Rapid Prototyping of a Stand-Alone Embedded Controller for a Stabilized Motion Platform

Ruben de Schipper, Prof. Ka C. Cheok
and Dr. G. E. Smid
Electrical & Systems Eng. Dept.
Oakland University
Rochester, Michigan 48309

Prof. J. B. Klaassens
Information Technology and Systems Dept.
Delft University of Technology
2600 GA, Delft
The Netherlands

Abstract

Microprocessors and peripheral devices are getting cheaper; new software makes programming microcontrollers easier; extensive computational and I/O capabilities, such as floating point calculations and encoded PWM, are now inclusive features of new generations of embedded microcontrollers. These are reasons to choose a microprocessor to implement control systems, rather than implementing by means of analog electronics or other techniques.

This paper describes a case study of how a self-levelling stabilization control system was prototyped and controlled by an analog electronics-controller, a PC-in-the-loop controller and an embedded Hitachi SH2 microcontroller. A working demonstrator-prototype has to be created in the fabrication process of a product. The functionality of the prototype is the key to whether or not investors will fund the project or even be interested in the product.

When designing a controller for a prototype, it is best to put most effort into the designing and tuning of the controller, rather than the actual implementation of the controller into hardware. In the study, the design and tuning was done with a PC-in-the-loop controller (PILC), using the Matlab/Simulink Real-Time Windows Target software. Once the PILC was finalized, the control scheme was directly ported onto an SH2-7055 embedded microcontroller. This was accomplished with a suite of rapid prototyping software, including MakeApp, TargetLink and Hitachi Embedded Workshop (see the websites [2], [3] and [5]).

The steps taken to accomplish a stand-alone embedded control system are described in this paper.

1 Introduction

Prototyping is an important part of any fabrication process. A good prototype helps investors to have confidence in the engineers that built the prototype and therefore are going to build the final product. So, a prototype can be the key to the final product, because without the confidence of investors, future funding of the project may be difficult to get.

Because of competition, a company wants to be the first to present their prototype of a new product. Speed is more important than detail, in the process of creating a prototype. This is why rapid prototyping is an important issue for the industry. When a prototype can be generated in short time, it will be easier for the company to find investors, because they would be the first in the industry to present their ideas.

In this paper, different ways of developing a prototype-controller are discussed. It will discuss the use of new software tools, to rapidly implement an embedded controller.

In Section 2, implementation of an analog controller is compared to those of PC-in-the-loop and embedded controllers. The main focus is the design of an embedded controller, using the SH2-Hitachi microprocessor. Section 3 gives an example of an actual controller that was first designed by means of an analog circuit. Afterwards, the controller was improved by translating it into an embedded system. Section 4 describes the results of building a controller for a motion platform. Section 5 gives some conclusions on using the SH2 microprocessor and the Hitachi software, for rapid prototyping of embedded controllers.

2 Analog Versus Digital Controllers

When a controller is needed for a product that is being developed, a choice has to be made as to how the controller will be designed. This section will discuss three ways of designing a control system, i.e. by means of an analog circuit, by means of a PC-in-the-loop controller and by means of software (an embedded system). First, the control system, by means of an analog circuit will be discussed. After that, the PC-in-the-loop controller will be discussed. Finally, the controller, by means of an embedded system, will be discussed. The microcontroller that will be discussed in detail, is the SH2 by Hitachi. Also, the software to program the microcontroller will be discussed.

2.1 Analog Controller

When a controller has to be built for a certain system, and this controller is not too complicated, a cheap way of building this controller is by means of analog electronics. When a PID-controller or Lead/Lag-controller with one input sensor is to be build, the electronics are straight forward. Transfer functions can easily be translated into analog electronics. OpAmps with capacitors and resistors in the feedback and feed forward loop are used to create the desired transfer functions. An example of an analog PID-controller and an analog Lead/Lag-controller are shown in Figure 2 and 3, respectively.

The advantage of using analog electronics is that the designed controller will be cheap. To be competitive in this industry, costs need to be kept low. This is a reason for them to choose for analog electronics, as opposed to an embedded digital controller.

2.2 PC-in-the-Loop Controller

A digital controller can be built, as a PC-in-the-loop controller (PILC) . The PC gets the sensor values through a data acquisition card, then calculates the output values, using a certain control strategy. To build a consistent controller, the PC has to process the values, real time. To build a real-time controller, the software that can be used is Real-Time Windows Target (see MathWorks' website [4]).

When using Real-Time Windows Target, Matlab/Simulink can still be used to design a control diagram. This diagram is translated into C-code. Real-Time Windows Target, runs this code at the highest CPU-priority, so that the CPU clock of the PC can be used to make the controller real-time.

Building a PILC is a way of designing a control, before implementing it into an embedded microcontroller. Using Real-Time Windows Target, it is easy to adapt the controller and try different controller strategies.

2.3 Embedded Microcontroller

Electronics are getting cheaper and so are microprocessors. Many applications use microcontrollers to replace functions that used to be built by means of analog electronics. This increase of microcontroller usage causes a chain reaction in the development of software to make the useability of the microcontrollers easier.

When designing a controller for a product that already contains a microprocessor for other purposes, there is no need to have a lot of extra analog electronics since they can be replaced by software.

Writing software instead of designing electronics is cheaper and in most cases even easier and faster than designing analog electronics. This paper describes the use of the Hitachi-SH2 microcontroller and the Hitachi software that is used to create the desired functionality of the SH2. The next sections tell more about how to use this software in order to design an embedded control system.

2.3.1 Simulink to C-code

A general way of designing a controller is by means of Matlab/Simulink. After designing, the result is a Simulink-model that represents the controller. To make this into an embedded control system, the model needs to be translated into C-functions. This can be done directly by using dSPACE's TargetLink. With this software package, it is possible to translate a Simulink model into a TargetLink model. This TargetLink model is basically the same as the Simulink model, but more parameters have to be set, e.g. datatype, such as *float*, *integer*, *char*. The TargetLink model can then be compiled into a C-function.

2.3.2 Configuring I/Os

The SH2 microcontroller, has many I/Os that need to be configured. The software that is used to do the I/O configuration is MakeApp by STENKIL. This is a windows based program that creates C-functions to configure the SH2's I/O ports. For example, when the user wants to configure three timers and one A/D conversion port, he/she sets it up easily in MakeApp and then has MakeApp build the C-code.

2.3.3 Hitachi Embedded Workshop (HEW)

To put all the C-functions together, before downloading them into the SH2, the Hitachi Embedded Workshop (HEW) is used. The user simply has to write the C-code for the function *main*, using the MakeApp and TargetLink C-functions. Furthermore, the user can write C-functions to be downloaded into the SH2. When all is put together, it is ready to be compiled and downloaded into the SH2.

The use of the software is straight forward, this allows the user to concentrate on designing, rather than writing software. A controller can be built and tuned in the familiar Simulink environment. It just takes a few step to then convert this Simulink model into an actual embedded control system.

3 Practical Example

In this section, an example is given of a controller that was first designed by means of analog electronics and was then improved by using a more sophisticated controller, downloaded into the Hitachi-SH2 microprocessor.

3.1 Stabilizing a Motion Platform

The purpose of the controller in this example is to stabilize a motion platform, which means a platform that has to stay level, regardless the surface that it is mounted on. The platform is shown in Figure 1.

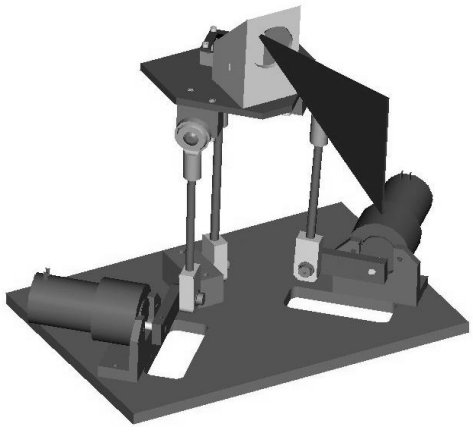

Figure 1: The motion platform that had to be stabilized.

The sensor that was used to measure the tilt angle of the platform is the *Analog Devices ADXL202*,

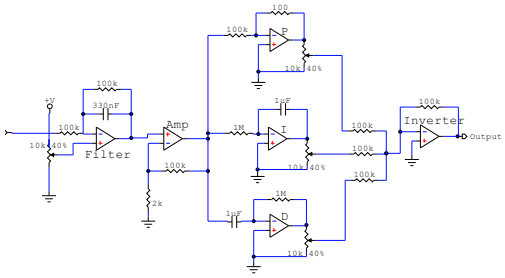

Figure 2: Electronic circuit for the analog PID-controller

which is a two-axis accelerometer. The accelerometer is mounted on top of the platform. When the platform (and hence the accelerometer) is level, no acceleration will be measured. When the platform is tilted the accelerometer will measure the component of gravitational acceleration, parallel to the platform. This measurement is linear with the tilt angle of the platform. The sensor signal was used as input for the controller.

3.2 The Analog Controller

The first controller that was built for this system, was an analog PID-controller, which is shown in Figure 2. The reason to build this controller by means of analog electronics was that it is easy to design, so that results can be achieved fast.

After this circuit was built, it became clear that debugging of the circuit, plus tuning of the controller took more time than expected. The circuit was designed to be able to tune the gains K_P, K_I and K_D. While tuning, it became clear that the input filter needed to be changed and that the K_D-gain needed an extra filter. These kind of changes take much more time to do in analog electronics, than in software.

After this conclusion was drawn, an improved controller was designed with a PC that uses a data-acquisition card to communicate with the system to be controlled. The result was a Lead/Lag controller, built in Matlab/Simulink. After this controller was tuned to its best performance, it was

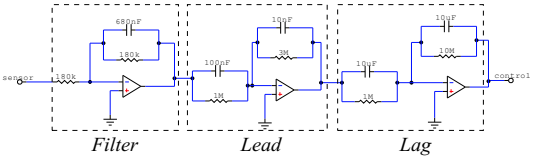

Figure 3: Electronic circuit for the analog Lead/Lag-controller

translated into an analog circuit, which is shown in Figure 3. The performance of this controller still did not meet the specifications, so an improved controller had to be designed. The tilt sensor (accelerometer) was the reason for the controller not to be accurate enough. This sensor is sensitive to more than just gravitational accelerations, for example when the platform is being tilted, the tangential component of the rotational acceleration is measured. In other words, this sensor is very sensitive to external disturbances.

To make the control system more accurate, a sensor was added as an input for the controller. A rate-gyro (BEI Gyrochip^{TM}, Model AQRS, see [1]) was used to measure the angular velocity of the platform. This sensor is more accurate than the accelerometer. The only disturbance on this sensor is the signal noise, no angular velocity, which is not related to a motion of the platform, is measured.

The two sensors were used in combination to achieve a much more accurate estimation of the tilt angle of the platform. To combine the two signals and estimate the angle of the platform, an estimator, based on Kalman filtering was used. A brief description of the Kalman filter is given in section 3.3

3.3 PC-in-the-Loop Controller

This section, briefly, describes the Kalman filter theory. For more details on this theory see [7] and [8]. Equations related to this example are specified in [6].

3.3.1 Platform State Description

To describe the platform in state space notation, three states are taken into account, based on two measurements for each axis, i.e. *tilt angle* (θ) for the accelerometer, *angular velocity* ($\dot{\theta}$) and *bias* (b) for the rate-gyro. The state space model of signals (states x_k) and measurements (outputs y_k) can be written as [8]

$$x_{k+1} \simeq \underbrace{(I + AT_s)}_{A_d} x_k + \underbrace{BT_s}_{B_d} u_k + \underbrace{GT_s}_{G_d} w_k \quad (1)$$

and

$$y_k = Cx_k + Du_k + v_k \quad (2)$$

which, for the platform, can be written as [6]

$$\begin{bmatrix} \theta \\ \dot{\theta} \\ b \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & T_s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ b \end{bmatrix}_k + \begin{bmatrix} w_\theta \\ w_{\dot{\theta}} \\ w_b \end{bmatrix}_k \quad (3)$$

$$\begin{bmatrix} y_{acc} \\ y_{gyro} \end{bmatrix}_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ b \end{bmatrix}_k + \begin{bmatrix} v_{\theta_m} \\ v_{\dot{\theta}_m} \end{bmatrix}_k \quad (4)$$

Since the input signal u is not used in this state space model, the matrices B and D disappear. The values w_θ, $w_{\dot{\theta}}$ and w_b represent distortions that cause the state of the system to change, e.g. external accelerations.

The fault, made in the measurements (measurement noise), is represented in the values of v_{θ_m} and $v_{\dot{\theta}_m}$. The covariance of these signals is used in the Kalman filter to estimate the different states.

3.3.2 Kalman Filter Theory

Estimating the states of the system can be done, using a Kalman filter. This way of filtering makes it possible to combine the signals, from the two sensors and let them weigh differently in the estimation of the states.

An estimation, based on the state space model, can be made just before the next sample. This *a priori* estimate $\hat{x}_k^-$ is used to estimate the state of the next sample, combining the measurement y_k and the *a priori* estimate $\hat{x}_k^-$.

$$\hat{x}_k^- = A_d \hat{x}_{k-1} \quad (5)$$

$$\hat{x}_k = \hat{x}_k^- + K(y_k - C\hat{x}_k^-) \quad (6)$$

where K is called the Kalman-gain matrix. The errors in estimation and their covariances are

$$e_k^- \equiv x_k - \hat{x}_k^- \quad (7)$$

$$P_k^- = E[e_k^- e_k^{-T}] \quad (8)$$

$$e_k \equiv x_k - \hat{x}_k \quad (9)$$

$$P_k = E[e_k e_k^T] \quad (10)$$

Since the sensor signals are assumed to be uncorrelated, by using equations 2 and 6 the covariance P_k is written as

$$P_k = (I - KC)P_k^-(I - KC)^T + KR_kK^T \quad (11)$$

where R_k is the covariance of v_k, $E[v_k {v_k}^T]$.

Obviously, the goal is to minimize P_k. Since the signals are uncorrelated P_k is a diagonal matrix and can be minimized by minimizing the sum of the elements on the diagonal. This is done by taking the derivative of *trace(P_k)* and setting that result equal to zero. The solution for the Kalman matrix is given by

$$K = P_k^- C^T (CP_k^- C^T + R_k)^{-1} \quad (12)$$

For the next sample, the *a priori* matrix P_{k+1}^- has to be calculated again. Using equations 1 and 5 the result is

$$P_{k+1}^- = A_d P_k {A_d}^T + Q_k \quad (13)$$

Where Q_k is the covariance of w_k, $E[w_k {w_k}^T]$.

Based on the preceding information, it can be seen that the equations for the Kalman filter fall into two groups: *time update* equations (5 and 13) and *measurement update* equations (12, 6 and 11). During the time update, the next *a priori* estimate for the state and the covariance of its error ($\hat{x}_k^-$ and P_{k+1}^-, respectively) are calculated. During the measurement update, the Kalman matrix is updated,

an estimation for the state (x_k) is made where the Kalman matrix weights the measurement and the *a priori* estimate. Also, the current value for the covariance of the estimation error (P_k) is calculated.

3.3.3 Kalman Filter for the Platform

For the platform, the covariances are considered to be constants. This means that the matrices R_k and Q_k don't change over time and the matrices P_k and K_k will converge to a certain value.

The matrix R was determined by measuring the covariance of the two sensors (accelerometer and the rate-gyro). Output data was captured of both sensors, during a zero input signal, i.e. *tilt angle* and *angle velocity* are zero.

The system noise w_k that is represented by the matrix Q_k wasn't measured, but the matrix Q_k was used to tune the dynamics of the estimator. The matrix Q_k was chosen constant, so that it becomes Q. Since the covariances of the system noise were considered to be uncorrelated, matrix Q becomes a diagonal matrix. By changing the values of the parameters on the diagonal, the dynamics of the estimator can be changed. The larger the values, the faster the estimator will respond to changes in the corresponding state, e.g. the value that corresponds to the *bias* of the system state vector was given a relatively small value, since the *bias* isn't suppose to change quickly. The dynamics of the estimator were tuned changing the Q matrix, by means of the 'trial and error' method.

Since Q and R are considered to be constant, the Kalman matrix K_k will converge quickly to a constant matrix K. This matrix can be calculated using the Matlab function *dlqe.m*.

Using the matrices R, Q and K the estimator was build in Matlab/Simulink (see Figure 4). The accelerometer and the rate-gyro are the input signals of the estimator and the three estimated states are the outputs, where both the estimates for *tilt angle* and *angular velocity* are used as input for the controller.

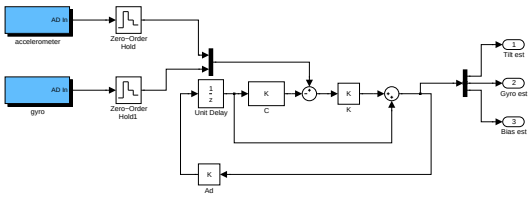

Figure 4: Kalman estimator model in Matlab/Simulink (PILC)

3.4 Embedded Digital Controller

The controller that uses the Kalman estimator, was first built by means of a Matlab/Simulink model that was tested and tuned, using a PC with a data-acquisition card. It was complicated to build this controller in analog electronics. Since the application as a whole uses a microcontroller for other functions, the idea was to use this microcontroller also to translate the Simulink model into an embedded system. The microcontroller that was used, is a Hitachi SH2-7055 Evaluation board (EVB). The microcontroller comes with Hitachi software that makes it easy to implement functions into the SH2. Hitachi Embedded Workbench (HEW) compiles C-code into assembly code that can be downloaded into the SH2. To generate the desired C-code a few steps have to be taken. Those steps were discussed in the sections 2.3.1, 2.3.2 and 2.3.3.

By designing and implementing the controller this way, a significant amount of time was dedicated to the actual design of the controller (through the use of Simulink). It took about four weeks to design and tune the controller and about two weeks to implement it into the SH2. The effort of building this controller was put into the design and tuning, as opposed to the analog approach of designing the controller. The analog PID controller that was built first, took about a week to design and than it took more than four weeks to implement and tune it. The time chart, shown in Figure 5 shows a rough estimate of the time that was devoted to development (design, tuning and implementation) of various controllers in this project.

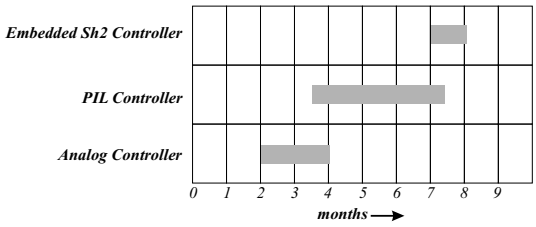

Figure 5: Time chart, showing an estimate of the time, devoted to the development of various controllers in the project.

4 Results

This section describes the results of building a controller for stabilizing a motion platform, first by means of an analog circuit and later by means of an embedded system, i.e. Hitachi SH2 microprocessor.

When the analog circuit was build to control the platform, it was discovered that the tilt sensor was very sensitive to external accelerations. This is shown in Figure 6. The signal of the accelerometer was captured while the motor was driven by alternating positive and negative pulses. When the motor was driven with a positive voltage, the accelerometer would measure an acceleration when the platform started moving and when the platform stopped moving, due to tangential accelerations.

The Kalman estimator was used to ignore accelerations other the gravity. The estimation of the platform angle, when changing as a square wave is shown in Figure 7. It can be seen that the peaks, caused by tangential accelerations disappear when the estimator is used. The estimator mainly uses the sensor signal of the rate-gyro. Measurements of the rate-gyro's are always related to movement of the platform. The Kalman filter estimates the tilt angle, as well as the angular velocity. The estimated angular velocity ($\hat{\dot{\theta}}$) can be used as the differential signal in the PID-controller. The rate-gyro sensor signal and the estimated value for the angular velocity are show in Figure 8. In this Figure, it can be seen that the Kalman filter also serves as a noise

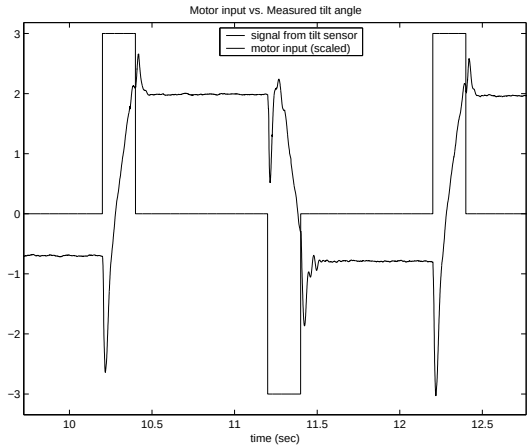

Figure 6: Disturbance in the sensor signal, due to tangential accelerations.

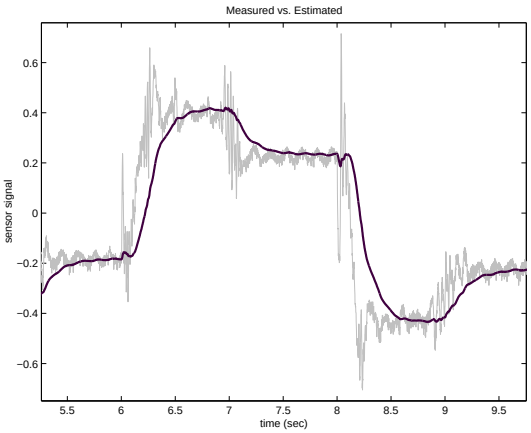

Figure 7: Estimated tilt angle of the platform vs. the measured tilt angle.

filter.

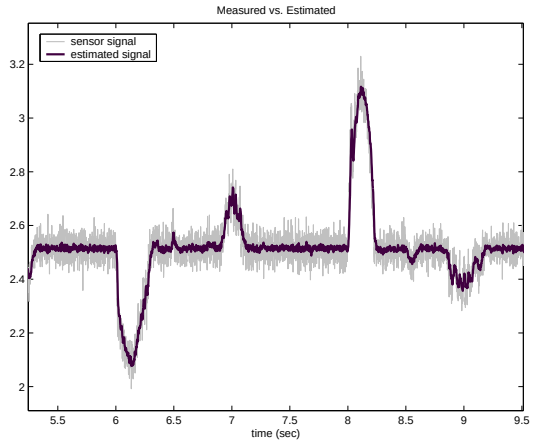

Figure 8: Estimated angular velocity of the platform vs. the rate-gyro sensor value.

5 Conclusions

In this case study, the advantages and disadvantages of designing and building a controller by means of analog electronics, as opposed to using a microcontroller and designing the controller by means of software, were considered and revealed in the process of implementing an actual control system. When a simple and cheap controller is desired, an analog solution will be sufficient. Since electronics are getting cheaper, using a microcontroller as a solution for a control problem, becomes an option even if the controller doesn't have to be very complicated.

Software to develop functions in a microprocessor is becoming more and more user-friendly as well. It doesn't take the skills of a computer engineer or programmer to translate a sophisticated controller, designed with Matlab/Simulink, into an embedded control system. Tools such as dSPACE's TargetLink, STENKIL's MakeApp and Hitachi's Embedded Workshop, make it possible to focus on designing the controller and not having to worry about implementing it into a microprocessor.

Acknowledgements

The authors would like to thank Mr. Donald J. McCune for his assistance with the SH2 software, Dr. Thomas Oho of Hitachi America Ltd, for the hardware/software support, Dr. Andreas Ruekgauer of dSpace Inc, for the software support and Mr. Paul Angott for the overall project and support.

References

[1] *BEI, Systron Donner International Division*. at http://www.systron.com.

[2] *Hitachi Embedded Workshop*. at http://www.hmse.com.

[3] *MakeApp*. at http://www.hmse.com.

[4] *Matlab/Simulink/Real-Time Windows*. at http://www.mathworks.com.

[5] *TargetLink*. at http://www.dspace.com.

[6] Prof. Ka C. Cheok and W. J. Young. *Pitch and Pitch Rate Estimator Formulation*. Oakland University, Rochester, Michigan 48309.

[7] A. Gelb, Jr. J. F. Kasper, Jr. R. A. Nash, C. F. Price, and Jr. A. A. Sutherland. *Applied Optimal Estimation*. The MIT Press, 1986.

[8] G. Welch and G. Bishop. *An Introduction to the Kalman Filter*. 1997. Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175.