A DECADE OF RAPID SOFTWARE DEVELOPMENT FOR CONTROL SYSTEM EXPERIMENTS.. LESSONS LEARNED

Amund Skavhaug (amund@itk.ntnu.no), Trygve Lunheim (trygvelu@itk.ntnu.no), Bjørnar Vik (bjoernar.vik@itk.ntnu.no) and Thor I. Fossen (tif@itk.ntnu.no)

Department of Engineering Cybernetics, Norwegian University of Science and Technology

Abstract: In this paper we show how the development of software for control system experiments has changed in our department the last 10 years. Different solutions for rapid development and deployment are presented, with our evaluation of cost vs benefits of using these when learning time, acquisition cost and required coding are considered. *Copyright © 2002 IFAC.*

Keywords: Control systems, software tools, laboratory education, operating systems, simulation, computer aided control system design.

1 INTRODUCTION

For students to gain insight in a problem, the physical manifestation in a real-life hands-on experiment is generally invaluable. For designing new control algorithms or new models of a system, at least a scaled down real physical model is, if not neccesary, at least highly desirable. For some simulations, the actual hardware utilized is needed for the results to be of any value. Unfortunately, resources to create experiments for students, both in available man-hours and certain expertise, are often lacking in universities today. For the industry available time for development has also decreased in the recent years. In other words: We would like to develop experiments with less need for expertise outside our own domain, and in shorter time.

Our department has close to 50 years of experience in designing and implementing software and hardware for advanced control system experiments. In the last decade we have seen new opportunities for developing systems faster, with greatly improved flexibility for changing parameters, and for doing brand new experiments on the same equipment with a simple exchange of SW. This has mostly been fueled by the PC revolution. Now low-cost hardware has computational capabilites only available in expensive special purpose systems just a few years ago. Commercial Off-The-Shelf (COTS) software packages give us graphical tools for expressing both logic and continuous algorithms without the need for expertise in either numerical methods or software engineering. Today, many systems can use automatic code generation to take care of the tedious task of writing the code.

Early in this decade use of the PC in real-time control was demonstrated by Schmid (1992). Baracos et.al. (2001) describe the use of COTS systems and new technologies in order to facilitate HIL simulations, as well as presenting OPAL-RT's RT-Lab system. Rabbath et.al (2000) demonstrated OPAL-RT's use in event-driven systems. Yao, Z. et.al. (2000) promotes the use of RT-Linux (Yodaiken, 1997) together with Matlab/Simulink and briefly compares Windows NT and QNX for similar use. Grega et.al. (1999), on a similar note present their solution for using Real-Time-Workshop together with Windows.

In this paper we present new tools, runtime systems and methods that have gained popularity, we give pros and cons learned from numerous projects and experiments done at our department, and present examples of these.

We conclude in principle that, used correctly, some of the newer systems and methods can be very useful, but that one must know their limitations, and be prepared to spend time to learn their intricacies.

1.1 Modern tools.

Below we give a brief description of some of the tools we have been using, and that are in common use elsewhere.

Matlab, Simulink, Stateflow and Real-Time Workshop

MathWorks Matlab seems to have become the de facto standard for control system engineering and education. Although expensive for the industry, the prices are kept low for universities. MATrix LABoratory, initially a tool for manipulating matrices without having to be an expert in numerical methods, has gradually gained more value with its various "toolboxes" with special solvers and tools for different problem areas, and add-ons that use the MATLAB engine in the background. The most well-known of these in the control engineering community is Simulink, a graphical tool where the user can create simulation models by picking function blocks and connecting defined inputs and outputs on these. This is a data-flow oriented approach, where every block might be another diagram, or just the interface to some code, written in e.g. C. Today one might compile Matlab/Simulink-code directly to a native executable for higher speed processing, or even use the functionallity of the Matlab/Simulink system from other software through the use of either DDE or COM on host OSes that supports this. A less known tool is StateFlow, which complements Simulink with its handling of event-based systems. This is done graphically with the use of Harel State Charts, quite similar to the UML State Charts. Together Simulink and StateFlow can be used to simulate systems with complex behaviour.

In order to be of use for more than the simulation and design of a control system, which was the initial use, MathWorks has created Real-Time Workshop (RTW). This is an add-on product that can generate code to be deployed on target systems directly from the Simulink/StateFlow diagrams.

Because of the periodic nature of most discrete control algorithms, implementing a standalone executive to run these is not a very difficult task. Without any additional third-party tools, RTW supports WindRivers' VxWorks and a "general RTOS[1]" target, as well as the possibility to run the control SW directly on top of MS-DOS. RTW uses a scripting/description language to facilitate the users' (or a third-party supplier's) adjustments of how the code is generated through its Target Language Compiler. Basically one might have a kind of *cyclic executive*, since due to the application and how the code is made, the usual misgivings for a "big while loop" are less valid. A very simple system such as this can in fact have inherently less jitter than the more advanced versions utilizing several tasks in a RTOS to achieve the same. Of course, when code is targeting a RTOS there are less restrictions on sample rates, and in some cases the load can even be shared by several CPUs. MathWorks has its own RTOS for PCs, "xPC Target", which can be bought as an additional product to RTW. The efficiency and applicability of RTW and "xPC Target" as well as some other vendors' solutions for simple target systems are discussed in Teng, F.C. (2000).

LabView from National Instruments is another dataflow oriented graphical tool where the user connects function-blocks of different complexity to build the application. These blocks are also hierarchical, and are often in the form of a "virtual instrument", e.g. a scope, a frequency counter, a voltage meter etc. It runs on Apple Macintosh, Sun Solaris and Windows NT systems. Since LabView runs as a user mode program on top of these non-real-time operating systems, it has generally been viewed as an interface generator for data-acquisition with relatively high-end equipment that takes care of buffering in HW[2]. Since its earliest incarnations, LabView has evolved from a system for controlling GPIB-equipment and simple data-acquisition, to a system that can be used for complex factory automation when used with add-ons to its full extent, but we will not discuss this further here.

At our department we use LabView in order to show students a tool commonly used in the industry. In experiments we only use this SW for design of userinterfaces. This is due to the high price of Real-time LabView, but more importantly because of our heavy use of MATLAB and its related products[3].

DSPACE has been around since the mid-eighties. Their main product line has been Texas Instruments DSP based processor cards, and I/O cards to go with these, and with SW for directly linking up with Matlab/Simulink for automatic generation, compilation and downloading of code that runs on this DSP-platform. This execution is uninterrupted by the host PC, where variables and measurment values can be shown

1. Using functionality that is commonly available in many RTOS, it is at least easy to port to the actual system.
2. Oscilloscopes, frequency analyzers, as well as PC I/O expansion cards with built in processing capabilites.
3. National Instruments has a product for mathematical analysis, HiQ, that can be used in conjunction with LabView.

in near real-time. Since the control SW basically runs in a loop on dedicated CPUs there are no issues with timeliness unless overload occurs.

2 EXPERIMENTS EVALUATED

Of the various experiments conducted over the last decade we have chosen the following experiments to demonstrate the different solutions:

- Mega-torque
- ABB-robot
- Cybership I (Model ship)
- Passivity based control of asynchronous AC-motor
- Cybership II (Model ship)
- GPS/INS laboratory
- 3DF Helicopter lab and Magnetic levitation lab
- CyberEagle

2.1 Mega-torque and ABB-robot

First we present two laboratories that were implemented using purely manual methods.

Mega-torque: In the early nineties we connected 2 large DC motors with a rubber-band loop made more flexible with a spring as part of it. Here one motor gave variable load, while the other should keep constant position/speed, depending on the experiment. This was designed as an educational lab for graduate students in control theory. Unfortunately the runtime system chosen was MS-DOS, where the control-system was supposed to be implemented as interrupt routines attached to the timer interrupt. The time used by the students was nearly exclusively spent on the intricacy of creating interrupt routines, how to use floating point operations inside these routines in a safe way, and with details of reprogramming the 8253 timer in a PC. In other words, the learning goals were not reached, and the teaching assistants spent far too much time on developing source-code stubs needed. We had already used VxWorks in several larger projects, but at the time, this was deemed too expensive, too difficult to use with the requirement of using UNIX workstations, and overkill for this project. This lab is no longer in use, and is mentioned here as a prime example of how *not* to do it.

ABB-robot: This robot had a control system running VxWorks and was used for several experiments related to graduate/Ph.D studies in the late eighties/early nineties. This was a cross-development system where a Solaris workstation was used for development, and a Motorola 68030-based controller mounted in a VME-rack was used as target. Development tools were, compared to todays standards, very crude; mainly Emacs for editing, GCC for compiling and GDB for debugging. Although we don't use this robot now, there is no obstacle keeping us from using Matlab and Simulink/RTW together with a new VxWorks version and updated BSPs, for a more modern approach.

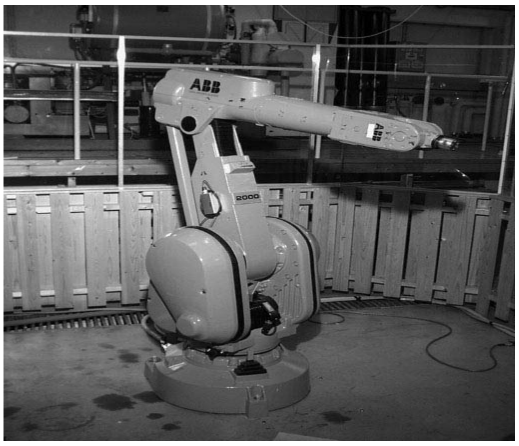


ABB-robot running VxWorks

2.2 Cybership I / Passivity based control of AC motor

Both of these projects used *DSPACE* as a main part of the solution.

Cybership I: This project was started around 1995, and was completed in its final form in 1998. The experimental setup consists of a basin 6x10m^2 with wind and wave generators, a model-ship with 4 thrusters locally controlled by a Motorola 68HC11, an infrared camera system consisting of 3 cameras connected to a black-box computer which computes the position of the ship, a PC running Windows NT and Matlab/Simulink, and finally the DSPACE system which runs the control algorithms, resulting in set-points transferred by radio-link to the ship.

The use of Simulink has been a great success for student education in non-linear ship control systems, already when the model was run by cable-control. Everything was built from scratch, and the effort to create both the HW and the SW used in the ship was in the area of 500 hours. Late in the project, it was discovered that the intended mechanism for handling the radio-transmission and control based on this was fatally flawed.

The DSPACE system did what was expected, but offered some problems regarding information on how to handle the position-information from the camera system delivered on a RS232 serial line.

Cybership II, PC based control system inside

Passivity based control of asynchronous AC-motor

Around 1993-1995 common PCs were not yet powerful enough to run the control algorithms for this kind of experiment. Here we used Matlab/Simulink/RTW with the DSPACE system. In-house our challenge was to make the electronics interfacing the DSPACE I/O cards with low noise levels in presence of a difficult RFI/EMC environment, as well as taking care of user safety in presence of high voltages. Although the DSPACE system was costly in acquisition, no time was spent in traditional hand-coding for this system, and the DSPs were fully capable of coping with the short periods required, even when running computationally intensive algorithms. For this experiment the choice of SW solutions was the right at the time, whereas for the HW we were left with little choice. This shows that for some advanced experiments off-the-shelf equipment may not be available, so one must either have the required expertise in-house, or be prepared to pay for external development.

2.3 Cybership II

Equipped with the knowledge of what went wrong, and what was most time-consuming, the next version of our ship was built in the period 1999 to 2001 with as much COTS equipment as possible, both for HW and SW. A Pentium class PC/104 embedded computer with commercial electrical driver cards for both propeller speed and the step motors for rudders. The radio communication is now based on commercial[1] 2.4GHz tranceivers interfaced directly to ordinary Ethernet cards. We chose QNX ver 4.25 as RTOS with OPAL-RT together with Matlab/Simulink.

Our use of ordinary Ethernet with TCP/IP for communication between the computer equipment made it possible to easily integrate a new position system based on hardware with drivers and run-time system only available for Windows 98, something that obviously required its own dedicated PC. The drivers for the I/O cards used vere not available, nor supported in any way by OPAL-RT. Therefore a consulting company was hired to write these.

1. Not the 802.11b standard though, we chose equipment from Breeze.com, due to their longer range, which might be useful for experiments done outdoors.

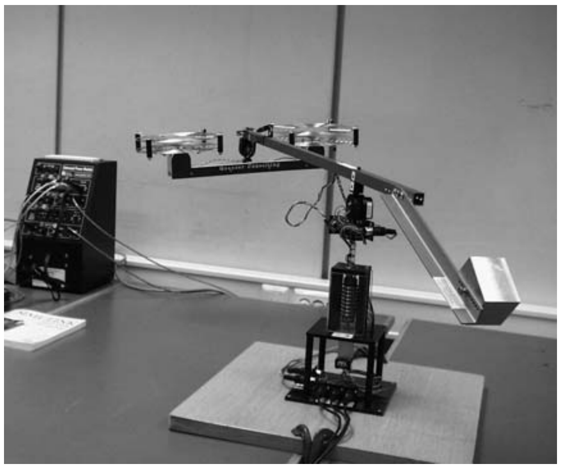
The GPS/INS laboratory

An interesting observation in this project, is that after some time of running different experiments, the main body of code has been hand-coded in C, and referenced in the Simulink system as external code. This is also the case for the main body of purely mathematical functionality.

2.4 GPS/INS laboratory

In order to study algorithms for integrating GPS information with INS measurements we have built a laboratory consisting of a Chrysler Voyager with several programmable GPS receivers connected to industrially hardened PCs with RS232 serial communication, and an advanced Litton IMU fiber-optic gyro with additional accelerometers connected to one of the PCs by RS485 serial-line SDLC communication.

Apart from the user interface, which is built with LabView, we used a software solution almost identical to the one used in the Cybership II. Here the very time-critical handling of the SDLC protocol required extensive knowledge of both the inner workings of QNX regarding memory, DMA and interrupt handling, and the ability to extract enough information from several sources about the actual circuits used in the SDLC PC-I/O card in order to write working code. In this case there was a QNX driver already written, and not just a simple in/out interface code for Simulink, which was the case for the Cybership II. Although the SW should have been the same, a strange error in serial communication with the GPS receivers happened. Seemingly random bytes disappeared from the information packages received. This was later shown to be caused by the third-party tool OPAL-RT having switched communication mode from raw to "terminal editing",



Helicopter laboratory

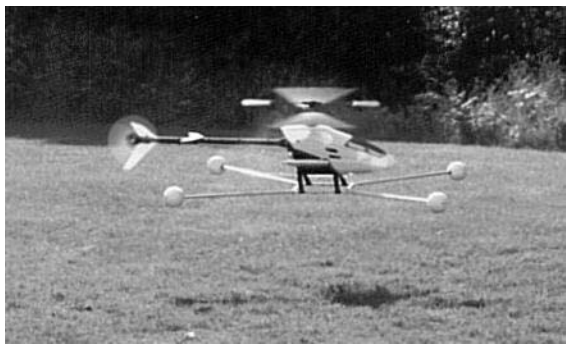
without any of us ever knowing when that happened. The same phenomenon happened for this lab as for the Cybership II. After some time, most of the code has been written manually. Still, the functionality of easily replacable parts of the system through the Simulink interface, makes us believe that this was still the right choice of software system.

2.5 3DF helicopter lab and Magnetic levitation lab

Both of these laboratories are purely for educational purposes, and had to be duplicated to make them available for several students at the same time,

3DF Helicopter: This is an experimental setup bought from Quanser consulting. The code is generated from Matlab/Simulink/RTW/WinCon-RTW addition, and runs on Windows NT with the RTX extension from VenturCom, which is used in our real-time programming class for graduate students. This combination was also supported by Quanser consulting. Apart from some problems in some of the special power supplies delivered, something which required our inhouse expertise in electronics to find and correct, these experiments have been used heavily over 2 years now, with the students focusing on linear control theory without having to worry about implementation questions. Although the price for each experimental setup might seem steep, approx. 10.000 $US plus a powerful PC, we have 5 identical versions of this lab.We believe this costed far less, and took less time to complete, than what would have been the case if we instead had our own electrical and mechanical workshop to implement something similar.

Magnetic levitation: Based on our experiences with the 3DF helicopter lab, we decided to use this solution for teaching non-linear control theory. We bought 3 kits to do magnetic levitation experiments, and used the exact same SW configuration. This has nearly been as un-eventful as expected. What we did learn, however, is the necessity of version control, especially when



CyberEagle, a computer controlled free flight helicopter.

dealing with third-party vendors. A different Windows NT Service Pack resulted in the need of a different version of RTX. The timing requirements for such an experiment is presented in (Verde C. and Fragaso, J.L. 1997).

2.6 CyberEagle

In this project being done now in 2001-2002, a helicopter with a gasoline fueled engine with the ability to lift a payload of approx. 2 kg has been equipped with low-cost accelerometers and some Atmel AVR microcontrollers that collect measurements from these, and a PC/104 card connected to a IEEE 802.11b wireless LAN that runs VxWorks[1] as well as the necessary equipment to control the hobby-type of servos found on model aircrafts such as this. The control system is developed on a portable computer running Windows NT and Simulink. RTW downloads the algorithms over the wireless LAN to the helicopter, and receives and displays monitored variables in the Simulink user interface.

Obviously a lot of traditional development both in HW and SW has been required for the microcontrollers and the instrumentation on the model. In addition such a system really requires that one learns quite a lot about the target OS used (VxWorks), even if this in principle is supported directly by RTW.

3 DISCUSSION

For some of our experiments we see that even if we use Simulink as the main tool, most of the code is written in a traditional way, and called as S-functions in the Simulink models. This is not true just for interface code, but also for mathematically oriented functionallity where complexity is high. In a way this seems to contradict the assumed benefits of using Simulink and RTW. Still we realize that most of the professors and

1. Where the VxWorks kernel is in fact downloaded with the radiolink.

Ph.D-students in control theory, at least at our department, do not have the necessary computer engineering and programming skills to implement their systems without the guidance of the codification framework given, and the system configuration help achieved by using RTW. A lot of work is done in simple diagrams created only in Simulink, and for changing parameters and interconnections, Simulink gives us a much faster turn-around time, as well as the obvious advantage of being less prone to errors.

In general, we have made the observation that systems and tools for rapid development of control systems SW accelerates development compared to traditional translation of models by hand-coding, but only if nothing goes wrong. When the unexpected occurs, we have in several occations found documentation lacking, both in detail and accuracy, and that the required knowledge of both the inner workings of all products as well as their interaction can become a major roadblock.

Also, many of the tools needed are from third-party vendors. In some systems we have ended up with as much as 5 different vendors of software products. This provides some interesting problems regarding both localization of errors and problems caused by the products requiring an exact version of each of the other products after an upgrade.

4 CONCLUSIONS

Our experiences show us that modern tools for rapid development of SW running on COTS PC-compatible hardware is a definite must-have for doing experimental work on the actual physical setup. However, for a one-off project the effort is probably no less than with traditional coding. And one must be aware of the need for writing low-level drivers and interface code in either case. Learning the idiosyncrasies of all the involved products takes both time and dedication. Flexibility of the former solution means that this is still what we advise, and use ourselves for new projects. If the same technological choices are made for several experiments large savings in development time is shown in the following experiments. For student textbook experiments our advice is to buy as complete systems as possible, e.g. both HW and SW setup included, and only expose the Simulink interface to the students, as this gives excellent opportunities to focus on the theoretical aspects.

Further work: Although we will continue to use our "buy" solution for student labs where possible, and our Simulink/OPAL-RT/QNX choice for advanced experiments, we are now also starting to use the Simulink/VxWorks combination in projects where industry is involved. This is essentially because of the demand caused by the large market share VxWorks holds for control system applications in Norwegian industry.

5 REFERENCES

Baracos, P., G. Murere, C.A. Rabbath and W. Jin (2001). Enabling PC-Based HIL Simulation for Automotive Applications. *IEEE IEMDC2000*, pp. 721 - 729.

Grega, W., K. Kolek and A. Turnau (1999). Rapid Prototyping Environment for Real-Time Control Education. *Real-Time Systems Education III, 1998. Proc.*, pp. 85-92

Quanser Consulting, 102 George Street, Hamilton, Ontario CANADA L8P 1E2, Tel: (905)570-1906, http://www.quanser.com

Rabbath, C.A., M. Abdoune and J. Belanger (2000). Effective Real-time simulations of event-based systems, *Proc. of the 2000 Winter Simulation conference*. pp. 232-238.

Schmid, Chr. (1992) Real-Time Control with CADACS-PC, in: M. Janshidi (ed.), *Recent Advances in Computer-Aided Control Systems Engineering*, Elsevier, pp. 337-355.

Teng, F.C. (2000) Real-time control using Matlab Simulink, *2000 IEEE Int. Conf. on Systems, Man, and Cybernetics*, **Vol. 4** , 2000 pp. 2697-2702.

Verde C. and J.L. Fragaso (1997). Implementation of controllers for a magnetic systems of laboratory, *Proc. of the 4th IFAC Symposium on Advances in control Education*, 1997, pp. 375-378.

Yao, Z., N.P. Costescu, S.P. Nagarkatti and D.M. Dawson (2000). Real-Time Linux Target: A MATLAB-Based Graphical Control Environment. *Proc. of the 2000 IEEE Int. Symp. on Computer-Aided Control System Design,* pp. 173-178.

Yodaiken, V. and M. Barabanov (1997). Real-time Linux, *Linux Journal*, Feb. 1997.