On model-based design for real-time embedded mechatronic systems using a “fail fast, fail forward” approach

Emiel Nuijten, system architect, Emiel.Nuijten@sioux.eu
Patrick van Bree, principle system designer mechatronics, Patrick.van.Bree@sioux.eu

Summary:

In the development of complex mechatronic systems controlled by real-time targets, model-based design significantly reduces the risks by enabling fast iterations of development cycles. The method provides an efficient and effective way of working between disciplines. Since hardly any knowledge-transfer is required, errors by miscommunication about the implementation of the mechatronic design in embedded software are avoided. Improving and adapting algorithms to the latest insights is encouraged. Certainly, in *“low-cost price, high-performance requirements”* projects with multiple design iterations and an exploring nature, it can be of a great importance to enable the designers to work in the environment they perform best.

Keywords: model-based design, mechatronic systems, real-time target, code-generation, embedded software, Simulink, Python, time to market, iterative approach, low-cost high-performance.

Introduction

At Sioux, we aim at developing challenging high-tech products with a competing cost price, a higher performance, and/or within a minimal timeframe. Our specialty is on high complexity, high mix and low volume. Cost optimization imposes challenges in the design due to introduced hardware imperfections of in low budget components. To deal with these challenges, we invest in mastering fast design increments. Within Sioux, an embedded platform is developed allowing the use of model-based software development to tackle the described challenge as part of a Model-based Systems Engineering (MBSE) vision on product development.

Real-time mechatronic systems

We develop state-of-the-art mechatronic products. In case of very highly complex or prototyping applications, we rely on high-end components and drives controlled by high-performance real-time targets. Such targets running e.g. Simulink Real-time are convenient since they provide a very efficient infrastructure where the use of models bridges the gap between simulation at a desktop and deploying the controller in closed loop with the actual hardware. The possibilities to automate test, to tune parameters and trace signals is available in both simulation and in target mode. Ideas can be tried out by the same designers that developed the algorithm, and best of all it can be done immediately with only a minimum amount of overhead.

The challenge of high-tech, low-cost

When we accept the challenge of combining high-performance with a competing cost price, both the high-end components and luxury real-time targets do not fit the profile of the final product. What will be the consequences of replacing luxury servomotors and encoders by low-cost alternatives? Can we, under these conditions, still deliver in time?

© 2019 Sioux

Considering low-cost, take as example an industrial servomotor with optical encoder. Perfectly accepted in one-off machine building. As product dimensions get smaller and quantities increase, the accepted cost for a position controlled motor drops rapidly. Questions arise whether a stepper motor (at a cost price of a few euros) would be acceptable instead. However, the cost price of the servo motor would be just within reach if we would lose the costly encoder and would still have sufficient trust in reaching our accuracy goals. In case a significantly better alternative pops up in a late development stage, do we dare to integrate it? How to manage this without introducing an unacceptable high risk?

Risk reduction by “failing fast, failing forward”

Having to change design concepts and control architectures within the product development phase is challenging and might feel a little uncomfortable. With a known tight time-schedule, having an efficient risk reduction method is important. This method is found in iterative development cycles enabling a fail fast, fail forward approach. The ideal domain for model-based systems engineering and model-based software design.

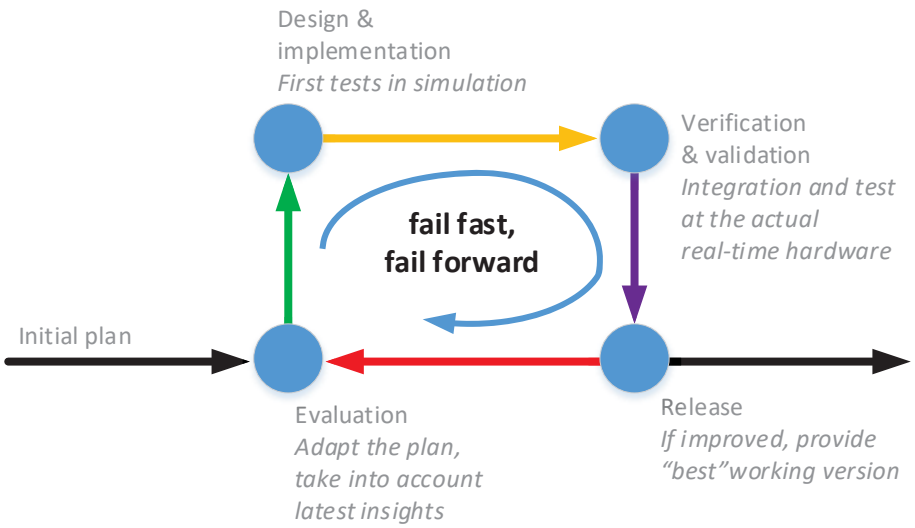

Figure 1 : Iterative product development cycles with a fail fast, fail forward approach.

It is all about communication …

In classical development of embedded systems, the mechatronics designers specify software requirements based on their calculations and simulations of functions and control loops. In a later stadium, the implementation is provided by software engineers. Knowledge transfer and interpretation of the requirements is a time consuming practice where misunderstandings are common and found only after the code has been completed and verified. Functions and logical rules written in C++ are easily checked, but performance of algorithms within feedback control loops is hard to verify without a proper (simulation) environment. In the latter case, the mechatronics designer has to become acquainted with the software platform and execute tests on the actual hardware where simulation, trace and debug options are more limited, far less accessible and more time consuming when compared with the environment in which the design is made.
Model-based design greatly reduces this effort. The mechatronics designer is operating in a domain he knows best and more importantly, the use of model-based tooling removes a very error prone communication step; there is no need to have the numerical algorithms and related functional behavior implemented manually in a different way than done during those early concept simulations.

© 2019 Sioux

Making the right split

It is important to utilize each other's strengths. An embedded software engineer is confident with platform specific implementation such as hardware abstraction, task scheduling, support for communication protocols and how to deal with memory storage and infrastructures to enable software updates.
The mechatronics designer has focus on functional behavior of system modules, dynamic behavior and control system design. Think of automatic calibrations, adaptations or learning algorithms and controller structures capable of dealing with imperfections in sensors and actuators. Those topics are far from trivial to specify upfront and their design may involve many iterations.

Making a split and being able to work in parallel, gives a pleasant balance in the development team. One group is totally focused on providing a well usable platform from software perspective. The other group focuses on reaching functional and dynamic objectives. Less information transfers are required for implementation and there is the benefit of having a graphical model for documentation. With this model-based approach, our mechatronic engineers give as feedback that they feel less restricted to refactor concepts, to try out improved or new algorithms and to find errors and weaknesses in less time.

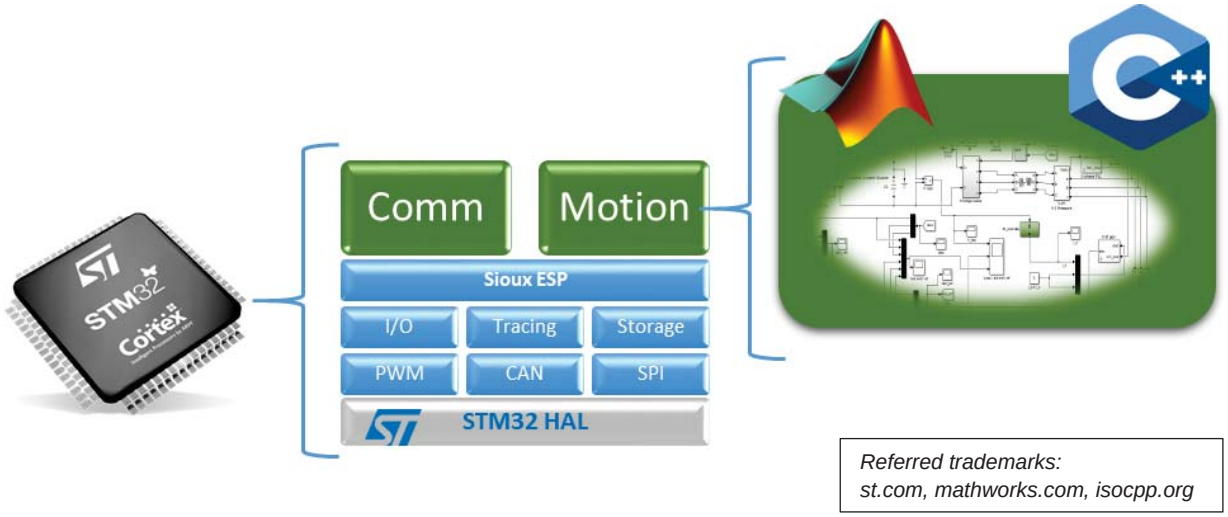

Figure 2 : Making the right split. Matlab-Simulink with code generation is used for motion control functionality. Software developers provide e.g. the hardware abstraction and communication specific functions.

A proper architecture and clean code

Model-based development may seem intuitive and easier to maintain, but finally it is all about well-organized code and well-defined architectures and interfaces in the same sense as writing software in any other way. The biggest benefits of model-based design are found in the understanding of the model and code across disciplines. This understanding is made possible by the "model" that serves as design, specification, code base, test instance and documentation. At Sioux, software engineers take care of defining architectures that support proper embedding of code generated from model-based designs. Besides *readability* and *maintainability, testability, modularity* and *reusability* of building blocks play a big role when activities in model-based controller design grow. Control engineers are in general well acquainted with Matlab-Simulink tooling that they use for analysis, design and simulation. With awareness of software design rules, they put themselves in the position to deliver high quality software from within the environment they function best.

© 2019 Sioux

A Simulink based motor control example

A motion system with tight requirements regarding positioning accuracy is developed. The product is intended for series production, which implies that for the sensor and actuator selection a cost price difference of below one euro is already significant. Next to performance and pricing, the availability of components during the product life cycle and for example volume restrictions imposed by the application will further limit the set of suitable electrical motors and encoders that can be used. In this section a typical way of working is described using an example in which the design is optimized to obtain a “good enough = within specification” positioning accuracy for an affordable price.

Feasibility of requirements

To verify the feasibility of the requirements with the initial design, experimental setups are built that are capable of controlling the (individual) processes. These type of setups typically focus on sub-functionalities and they will not be integrated into a complete machine. In this stadium, high-end drives, highly accurate encoders and powerful real-time targets are used to obtain full understanding, to benchmark the performance and to sharpen the requirements. The test results e.g. provide the required torques and characteristics of the disturbance force. With this information, the requirements towards motors and encoders are specified on which basis the cost price estimation is updated. A conflict between requirements and cost targets will immediately become visible.

Prototyping and early testing

The prototype, which is based on an embedded target, supports all core functionalities. Preferably, it shall be integrated and tested at the customer. As a result, a major risk reduction is achieved since missing requirements and misinterpretation of requirements become identified in an early phase of the project. Next to that, a functional prototype will put developers from other “domains” (e.g. mechanical designers, testers) in the position to evaluate their contribution to the project.

The request of being able to test functionality in an early phase usually conflicts with the time needed to develop compensation algorithms for artifacts introduced by low-cost hardware. To solve this conflict, alternative high-end components are used that do not require extensive compensation and calibration. In a later design phase, these more expensive components are replaced by the more affordable versions plus compensation in software. The system architecture is already prepared to support this future design iteration. The Simulink model for the prototype already supports all functions like command handling, trajectory generation and feedback control.

An alternative encoder solution

In the motor control example, the prototype contains an optical encoder for measuring the motor position. For the final product, this solution is too expensive. However, for the prototype it enables us to provide a working solution. In the next phase of the project, an alternative for the encoder needs to be integrated. The alternatives are:

- a custom low-cost magnetic rotary encoder as part of one of the gears in the drivetrain,
- gear-tooth sensing with event driven position correction,
- motor angle estimation based on sensing the magnetic field induced by the rotor magnet.

Together with the supplier of the electrical motor, the choice is made to replace the motor’s digital hall sensors used for commutation by analog hall sensors. The motor commutation switching angles must now be determined in software within the microcontroller, instead of directly by the digital hall logic. The position-sensing algorithm in Simulink supports the use of an incremental encoder for the prototype and the architecture is in this stage already prepared for the use of an absolute motor angle.

© 2019 Sioux

Variant handling

The Simulink model is extended with options to select either the incremental encoder and motor commutation based on logic or commutation and position control based on angle estimation. The result is that the same software version can be executed at different hardware versions. This step requires a tight coordination between what happens in Simulink and what happens deep in the embedded software close to the hardware layers. Before the customized motors arrive, the angle estimation algorithms are implemented and tested in simulation. In parallel, the position controller is tuned using the actual setup with the incremental encoder. Both activities take into account the same version controlled model.

Iteratively towards performance

The cost price issue is solved by removing the optical encoder. The changes in the electronic layout and software to work with analog hall instead of digital had only small impact. What is left is the resulting position accuracy. Ideally, the analog hall sensors provide three sinusoidal signals with equal offset, equal amplitude and a phase shift of 120 degrees. However, differences in the rotor magnetization for different pole-pairs and tolerances in the sensor placement will lead to a significant error in angular position. A simple Clarke transform is insufficient; algorithms are needed to obtain an accurate motor position and motor velocity estimate. The controller design is first made suitable to provide minimal but robust performance with the disturbed angular position. Next, many iterations that improve performance follow.

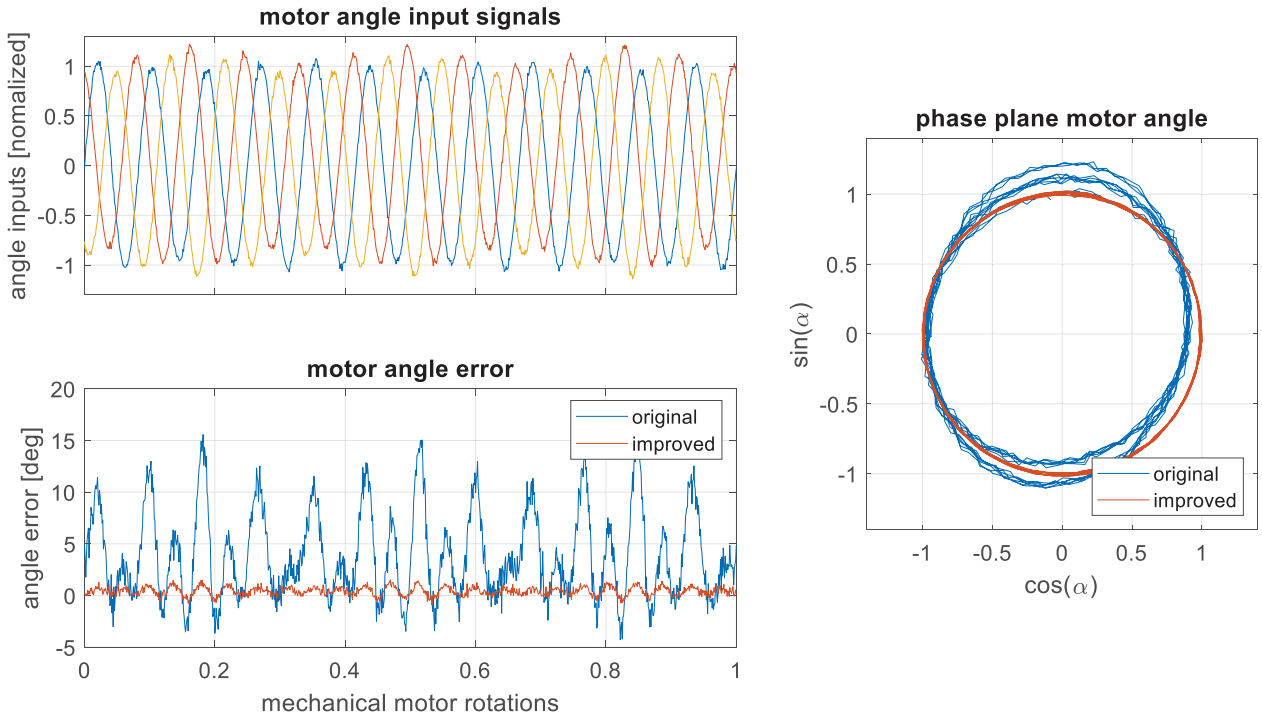

Figure 3: Motor angle estimation based on the analog hall custom encoder option. Ideally, three sine waves with equal amplitude and 120 degrees phase shift are received. Due to production tolerances in the motor, the analog hall signals are far from ideal. However, an accurate angular position can still be derived by advanced estimation algorithms that benefit from the repetitiveness of the disturbances.

© 2019 Sioux

Simulink and Python as tools for fast iterations

The development framework is extended with a Python environment in which test scenarios can be scripted. Thus, it becomes possible to obtain experimental data from a large batch of motors. Automated tests can interact with the actual hardware as well as with the simulation model. Performance can be evaluated based on traces from process variables but also on internal signals from the controller. By making use of the option to tune parameters online, as in Simulink, the different options can be compared rapidly. Experiments confirm that the disturbances repeat themselves with the mechanical motor angle. This insight holds the key for increased accuracy.

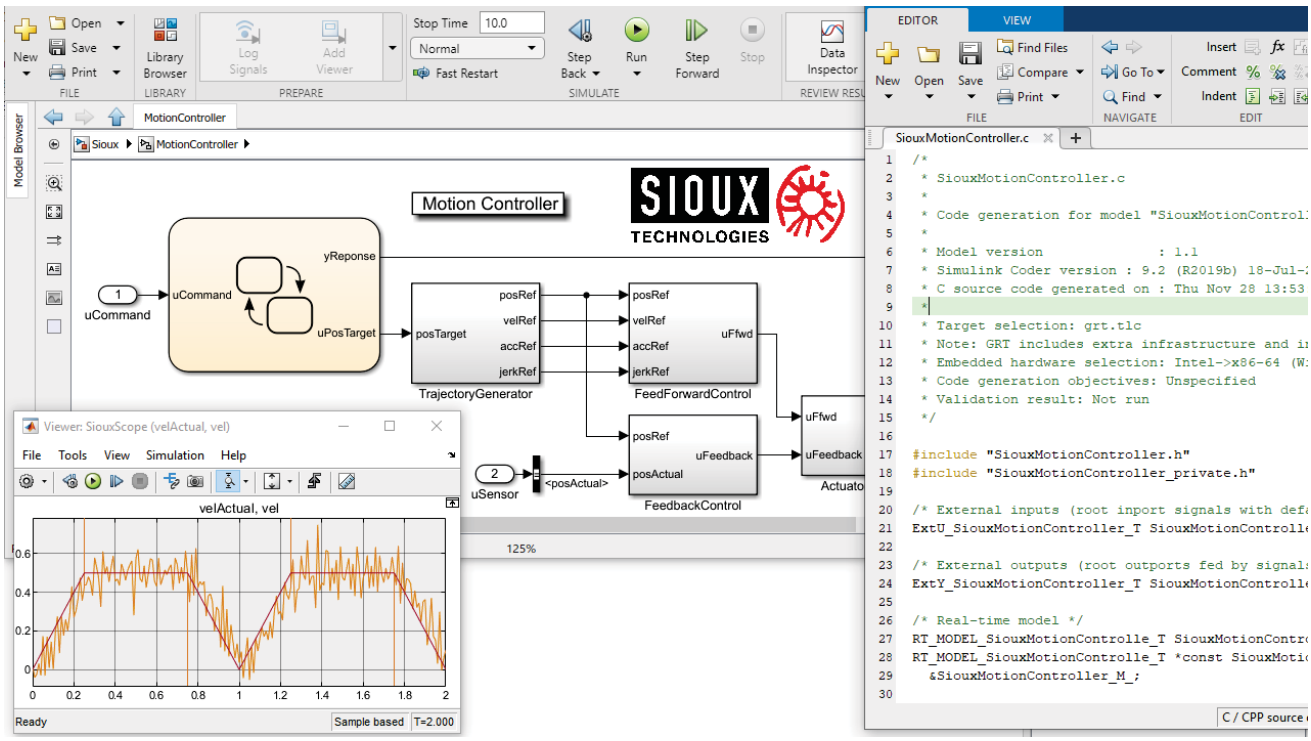

Figure 3 : Simulink environment in which implementation, simulation, documentation and code-generation take place.

Algorithm development

The simulation model is extended to cover the different characteristics of the disturbances. The suitability of an algorithm design change is evaluated based on both simulation and real-life testing, using the exact same model. In one case, the controller runs in simulation mode interacting with an environmental plant model, in the other case as generated code embedded in the application. With the combination of a calibration-procedure, that takes into account position dependent disturbances, and a model-based observer, the positioning accuracy of the overall system is brought within specification.

© 2019 Sioux

Keeping responsibilities with the right person

The angle estimation problem is just an example where algorithms solve imperfections of the hardware. These imperfections were in no way specified in a datasheet and not known beforehand. Using Matlab-Simulink the mechatronics designers gain in flexibility and are in full control of the functions: design, implementation and test without the need for in-depth knowledge of embedded software or error prone knowledge transfers. Barriers to improve and adapt functions to new insights or to evaluate alternatives are at a very minimum. Within a minimum amount of time, it is known whether the idea fails or if we made a step forward.

Suitability for embedded systems

Fast iterations and evaluating multiple concepts are a challenge on their own. We experience that using Matlab-Simulink as model-based language and having higher-level languages such as python for test-frameworks and automation greatly reduces the effort spent when comparing to other projects where classical specification and programming approaches are used. The model-based development approach is very compatible with microcontroller-based development. Modern affordable microcontrollers such as the STM32 series bring sufficient capabilities to even perform live sample-synchronous measurements for control loop analysis and can operate efficiently in floating point. With the right tooling, a mechatronics engineer can visualize signal traces at sample rate and perform FRF analysis in applications. Test frameworks written in high-level languages such as Python form a flexible and low-cost alternative to data (post) processing in Mathworks tooling.

I-Mech: Intelligent Motion Control Platform for Smart Mechatronic Systems

For effective and efficient development of mechatronic systems, much more is needed than the topics mentioned in this paper. The presented approach refers to development methods of real-time embedded controllers. These type of sub-systems are close to the hardware to be controlled, and contain smart sensors, smart actuators and smart power electronics. Within the architecture defined in the so-called I-Mech project this level maps to the “instrumentation layer”. Within I-Mech, a consortium of >30 participants of both industry and academy work together to bridge the gap between the latest research results and best industrial practice in advanced mechatronic motion control systems. On multiple levels model-based techniques are exploited to keep grip on complexity. See www.i-mech.eu for a detailed description and results obtained with pilots and demonstrators.

Conclusion

Within Sioux, the challenge of developing embedded real-time controllers for complex mechatronic systems is tackled using model-based design techniques. They enable us to define proper architectures and to generate clean code with minimal risk and within minimal time. The mechatronic system designer, as the owner of the functionality to be developed, is put in a position from which he/she can design, deploy and evaluate increments without being dependent on other disciplines. The development of low-cost - high-performance systems, in which the imperfections of sensors and actuators are corrected in software, is kept manageable, also for embedded targets.

Based on our experience with Matlab-Simulink in many projects, we can recommend following article for further reading, focused on the ROI impact of model-based design and the benefits of early testing and integration. Try the QR code.

http://tiny.cc/mvr8gz

© 2019 Sioux