Dymola and Modelica_EmbeddedSystems in Teaching-Experiences from a Project Course

Åkesson, Johan; Elmqvist, Hilding; Nordström, Ulf

2009

Dymola and Modelica_EmbeddedSystems in Teaching – Experiences from a Project Course

Johan Åkesson[ab] Ulf Nordström[c] Hilding Elmqvist[c]
[a]Department of Automatic Control, Lund University, Sweden
[b]Modelon AB, Lund, Sweden
[c]Dassault Systèmes, Lund, Sweden (Dynasim)
Johan.Akesson@control.lth.se Ulf.Nordstrom@3ds.com Hilding.Elmqvist@3ds.com

Abstract

This contribution presents experiences from a master level project course where the Modelica-based tool Dymola, supporting embedded control system design, has been used. In a recent initiative, the Modelica language has been enhanced to support modeling of embedded systems and code generation targeted at micro processors.[1] The new specification is supported by Dymola and enables wide range of design tasks to be performed in a unified framework. Such tasks include software in the loop simulation to test controller code in simulation, hardware in the loop simulation, and final deployment on the target. In the context of teaching, the new features of Modelica/Dymola enable universities to offer a realistic environment providing students with hands on experiences from model-based control system development.

Keywords: Modelica; Dymola; Embedded Control Systems; Teaching

1 Introduction

Much effort is devoted to studies of analysis and synthesis methods in engineering programs oriented towards systems and control. Often, the course material is mainly of theoretical nature, sometimes complemented with laboratory sessions. To further strengthen the practical skills of the students, a project course, "Projects in Automatic Control" is offered by the Department of Automatic Control, Lund University. The main themes of the course are practical application of theoretical skills acquired in previous courses and working in teams.

This contribution describes two projects that were part of the course of 2009, where Dymola and Modelica_EmbeddedSystems was used to develop control systems for Segway-type robots, Figure 1, built using the Lego Mindstorms NXT platform.

The paper is outlined as follows. In Section 2, an overview is given over different approaches to teaching embedded systems and control, and in Section 3 the Project in Automatic Control course is described. Section 4 and 5 describes, respectively, the Modelica_EmbeddedSystems library and the LEGO_Mindstorms library. In Section 6, some common usage scenarios are discussed and in Section 7 the fixed point code generation module of Dymola is outlined. The paper ends with a review of the course results in Section 8 and a summary in Section 9.

Figure 1. Lego Segway

[1] This effort has been performed within the EUROSYSLIB project.

2 Background

Teaching of embedded control systems requires insights into different disciplines, including mathematical modeling, control system design, and computer science. In the latter case, real-time systems are particularly important. Embedded control systems are distinguished by the complex interplay between the behaviors of the controller to be implemented, typically designed in continuous time, the discretization method used in order to obtain a discrete time approximation of the controller, and the properties of the execution environment. In order to analyze the closed loop behavior of the controlled system, all three aspects need to be attended to.

Teaching of embedded systems can be approached in several ways, using different levels of abstraction. At the lowest level, control systems are encoded in C, or even assembly. The control system is then typically run without an operating system and periodic processes, or tasks, are mapped onto timer interrupts. Using this approach, the modeling and control systems design is typically done prior to the encoding phase, using different tools and methodologies. It is also common that controllers need to be translated into fixed-point arithmetics. From a pedagogical perspective, this method has distinct advantages and disadvantages. Coding an embedded control system in a low level language, perhaps including manual fixed-point conversion, does indeed promote understanding of the tasks involved. Also, mapping of periodic tasks onto hardware interrupts further strengthens the student's understanding of the methods involved. On the other hand, modeling and control system design is disjoint from the actual encoding and execution of the control system. Debugging is often further complicated by limited means to log signals in the embedded control application.

At the next level of abstraction, a high-level language, relative to C or assembly, can be used for implementation. For example, Java offers suitable abstractions for creating periodic tasks, e.g., threads and synchronization. Also, there are platforms providing Java support, including Lego Mindstorms NXT. Modeling and control system design, proceeds, however, as with the previous approach, and is typically disjoint from the actual implementation. Never the less, this approach captures important aspects of embedded control system, such as multi-threaded applications and the consequences thereof.

In order to promote joint modeling, control systems design and implementation, tools like Real-Time Workshop for Matlab/Simulink are available. A similar tool is Scilab/Scicos. Such tools offer strong support for block-based modeling, which is well suited for development of control systems. Real-Time Workshop may then translate the block-oriented graphical Simulink model into executable C code, which in turn can be compiled and downloaded to the target processor. Using the simulation capabilities of Simulink, the control system can be simulated together with a model of a physical plant in order to assess the closed loop behavior prior to deployment. There is also a toolbox for fixed-point arithmetics available for Simulink as well as a freely available toolbox for simulation of the temporal behavior of embedded kernels and computer networks, TrueTime [1].

The approach taken in this paper is similar to that of Matlab/Simulink and Real-Time Workshop. The simulation software Dymola is used for physical modeling as well as development of the control system. Modelica is used as modeling and implementation language. As compared to Matlab/Simulink, Modelica offers stronger support for physical modeling, and supports advanced modeling concepts such as object orientation, equations, and acausal connections between components. An additional advantage of Modelica is that the code is available to the user, which adds to the transparency of the method. Using novel features of Dymola and additions to the Modelica language explored in the Modelica_Embedded library, it is possible to generate C code, automatically translated to fixed-point if desired, corresponding to the control system. The generated C code may then be either compiled and downloaded to the target or compiled and linked with a simulation executable. The latter case enables detailed study, in simulation, of the closed loop behavior of the system prior to deployment.

The method of automatic code generation from a high-level description is a novel addition to the course portfolio of Automatic Control. Joint control system design and implementation on embedded platforms has been a long-standing theme of the department, both in research and in teaching, but so far, C and Java (and previously also Modula-2) has been used as implementation languages. Dymola and Modelica therefore offer an appealing complement for providing the students with experiences from a different environment.

3 Project in Automatic Control

The Department of Automatic Control has a long tradition of laboratory work in control education. Laboratory sessions are included in all theoretically oriented courses and some courses also offer small projects. In order to further strengthen the practical

and experimental skills of the students, a dedicated project course is offered to master's level students. The course gives 7.5 ECTS units and is categorized as advanced level. The syllabus of the course changes each year depending on the number of students and the availability of interesting projects, usually with connection to research or industrial applications. The projects are typically set up so that the students need to go through several steps in the design cycle, including mathematical modelling, parameter identification based on measurement data, control design, control system implementation, user interaction and testing. The examination of the course consists of weekly meetings with a teacher, a written report and an oral presentation.

The students in the course have in most cases taken several control courses covering topics such as linear and non-linear control system design, multivariable control, sampled systems and real-time systems. For a list of courses offered by the Department of Automatic Control, see [2]. In the project course, the students need to apply their knowledge from previous courses in order to solve a larger design problem in a team consisting of three to five students. More often than not, the course helps the students to put their theoretical knowledge into a practical perspective where sensors and actuation, unit conversions, and limited computing resources play important roles.

For the course as of spring 2009, the Lego Mindstorms NXT [6] platform was selected as a basis for the course projects. The platform features several possibilities for sensors and actuations, also from third party manufacturers, and the embedded micro processor can be programmed in several ways using e.g., C/C++, NXC or Java. Out of 22 students in total, two teams of five students in each were selected to perform projects where the Dymola software was used for modeling, control design and embedded code generation.

3.1 Project infrastructure support

In order to emphasize and support the collaborative character of the projects, a version control repository and a web-based tool for project planning were made available for each project group. As for version control, Subversion [3] was used and Trac [4] was used as project planning platform. The objective of introducing these tools in the course was to add an additional element of industrial realism to the projects. Also, the students were required to prepare each weekly meeting with their teacher by updating the Trac site to reflect the current status of the project.

Throughout the projects, the students had access to a lab where the Lego sets and computers for development were available.

3.2 Tutorials

All students in the course were offered a tutorial on how to use Trac and Subversion, since few had any experience of such tools. The students participating in the Dymola projects were offered additional tutorial lectures in order to get started with the course work. A basic tutorial on how to operate the Lego Mindstorms NXT hardware was offered in the beginning of the course, with the objective of introducing the students to basic operation such as reading from sensors, compilation of programs, and downloading and running programs. Since the students lacked previous experience with Modelica, an introductory lecture was given. The tutorial covered basic Modelica features, including textual and graphical modeling, as well as an introduction to Dymola. Finally, a lecture on advanced Modelica and multibody modeling was offered, covering also the animation features of Dymola. The final lecture was given by personel from Dynasim, whereas the three first were given by personel from the Department of Automatic Control.

The initial tutorial lectures given early in the course provided the students with sufficient information to get started with Modelica and Dymola. However, some additional support in the form of informal tutorials in front of the computer was also needed, especially in order for the students to learn how to use the new advanced features related to Modelica_EmbeddedSystems and code generation.

3.3 Project task

The task for the students to solve was to construct a Segway-type robot, see Figure 1, and to develop a model-based stabilizing control scheme using Modelica and Dymola. Dynamic modeling of the robot was done using the multi-body library in Dymola. While modeling of the mechanical parts is fairly straightforward, the Lego servos pose a challenge. In order to obtain a good model for these, identification experiments need to be performed. This was made possible by the data-logging feature of Dymola; a small Modelica test program was downloaded and the resulting signals were logged back to Dymola over the BlueTooth communication link. Having constructed a dynamic model, a linearized approximation can be derived and exported from Dymola. Both groups opted to use a state feedback controller designed using Control Systems Toolbox in Matlab.

Given the controller, sensors and sensor processing needs to be considered. The Lego servos have built-in angular measurements, and in addition, one rate gyro and one accelerometer were available to each group. In the final step, the control system was designed using blocks from the Modelica Standard Library, and the details of the embedded platform were set up. As a parallel task, animation of the robot was set up in Dymola.

4 Modelica_EmbeddedSystems

The Modelica_EmbeddedSystems library [5] was used to set up the models for use with embedded systems. Using components from the library, target configuration records and communication points are inserted in the models containing properties of the target system and computational tasks.

4.1 Communication points

One of the key components of the library is the `CommunicateReal` block. It is used to set up communication with external I/O ports of the target system or to model the interface.

In the LEGO_Mindstorms Modelica library, described in a later section, I/O communications blocks were implemented such that they fit in the framework set by the `CommunicateReal` block. The design allows for straight forward use of the GUI (parameter dialog) to enable access to the external I/O blocks by a simple pull-down menu, depicted in Figure 2.

A user could thus implement new I/O blocks that would end up in the same dialog for selection.

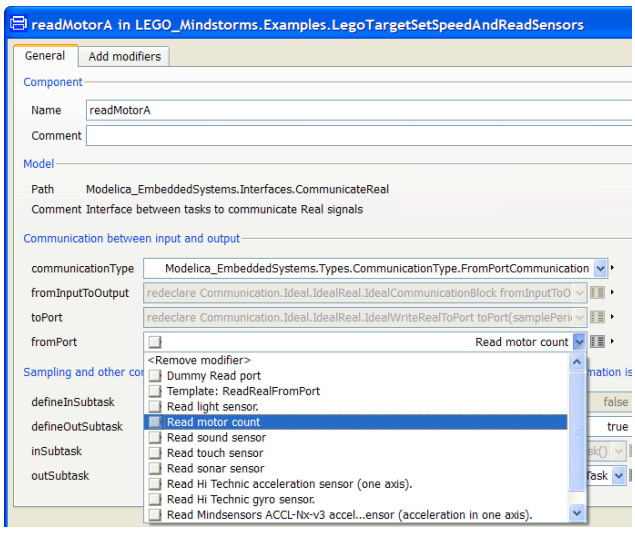

Figure 2. Lego I/O blocks in the CommunicateReal parameter dialog

4.2 Configuration records

Another key component of the library is the configuration record that is used to configure the models with respect to the target platform and task partitioning. A configuration record is a user configurable nested record (record containing records). Depending on the problem, the record could contain one or more targets, tasks and subtask. A simple example is depicted below, Figure 3, where there is just one target, one task and one subtask. The additional block with a Bluetooth icon is from the LEGO_Mindstorms library and is used to select virtual COM ports for Bluetooth communication.

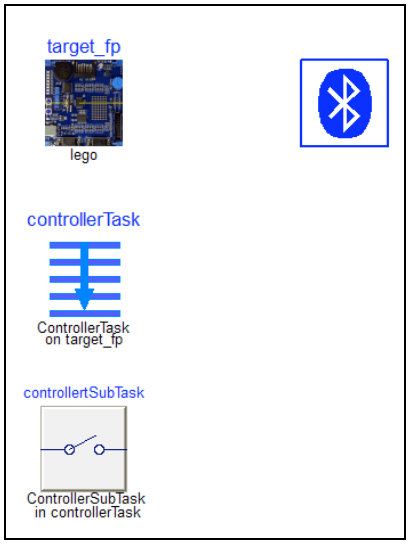

Figure 3. Example of configuration record

5 Dymola Lego Mindstorms API

The Lego Mindstorms NXT device can run under several operating systems. For this project course the nxtOSEK [7] open source real-time operating system was selected due to its openness and well documented C API for sensors, motors and other devices (including some third party sensors). It provides a C programming environment using a GCC tool chain and comes with an extensive set of samples that help the students to get a throughout understanding of the platform and interaction with sensors and actuators. Based on these samples a main program was developed as a wrapper to the Dymola generated model code and variable declarations. The main program handles initialization and termination of sensors and Bluetooth communication (invoking the hook routines described in the nxtOSEK C API Reference [7]) and mapping of system time to fixed-point time while the Dymola generated code that is included performs all the computations.

5.1 LEGO_Mindstorms library

The LEGO_Mindstroms library, Figure 4, has been developed for education and implements communication blocks and a small set of examples and additional components.

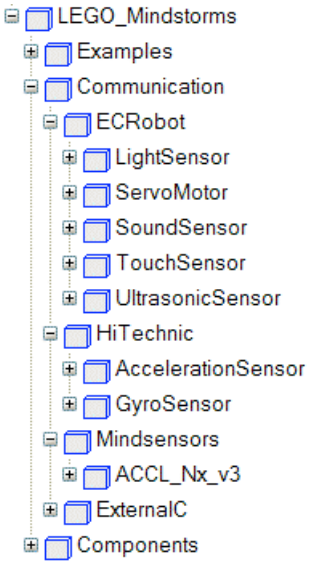

Figure 4. LEGO _Mindstorms library

The communication blocks can be used in models to map Modelica variables to low-level C functions on the Lego Mindstorms NXT device. An example would be to map the output of a speed controller to the servo motors and to feed the same controller with data from the ultrasonic sensor for obstacle detection.

The communication blocks provide the mapping to selected function of the API for interaction with sensors and actuators. Also included are some third party sensors from HiTechnic [8] and Mindsensors [9]. The design extends from the Modelica_EmbeddedSystems architecture in such a way that the various blocks can be conveniently selected from a drop down list in the parameter dialog of the `CommunicateReal` block. This enables the students to easily configure the interaction with sensors and actuators in their models.

In addition to the standard sensors of the NXT device the students had access to third party sensors, some included in the C API for nxtOSEK and some not included. Currently the following sensors and actuators are supported:

- ECRobot
  - Light sensor
  - Servo sensor
  - Sound sensor
  - Touch sensor
  - Ultrasonic sensor
- HiTechnic
  - Acceleration sensor (NAC1040)
  - Gyro sensor (NGY1044)
- Mindsensors
  - Acceleration sensor (ACCL-Nx-v3)

The `ECRobot` sub package contains the interface blocks for the standard Lego Mindstorms I/O devises. The blocks contain a mapping to the corresponding nxtOSEK C API functions and utilises the Modelica external function concept. Below is a simple example using the Touch Sensor. As can be seen in Figure 5 the Touch Sensor API takes an U8 (unsigned 8-bit integer) as argument and returns an U8.

Touch Sensor API
U8 ecrobot_get_touch_sensor(U8 *port_id*)

Figure 5. Touch Sensor API (in nxtOSEK)

The corresponding function in Modelica that maps to this is depicted below in Figure 6.

```
function ecrobot_get_touch_sensor
  input Integer port_id(min=0, max=3);
  input Real Time;
  output Real signal;
external "C" signal =
  ecrobot_get_touch_sensor(port_id);
end ecrobot_get_touch_sensor;
```

Figure 6. Modelica function mapping

A block that can be used in the `CommunicateReal` block of Modelica_EmbeddedSystems is constructed by extending from the appropriate base class and calling the mapping function, see Figure 7.

```
block ecrobot_get_touch_sensor
  extends ...PartialReadRealFromPort(
    minValue=0,
    maxValue=1);
  parameter Integer port_id(
    min=0,
    max=3) = 0;
equation
  y = ecrobot_get_touch_sensor(port_id, time);
end ecrobot_get_touch_sensor;
```

Figure 7. Block calling the mapping function (paths have been shortened to fit in the picture)

Note that in this first implementation the return type of the Modelica function is `Real` even though the C function returns an integer (U8). This was done to simplify usage for the students but should be re-

designed for a final version of the library. The type conversions are handled by Dymola and the C compiler automatically. All of the sensors and actuators in this sub package are standard Lego sensors and they are all included in the nxtOSEK C API.

The `HiTechnic` sub package contains the interface blocks to two third party sensors from HiTechnic, a gyro sensor and an acceleration sensor. Both sensors are available in the C API which make the Modelica implementation straight forward with one exception. The API for the acceleration sensor, Figure 8 below, differs in that it takes an integer array to store the results in.

HiTechnic Acceleration Sensor API
void ecrobot_get_accel_sensor(U8 *port_id*, S16 *buf*[3])

Figure 8. Acceleration sensor API (in nxtOSEK)

For this sensor a special wrapper has been written in C to extract only one of the elements since the `CommunicateReal` block in Modelica_EmbeddedSystems currently only supports scalars. In Modelica you then choose with a parameter which axis to read from. The drawback is that you need three blocks to read all axes compared to just one function call if using the API as it is.

The `Mindsensors` sub package contains an additional third party sensor: the ACCL-Nx-v3 acceleration sensor from Mindsensors. This sensor can be used either as a tilt sensor or to measure acceleration in any of the x-, y- or z-axis. It is more sensitive than the acceleration sensor from HiTechnic and returns the measured acceleration in units of milli-G, where G is the gravitational unit. This sensor was not represented in the nxtOSEK C API so API functions had to be written manually and supplied to the students.

The `Components.BlueTooth` sub package contains a block that is used to set up Bluetooth communication from the Lego NXT to dymosim (the standard Dymola simulator). It is designed and tested only for Windows and uses virtual COM ports for Bluetooth communication.

5.2 Main program

The main program is based on the sample programs from the nxtOSEK distribution and acts as a wrapper to the model code generated by Dymola. It also handles mapping of system clock to fixed-point time (currently hard coded to 10 fractional bits) and provides some wrappers and API functions for third party sensors. Below in Figure 9 the main program is outlined in pseudo code.

```
/* OSEK include files */
/* ... */
#include "target_port.h"

/* OSEK declarations */
/* ... */

/* Include fixedpoint variable declarations */
#include "declarations.c"
/* Include API to sensors from Mindsensors */
#include "mindsensors.c"

/* OSEK hooks */
/* ...*/

/* Task executed every 10msec */
{
    /* map system time to fixedpoint time */
    timeFP = ...

    /* include fixedpoint equations */
    #include "equations.c"

    /* display time in seconds*/
    ...
    display_string("TIME:");
    display_int(timeFP0_0/1024, 0);
    display_update();
    ...
}
```

Figure 9. Main program pseudo code

The students could easily modify the program for more advanced use of the display, reconfiguring, adding or removing sensors etc. It is also possible to access all the fixed-point variables for online debugging etc. using their fixed-point representation (integer values used to store the signals). For more convenient debugging the variables can be sent to Dymola using the Bluetooth connection.

6 Dymola and code generation

6.1 Configuring the model for fixed-point

The Lego Mindstorms NXT device does not have hardware support for floating-point arithmetic's and in order to avoid using computationally heavy and memory consuming floating point math libraries, fixed-point code is preferred. In order to use the fixed-point code generation capabilities of Dymola the model must be annotated with additional information. This is done using the *min* and *max* attributes to specify the range of a variable and the relative resolution with newly introduced experimental annotation, *annotation(mapping(resolution=0.001))*. In Figure 10 an example of setting the resolution for two variables of a component is shown. Note that this experimental annotation can be set as a modifier. The information is then used during translation to

allocate integer and fractional bits for the fixed-point variables.

```
Modelica.Blocks.Sources.Ramp ramp(
  height(min=0, max=100) = 100
   annotation (mapping(resolution=0.001)),
  y(min=0, max=100)
   annotation (mapping(resolution=0.01)));
```

Figure 10. Fixed-point annotated component

6.2 Code output

When configured for external code generation, Dymola generates two files, namely declaration.c and equations.c with fixed-point code to be included in the main program. The code is well documented and includes the original Modelica code and the assigned fixed-point format in Q-notation. An example of declatation.c can be seen in Figure 11, note the full Modelica declaration from where the variable originates and the Q-notation indicating the number of integer- and fractional bits.

```
/* output Modelica.Blocks.Interfaces.RealOutput ramp.y(
   min = 0.0, max = 100.0) annotation(mapping(
   resolution = 0.01)); */
int ramp_yFP = 0;   /* Q[7, 0] */

/* parameter Modelica.SIunits.Time ramp.duration(
   min = 0.0, max = 50.0) = 10
   annotation(mapping(resolution = 0.001)); */
int ramp_durationFP = 320;  /* Q[6, 5] */

/* parameter Real ramp.height(min = 0.0,
   max = 100.0) = 100  annotation(mapping(
   resolution = 0.001)); */
int ramp_heightFP = 1600;   /* Q[7, 4] */
```

Figure 11. Declaration of fixed-point variables

All computations are collected in the file equations.c. Just as for the declarations, the eqation file includes the original Modelica eqation as a comment for traceability. Below in Figure 12 is an example of generated fixed-point code for a ramp-function.

```
/* ramp.y = ramp.offset+(if time < ramp.startTime
  then 0 else (if time < ramp.startTime+ramp.duration
  then (time-ramp.startTime)*ramp.height/
  ramp.duration else ramp.height)); */
ramp_yFP = (((ramp_offsetFP << 4) + ((((timeFP0_0 << 4)
  < ramp_startTimeFP) ? (0 << 4) : ((((timeFP0_0 << 4)
  < (ramp_startTimeFP + (ramp_durationFP << 9))) ?
  ((((((timeFP0_0 << 4) - ramp_startTimeFP) / 32) *
  (ramp_heightFP / 32)) / ramp_durationFP) << 1) :
  ramp_heightFP)))))) / 16;
```

Figure 12. Fixed-point code for an equation

6.3 Bluetooth data logging

It can be a very hard task to debug code in embedded systems. To make debugging easier, Dymola generates code (for the Lego Mindstorms NXT target) to send the internal variables of the target in fixed-point representation to Dymola using Bluetooth. The received values are automatically re-scaled to their corresponding Real (SIunit) values. This enables real-time plotting of the internal variables of the target as well as storing the data.

7 Scenarios

In this course, Dymola and Modelica_EmbeddedSystems were used in several of the scenarios the students were faced with. Typical such scenarios are plant modelling and controller design including development and tuning using Model-in-the-Loop (MIL) simulation and Software-in-the-Loop (SIL) simulation. Also for final production code generation and deployment Dymola was used (in combination with other tools; Cygwin, GCC to name the most important).

7.1 Model in the Loop simulation

MIL simulations were performed to test the control strategy with the student's model of the robot. These simulations are typically done with continuous time (ideal) controllers without taking into account effects of sample, communication delays, fixed-point arithmetic's etc. It serves as a foundation, to validate that the control strategy is feasible.

To set up the model for MIL simulation one uses communication blocks from Modelica_EmbeddedSystems. These blocks are inserted between different parts of the model, for example controller and plant, to define the border between different tasks.

7.2 Software in the Loop simulation

The next logical step after MIL simulation is SIL simulation where more detail is included in the controller (non-ideal), in this course, the effects of fixed-point arithmetic's in particular.

The model is prepared for SIL simulation by using the Modelica extends mechanism (inheritance) together with a modifier with another configuration record to indicate that the target of the control task does not have a floating-point arithmetic unit. This means that the original model is not changed which is a great benefit in larger projects. The reconfiguring described above is a very simple modification of the model assuming that the model was correctly partitioned for MIL simulation and that the configuration records was set up containing all necessary details. Below, in Figure 13, is an example plot showing the effects of fixed-point arithmetic's on a

PI controller with low resolution driving a simple drive train.

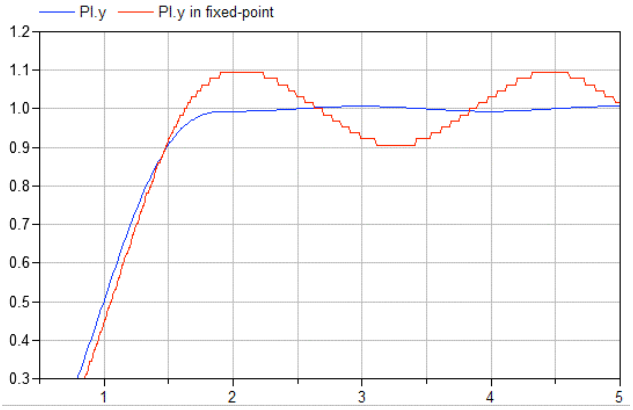

Figure 13. Plot of control signal for a system in closed loop in floating-point vs. low resolution fixed-point

7.3 Production code

Dymola was used for this final stage to generate fixed-point C code for the model equations. This code combined with the main program described in an earlier section can be downloaded to the Lego Mindstorms NXT device and run.

As for the case above, SIL simulation, the reconfiguration is very simple to do. Again the Modelica extends mechanism is used together with a modifier to change the configuration record. This new configuration record specifies the target to be a Lego Mindstroms NXT unit without a floating-point arithmetic support. Dymola could then recognize this target and generate code to fit with the main program. Code is also generated for dymosim which is running in parallel with the Lego controller to collect variable data and convert them for logging, plotting, animation and debugging using Bluetooth, more on this in Section 1. Below in Figure 14 an animation of the Lego robot can be seen.

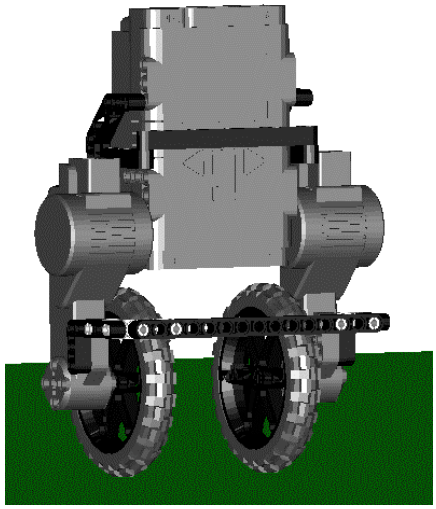

Figure 14. Dymola animation of Lego segway

8 Student results and experiences

Both project groups working with Dymola reached their goal of designing a stabilizing controller based on their multi-body models. The approach was very similar in both cases, and followed largely the steps outlined in Section 3.3. However, the students ran into numerous problems on their way, which needed attention.

While the students quickly constructed mechanical models for their robots, the servos posed a challenge, both in terms of unknown dynamics and in terms of how to connect such a model once available to the mechanical parts. Much time was devoted to solve this problem. The diagram layer for a mechanical model constructed by one of the student groups is shown in Figure 15.

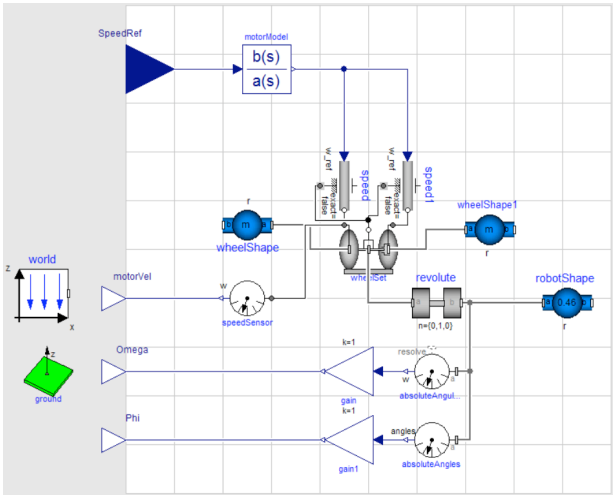

Figure 15. A Modelica model for a Segway robot constructed by one of the student groups.

Once a complete model for the robot had been constructed, linearizations were computed to use in the control design. In initial attempts, the linearized models were of high order, in some cases due to high-order servo models derived by means of black-box systems identification. While the high-order models did not impair the possibility to design controllers, problems arose in the controller implementation phase where the availability of measurement signals was limited. In order to solve this problem, simpler models were derived, in particular by simplifying the servo models, and increased attention was given to the available sensors. In the end, the complexity of the controllers was matched to the available measurement signals, but without compromising the model-based approach.

The students experienced some problems with specification of the mapping of controllers onto hardware and the fixed-point code generation in Dymola. Most of the problems were a result of the beta-status of these features at the time of the course. The problems where, however, quickly solved and did not significantly hinder the students in their work.

The reactions from the students were overly positive: *"great to apply knowledge from previous courses in practice"* and *"appreciated the opportunity to work with an industrially relevant tool like Dymola"* were some of the comments. While the students in some cases were a bit disappointed by implementing only stabilization but not remote control the general opinion seems to be that they learnt a lot. Not the least to put their theoretical knowledge into a practical perspective.

9 Summary and conclusions

In this paper, we have reported an application of Modelica in education. Modelica, Dymola, and in particular Modelica_EmbeddedSystems have been used in a master's level course; Project in Automatic Control. The experiences are very encouraging and the tools and methods used in the course of 2009 will be used also in the next year's course.

References

[1] Cervin, A., Henriksson, D., Lincoln, B., Eker, J., Årzén, K-E.: How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime. *IEEE Control Systems Magazine,* **23:3** pp. 16-30, June 2003.

[2] Courses at Automatic Control: http://www.control.lth.se/education/civing.html

[3] Pilato, C., Collins-Sussman, B., Fitzpatrick, B. (2008): *Version Control with Subversion.* O'Reilly Media, Inc.

[4] Trac webpage: http://trac.edgewall.org/

[5] Elmqvist, H., Otter, M., H.,Henriksson, D., Thiele, B.,Mattson, S.E.: Modelica for Embedded Systems, Modelica Conference 2009.

[6] Lego Mindstorms webpage: http://mindstorms.lego.com/

[7] NxtOSEK webpage: http://lejos-osek.sourceforge.net/

[8] HiTechnic webpage: http://www.hitechnic.com/

[9] Mindsensors webpage: http://www.mindsensors.com/