real NXT.
'Platform', 'Task stack size', and 'Resource' parameters affect only RTW-EC code generation.

```
-Function-Call name: to specify Function names
-Function-Call source: to specify the Function-Call sources.
  0 = Initialization
  Number of periodical samples = Periodical execution
  -1 = External event triggered execution
  (External event triggered exection is detected only at its rising edge in simulation)
 'External event triggered execution' is valid for only simulation, not supported for the real target.
-Sample time[sec]: It has to be 0.001
-Platform: to specify run-time platform in the real NXT.
  LEJOS C: using simple polling based multi-rate scheduler
  LEJOS+OSEK: TOPPERS OSEK is used to execute RTW-EC generated C functions in
  multi-tasking environment
-Task stack size[bytes]: to specify each OSEK Task (Function-Call) stack size.
-Resource: to specify OSEK resources
```

Parameters

Function-Call name(i.e. 'Fcn', 'Fcn1', 'Fcn2'): 'Fcn_Init', 'Fcn_LowSpeed'

Function-Call source(i.e. [0 -1 10]): [0 50]

Sample time[sec]: 0.001

Platform: LEJOS C

Task stack size(i.e. [512 512 512]): [512 512]

Resource(i.e. 'resA', 'resB'):

Block Parameters dialog of *Exported Function-Calls Scheduler*

Inter *Function-Call Subsystems* communication via *Data Store Read/Write* blocks:

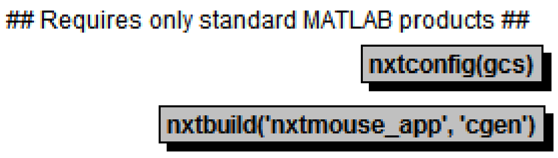
Function-Call Subsystem is a unique feature in terms of Simulink modeling semantics. In general, execution order of each block in Simulink model is determined by its context (i.e. connection of signal lines). However, *Function-Call Subsystem* itself has no context for execution timing and the execution is always controlled by its caller (i.e. Stateflow Event). These characteristics sometimes lead to difficulty regarding data transfers between *Function-Call Subsystems*. For more detailed information, please type the following command in MATLAB command window.

```
>>sl_subsys_semantics
```

In the ECRobot NXT demo, inter *Function-Call Subsystems* communication has to be done by *Data Store Read/Write* blocks. ECRobot device blocks like *Light Sensor Interface*/*Light Sensor Read* blocks are actually masked *Data Store Read/Write* blocks and the block identifiers are resolved by custom NXT Signal Objects during model update. Use of *Data Store Read/Write* blocks solves potential issues laid down on *Function-Call Subsystems* communication. However, there is one thing to be reminded when using *Data Store Read/Write* block. *Data Store Read/Write* blocks cut off signal flow between the blocks; therefore, Simulink may determine an unexpected execution order for the blocks, thus users may need to explicitly specify the execution priority for each *Data Store Read/Write* block in block property dialog.

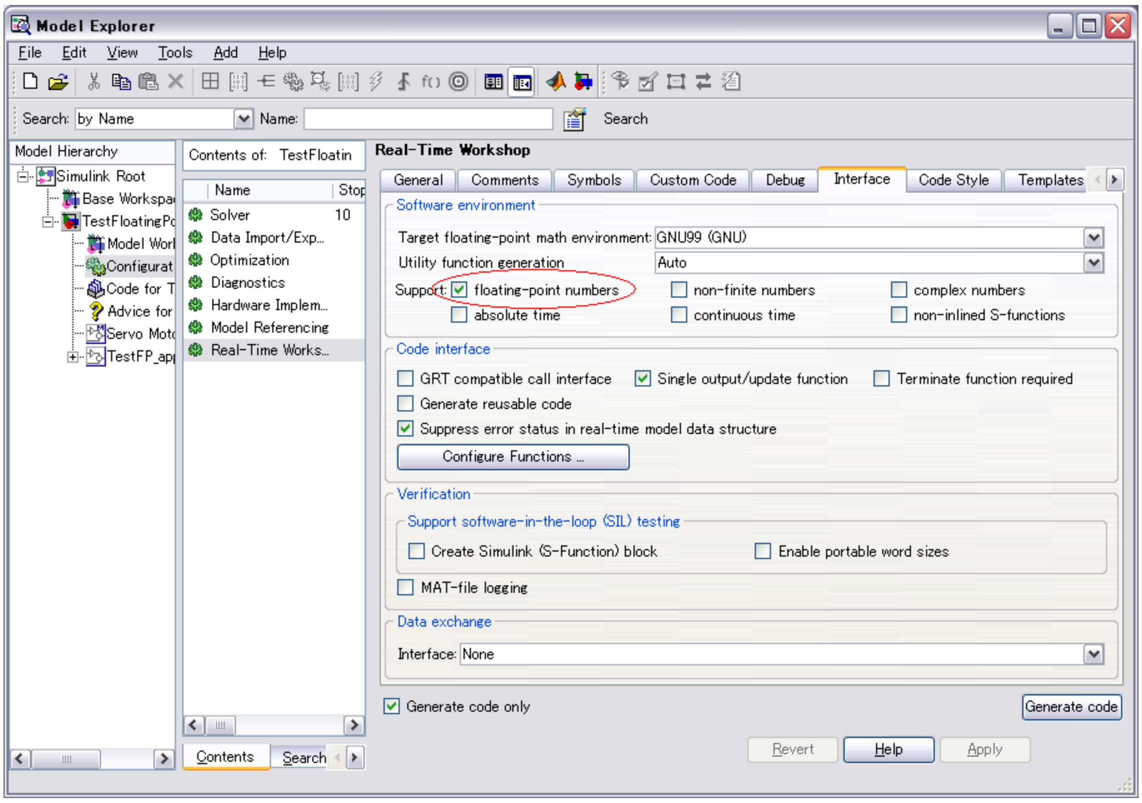
nxtconfig API for setting configuration parameters:

nxtconfig API is an M function and this API sets all necessary configuration parameters for the ECRobot NXT controller model. This kind of automation is also a significant factor for large scale Model-Based Design process, especially for Production Code Generation use cases using Real-Time Workshop Embedded Coder.


```
## Requires only standard MATLAB products ##
nxtconfig(gcs)
nxtbuild('nxtmouse_app', 'cgen')
```

Annotation callback for nxtconfig (nxtmouse_ctrl.mdl)

Since the initial release of ECRobot NXT demo, a lot of feedbacks have been received and most of users are in academic area and academic users may want to use more variety types of blocks and floating-point data types. Therefore, ECRobot NXT V3.03 supports not only fixed-point/integer code generation, but also floating-point code generation (both of single and double). nxtconfig API turns off support of ‘floating-point numbers’ in Real-Time Workshop option pane, therefore, user needs to turn on this option manually when doing floating-point code generation. TestFloatingPoint.mdl in samples directory is an example of floating-point code generation.



Real-Time Workshop — Interface

Software environment

Target floating-point math environment: GNU99 (GNU)

Utility function generation: Auto

Support: [✓] floating-point numbers [ ] non-finite numbers [ ] complex numbers

[ ] absolute time [ ] continuous time [ ] non-inlined S-functions

Code interface

[ ] GRT compatible call interface [✓] Single output/update function [ ] Terminate function required

[ ] Generate reusable code

[✓] Suppress error status in real-time model data structure

Configure Functions ...

Verification

Support software-in-the-loop (SIL) testing

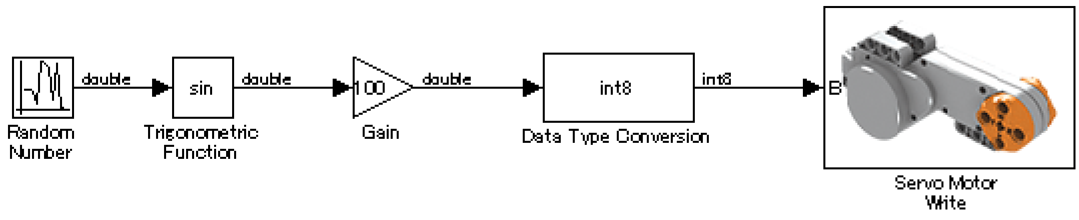
[ ] Create Simulink (S-Function) block [ ] Enable portable word sizes

[ ] MAT-file logging

Data exchange

Interface: None

[✓] Generate code only Generate code



'floating-point numbers' option and logic of TestFloatingPoint.mdl

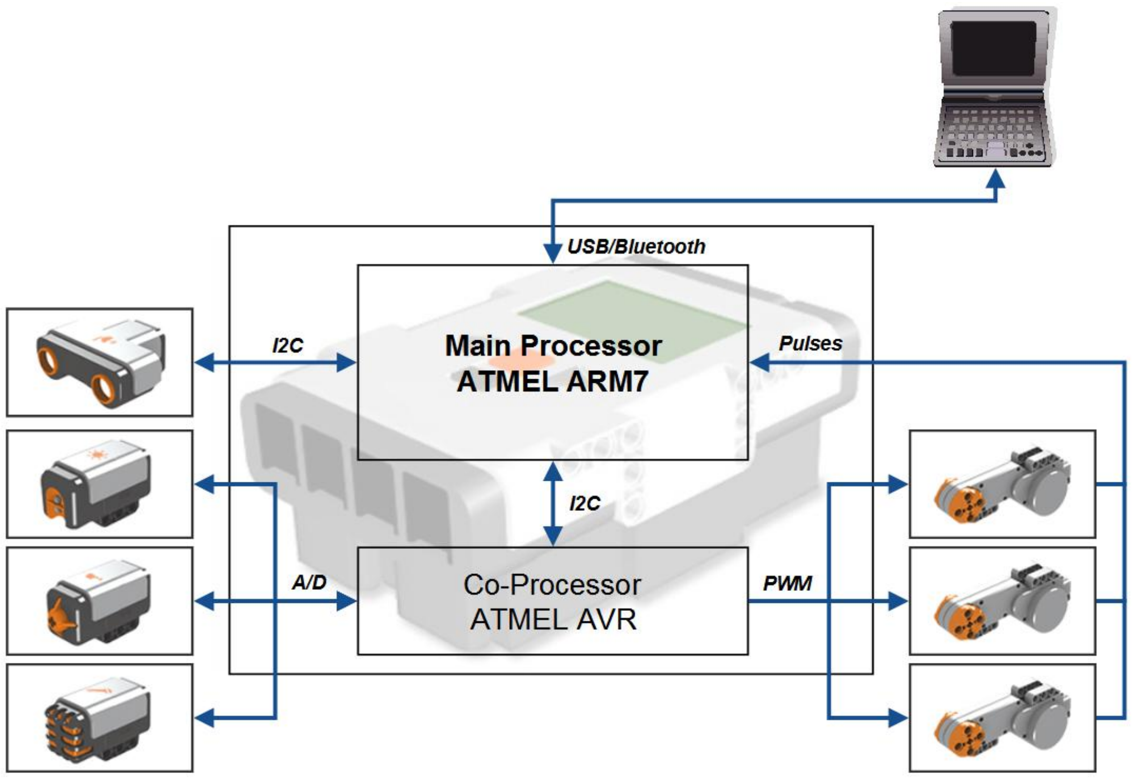
ATMEL AT91SAM7S256 (ARM7 CPU core) in the NXT does not have Floating Point Unit and some math function blocks require C math library in the generated code, therefore please remind that floating-point code generation consumes more memory and has less performance compared to fixed-point/integer code generation.

ECRobot NXT run-time environment

This chapter describes how the ECRobot NXT application program runs in the real NXT.

System architecture of LEGO Mindstorms NXT:

The below figure provides a graphical overview of LEGO Mindstorms NXT system architecture according to LEGO Mindstorms NXT Hardware Developer Kit.

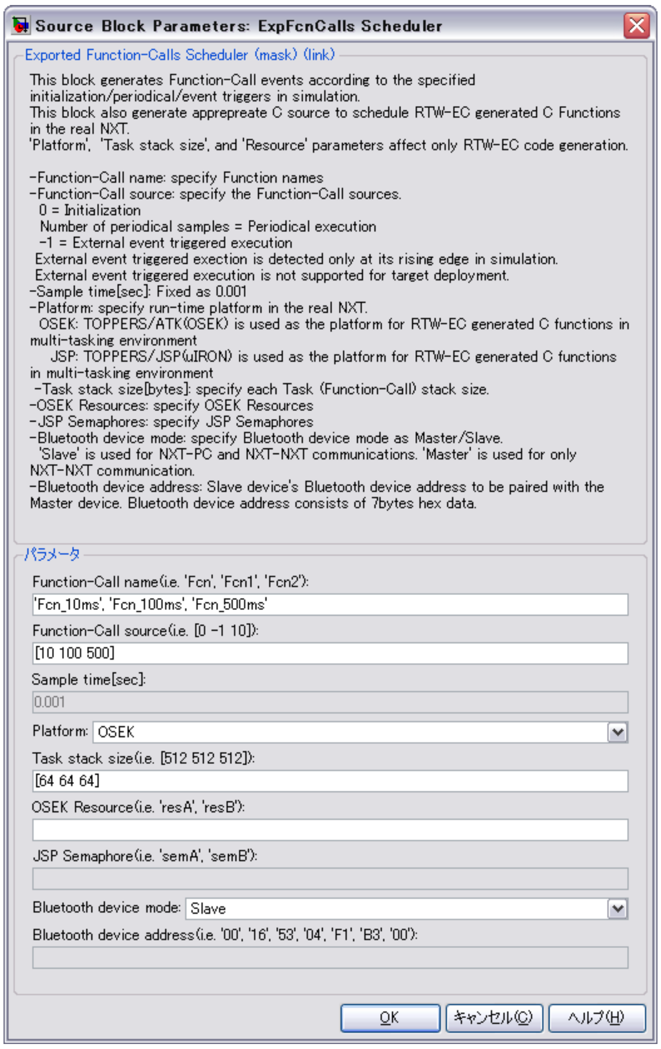


Graphical overview of LEGO Mindstorms NXT system architecture

This figure shows that the communication between the main ARM7 processor (ATMEL AT91SAM7S256) and Sensors/Servo Motors is done via the co-processor (ATMEL AVR), except for the Ultrasonic Sensor and acquisition of Servo Motor revolutions. For ECRobot NXT applications, the most important factor to access the Sensors/Servo Motors is the communication with the co-processor via the I2C serial bus. This system architecture definitely influences the software run-time environment of the ECRobot NXT. The main ARM7 processor accesses the Sensors (to read sensor A/D value) and Servo Motors (to set PWM duty ratio) independently every 2 milli seconds through a 1 milli second periodic Interrupt Service Routine of LEJOS NXJ firmware. Servo Motors revolutions are directly captured by pulse triggered Interrupt Service Routines of the LEJOS NXJ firmware in the main ARM7 processor. The Ultrasonic Sensor has its brain directly communicate with the main ARM7 processor via another I2C communication channel.

nxtOSEK(TOPPERS/OSEK or TOPPERS/JSP) Rate Monotonic Scheduler:

ECRobot NXT generates OSEK/JSP based Rate Monotonic Scheduler.



Source Block Parameters: ExpFcnCalls Scheduler

Exported Function-Calls Scheduler (mask) (link)

This block generates Function-Call events according to the specified
initialization/periodical/event triggers in simulation.
This block also generate apprepreate C source to schedule RTW-EC generated C Functions
in the real NXT.
'Platform', 'Task stack size', and 'Resource' parameters affect only RTW-EC code generation.

-Function-Call name: specify Function names
-Function-Call source: specify the Function-Call sources.
 0 = Initialization
 Number of periodical samples = Periodical execution
 -1 = External event triggered execution
 External event triggered exection is detected only at its rising edge in simulation.
 External event triggered execution is not supported for target deployment.
-Sample time[sec]: Fixed as 0.001
-Platform: specify run-time platform in the real NXT.
 OSEK: TOPPERS/ATK(OSEK) is used as the platform for RTW-EC generated C functions in
multi-tasking environment
 JSP: TOPPERS/JSP(uIRON) is used as the platform for RTW-EC generated C functions
in multi-tasking environment
 -Task stack size[bytes]: specify each Task (Function-Call) stack size.
-OSEK Resources: specify OSEK Resources
-JSP Semaphores: specify JSP Semaphores
-Bluetooth device mode: specify Bluetooth device mode as Master/Slave.
 'Slave' is used for NXT-PC and NXT-NXT communications. 'Master' is used for only
NXT-NXT communication.
-Bluetooth device address: Slave device's Bluetooth device address to be paired with the
Master device. Bluetooth device address consists of 7bytes hex data.

パラメータ

Function-Call name(i.e. 'Fcn', 'Fcn1', 'Fcn2'): 'Fcn_10ms', 'Fcn_100ms', 'Fcn_500ms'

Function-Call source(i.e. [0 -1 10]): [10 100 500]

Sample time[sec]: 0.001

Platform: OSEK

Task stack size(i.e. [512 512 512]): [64 64 64]

OSEK Resource(i.e. 'resA', 'resB'):

JSP Semaphore(i.e. 'semA', 'semB'):

Bluetooth device mode: Slave

Bluetooth device address(i.e. '00', '16', '53', '04', 'F1', 'B3', '00'):

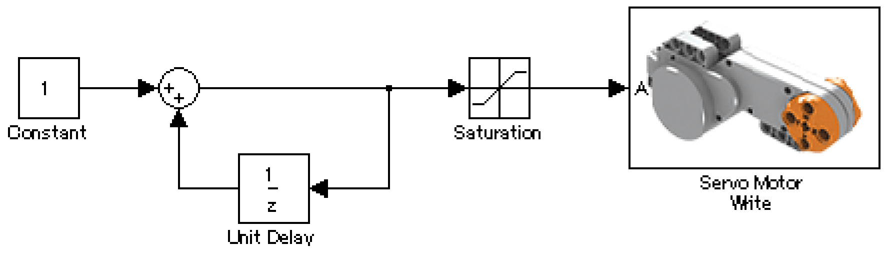
In the Exported Function-Calls Scheduler block, there is Platform parameter to configure platform software on the NXT. Since Embedded Coder Robot NXT v3.16, TOPPERS/OSEK or TOPPERS/JSP can be configured.

In the generated ecrobot_main.c file, there are Tasks definitions and the specified Function-Calls are mapped into the Tasks. Execution timing of each Tasks is configured in the automatically generated RTOS configuration file (TOPPERS/OSEK: ecrobot.oil, TOPPERS/JSP: ecorobot.cfg). From MBD engineer’s point of view, MBD engineers do not need to take care of what kind of RTOS is used, they just need to consider about execution timing (i.e. initialize, periodical execution) on application level.

ECRobot Interface API:

In the lejos_osek/ecrobot directory, there are ECRobot NXT interface C source files (ecrobot_interface.c, ecrobot_interface.h). These C source files include device APIs that access LEJOS NXJ I/O driver. Most of the published ECRobot Interface APIs have corresponding ECRobot NXT device blocks.

For example, Fcn_10ms C function is generated from Fcn_10ms *Function-Call Subsystem* in TestMotor.mdl. In Fcn_10ms *Function-Call Subsystem* there is a *Servo Motor Write* block, and a corresponding published device API is implemented in the Fcn_10ms C function. This device API is implemented by using a custom Simulink Signal Object which is called NXT.Signal.



```
void Fcn_10ms(void)
{
  int8_T rtb_UnitDelay;

  rtDWork.UnitDelay_DSTATE++;
  rtb_UnitDelay = rtDWork.UnitDelay_DSTATE;
  ecrobot_set_motor_speed(NXT_PORT_A, (int8_T)rt_SATURATE(rtb_UnitDelay, -100,
    100));
}
```

Logic of Fcn_10ms *Function-Call Subsystem* and generated C function

ECRobot NXT demo

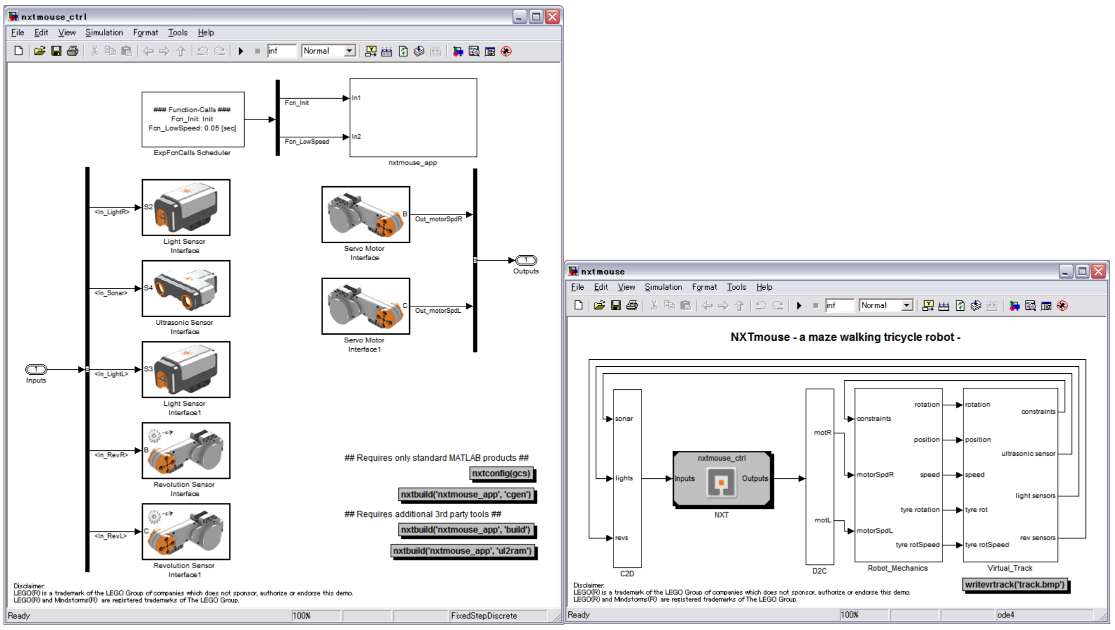
ECRobot NXT provides three application demos: NXTmouse, NXTracer, and NXTway. Each demo includes objectives to evolve the shipped robots. These objectives offer readers good opportunities to experience Model-Based Design using MATLAB and Simulink software products.

NXTmouse

NXTmouse is a maze walking tricycle robot and has one Ultrasonic Sensor, two Light Sensors (not used in the shipped controller model) and two Servo Motors. This robot detects walls by the Ultrasonic Sensor and changes direction. NXTmouse demo files are stored in nxtmouse directory.

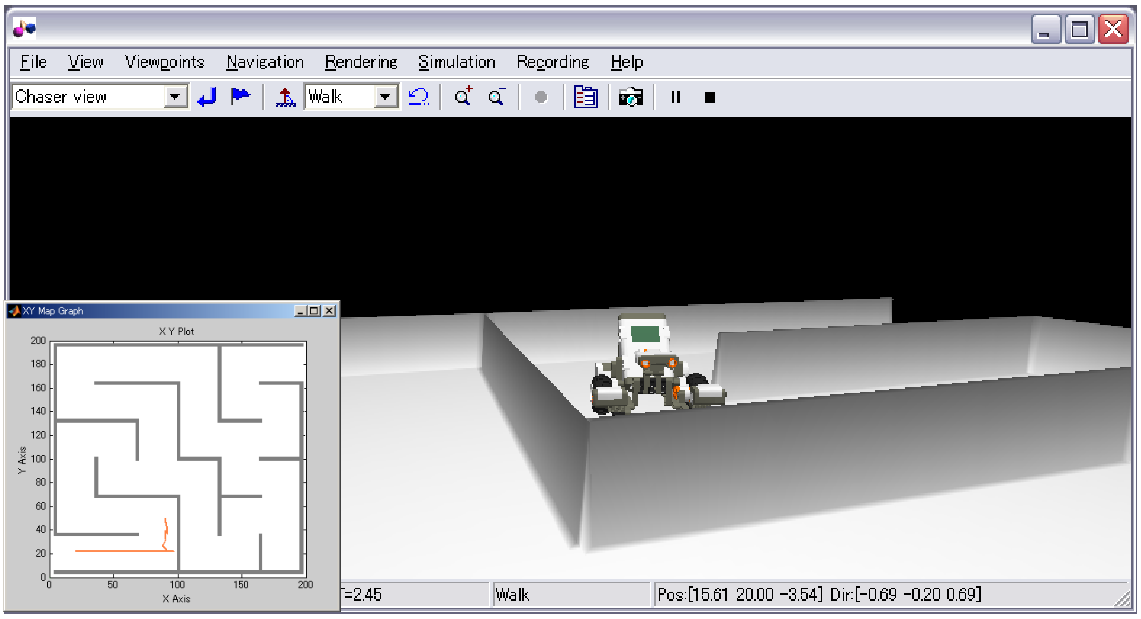
nxtmouse.mdl
nxtmouse_ctrl.mdl
param.m
track.bmp
track.wrl
vrnxtcartrack.wrl

nxtmouse.mdl is the main model file that includes a robot plant model and a virtual track model. The control strategy is modeled in nxtmouse_ctrl.mdl file and is referenced via a *Model* block in nxtmouse.mdl.



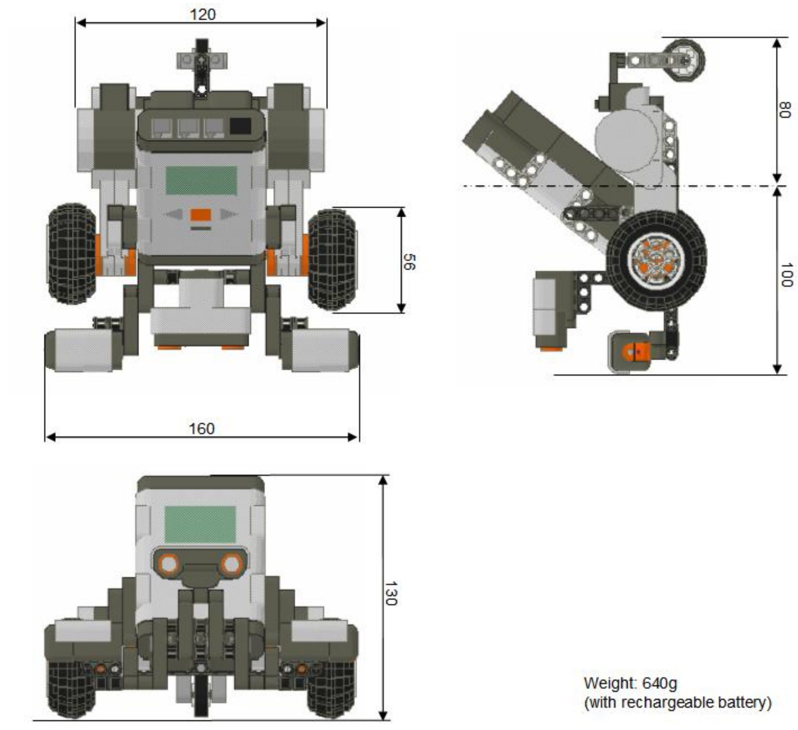
Controller model (nxtmouse_ctrl.mdl) and plant model (nxtmouse.mdl)

During simulation, a Virtual Reality Viewer window and an XY Map Graph figure come up to monitor behavior of the robot in a maze.



XY Map Graph and Virtual Reality Viewer (for NXTmouse)

NXTmouse has the following specifications. A real NXTmouse can be built by using a LEGO Mindstorms NXT Educational Kit (and an additional Light Sensor), but detailed parts construction is different from the model used in the Virtual Reality Viewer.



Specifications of NXTmouse

Model-Based Design Experience with NXTmouse:

Objective: Development of a maze solving robot

Maze solving algorithms have been a popular topic in autonomous robotics. In the web, there are many good examples (i.e. micro mouse: http://micromouse.cannock.ac.uk/index.htm). So this objective allows readers to develop a maze solving robot. Maze solver requires handling matrix data to analyze the maze, so it is also a good application for MATLAB and Simulink. There are several hints that may help readers to develop a maze solving robot.

Hint-1: Make the robot recognize walls in three directions (front/right/left sides of the robot)

NXTmouse has two different types of sensors (one Ultrasonic Sensor and two Light Sensors) to detect the walls. Both of the Ultrasonic Sensor and the Light Sensor can detect the walls without contact; however each sensor has pros and cons, so please note the characteristics of each sensor.

According to the specification of the Ultrasonic Sensor in LEGO's official site: http://mindstorms.lego.com/eng/Overview/Ultrasonic_Sensor.aspx

"The Ultrasonic Sensor measures distance in centimeters and in inches. It is able to measure distances from 0 to 255 centimeters with a precision of +/- 3 cm."

Therefore, we assume the Ultrasonic Sensor has the following specification:

Measurement range: 0 to 255 [cm]

(If the sensor was not ready for measurement, the sensor returns -1)

Measurement width: 160 [mm] (same as the width of the robot)

Measurement period: 50 [msec]

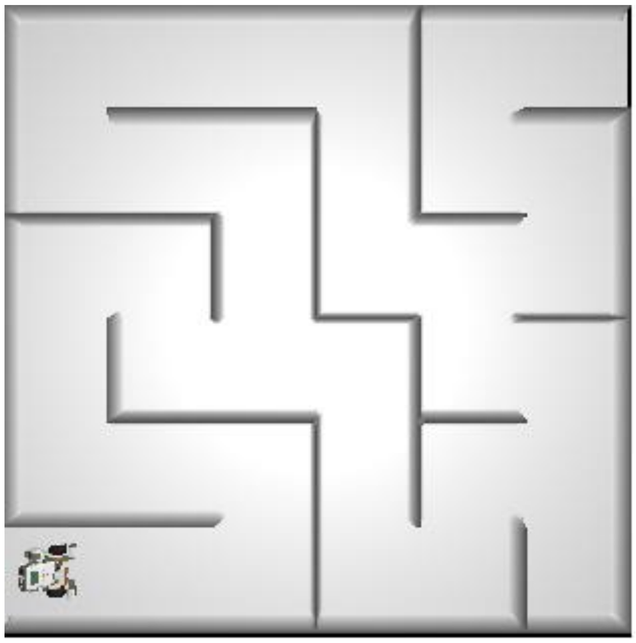
(When ambient temperature is 20 [°C], speed of sound is 343.7 [m/s]; therefore, the elapsed time to measure the maximum distance is calculated as: 2.55*2/343.7 = 0.0148 [sec])

Measurement data by the Ultrasonic Sensor in the real world is not stable due to shape of the obstacles, sensor angle against the obstacles, and temperature…, therefore, noise data is mixed to mimic the real Ultrasonic Sensor during simulation. In the Virtual Reality graphics, the surfaces of the walls are drawn as flat. However, it may cause unstable Ultrasonic Sensor measurement depending on the sensor angle against the wall in reality.

Two Light Sensors are set to detect the walls that are located on the right and left sides of the robot. The Light Sensor value varies depending on the environment (under daylight, room light, color of the walls…) and does not have linear characteristics with respect to the distance to the walls, so new algorithms are required to detect the wall effectively during run-time.

Hint-2: Make the robot recognize position and rotation angle

The robot can recognize the Servo Motor revolutions angle in degrees by using 2 Revolution Sensors, therefore the robot can recognize the position and the rotation angle. To stop at the target position and angle, control algorithms (i.e. PI controller) may be needed. The maze consists of 6 x 6 cells, with the size of a cell being 30 x 30 [cm or pixels]. Initially the robot is located at [22 15] in the maze. This means that the robot is located at the center of the first cell. If the robot can recognize the position and the rotation angle, the robot will move around much more effectively and exit the maze more quickly.

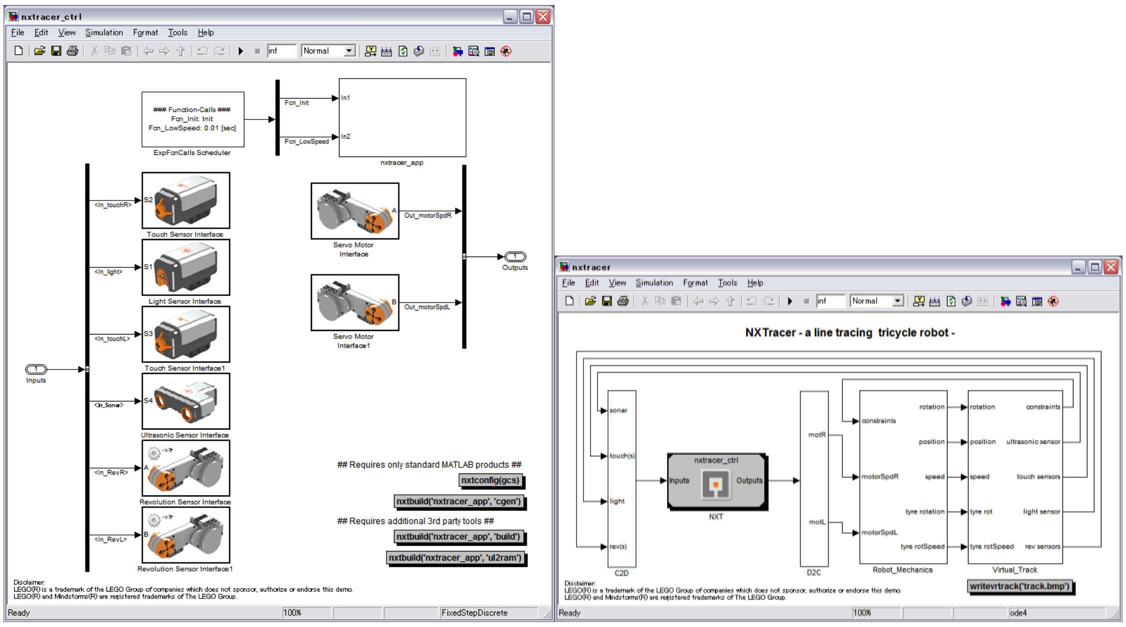

NXTmouse in the maze

NXTracer

NXTracer is a line tracing tricycle robot and has one Light Sensor, two Touch Sensors, one Ultrasonic Sensor and two Servo Motors. The Touch Sensors and the Ultrasonic Sensor are not used in the shipped controller model. This robot detects a black line by the Light Sensor and follows the black line on the track. NXTracer demo files are stored in nxtracer directory.

- nxtracer.mdl
- nxtracer_ctrl.mdl
- param.m
- track.bmp
- track.wrl
- vrnxtcartrack.wrl

nxtracer.mdl is the main model file that includes a robot plant model and virtual track model. The Control strategy is modeled in nxtracer_ctrl.mdl file that is referenced via a *Model* block in nxtracer.mdl.



```
## Requires only standard MATLAB products ##
nxtconfig(gcs)
nxtbuild('nxtracer_app', 'cgen')
## Requires additional 3rd party tools ##
nxtbuild('nxtracer_app', 'build')
nxtbuild('nxtracer_app', 'ul2ram')
```

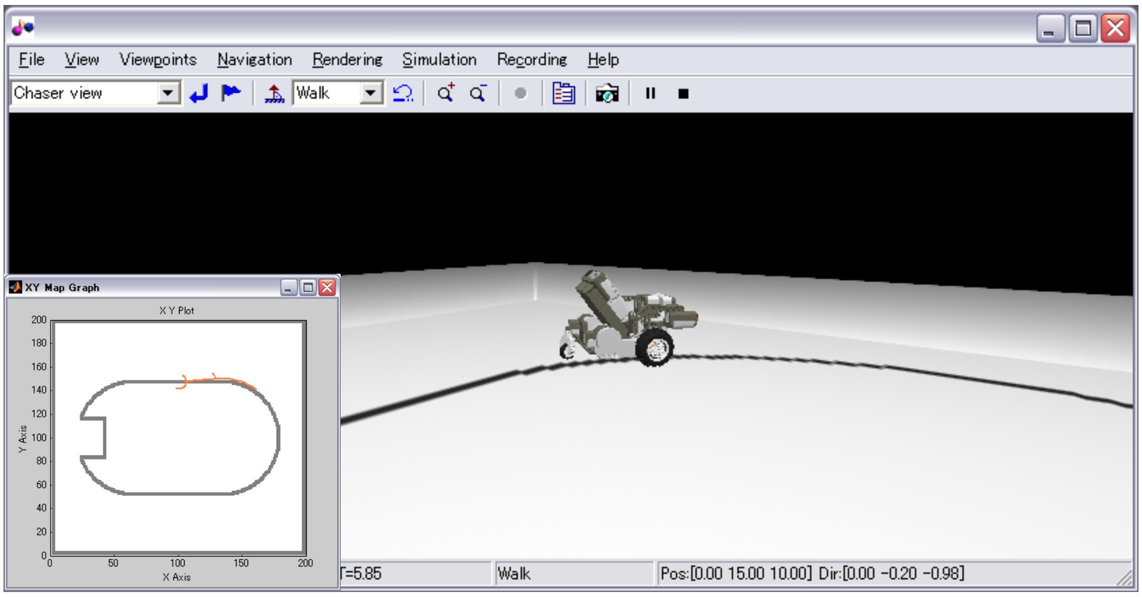
Disclaimer:
LEGO(R) is a trademark of the LEGO Group of companies which does not sponsor, authorize or endorse this demo.
LEGO(R) and Mindstorms(R) are registered trademarks of The LEGO Group.

```
writevrtrack('track.bmp')
```

Disclaimer:
LEGO(R) is a trademark of the LEGO Group of companies which does not sponsor, authorize or endorse this demo.
LEGO(R) and Mindstorms(R) are registered trademarks of The LEGO Group.

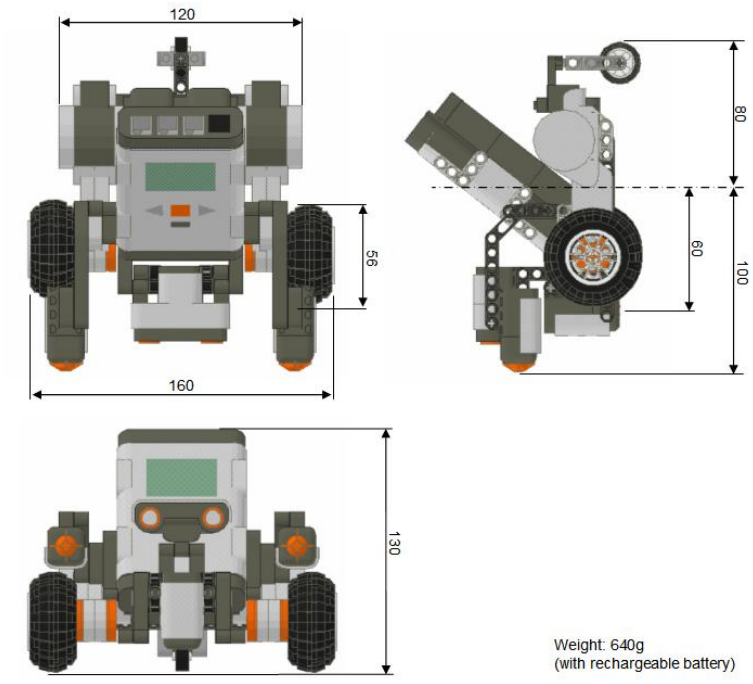
nxtracer_ctrl.mdl and nxtracer.mdl

When the simulation starts, the Virtual Reality Viewer window and XY Map Graph figure come up to monitor the behavior of the robot on virtual track.



XY Map Graph and Virtual Reality Viewer (for NXTracer)

The NXTracer has the following specifications. A real NXTracer can be built using a LEGO Mindstorms NXT Educational Kit, but detailed parts construction is different from the model used in the Virtual Reality Viewer.



Specifications of NXTmouse

Model-Based Design Experience with NXTracer:

Objective: Developm a faster line tracing robot that has self- learning

The shipped NXTracer already has a dynamic threshold level configuration feature to recognize the drawn line in robust way. However, the line tracing algorithm is simple and not very effective. So this objective allows readers to improve the robot to make it trace the line faster. There are several hints that may help readers develop a faster line tracing robot.

Hint-1: The robot may memorize the line layout during the first round

As described in the Model-Based Design Experience with NXTmouse, the robot can recognize the position and the rotation angle. So an application can be developed that allows the robot to memorize the line layout during the first round as a kind of self-learning feature.

Hint-2: The robot may trace the line faster and faster after each round

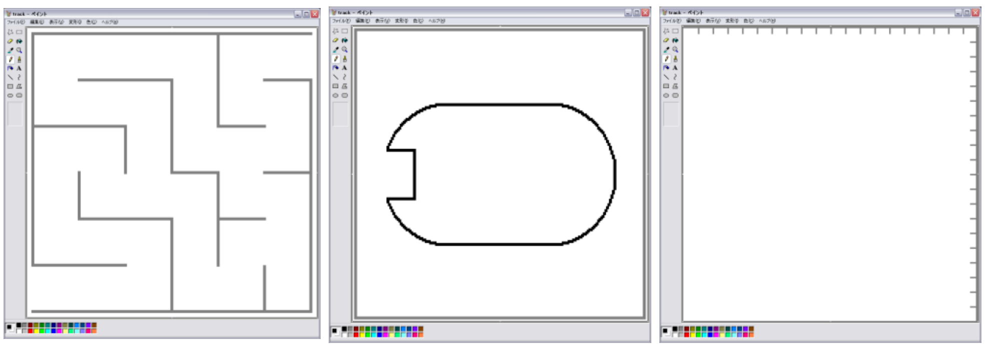
After the first round, the robot may be able to trace the line more effectively based on the self-learned line layout obtained during the first round. The self-learning algorithm may be continuously improved through the subsequent rounds on the line track.

Tips of ECRobot NXT

In this chapter, several tips related to ECRobot NXT are described.

How to design an original virtual track:

The virtual track in the ECRobot NXT demo is kind of a unique test harness for verifying the controller model. ECRobot NXT provides the features that a user can use to easily design their own virtual track and test the robot. In nxtmouse/nxtrace/nxtway directories, there is a bit map file named track.bmp. track.bmp is the design source for the virtual track.



track.bmp files for NXTmouse/NXTracer/NXTway

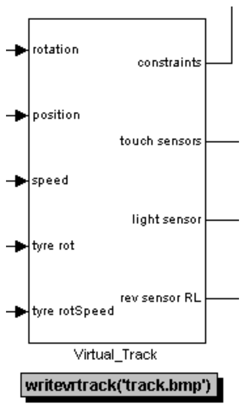
The following rules should be applied for the design of the virtual track:

Size of the virtual track: 200 x 200 [pixels]

Save as monochrome 256 colored bit map file

Black color represents line/ Gray color represents wall/ White color represents floor

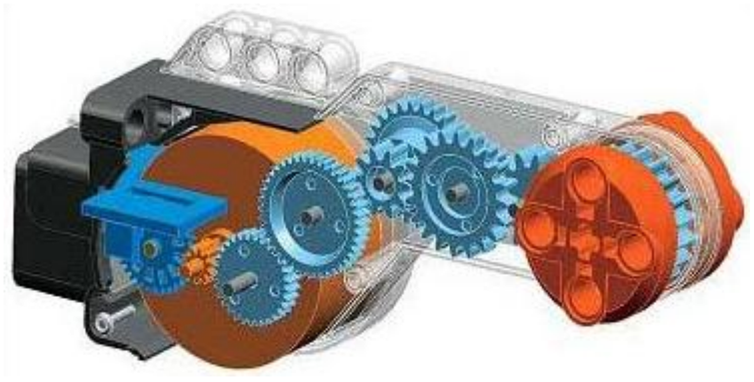
After designing the original virtual track and having saved it, then the MATLAB function that is called 'writevrtrack' automatically generates VRML file (track.wrl) for simulation. In the ECRobot demo models, annotation callbacks are used for updating virtual tracks.



Annotation callback for writevrtrack in nxtmouse.mdl

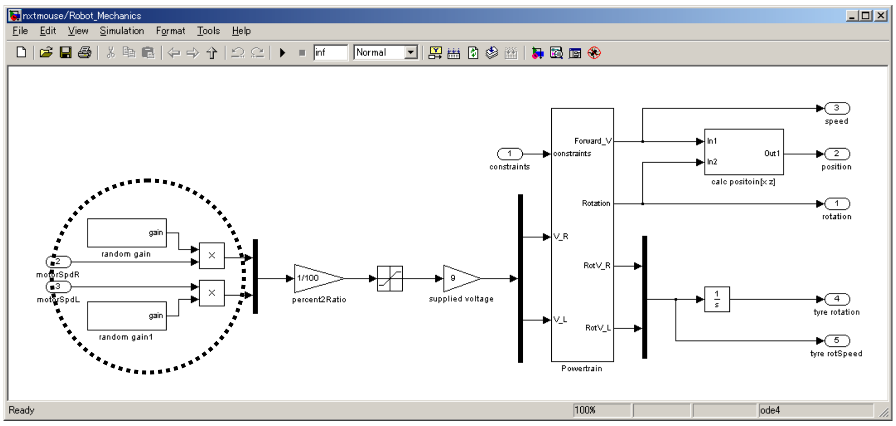
Why the robots can not move smoothly:

When you try the “Model-Based Design Experience” in the ECRobot demo, you may encounter some difficulties to control the robots as you expected (e.g., robot can not move straight, simulation result are different for each simulation…). This behavior is intentionally designed in the robot plant models. In general, control designs assume linearization of plant system, however, in the real world, any simple control system contains non-linear factors that may cause problems in the real world control system. For example, the Servo Motor of LEGO Mindstorms NXT uses many gears to get high torque. These gears are not precisely manufactured and assembled; therefore, this results in certain amount of mechanical backlash. In the Servo Motor plant model, there is a backlash block to mimic this. Effects of backlash can be seen especially in NXTway demo. (The robot does not move like a typical inverted pendulum model)



Transparent picture of a NXT Servo Motor

Additionally, inside of the Robot_Mechanics Subsystem in the demo models, Servo Motor controls for the right and left wheels have random error at every simulation. Therefore, the robot can not move straight. This is a common issue in the real hardware due to mechanical errors of motors, gears, wheels, and floor surfaces.



Random error gain for Servo Motors in Robot_Mechanics Subsystem (nxtmouse.mdl)

Preserved signal identifiers for NXT device blocks:

NXT device blocks such as *Light Sensor Interface…* are masked Subsystems and inside of the Subsystems, *Data Store Read/Write* blocks are used. The identifiers of the *Data Store Read/Write* blocks are automatically defined based on the input port configuration when updating model. Therefore, the following preserved signal identifiers should not be used to avoid naming conflicts:

NXT device	Preserved identifier
Light Sensor	LightSensor + S1/S2/S3/S4
Touch Sensor	TouchSensor + S1/S2/S3/S4
Sound Sensor	SoundSensor + S1/S2/S3/S4
Ultrasonic Sensor	UltrasonicSensor + S1/S2/S3/S4
Revolution Sensor	RevolutionSensor + A/B/C
Servo Motor	ServoMotor + A/B/C
Battery Voltage	BatteryVoltage
System Clock	SystemClock

How to upload an ECRobot NXT application program to the FLASH memory on the NXT:

In the chapter of ECRobot NXT target deployment, the ECRobot NXT application program was uploaded to the RAM of AT91SAM7S256 on the NXT. The uploaded application program on the RAM is gone once the NXT is turned off. ECRobot NXT also supports program upload to the FLASH memory of AT91SAM7S256 on the NXT. However, it was not described in ECRobot NXT target deployment section due to several reasons. The most important of these is that the lifecycle of the FLASH memory on the NXT is limited and uploading the program to the FLASH may cause a serious damage to the NXT, therefore, an optional usage of ‘nxtbuild’ M function exists but is not published.

How to design LEGO Mindstorms NXT 3-D models for Virtual Reality Toolbox:

Around LEGO and Mindstorms, many interesting tools and projects exist. Most of them are created by some voluntary people or open-source projects. Thanks to the great help of the below tools, LEGO Mindstorms NXT 3-D virtual reality models in ECRobot NXT could be created.

LDraw:

LDraw is a huge 3-D LEGO brick library.

http://www.ldraw.org/

LDraw library for Mindstorms NXT:

Original creator of NXTway is also the author of LDraw librariy for NXT. The library has not been certified as an official LDraw library yet, however, the quality of the library is excellent.

http://philohome.com/

MLCAD:

MLCAD is a LEGO brick specific 3-D CAD and LDraw Brick library can be used as parts.

http://www.lm-software.com/mlcad/

AC3D CAD:

AC3D CAD is a 3-D CAD and it enables converting an LDraw file to be a VRML file. AC3D CAD is proprietary software, but has a free trial version available for 14 days.

http://www.ac3d.org/

About pictures used in ECRobot NXT:

Pictures used in ECRobot NXT were quoted from the following URLs.

Pictures of LEGO Mindstorms NXT devices could be downloaded from LEGO’s official site.

http://mindstorms.lego.com/download/default.aspx

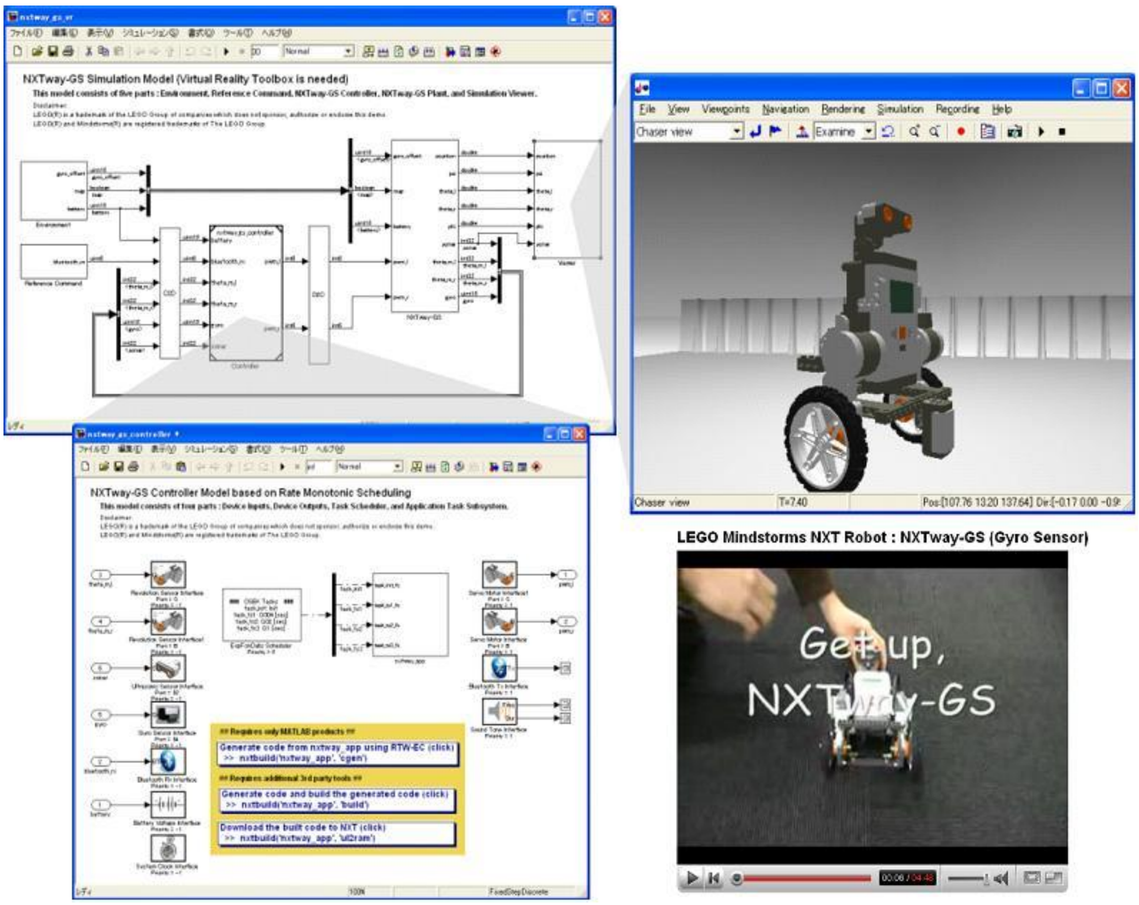
The picture of *Bluetooth Receive Message Interface/Read* blocks was quoted from the following URL: http://www.deviantart.com/deviation/23759806/

Application demos based on Embedded Coder Robot NXT:

In the MathWorks File Exchange, there is several application demos based on Embedded Coder Robot NXT.

NXTway-GS: http://www.mathworks.com/matlabcentral/fileexchange/19147

NXTway-GS is a self-balancing two wheeled robot with a HiTechnic Gyro Sensor.

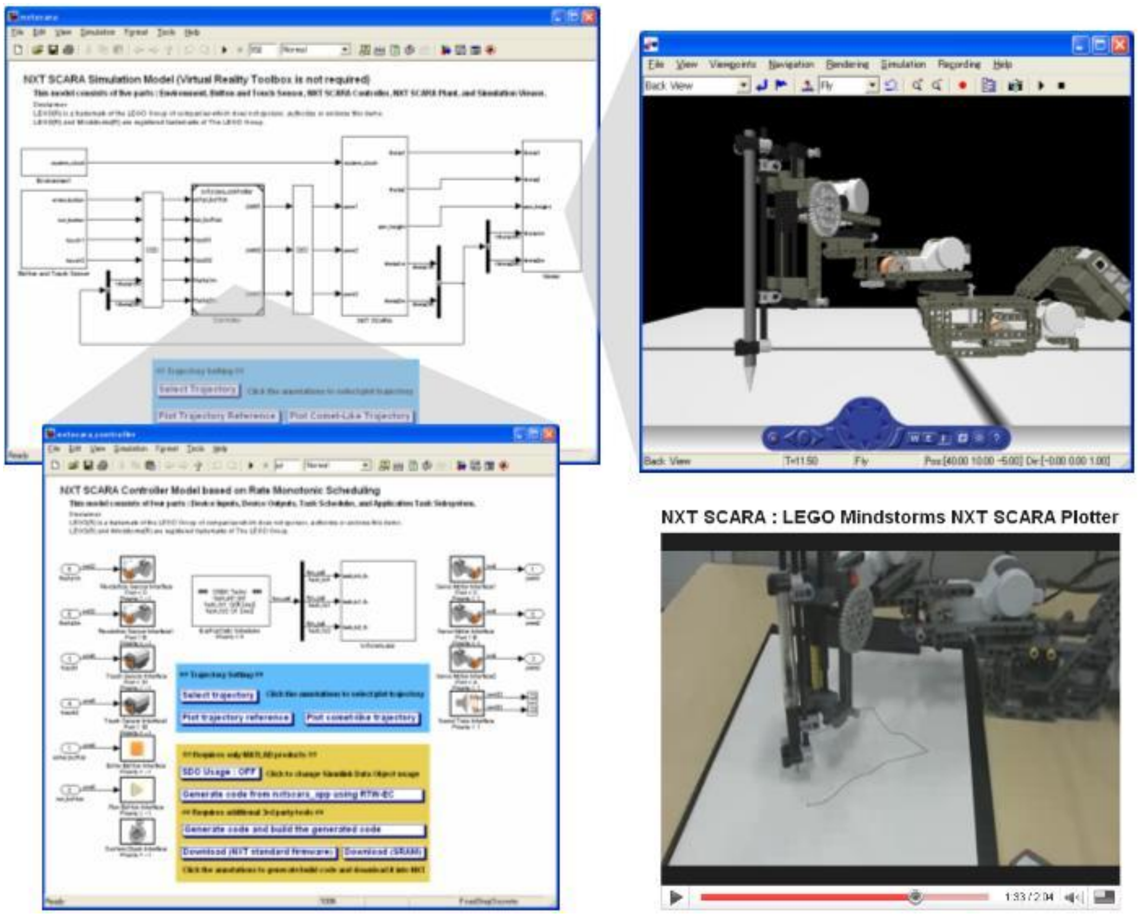

NXTway-GS Simulation Model (Virtual Reality Toolbox is needed)

NXTway-GS Controller Model based on Rate Monotonic Scheduling

LEGO Mindstorms NXT Robot : NXTway-GS (Gyro Sensor)

NXT SCARA: http://www.mathworks.com/matlabcentral/fileexchange/22126

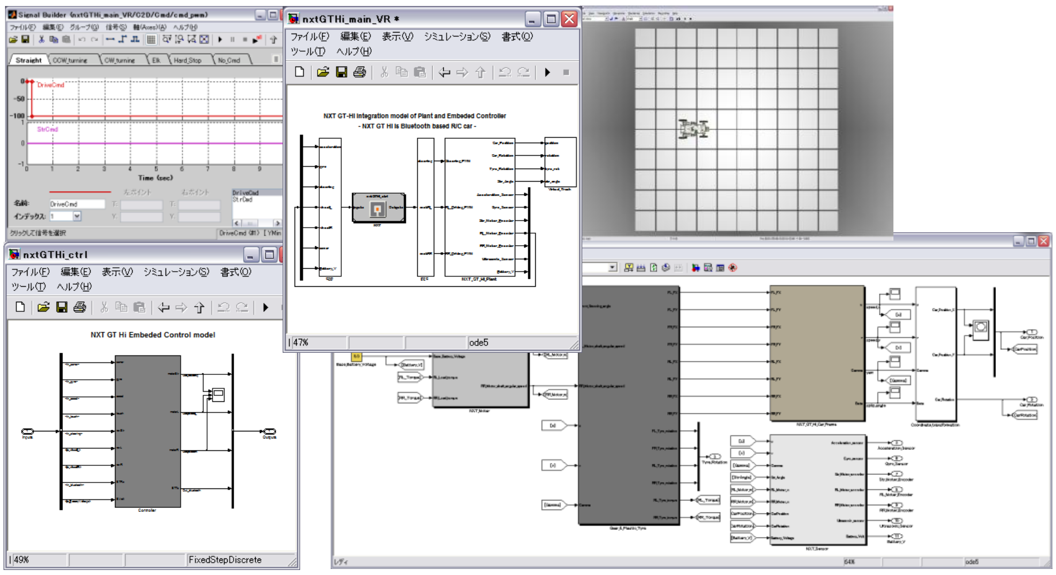
NXT SCARA is a 2-link SCARA plotter. SCARA stands for Selective Compliance Assembly Robot Arm.



NXT SCARA Simulation Model (Virtual Reality Toolbox is not required)
NXT SCARA Controller Model based on Rate Monotonic Scheduling
Select Trajectory
Plot Trajectory Reference
Plot Comet-Like Trajectory
Select trajectory
Plot trajectory reference
Plot comet-like trajectory
SDO Usage : OFF
Generate code and build the generated code
Download (NXT standard firmware)
Download (SRAM)
Back View
NXT SCARA : LEGO Mindstorms NXT SCARA Plotter
1:33 / 2:04

NXT GT-Hi: http://www.mathworks.com/matlabcentral/fileexchange/22411

NXT GT-Hi is a Bluetooth R/C car with ESC (Electronic Stability Control) using a HiTechnic Acceleration Sensor and a HiTechnic Gyro Sensor. This demo illustrates Model-Based Design for Electrical Automobile Chassis Control in controller design and plant design.



nxtGTHi_main_VR
NXT GT-HI Integration model of Plant and Embeded Controller
- NXT GT HI is Bluetooth based R/C car -
nxtGTHi_ctrl
NXT GT Hi Embeded Control model
Straight
CCW_turning
CW_turning
Elk
Hard_Stop
No_Cmd
DriveCmd
StrCmd
Time (sec)
ode5
FixedStepDiscrete