Electrifier vos systèmes avec Simulink

Kevin Roblet, MathWorks
Morgan Fremovici, MathWorks
no MBD
no electrical skill

MBD + electrical skills
Skilled with motors but not that much with MBD

Skilled with MBD but not that much with motors
I am skilled with MBD with good ROI. Let’s **address new markets**!

I know how to get the best of IA with simulation. Let’s differentiate with **innovative features**!

I do not know that much about motors. Let’s take this opportunity to **upskill**!
models to capture knowledge and as a collaboration and communication tool
Our motor expertise is critical, how do we...
... ensure **knowledge flow** in the company?
... handle software **maintainability**?
... **train** new hires?

Raw material prices, supply chain disruptions... we need to...
... **depend less on prototypes**
... get more job done in simulation
... be **less dependent on specific chips**

Our customers have new expectations, let's...
... **deploy on FPGA**!
... **achieve certification**!
models to try out new ideas through short agile iteration cycles
Motor Control Blockset
Concevoir et implémerter des algorithmes de contrôle moteur
Managing Model-Based Design

Knowledge
Agile
Resilient
Lean

Time to Market
Cost
Innovation
Quality
Upskill

Roger Aarenstrup
8 Core Concepts
Model-Based Design

- Executable Specification
- Model elaboration
- System-level simulation
- What-if analysis
- Continuous test and verification
- Virtual prototyping
- Automation
- Knowledge capture

TIME TO MARKET
COST
INNOVATION
QUALITY
UPSKILL

KNOWLEDGE
AGILE
RESILIENT
LEAN
CALIBRATE SENSORS → ESTIMATE MOTOR PARAMETERS → MODEL MOTOR & INVERTER → DESIGN FOC ALGORITHM → TUNE CONTROLLER GAINS → VERIFY IN DESKTOP SIMULATION → GENERATE CODE → VALIDATE ON HARDWARE

Workflow step
Agenda

From Desktop Simulation to Software Deployment

- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model

- Algorithm design with simulation
  - Field-Oriented control
  - Controller tuning
  - Calibration

- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test
Agenda
From Desktop Simulation to Software Deployment

- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model

- Algorithm design with simulation
  - Field-Oriented control
  - Controller tuning
  - Calibration

- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test
Motor Parameters Estimation

Plant Modeling

Two types of parameter estimation methods:

- Parameter Estimation with Instrumented Test
- Parameter Estimation using Operation Data
Other techniques to parameterize motor models

**Bonus**

**PMSM Parameterization from Datasheet**

Two test harnesses that add confidence that a PMSM is correctly parameterized from a datasheet. It also calculates motor efficiency at

From datasheet

**Simscape Electrical**

---

**Import IPMSM Flux Linkage Data from ANSYS Maxwell**

Import a motor design from ANSYS® Maxwell® into a Simscape™ simulation.

From ANSYS Maxwell, JMAG, Motor-CAD FEA tools

**Simscape Electrical**

---

**Import IPMSM Flux Linkage Data from Motor-CAD**

Import a motor design from Motor-CAD into a Simscape™ simulation.

From dy no data

**Powertrain Blockset**

---

**Generate Parameters for Flux-Based PMSM Block**

Using MathWorks tools, you can create lookup tables for an interior permanent magnet synchronous motor (PMSM) controller that characterizes the d-axis and q-axis current as a function of d-axis and q-axis flux.

To generate the flux parameters for the Flux-Based PMSM block, follow these workflow steps. Example script `CreatingIdqTable.m` calls `gridfit` to model the current surface using scattered or semi-scattered flux data.

<table>
<thead>
<tr>
<th>Workflow</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1. Load and Preprocess Data</td>
<td>Load and preprocess this nonlinear motor flux data from dynamometer testing or finite element analysis (FEA):</td>
</tr>
<tr>
<td></td>
<td>• d- and q- axis current</td>
</tr>
<tr>
<td></td>
<td>• d- and q- axis flux</td>
</tr>
<tr>
<td></td>
<td>• Electromagnetic motor torque</td>
</tr>
<tr>
<td>Step 2. Generate Evenly Spaced Table Data From Scattered Data</td>
<td>Use the <code>gridfit</code> function to generate evenly spaced data. Visualize the flux surface plots.</td>
</tr>
</tbody>
</table>

Step 3: Set Block Parameters

Set workspace variables that you can use for the Flux-Based PM Controller block parameters.
Motor Parameters Estimation - Instrumented Test

Plant Modeling
Agenda

From Desktop Simulation to Software Deployment

- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model

- Algorithm design with simulation
  - Field-Oriented control
  - Controller tuning
  - Calibration

- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test

Workflow step
Motor and Inverter Modeling

Choose the right level of fidelity

- Use linear lumped-parameter model shipped with Motor Control Blockset
Model Fidelity

Plant Modeling

- Linear Lumped Parameter
  - Motor Control Blockset
  - Simscape Electrical

- Saturation
  - Simscape Electrical

- Saturation & Spatial Harmonics
  - Simscape Electrical
Simscape Products

Plant Modeling

- Simscape platform
  - Foundation libraries in many domains
  - Language for defining custom blocks
    - Extension of MATLAB
  - Simulation engine and custom diagnostics

- Simscape add-on libraries
  - Extend foundation domains with components, effects, parameterizations
  - Multibody simulation
  - Editing Mode permits use of add-ons with Simscape license only
  - Models can be converted to C code
Trade Off - Balance Model Fidelity vs Simulation Speed

Plant Modeling
Agenda
From Desktop Simulation to Software Deployment

- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model

- Algorithm design with simulation
  - Field-Oriented control
  - Controller tuning
  - Calibration

- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test
Field-Oriented control

Controller tuning

Calibration
Modeling Field-Oriented Control (FOC)

A word about transforms

Measured current (A, B, C) in time domain

Current control (d, q)

Clarke transform (abc ↔ αβ)
Park transform (αβ ↔ dq)
Modeling Field-Oriented Control (FOC)

Algorithm Design

[Diagram of the Field-Oriented Control system]

- Speed Controller
- Current Reference Generator
- Current Controller
- Space Vector Generator
- Power Inverter

Symbols:
- $\omega_{ref}$: Reference Angular Speed
- $T_{ref}$: Reference Torque
- $i_{d,ref}$: Reference Direct Component of Current
- $i_{q,ref}$: Reference Quadrature Component of Current
- $v_{d,ref}$: Reference Direct Component of Voltage
- $v_{q,ref}$: Reference Quadrature Component of Voltage
- $v_{a,ref}$: Reference Alpha Component of Voltage
- $v_{b,ref}$: Reference Beta Component of Voltage
- $v_{DC}$: DC Voltage
- $\theta_e$: Electrical Angle
- $\alpha$, $\beta$: Alpha and Beta components
- $i_a$, $i_b$: Phase A and B Currents

Control algorithm vs. Physical system
Modeling Field-Oriented Control (FOC)

Overview of the model
Field-Oriented control

Controller tuning

Calibration
Modeling Field-Oriented Control (FOC)

Empirical Computation

Motor Control Blockset

FOC Autotuner

Motor Control Blockset

Classic Control Theory

Simulink Control Design
Autotuning controller gains

Overview of the workflow
Field-Oriented control
Controller tuning
Calibration
FOC Circles tracing

mcbPMSMConstraintCurves

Run this section

```matlab
inverter.V_dc = 24;
inverter.p = 4;
inverter.Rs = 0.15;
inverter.Ld = 1e-3;
inverter.Lq = inverter.Ld * 2.3;
inverter.B = 1.16e-5;
inverter.FluxPM = 5.2e-3;
w_rpm = 3500;
T_load = 0.01;
inverter.Irated = 2.5;
mcbPMSMCharacteristics(inverter,'speed',w_rpm,'torque',T_load);
xlim([-10.9, 6.0]);
ylim([-12.0, 7.7]);
legend("Position", [0.32518, 0.11984, 0.61786, 0.2869])
```

Calibrating Optimal PMSM Torque Control with Field-Weakening Using Model-Based Calibration

By Dakai Hu, MathWorks

Permanent magnet synchronous motor (PMSM) calibration is an indispensable step in the design of high-performance electric traction drive controls. Traditionally, the calibration process involves extensive hardware dynamometer (dyno) testing and data processing, and its accuracy depends largely on the expertise of the calibration engineer.

Model-based calibration standardizes the PMSM calibration process, reduces unnecessary testing, and generates consistent results. It is an industry-proven, automated workflow that uses statistical modeling and numeric optimization to optimally calibrate complex nonlinear systems. It can be used in a wide range of applications and is well known for being adopted in internal combustion engine control calibration. When applied to e-motor control calibration, the model-based calibration workflow can help motor control engineers achieve optimal torque and field-weakening control for PMSMs.

PMSM Characterization and Calibration: Challenges and Requirements

PMSMs stand out from other types of e-motors because of their high efficiency and torque density. This is because the permanent magnets inside the machine can generate substantial air gap magnetic flux without external excitation. This special trait makes a PMSM an excellent candidate for both non-traction and traction motor drive applications.
Model-Based Calibration Workflow

DoE → Data Modeling → Calibration

Implementation
Model-Based Calibration Workflow

DoE → Data Modeling → Calibration

Implementation
Fitting Models
Calibration

Goal: find the best \((id, iq)\) operating points that can achieve pre-set optimization objective while satisfying certain physical constraints.
Model-Based Calibration Workflow

DoE → Data Modeling → Calibration

Implementation
Calibration Results – Fill Calibration Tables

\( i_q \) table (after clipping max torque)

\( i_q \) table (based on percentage of max torque)
Agenda
From Desktop Simulation to Software Deployment

- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model

- Algorithm design with simulation
  - Field-Oriented control
  - Controller tuning
  - Calibration

- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test
Simple Embedded Software Architecture

Most Development is on Core Software Algorithms

Communication Interfaces

Input Drivers

Core Software Algorithms and Logic

Scheduler/Operating System And Support Utilities

Output Drivers

Special Device Drivers

Actuators

Special Interfaces

ASAP2

CCP

Sensors

Drivers

Drivers
Custom Target, Hardware Support Package or Specialized toolbox

Hardware Support Packages: [https://www.mathworks.com/hardware-support/home.html](https://www.mathworks.com/hardware-support/home.html)
C2000 Microcontroller Blockset

**Supported devices:**
- F2802x/3x/5x/6x/07x/004x
- F2833x/32x/37xS/37xD/38xS/38xD
- Fixed-point F280x/1x
- F280013x / F280015x

F28379D LaunchPad

**Scheduling the generated code:**
- Periodic tasks
- Idle tasks
- Interrupts (Hardware, Software)
- Advanced concepts:
  - Pre-emptive rate-monotonic scheduler
  - Base rate interrupt replacement
  - Peripheral triggers (launch A/D conversion from PWM)
  - Running on the CLA
  - Loading in Flash, running in RAM
  - Using DMA
Supported TI C2000 drivers

- ADC, AIO, Comparator,
- GPIO, eQEP, ePWM, eCAP,
- eCAN, I2C, SCI, SPI, LIN
- Watchdog, DMA

- Motor control position sensing
  - Optical encoder (using eQEP)
  - Hall sensors (using eCAP)
  - Sensorless (using SMO)
Prepare the Model for Code Generation Using Supported TI C2000 Drivers Blocks
Deployment on the Target

- Generate code (floating and fixed-point)
- Use host model to control and debug
- Validate on hardware
Software-In-the-Loop (SIL) Testing

- Show equivalence, model to code
- Assess code execution time
- Collect code coverage

1. Desktop Simulation (on PC)
2. Model
3. Code Generator
4. Generated Code
5. PC Compiler
6. Object File
7. Object Code Execution (on PC)
8. Results
9. Test Vectors

Compare: == ?
Processor-In-the-Loop (PIL) Testing

- Verify numerical equivalence
- Assess target execution time
- Collect on target code coverage

Model

Desktop Simulation (on PC)

Generated Code

Cross Compiler

Object File

Object Code Execution (on target)

Test Vectors

Code Generator

Compare

Results

Results

Generated Code

= ?
Verify and Profile Code Using Processor-In-the-Loop (PIL) Testing

Code Execution Profiling Report for mcb_pmsm_foc_sim_v2/Current Control1

The code execution profiling report provides metrics based on data collected from a SIL or PIL execution. Execution times are calculated from data recorded by instrumentation probes added to the SIL or PIL test harness or inside the code generated for each component. See Code Execution Profiling for more information.

1. Summary

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time</td>
<td>50681790 ns</td>
</tr>
<tr>
<td>Unit of time</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>report(executionProfile, 'Units', 'seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');</td>
</tr>
<tr>
<td>Timer frequency (ticks per second)</td>
<td>2e-08</td>
</tr>
<tr>
<td>Profiling data created</td>
<td>16-Jan-2020 18:09:48</td>
</tr>
</tbody>
</table>

2. Profiled Sections of Code

<table>
<thead>
<tr>
<th>Section</th>
<th>Maximum Execution Time in ns</th>
<th>Average Execution Time in ns</th>
<th>Maximum Self Time in ns</th>
<th>Average Self Time in ns</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current_initialize</td>
<td>2260</td>
<td>2260</td>
<td>1365</td>
<td>1365</td>
<td>1</td>
</tr>
<tr>
<td>Current_step [5e-05 0]</td>
<td>5135</td>
<td>5067</td>
<td>5135</td>
<td>5067</td>
<td>10001</td>
</tr>
<tr>
<td>Current_terminate</td>
<td>540</td>
<td>540</td>
<td>540</td>
<td>540</td>
<td>1</td>
</tr>
</tbody>
</table>

3. CPU Utilization

<table>
<thead>
<tr>
<th>Task</th>
<th>Average CPU Utilization</th>
<th>Maximum CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current_step [5e-05 0]</td>
<td>10.13%</td>
<td>10.27%</td>
</tr>
<tr>
<td>Overall CPU Utilization</td>
<td>10.13%</td>
<td>10.27%</td>
</tr>
</tbody>
</table>

Profiling in real time execution
Deployment on the Target

- Generate code (floating and fixed-point)
- Use host model to control and debug
- Validate on hardware
Simple Embedded Software Architecture

Most Development is on Core Software Algorithms

Communication Interfaces

Input Drivers

Core Software Algorithms and Logic

Output Drivers

Special Device Drivers

Scheduler/Operating System And Support Utilities

Actuators

Special Interfaces

CCP

ASAP2

Drivers

Drivers

Sensors

Drivers

Drivers
Integrating Generated Controller Code with an Embedded Software Project

Execute at 20kHz

Controller

Command
ADC
Encoder

PWM

Embedded Software Project
Integrate Generated Controller Code with Your Hand-Coded Software Project

Embedded Software Project Pseudo-Code

```c
main()
{
    adcInit();
    encoderInit();
    pwmInit();
    controllerInit();
    while(1) {
    }
}

interruptServiceRoutine()
{
    AdcStruct = readAdcCountFromDriver();
    EncoderStruct = readEncoderCountFromDriver();

    PwmStruct = controllerStep(AdcStruct, EncoderStruct);

    writePwmCountToDriver(PwmStruct);
}
```
### Customize Generated Code

#### Speed Control Algorithm

![Diagram of Speed Control Algorithm](image)

**Copyright 2021 The MathWorks, Inc.**

**Explains more:**
1. Edit motor & inverter parameters
2. Generate C code using the ‘Embedded Coder’ app
3. Integrate generated code with driver code
4. Control motor via host model

<table>
<thead>
<tr>
<th>Code Mapping</th>
<th>Component Interface</th>
<th>Function Default</th>
<th>Function Name</th>
<th>Function Name ( recreated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Mode default</td>
<td>speed.controls.alarm</td>
<td>speed.controls.alarm (recreated)</td>
<td>speed.controls.alarm (recreated)</td>
</tr>
<tr>
<td>Description</td>
<td>Mode default</td>
<td>speed.controls.alarm</td>
<td>speed.controls.alarm (recreated)</td>
<td>speed.controls.alarm (recreated)</td>
</tr>
</tbody>
</table>

---

**Code Snippet:**

```c
// Code generated for Simulink model 'speed.controls.alarm'.

// Copyright 2021 The MathWorks, Inc.

// This code is generated by the Embedded Coder application.

// Generated by: Embedded Coder 12.1 (R2021b) 25-Mar-2021 14:03:35

// Embedded Coder Environment: Microsoft Windows 64-bit 8.1 (build 7601)

// File: speed.controls.alarm

// This file contains the code generated by the 'speed.controls.alarm' model.

// This code contains:

// - Model-Generated Code

// Generated code:

// - Function Definitions

// - Include Files

// - Variables

// - Functions

// - Constants

// - Preprocessor Definitions

// - Data Types

// - Data Structures

// - Structures

// - Typedefs

// - Enumerations

// - Variables

// - Annotations

// - Comments

// Generated code:

// - Function Definitions

// - Include Files

// - Variables

// - Functions

// - Constants

// - Preprocessor Definitions

// - Data Types

// - Data Structures

// - Structures

// - Typedefs

// - Enumerations

// Generated code:

// - Function Definitions

// - Include Files

// - Variables

// - Functions

// - Constants

// - Preprocessor Definitions

// - Data Types

// - Data Structures

// - Structures

// - Typedefs

// - Enumerations

// Generated code:

// - Function Definitions

// - Include Files

// - Variables

// - Functions

// - Constants

// - Preprocessor Definitions

// - Data Types

// - Data Structures

// - Structures

// - Typedefs

// - Enumerations

// Generated code:

// - Function Definitions

// - Include Files

// - Variables

// - Functions

// - Constants

// - Preprocessor Definitions

// - Data Types

// - Data Structures

// - Structures

// - Typedefs

// - Enumerations
```
Basic generated code architecture & interface

- **Basic generated functions format:**
  - `void modelname_initialize(void)`: to call at system initialization
  - `Void modelname_step(void)`: to call each processing step (on timer or interrupt)
  - `Void modelname_terminate(void)`: to call at system shutdown.

- **Several possible customisation using Embedded coder**
  - Functions / files names
  - Function interface (return & argument passing mode).
  - Several step functions for concurrent tasking.
  - ...

  example: `int MyModel_step(int inputs, *int params, *int dworks)`
Complete Model-Based Design Workflow

Get the complete confidence in your design

1. Textual Requirements
2. Executable Specification
3. Model used for production code generation
4. Generated C/C++ code
5. Code Generation
6. Compilation and Linking
7. Component and system testing
8. Review and static analysis
9. Equivalence checking
10. Static Code Analysis
11. Equivalence testing

Get the complete confidence in your design.
**Complete Model-Based Design Workflow**

Get the complete confidence in your design
Key Takeaways

- Model-based design for motor control enables you
  - To make faster time to market.
  - To be more resilient to external disturbances
  - Increase collaboration
  - Share knowledge

- Motor Control Blockset, using reference examples & built-in blocks, enables you
  - To minimize even more development time
  - Upskill yourself
  - To experiment easily new control technics
Q&A