From concept model to production code

In the industry, model-based design is utilized more and more often. Because changing a conventional development process to a model-based process is not simple, we take a look at a practical approach.

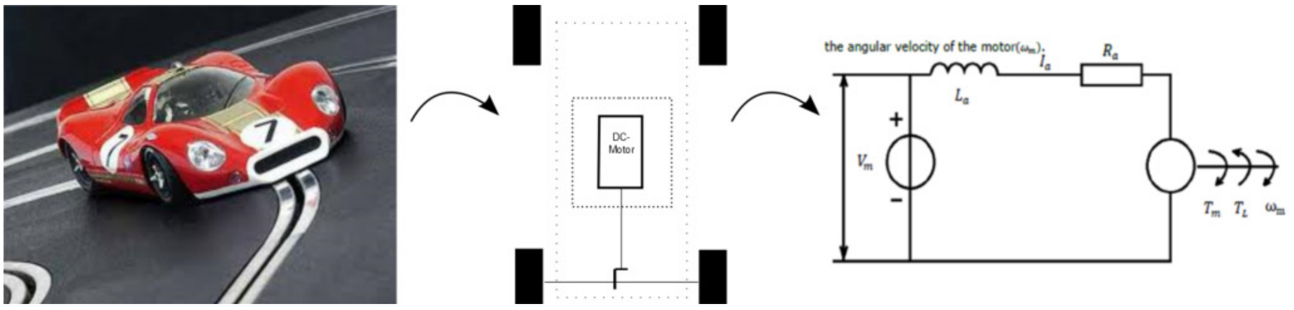

Figure 1: Slot car representing a linear actuator system and schematic model of a DC motor

The model based design (MBD) process relies on mathematical models and simulation. Adjusting to this new process can be overwhelming when not properly implemented. We present a practical approach to the MBD process, using an example to explain the critical choices that need to be made in order to start with a concept and end up with production code.

We want to illustrate the steps that are needed to go from concept model to production code in a quick development process. The process is applied to a linear actuator based on an electric DC motor. A linear actuator is used in many applications, for example agricultural tools, printers, CD players, etc. For simplification purposes, in this article the linear actuator is replaced by a slot car. Of course, the presented techniques are valid for a large range of control problems. The idea of the slot-car example is as follows: A slot car racetrack is modified so that one car is controlled by a computer. The goal is that the slot car is accelerated from a starting position as fast as possible and crosses a finish line. However, it should brake as well, because of a fictive wall shortly after the finish line. Controllable parameters are the voltage and current supply to the slot car. Position sensors for feedback are also employed.

The challenge in this example is the development of the controller. It is a perfect example to show the model based design process for a quick translation from concept model to production code. In the example, we will make use of a plant model for the development of a control design solving the control problem. The developed controller is discretized, CAN-communication is added, other software limitations are taken into account, and fault behavior is introduced. All these steps have their limitations and affect the performance of the control software on the production ECU. The plant model is used for testing and verifying the performance at each development step. This helps detecting limitations and hick-ups early in the development process. Solving these issues as early as possible in the development process is the key to a quick and cost-effective translation from concept to production code.

Plant model

Before the development the actual controller, a plant model has to be made. A plant model helps the development process in several ways: By making a plant model, the developer gets a lot of insights in the system and how it behaves. This helps to focus on the actual control problem. Furthermore, a plant model is useful to do quick 'verification' iterations during the controller development. Creating a plant model requires a significant effort early in the project. However, it supports the development steps of the control system and helps getting closer to a first-time-right concept.

A plant model can be made in several ways, from very simple to extremely complex. Creating an appropriate plant model is a study on its own. One can easily loose oneself in making models too detailed. It is best to keep the model as simple as possible. A simple model is easy to maintain and already gives a lot of insight. If really needed, the model can be extended with additional complexity. In the slot car example, this means that we have to create a plant model of the slot car itself. It does not make sense to model the slot car as a multi-body dynamic model, which takes friction losses of the air drag, slip of the wheels, thermodynamics of the motor etc. into account. It has to be kept as simple as possible. If a linear actuator were modeled, the plant model would also not take every possible influence factor into account.

We start with a model of a DC motor, the heart of the slot car (see Figure 2). If it turns out that the dynamics of the DC motor are not sufficient to cover the behavior of the real plant, one can decide to extend the model with, for example, a load estimator (mainly friction in driveline). The DC motor is modeled according to the first-principles by using the differential equations in the electrical and mechanical domain.

$U_m - R_a I_a - L_a \frac{dI_a}{dt} = U_{emf}$, with $U, R, I,$ and L voltage, resistance, current, and inductance. Indices m, a, and emf represent the motor, armature, and electromotive forces.

$T_m - T_L = J\frac{d\omega_m}{dt} + D_m\omega_m$, with T, J, D and ω as torque, inertia, dynamic friction, and rotational speed. Indices m and L represent motor and load.

The coupling between both equations is given by $U_{emf} = K\omega_m$ and $T_m = KI_a$, with K as a motor constant.

When these equations are rewritten and transferred to the frequency domain, it results in the transfer function:

$$tf(s) = \frac{\omega_m(s)}{U_m(s)} = \frac{\frac{K}{L_a J_m}}{s^2 + \frac{(D_m L_a + J_m R_a)}{L_a J_m}s + \frac{(K^2 + R_a D_m)}{L_a J_m}}$$

Of course it is of great importance to use the correct parameters in the equation to have a system response which reflects the real plant. Some parameters can be measured, derived or are given by the supplier. A way to get the unknown parameters is to do verification measurements on the plant and derive these parameters from the real plant response. This is also a check if the plant behavior is sufficiently covered by the model.

Controller design

Once a plant model is created, it can be used efficiently for the controller development (Figure 3). Classical control theory is used to come up with a controller, which fulfills the set requirements. By using the plant model, the performance is easily verified and visualized, as can be seen in Figure 4.

Theoretically, the developed controller should perform very well and in most organizations, this is the end of the R&D process. Since – in theory – it has been proven that the system works, the concept is given to the software developers with the request to “Please implement this in an embedded system”. However, in the next development steps, when it has to actually be implemented on an ECU controller, performance might, and in most situations, will be affected.

Discrete controller

Once it has been proven in theory that the concept works, it must be further developed to production code. One of the necessary steps is to discretize the controller. In the end, ECUs are not able to run continuous calculations and they have limitations. By discretizing the controller to a certain sample rate, the behavior of the system is changed (see Figure 5). It is therefore of great importance to recheck the performance of the controller after discretization.

In case of the slot car, the open loop transfer function in the continuous domain looks like this:

$$G(s) = \frac{V_{vehicle}(s)}{U_m(s)} = \frac{A_c}{s^2 + B_c s + C_c}$$

▷

After discretizing it to a certain sample rate, the transfer function changes to:

$$tf\,(G) = \frac{A_d\,Z + B_d}{Z^2 + C_d\,Z + D_d}$$

Note that the values A_d ... D_d differ when the sample rate changes.

If the discretized controller is used without rechecking the performance, stability, and robustness, there is a serious risk of malfunction. With the discretizing, an additional delay is introduced, which especially affects the phase of the system. In the continuous domain, the controller is tuned with a bandwidth around 200 Hz, where the classical control theory stability margins are fine. If the discretized controller of 1 ms is used, one can see in Figure 6 that the phase margin is critically low, resulting in an instable system. For a 10 ms discretized controller it is not even possible to reach the bandwidth of 200 Hz. To ensure correct behavior, the controller must be re-tuned. Mostly, the bandwidth has to drop significantly to guarantee a stable system. The example indicates the importance of taking these limitations into account during the development process. This avoids that R&D comes up with a controller bandwidth that is out of reach of the embedded system.

Software code must have protections against division by zero, must make decisions on signed versus unsigned variables, data types based on the possible range of a variable, etc. And secondly, solutions that could potentially work might have a significant impact on processor load and memory usage.

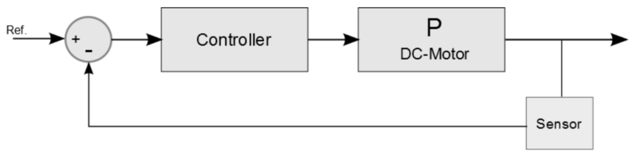

Figure 2: Controller and plant

CAN controller

The steps mentioned above still leave out an important factor: the CAN network. In typical automotive control systems, multiple control units hold sections of the complete system. In our example, a smart sensor/actuator system handles all the measurements and power electronics while the control algorithm itself runs on a separate ECU. In between is the CAN network. The CAN network has its own message rate, which acts as another discrete sample rate. However, the difficulty with CAN messages is that the actual sample rate is not constant. CAN messages are sent based on priority when there is room available on the bus. Depending on the priority of the message and bus load, additional delays may be occur. The control system must be prepared to work with a bandwidth of CAN delays. Therefore, some tolerance must be engineered in for controller stability and for controller settings to meet the requirements.

Another potential delay in the process is the sensor signal processing. However, this delay is stable and less significant. Taking these types of issues into account early

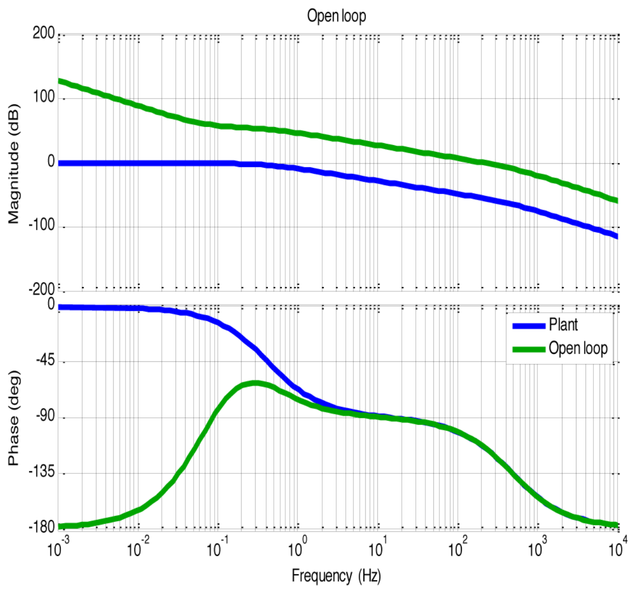

Figure 3: Bode plot of the plant and open loop controlled plant

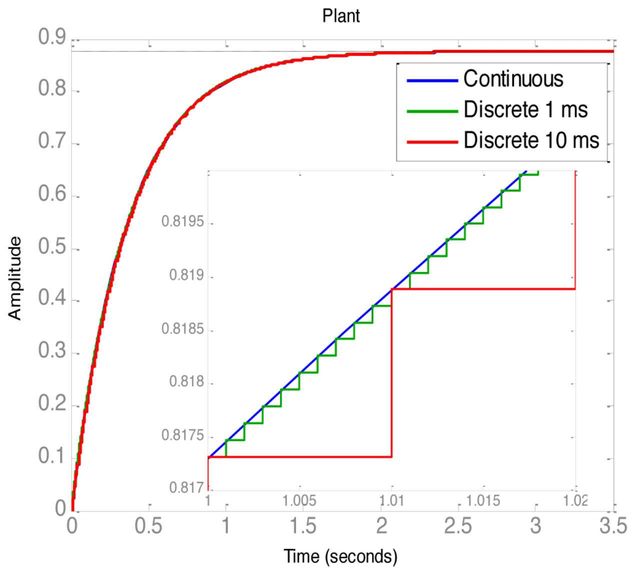

Figure 4: Effect of discretization

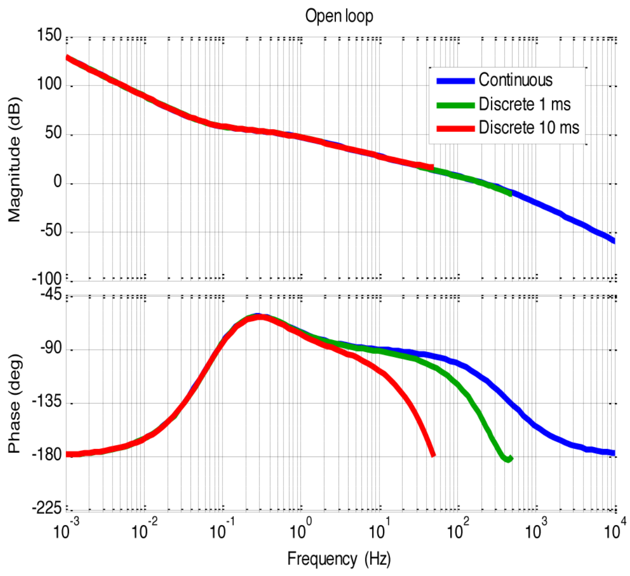

Figure 5: Bode plot of the controlled system in continuous and discrete sampling steps

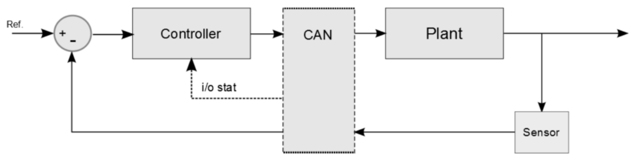

Figure 6: Controller and plant with CAN network interaction

in the development process when the controller settings are determined, limits time consuming rework during the project.

Fault behavior

A proper controller design takes into account what to do when errors occur. In the example with the CAN network interaction, it must be prevented that extreme delays of CAN messages cause unstable behavior.

For a linear actuator that has a smart sensor/actuator setup, a problem can occur when the actuator command is sent out but not quickly followed by a new actuator command. It is possible that the linear actuator is sent to its end position at maximum speed and does not slow down or stop when it reaches the end position. When trying to act on sensor signals, if the sensor CAN message has a very large delay it might not be received at all. And naturally, the sensor signal may have an error in it as well.

To prevent unwanted scenarios and damage, robustness for faulty behavior must be implemented in the controller design. Preferably, the controller enters a special safe state when faults are detected. In automotive applications, this state is typically known as "limp-home" or "default" state.

Results

The chosen approach to the linear actuator/the slot-car, provides a pragmatic approach that is time and cost effective. Quick development iterations are available to the concept developers and work can start using the plant model when the actual plant is not even available yet. The same is true for when the actual ECU target is not available yet.

The chosen approach with different pragmatic testing steps naturally guides developers to gradually build up the realism of testing scenarios and equipment. Step by step, the developers are faced with real life restrictions. This helps to make the control system more suited for its real-life environment. Involvement of the original concept developer in this process makes sure that the optimum result is achieved given the limitations of discretization, CAN networks, and other items. Fault behavior and actions to be taken towards a safe response are best identified early in the process. The developer of the plant model is also most aware of system responses and the possible risks of incorrect controller behavior.

Avoiding the pitfalls of the MBD approach

The plant model approach takes more effort at the start of the project. However, it provides many efficiency gains later on in the development process. Classis control theory

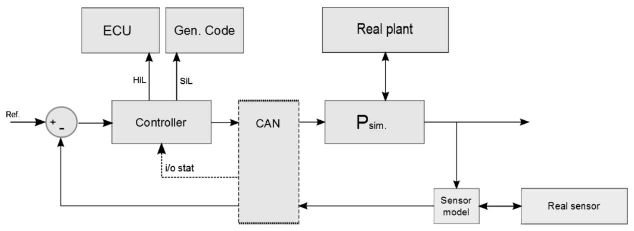

Figure 7: Controller and plant can be exchanged in development steps with simulations, software code, and real components

is still very usable for the concept phase and can be combined with the plant model. There are, however, significant pitfalls in the process.

If departments isolate themselves from each other, the benefits of the MBD approach cannot be achieved. The pitfalls of discretization, CAN network delays, and variation etc. must be tackled in cooperation between departments. Pitfalls can be identified more easily when tests and validation steps occur as early as possible. During the further course of the project, the tests involve more variables and unknowns, more actual components and targets and this gradually brings the control system to a production-ready status. Handling the testing and verification/validation steps late in the process mostly results in a large amount of rework, which is costly and time consuming. ◀

Authors

Mark Maessen, Judith Vliegen
Brace Automotive
www.brace-automotive.com