MATLAB EXPO

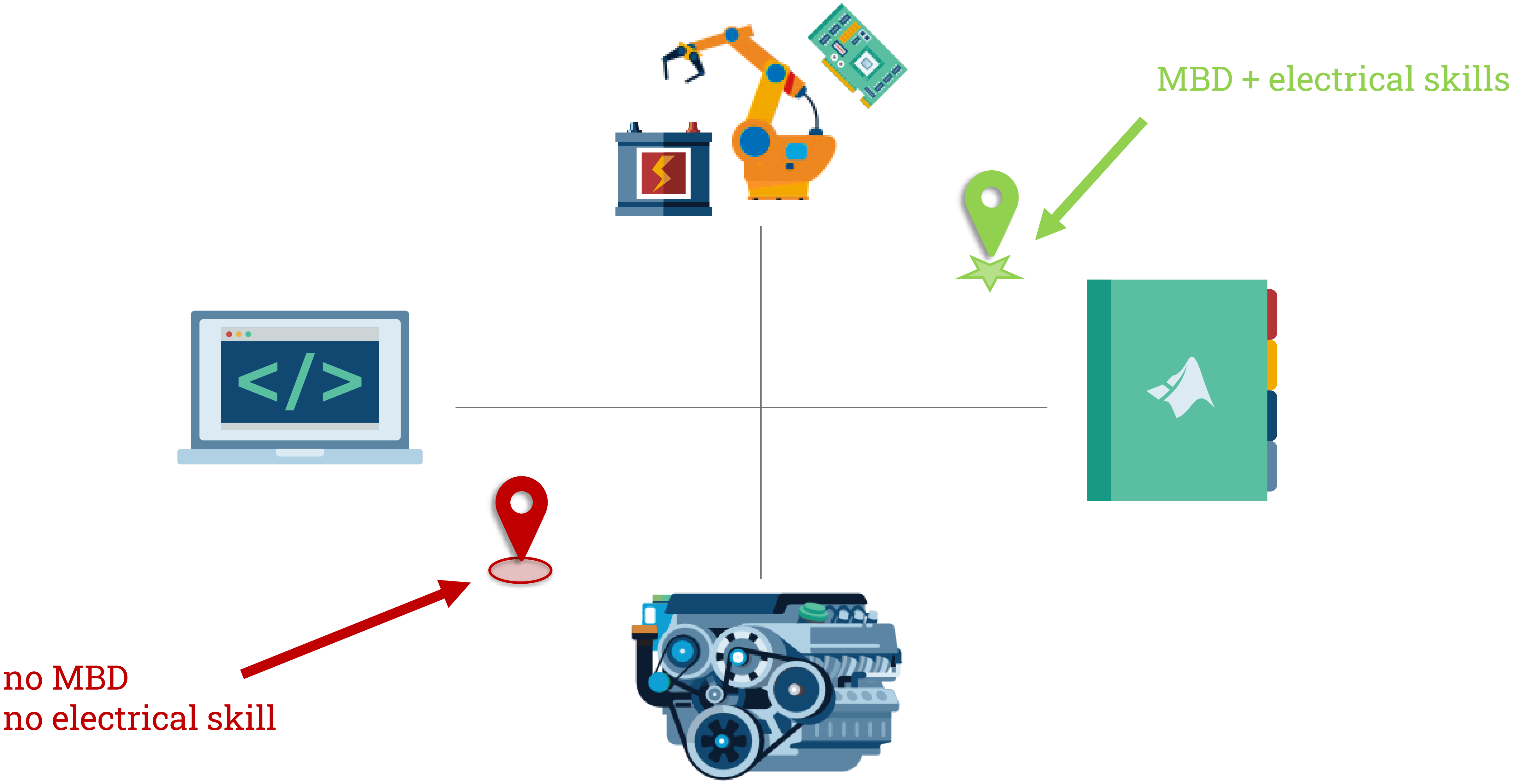
FRANCE

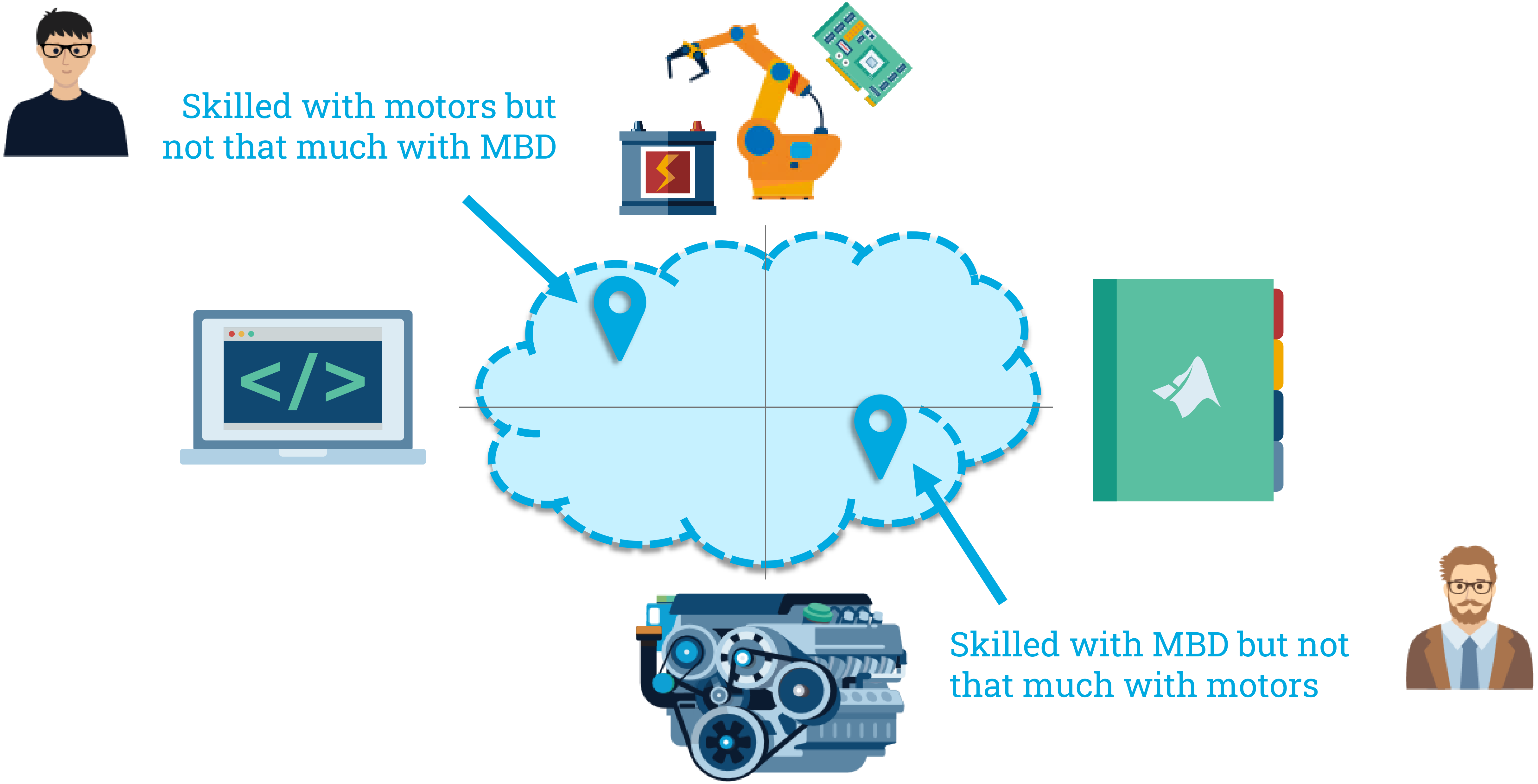
Electrifier vos systèmes avec Simulink

Kevin Roblet, MathWorks

Morgan Fremovici, MathWorks

MathWorks®

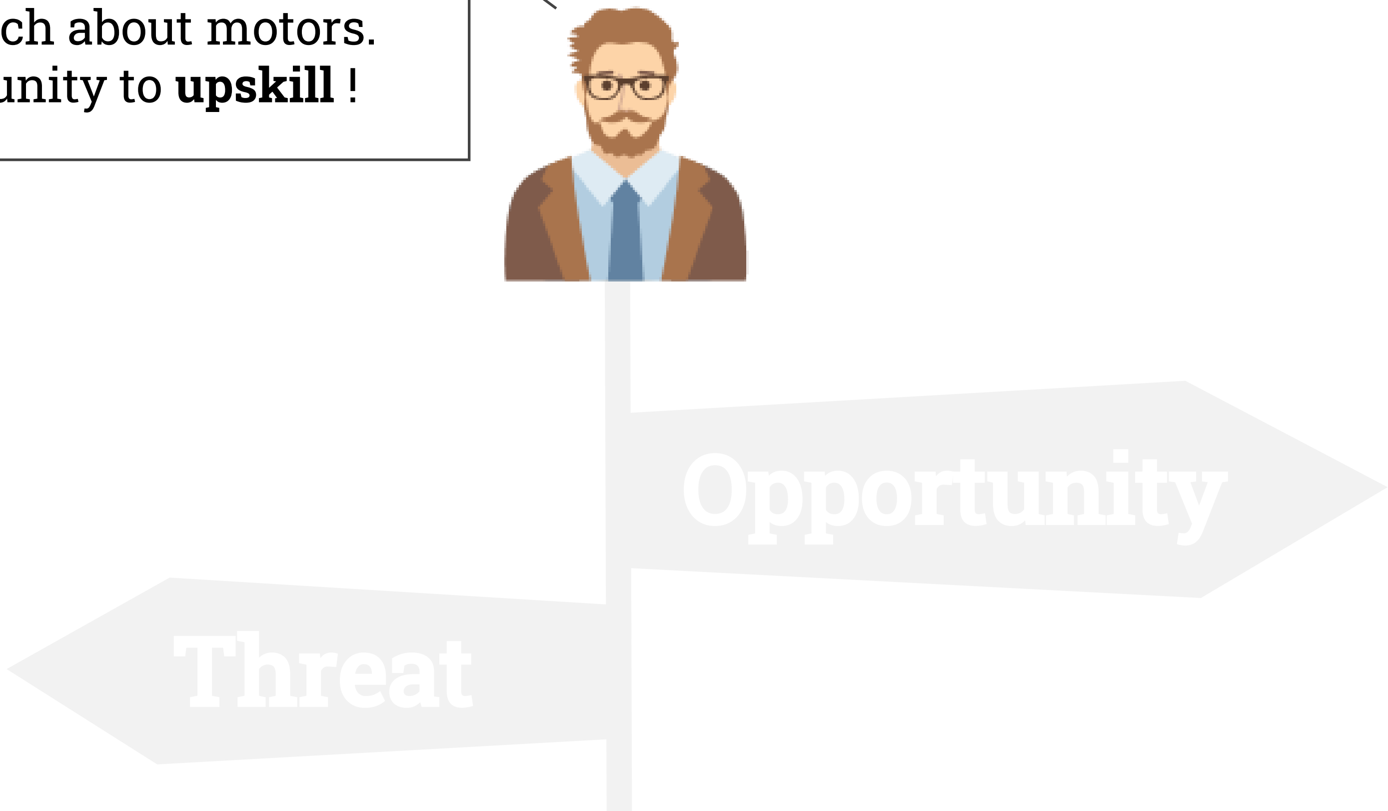

MBD + electrical skills
no MBD
no electrical skill


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** !

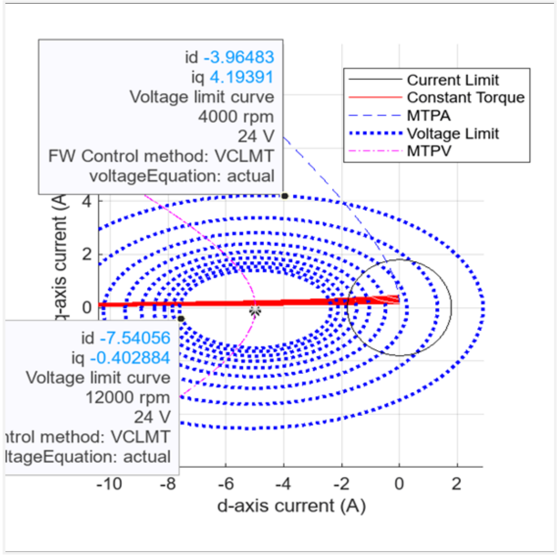
I know how to get the best of IA with simulation.
Let's differentiate with **innovative features** !

I do not know that much about motors.
Let's take this opportunity to **upskill** !

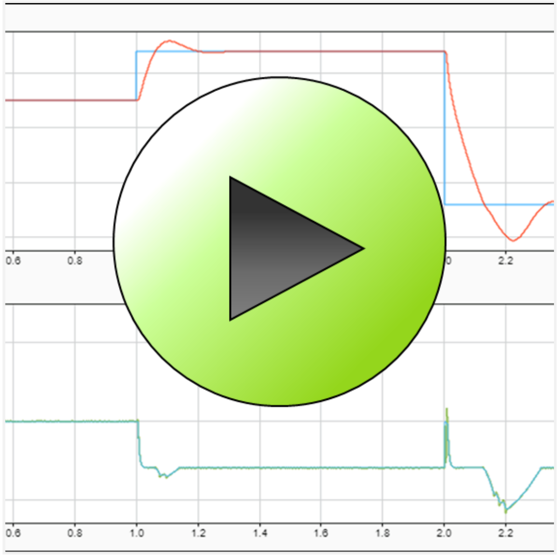


models to capture knowledge and as a collaboration and communication tool

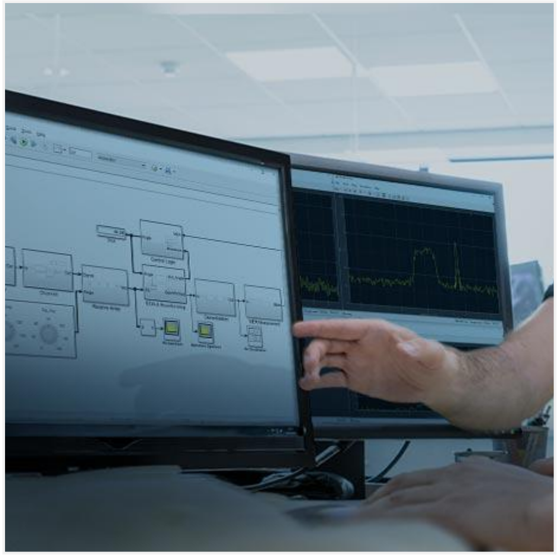
Documentation



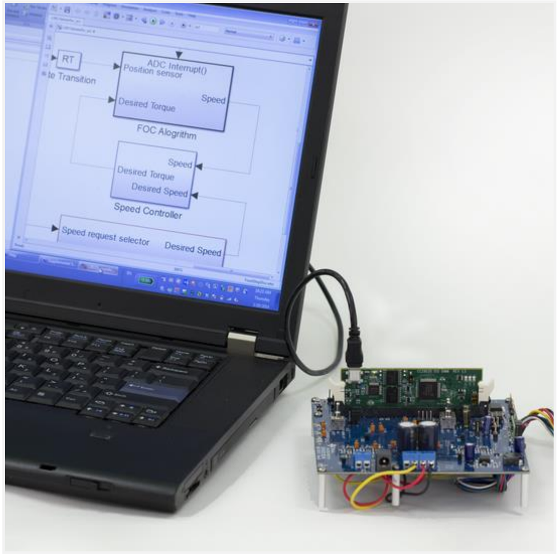
Simulation



Collaboration



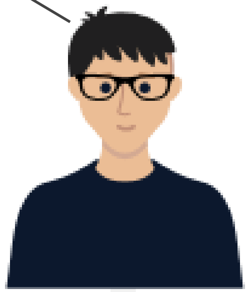
Hands-on kit



Our motor expertise is critical, how do we…
… ensure **knowledge flow** in the company ?
… handle software **maintainability** ?
… **train** new hires ?

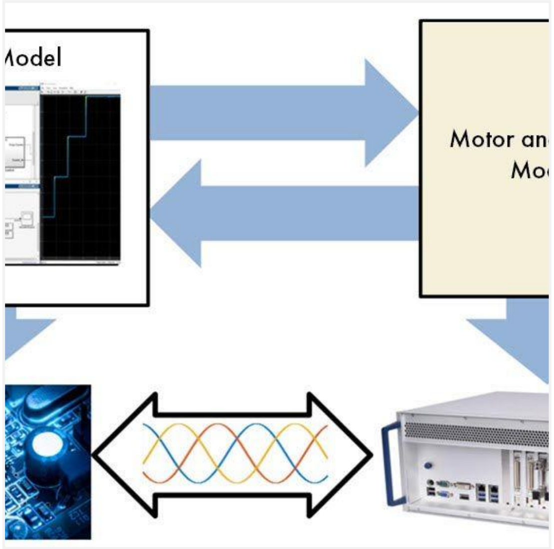
Raw material prices, supply chain disruptions… we need to…
… **depend less on prototypes**
… **get more job done in simulation**
… be **less dependent on specific chips**

Our customers have new expectations, let's…
… deploy on **FPGA** !
… achieve **certification** !

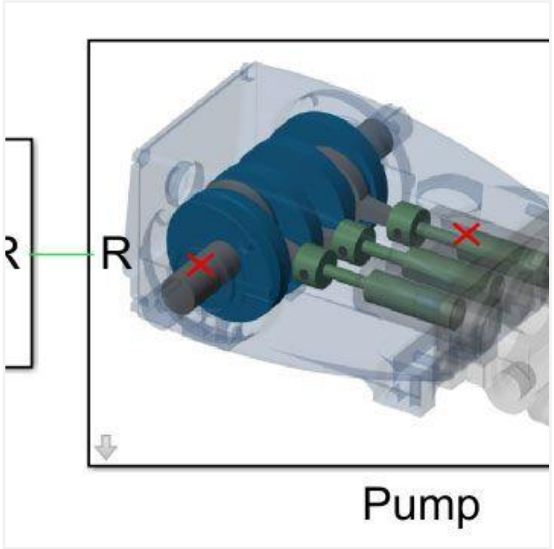


models to try out new ideas through short agile iteration cycles

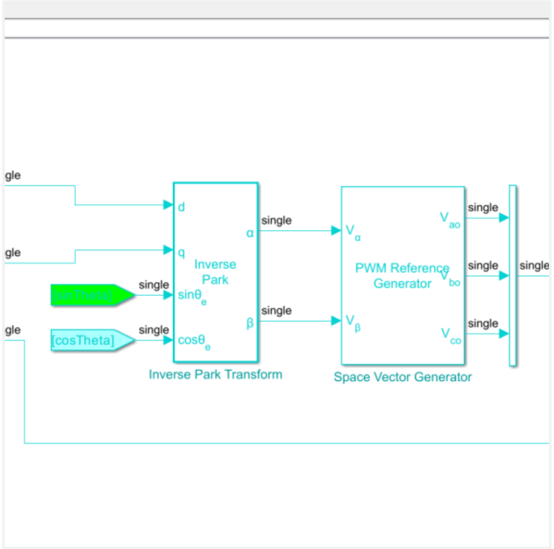
Virtual prototyping



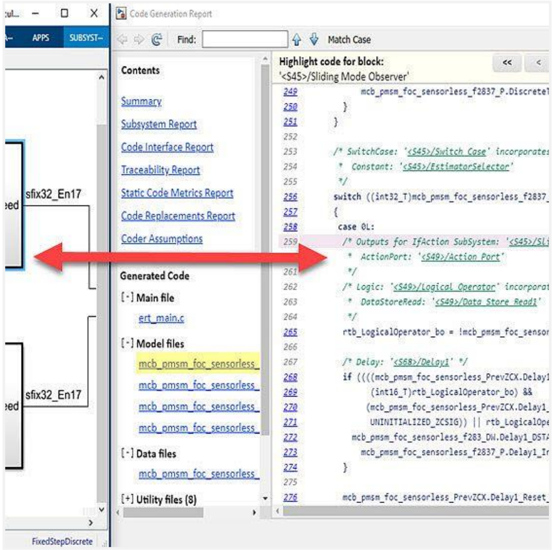
System-level simulation



Model elaboration

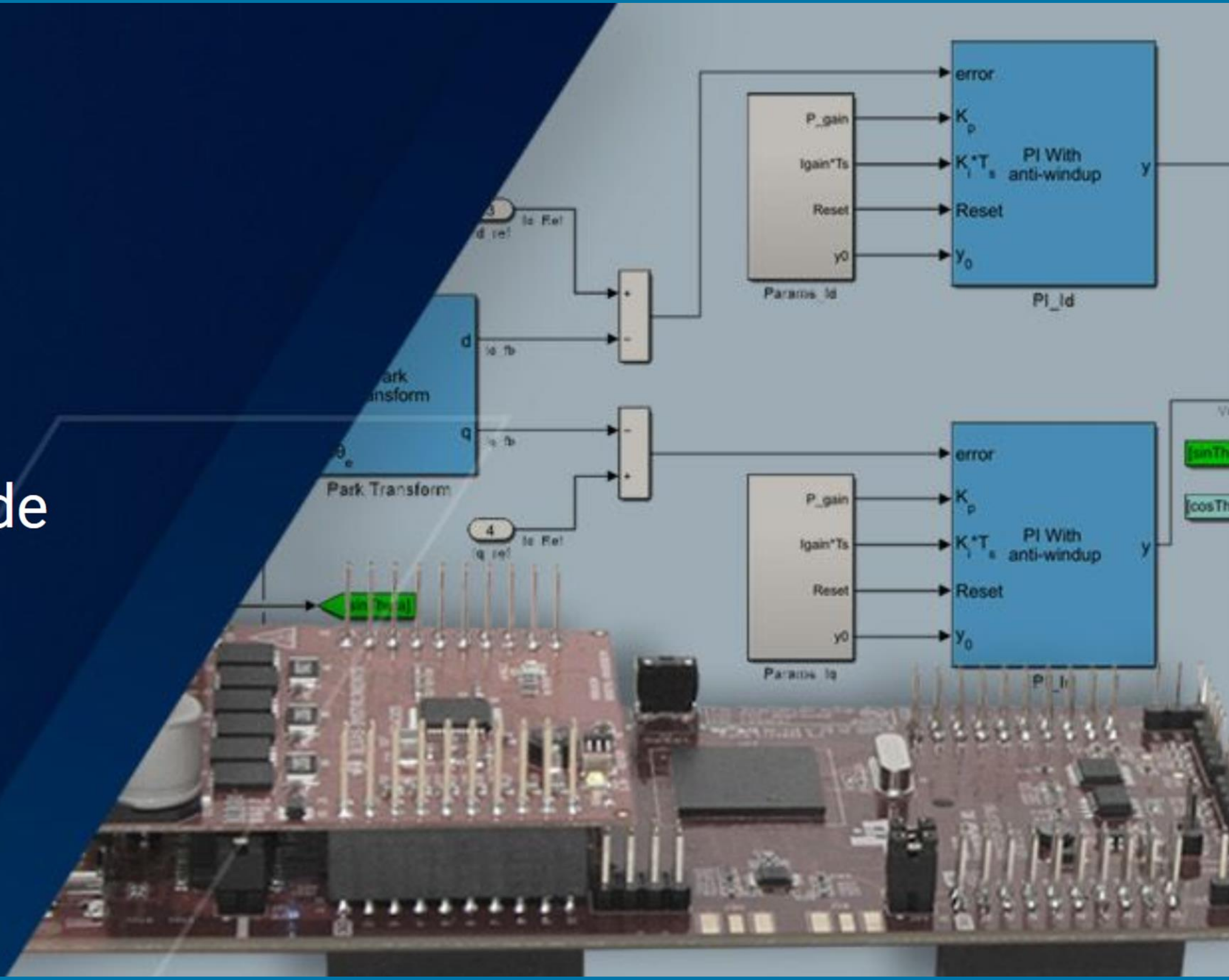


Automation



Motor Control Blockset

Concevoir et implémenter des algorithmes de contrôle moteur



KNOWLEDGE

AGILE

RESILIENT

LEAN

TIME TO MARKET

COST

INNOVATION

QUALITY

UPSKILL

KNOWLEDGE

AGILE

RESILIENT

LEAN

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

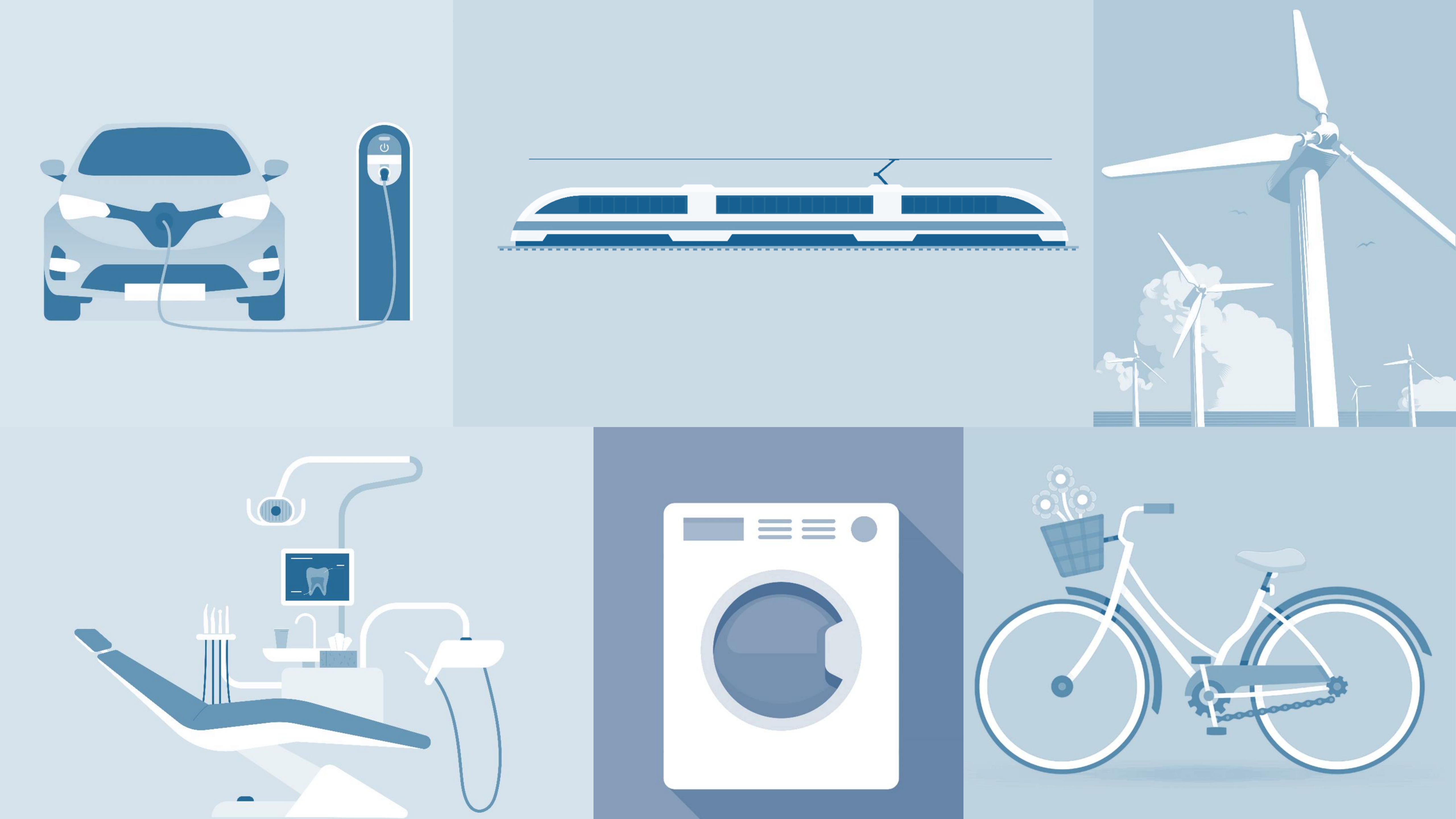
TIME TO MARKET

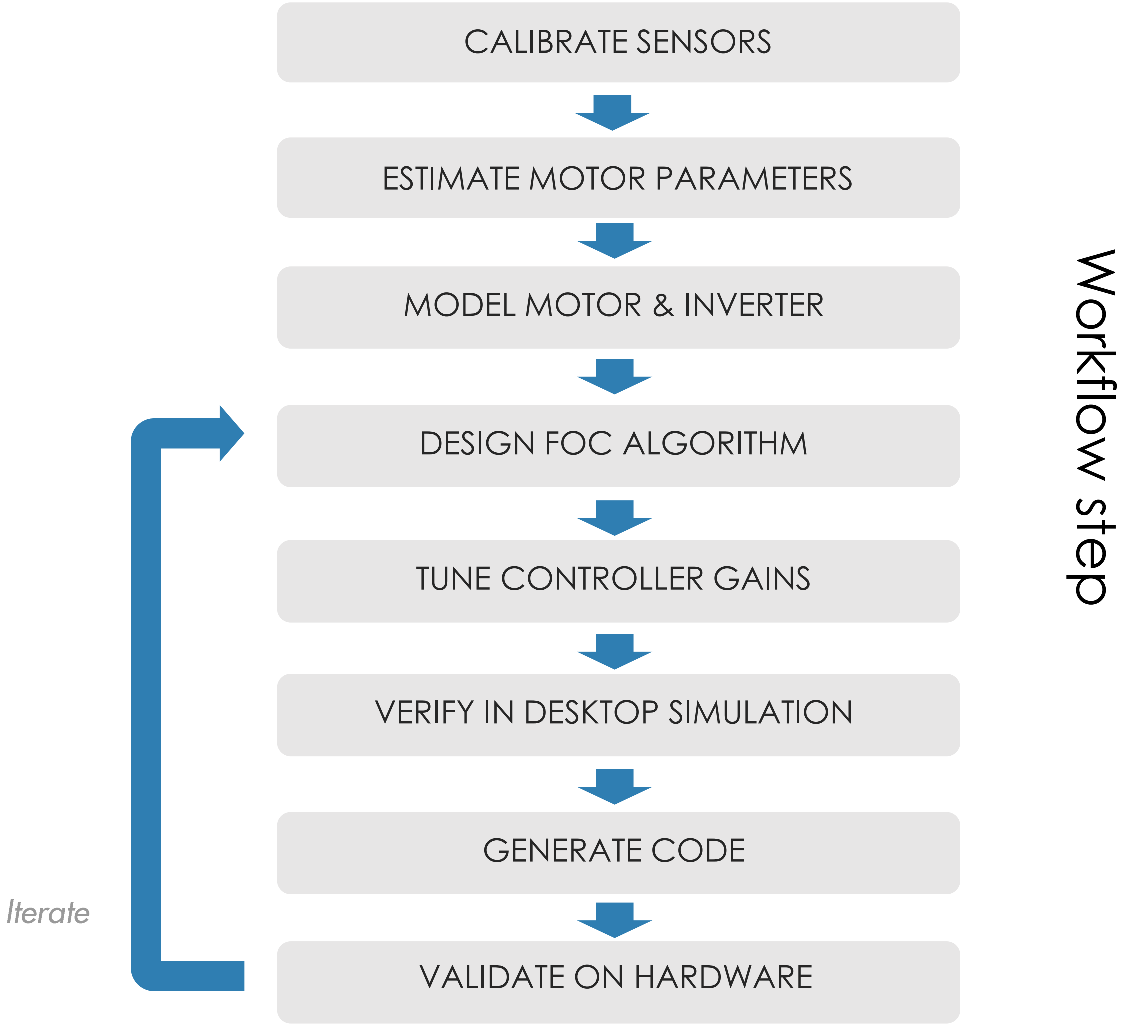
COST

INNOVATION

QUALITY

UPSKILL

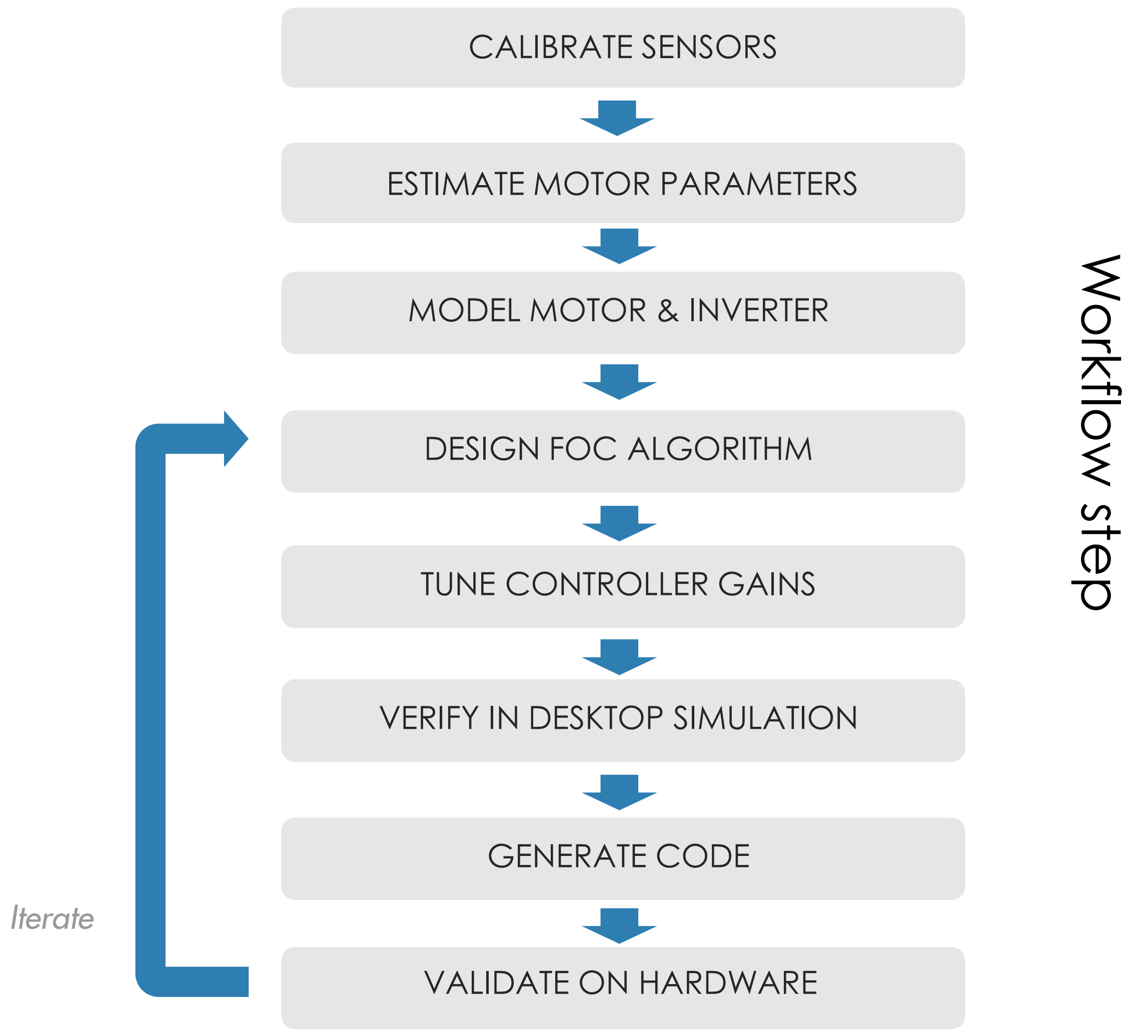



CALIBRATE SENSORS
ESTIMATE MOTOR PARAMETERS
MODEL MOTOR & INVERTER
DESIGN FOC ALGORITHM
TUNE CONTROLLER GAINS
VERIFY IN DESKTOP SIMULATION
GENERATE CODE
VALIDATE ON HARDWARE
Iterate
Workflow step

Agenda

From Desktop Simulation to Software Deployment

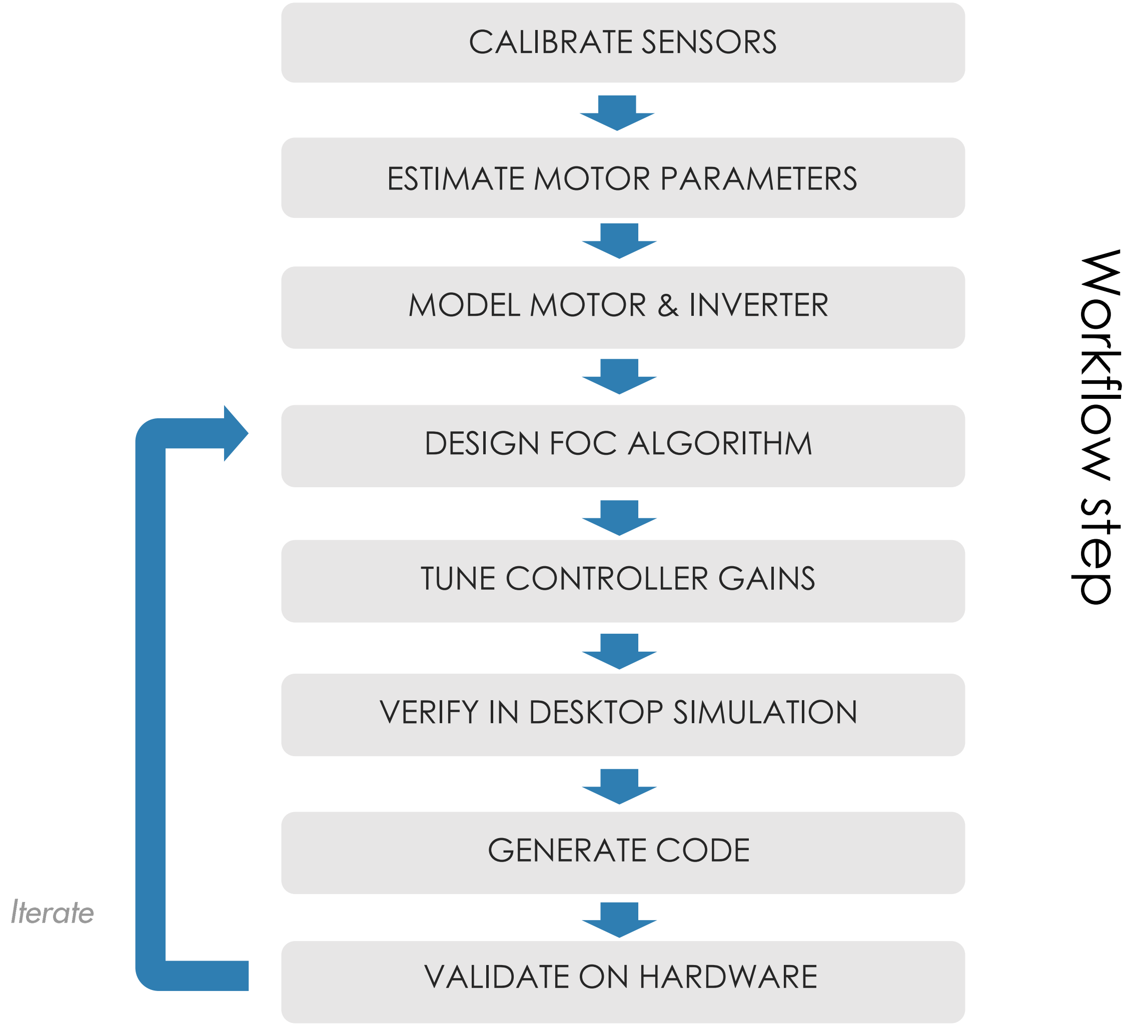
- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model
- Algorithm design with simulation
  - Field-Oriented control
  - Controller tuning
  - Calibration
- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test



Agenda

From Desktop Simulation to Software Deployment

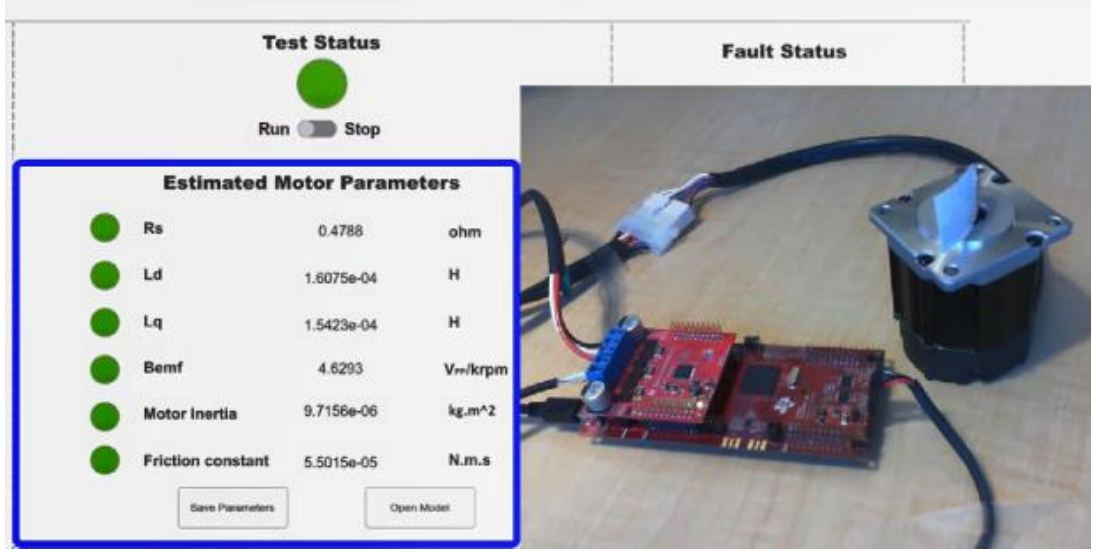
- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model
- Algorithm design with simulation
  - Field-Oriented control
  - Controller tuning
  - Calibration
- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test



Motor Parameters Estimation

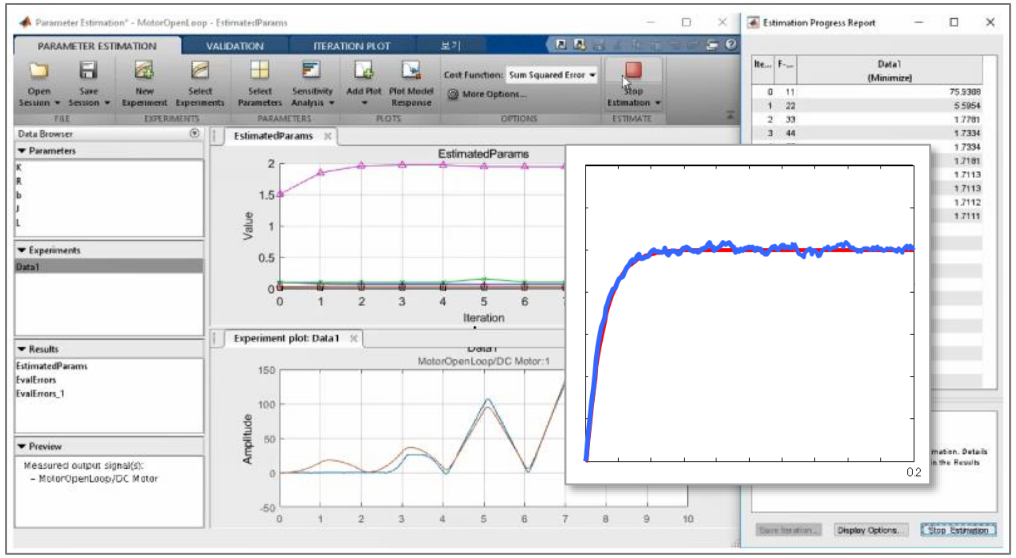
Plant Modeling

Two types of parameter estimation methods:



Parameter Estimation with Instrumented Test

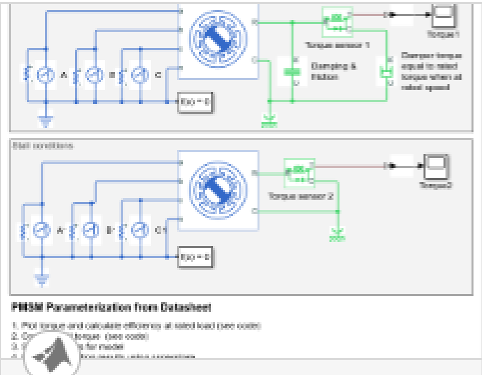
or



Parameter Estimation using Operation Data

Other techniques to parameterize motor models

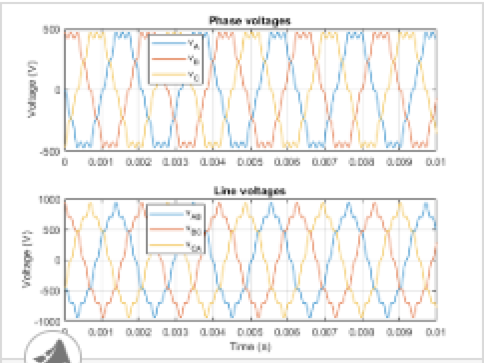
Bonus



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

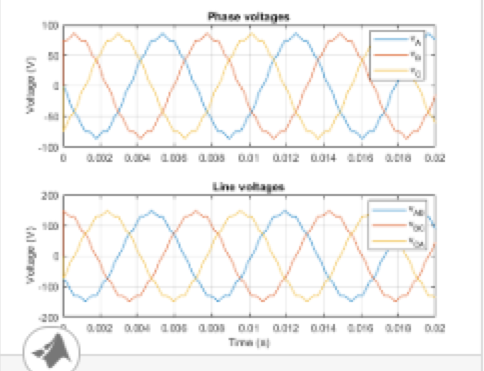
Open Model



Import IPMSM Flux Linkage Data from ANSYS Maxwell

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

Open Model



Import IPMSM Flux Linkage Data from Motor-CAD

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

Open Model

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.

Workflow	Description
Step 1: Load and Preprocess Data	Load and preprocess this nonlinear motor flux data from dynamometer testing or finite element analysis (FEA): • *d*- and *q*- axis current • *d*- and *q*- axis flux • Electromagnetic motor torque
Step 2: Generate Evenly Spaced Table Data From Scattered Data	Use the `gridfit` function to generate evenly spaced data. Visualize the flux surface plots.
Step 3: Set Block Parameters	Set workspace variables that you can use for the Flux-Based PM Controller block parameters.

From datasheet

Simscape Electrical

From ANSYS Maxwell, JMAG, Motor-CAD FEA tools

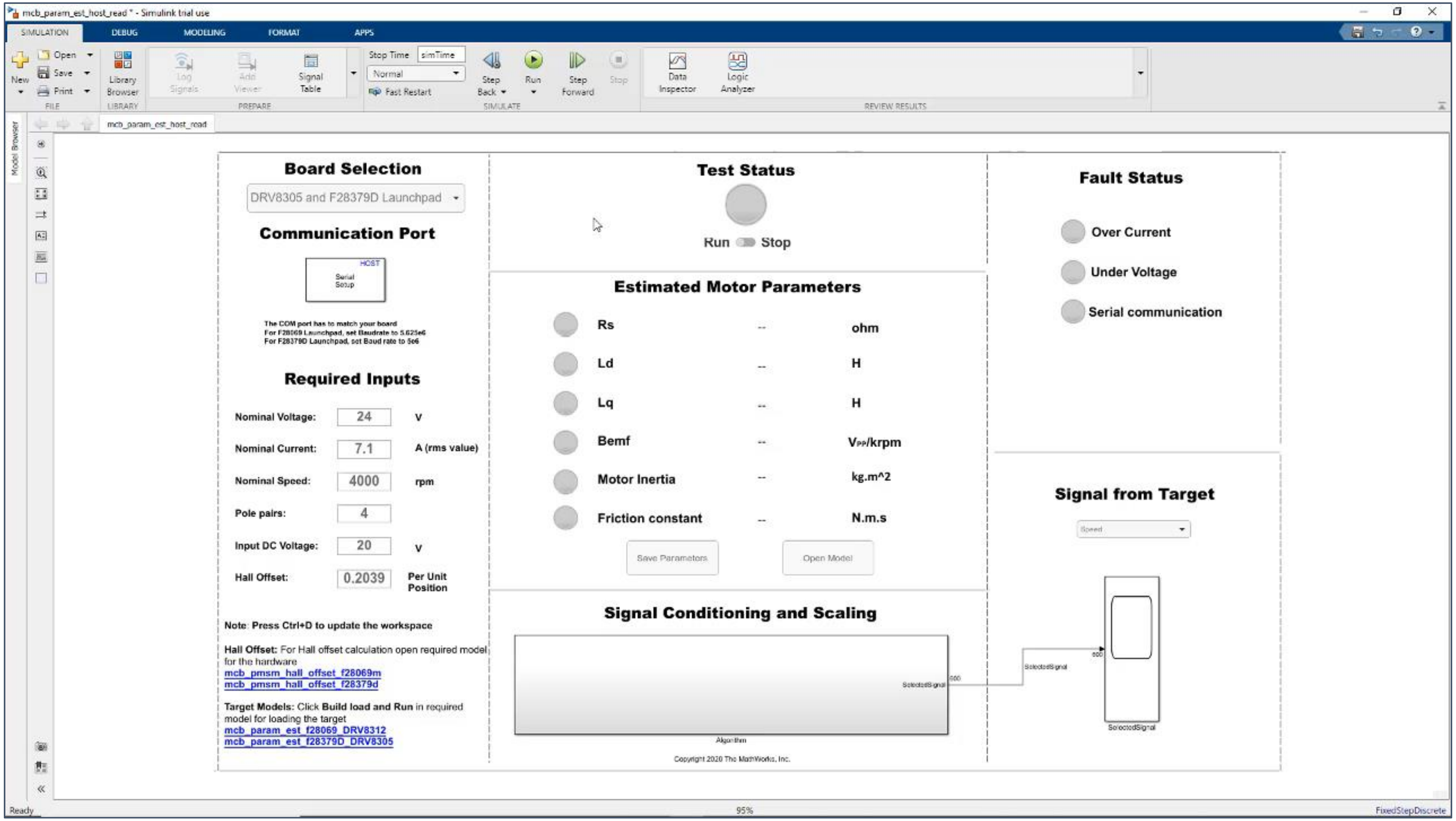
Simscape Electrical

From dyno data

Powertrain Blockset

Motor Parameters Estimation - Instrumented Test

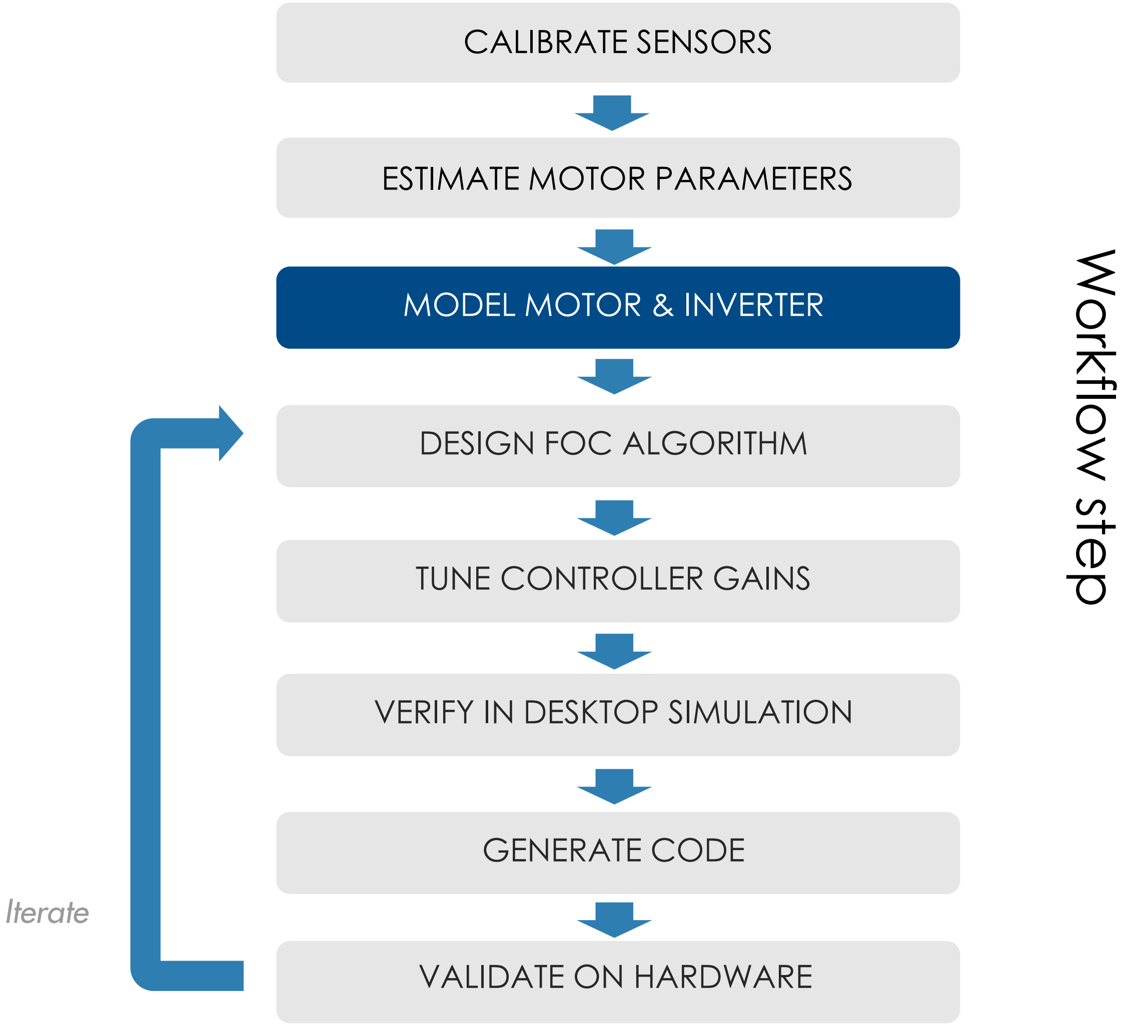
Plant Modeling



Agenda

From Desktop Simulation to Software Deployment

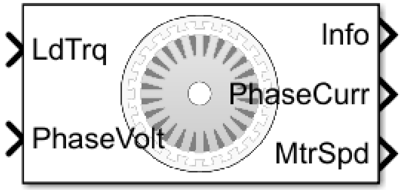
- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model
- Algorithm design with simulation
  - Field-Oriented control
  - Controller tuning
  - Calibration
- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test



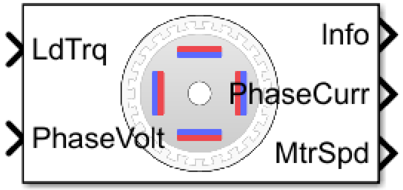
Motor and Inverter Modeling

Choose the right level of fidelity

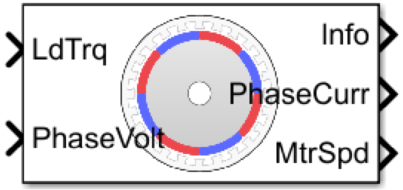
- Use linear lumped-parameter model shipped with Motor Control Blockset



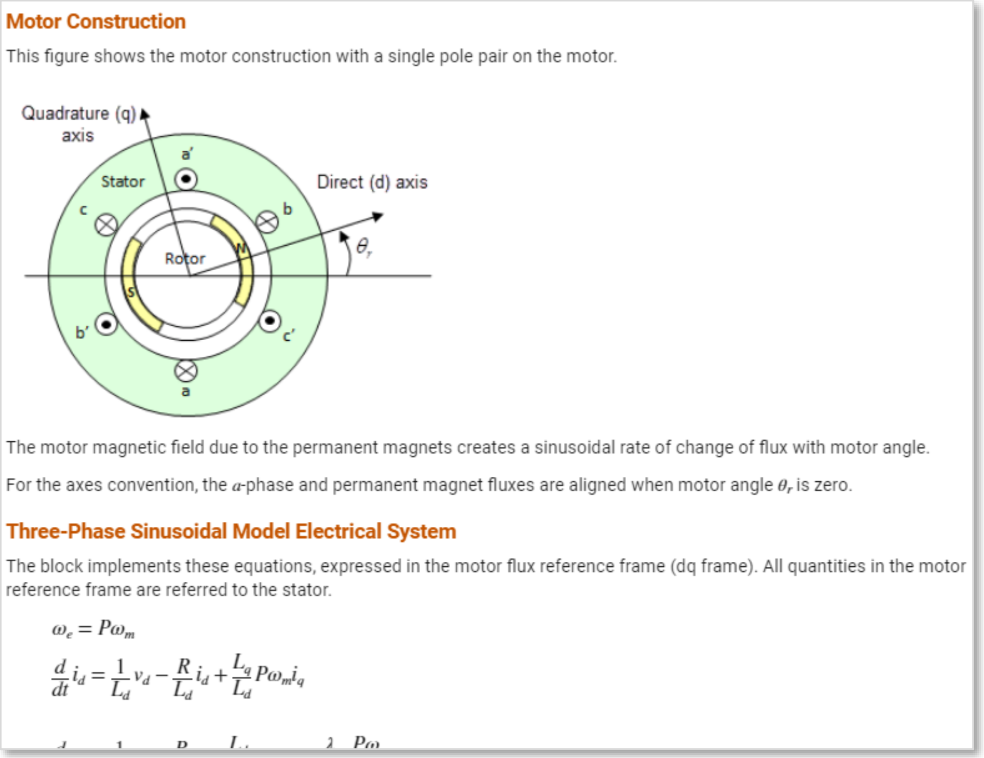
Induction Motor



Interior PMSM



Surface Mount PMSM


Motor Construction

This figure shows the motor construction with a single pole pair on the motor.

The motor magnetic field due to the permanent magnets creates a sinusoidal rate of change of flux with motor angle.

For the axes convention, the *a*-phase and permanent magnet fluxes are aligned when motor angle θ_r is zero.

Three-Phase Sinusoidal Model Electrical System

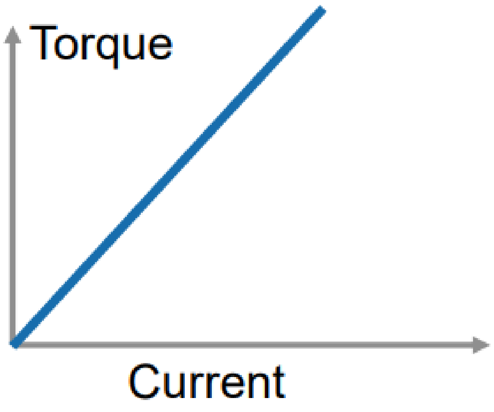
The block implements these equations, expressed in the motor flux reference frame (dq frame). All quantities in the motor reference frame are referred to the stator.

$$\omega_e = P\omega_m$$

$$\frac{d}{dt}i_d = \frac{1}{L_d}v_d - \frac{R}{L_d}i_d + \frac{L_q}{L_d}P\omega_m i_q$$

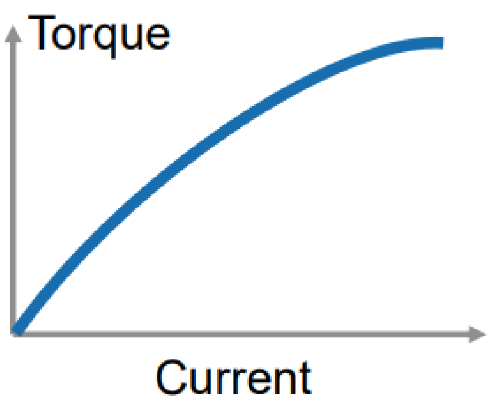
Model Fidelity

Plant Modeling



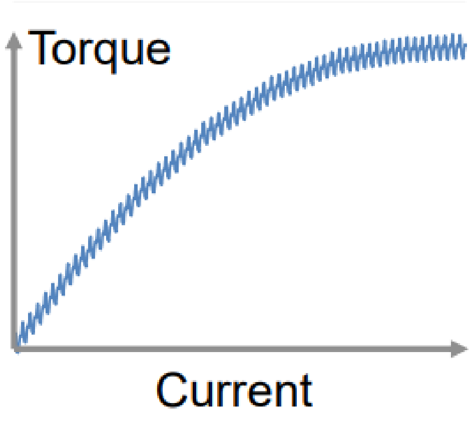
Linear Lumped Parameter

Motor Control Blockset
Simscape Electrical



Saturation

Simscape Electrical



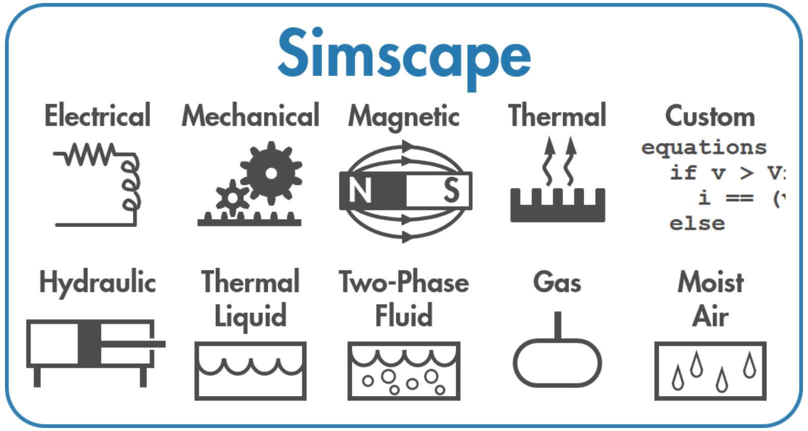
Saturation &
Spatial Harmonics

Simscape Electrical

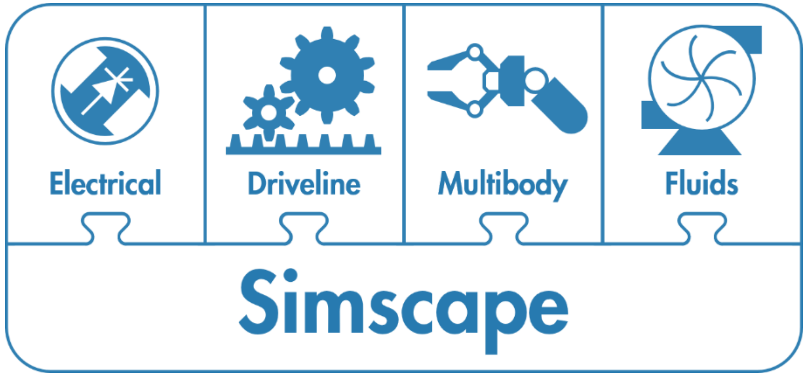
Simscape Products

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



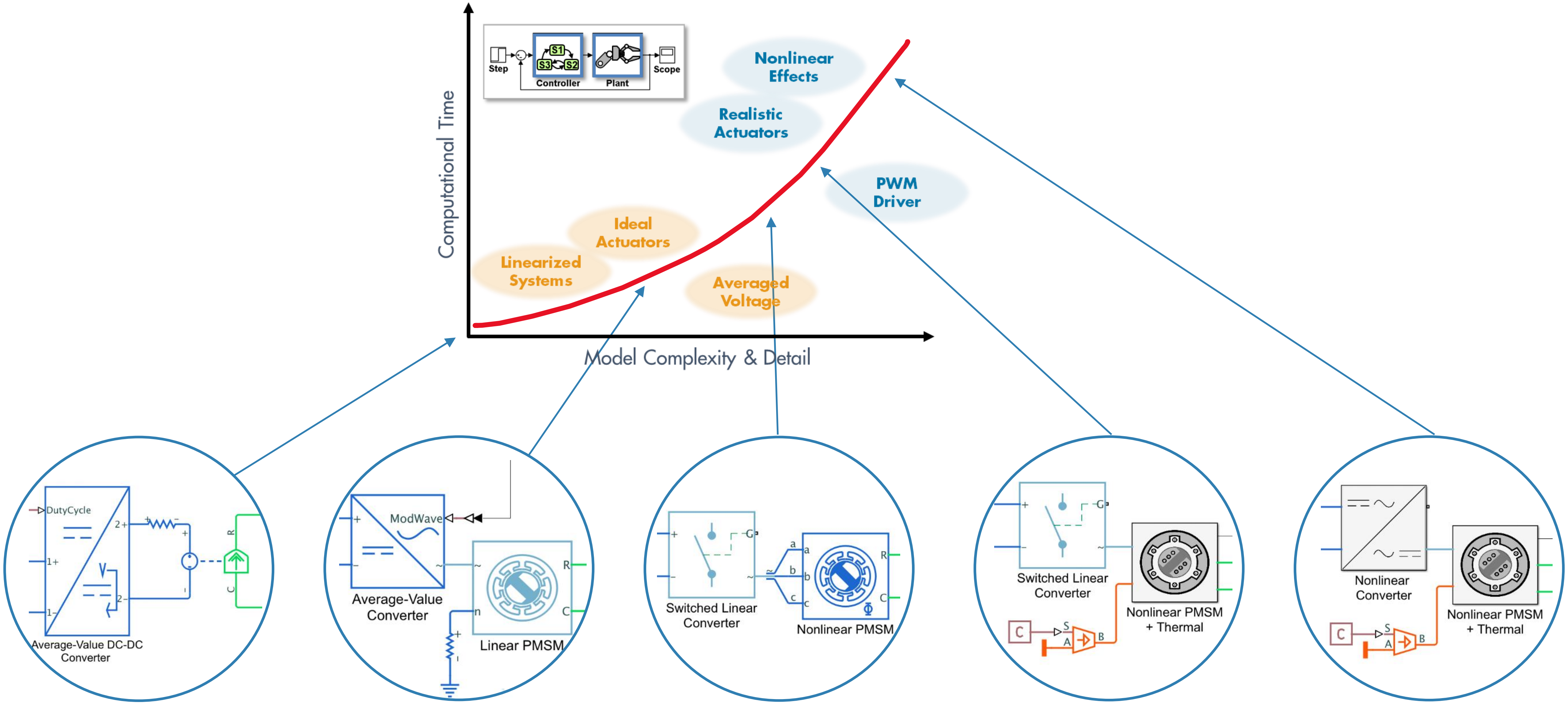
Simscape Foundation



Simscape Add-Ons

Trade Off - Balance Model Fidelity vs Simulation Speed

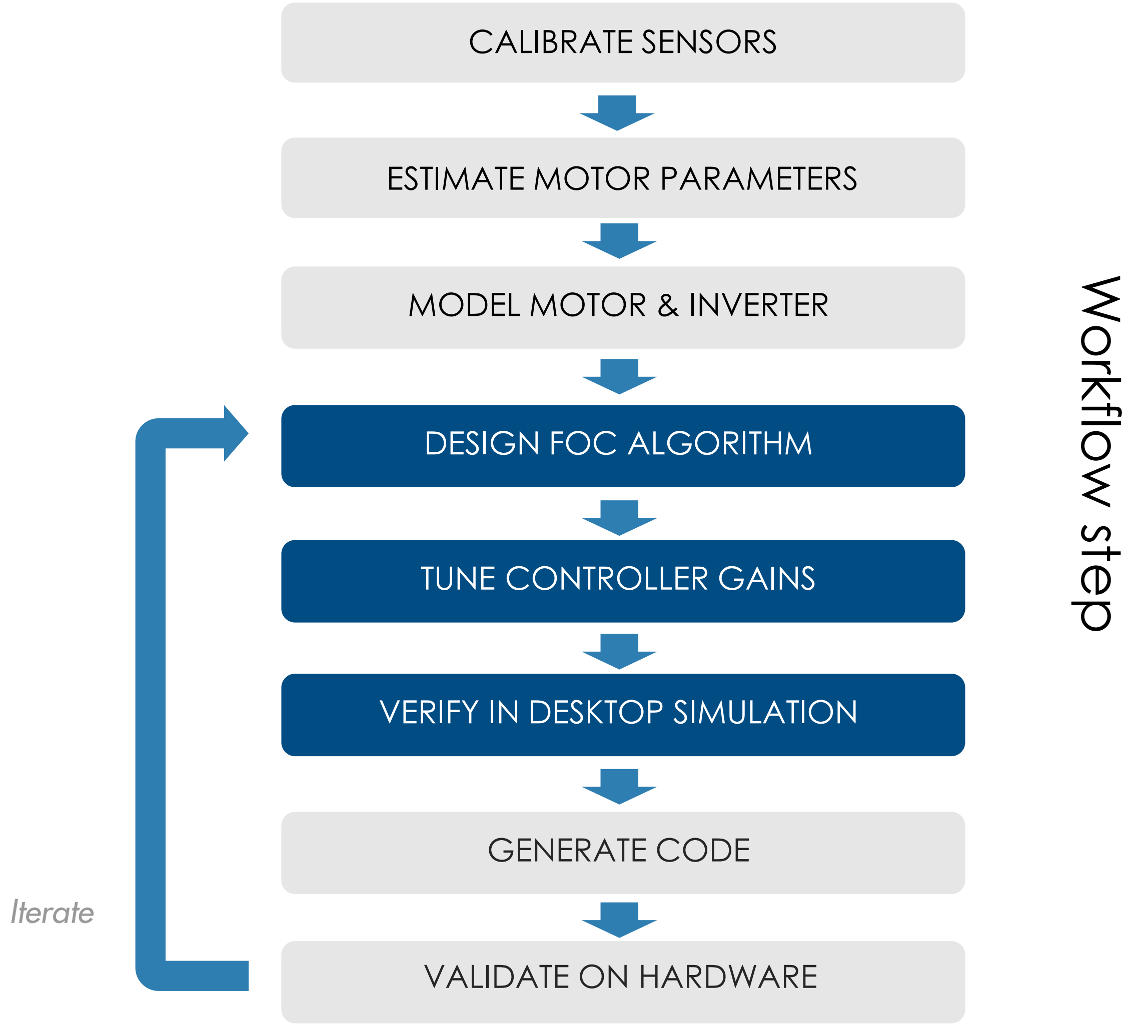
Plant Modeling



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

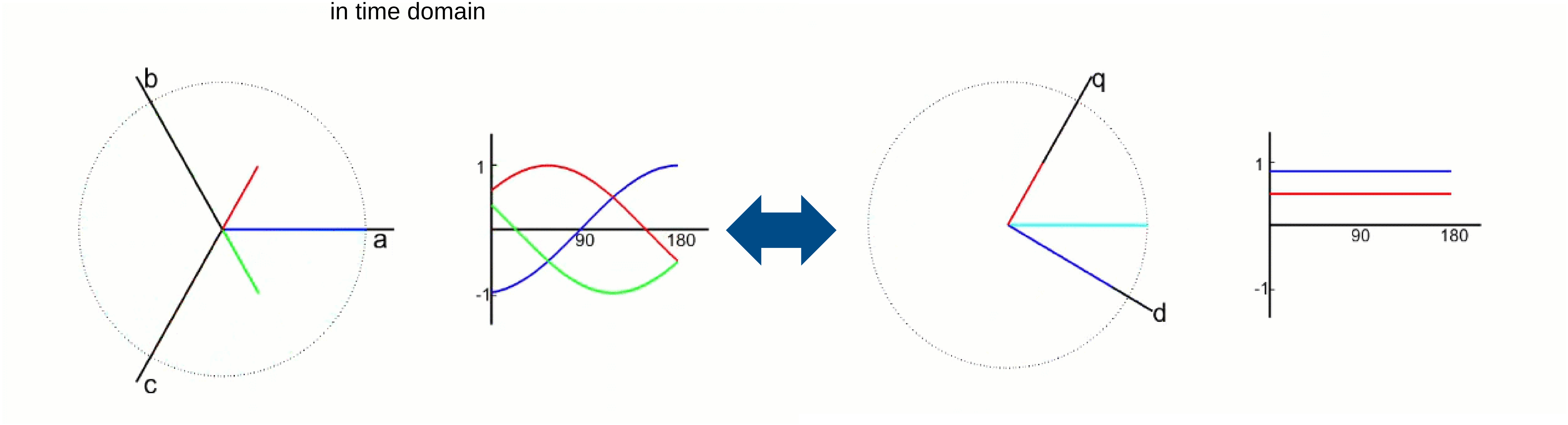
Calibration

Modeling Field-Oriented Control (FOC)

A word about transforms

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

Current control (d, q)

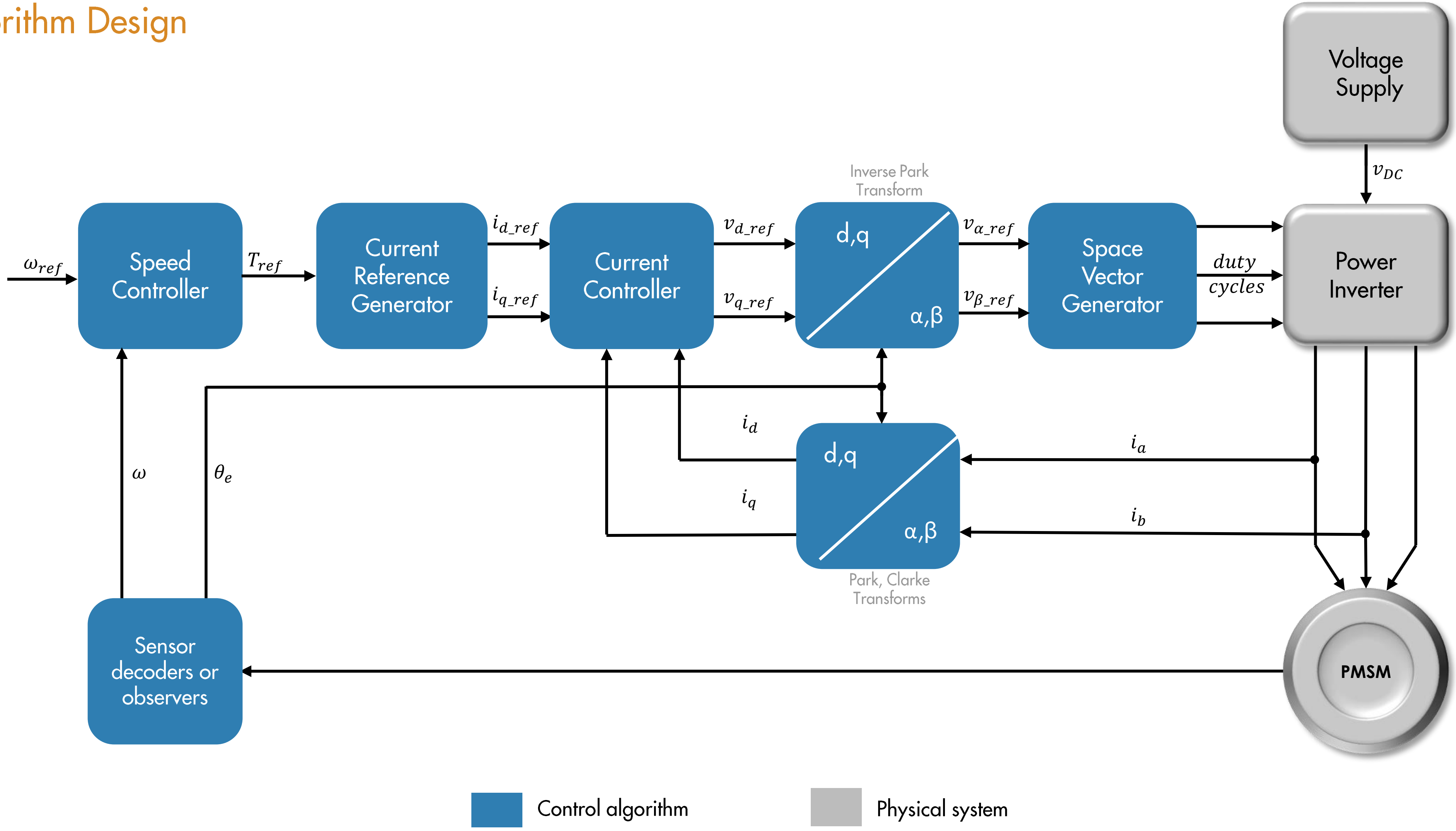


Clarke transform (abc ←→$\alpha\beta$)

Park transform ($\alpha\beta$ ←→ dq)

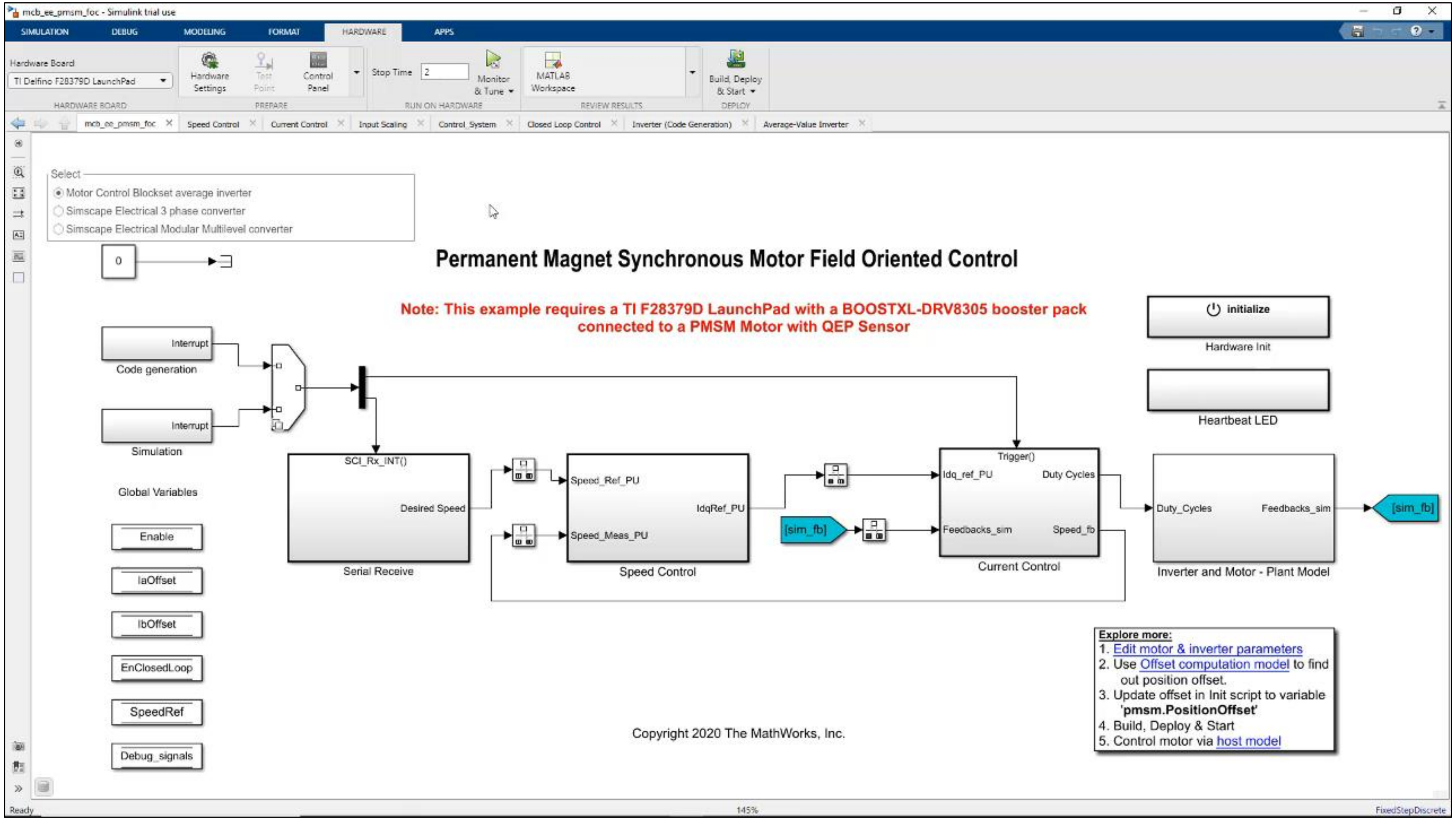
Modeling Field-Oriented Control (FOC)

Algorithm Design



Modeling Field-Oriented Control (FOC)

Overview of the model



Field-Oriented control

Controller tuning

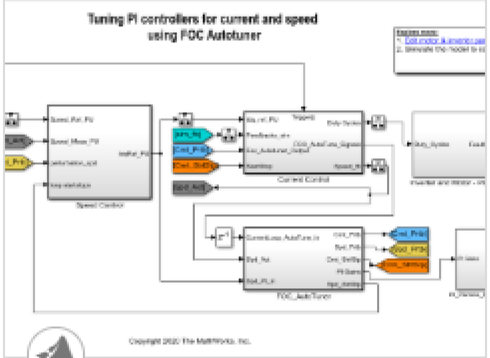
Calibration

Modeling Field-Oriented Control (FOC)

```
%% Set PWM Switching frequency
PWM_frequency   = 20e3;    %Hz          // converter s/w freq
T_pwm           = 1/PWM_frequency;  %s  // PWM switching time period

%% Set Sample Times
Ts              = T_pwm;        %sec        // sample time for controller
Ts_simulink     = T_pwm/2;      %sec        // simulation time step for model simulation
Ts_motor        = T_pwm/2;      %Sec        // simulation sample time
Ts_inverter     = T_pwm/2;      %sec        // simulation time step for average value inverter
Ts_speed        = 10*Ts;        %Sec        // sample time for speed controller

%% Set data type for controller & code-gen
% dataType = fixdt(1,32,17);    % Fixed point code-generation
dataType = 'single';            % Floating point code-generation

%% System Parameters // Hardware parameters

pmsm = mcb_SetPMSMMotorParameters('BLY171D');

%% Parameters below are not mandatory for offset computation

inverter = mcb_SetInverterParameters('DRV8312-C2-KIT');

inverter.ADCOffsetCalibEnable = 1; % Enable: 1, Disable:0

target = mcb_SetProcessorDetails('F28069M',PWM_frequency);

%% Derive Characteristics
pmsm.N_base = mcb_getBaseSpeed(pmsm,inverter); %rpm // Base speed of motor at given Vdc
% mcb_getCharacteristics(pmsm,inverter);

%% PU System details // Set base values for pu conversion

PU_System = mcb_SetPUSystem(pmsm,inverter);

%% Controller design // Get ballpark values!

PI_params = mcb.internal.SetControllerParameters(pmsm,inverter,PU_System,T_pwm,Ts,Ts_speed);
```

Empirical Computation

Motor Control Blockset



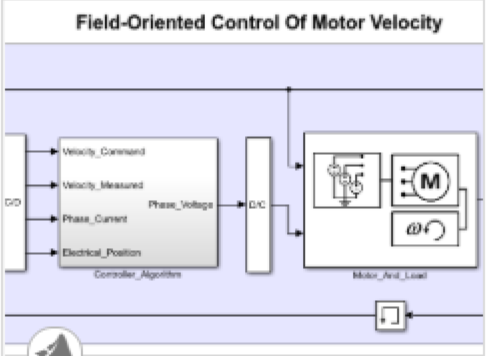
Tune PI controllers by Using Field Oriented Control (FOC) Autotuner

Computes the gain values of the PI controllers within the speed and current controllers by using the Field Oriented Control Autotuner block.

Open Example

FOC Autotuner

Motor Control Blockset



Tune Field-Oriented Controllers Using SYSTUNE

Tune a field-oriented controller for an asynchronous machine in one simulation.

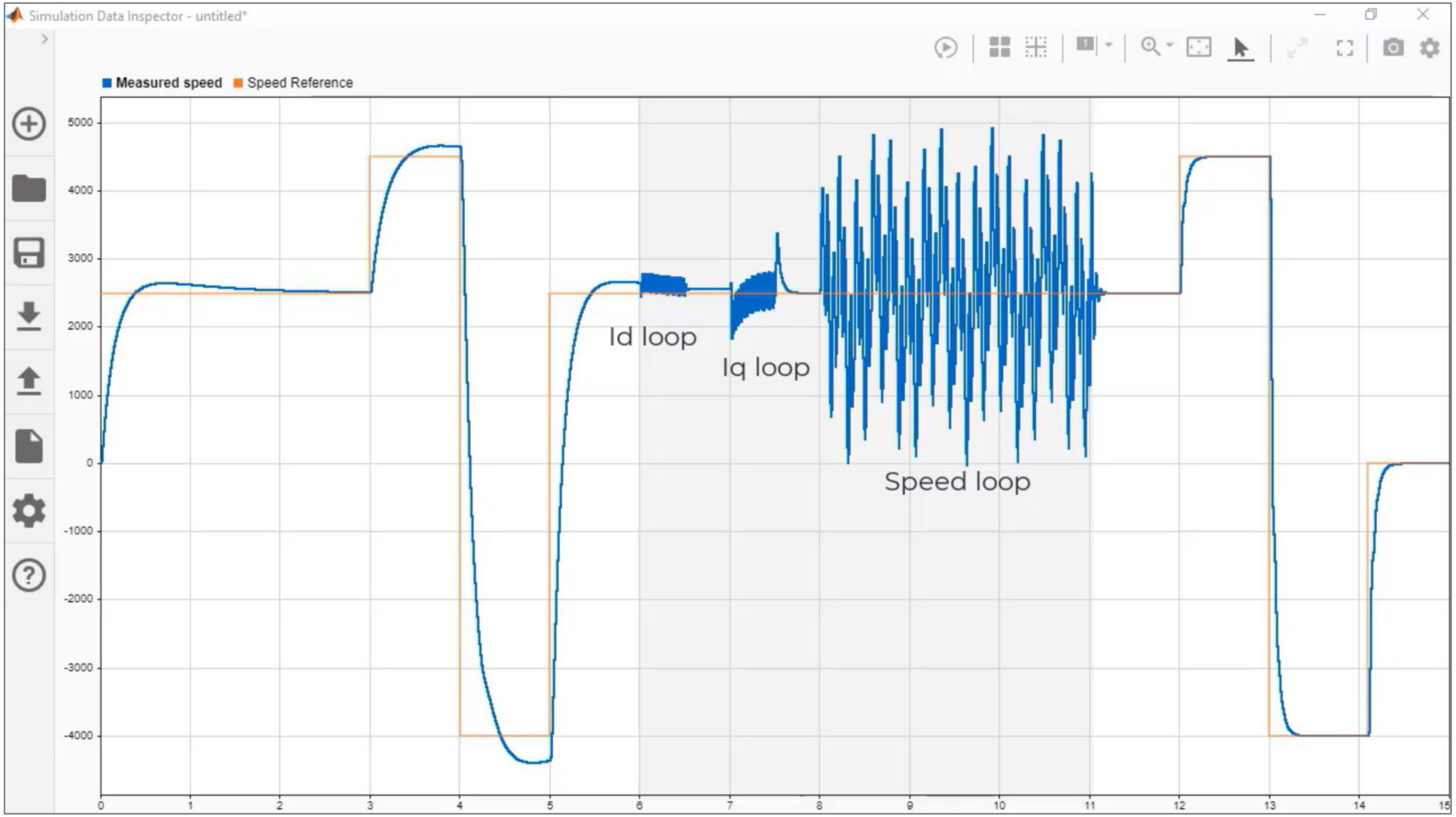
Open Script

Classic Control Theory

Simulink Control Design

Autotuning controller gains

Overview of the workflow



Field-Oriented control

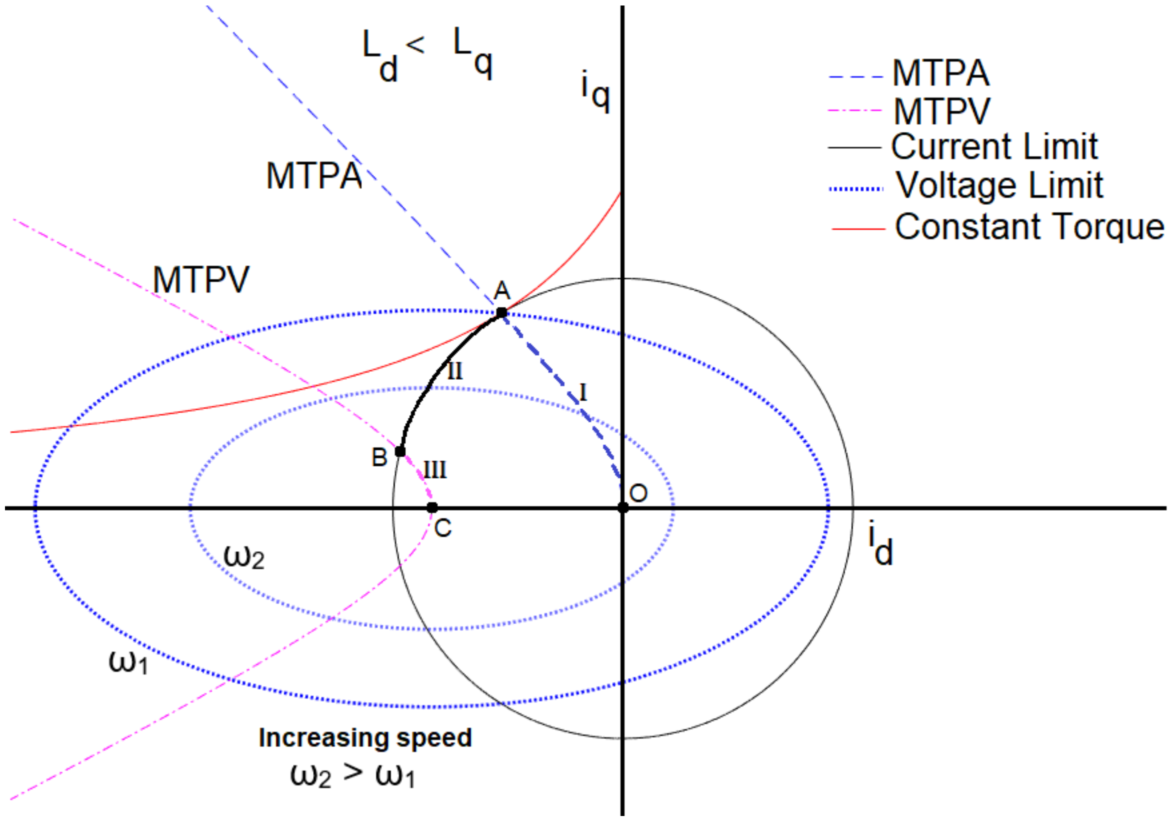
Controller tuning

Calibration

FOC Circles tracing

mcbPMSMConstraintCurves

```
Run this section                                        Get
inverter.V_dc= 24 ;
pmsm.p=4;
pmsm.Rs= 0.15 ;
pmsm.Ld=1e-3;
pmsm.Lq=pmsm.Ld* 2.3 ;
pmsm.B=1.16e-5;
pmsm.FluxPM=5.2e-3;

w_rpm= 3500 ;
T_load= 0.01 ;
pmsm.I_rated= 2.5 ;
mcbPMSMCharacteristics(pmsm,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])
```



https://www.mathworks.com/help/mcb/ug/pmsm-characteristics-constraint-curves.html

MathWorks

MathWorks® Products Solutions Academia Support Community Events

Technical Articles and Newsletters

Search Technical Articles

Overview | Search Technical Articles | Newsletters | Cleve's Corner Collection | Sign Up

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.

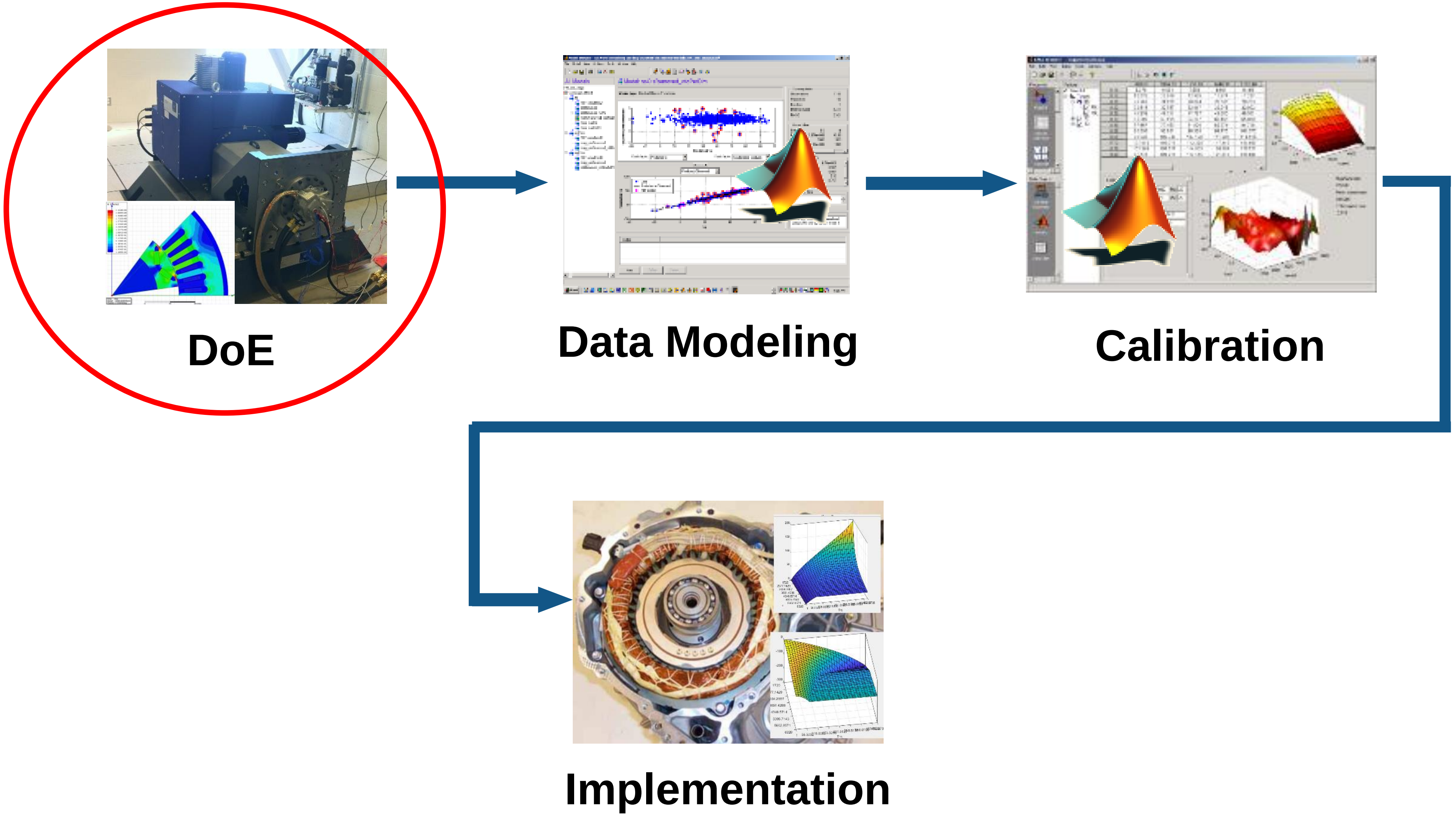
Model-based calibration standardizes the PMSM calibration process, reduces unnecessary testing, and generates consistent results. It is an industry-proven, automated workflow that uses statistical modeling and numeric optimization to optimally calibrate complex nonlinear systems. It can be used in a 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

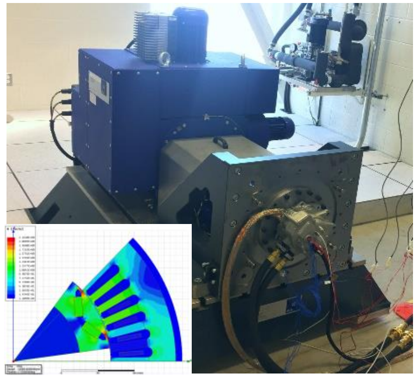
PMSMs stand out from other types of e-motors because of their high efficiency and torque density. This is because the permanent magnets inside the machine can generate substantial air gap magnetic flux without external excitation. This special trait makes a PMSM an excellent candidate for both non-traction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

MathWorks technical article

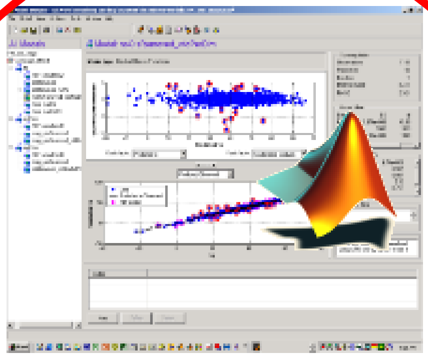
Model-Based Calibration Workflow



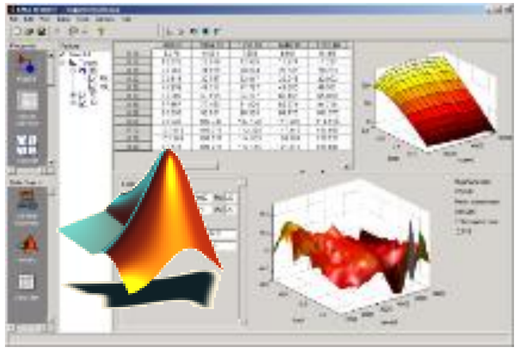
Model-Based Calibration Workflow



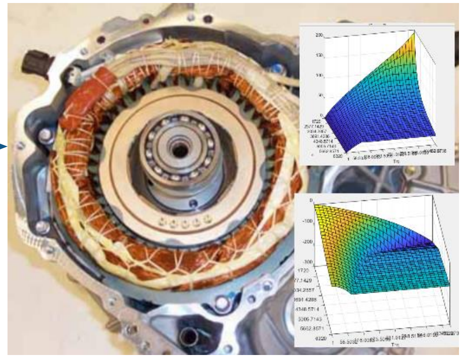
DoE



Data Modeling

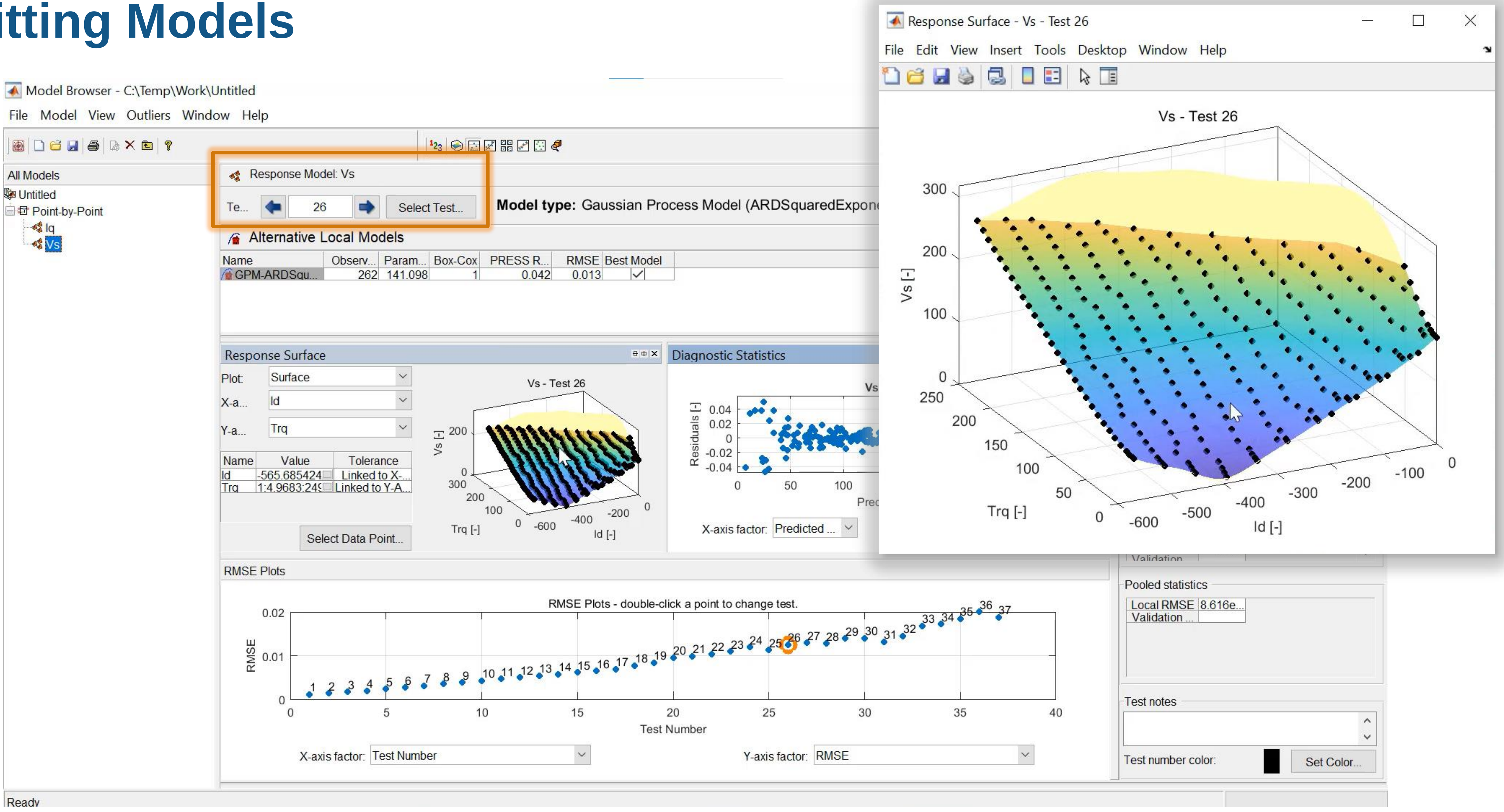


Calibration

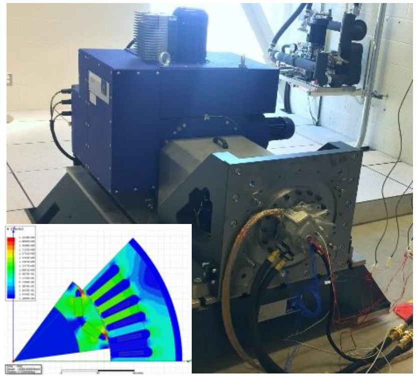


Implementation

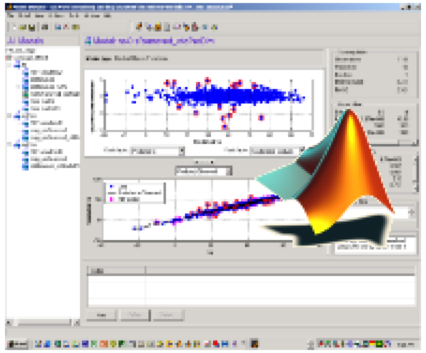
Fitting Models



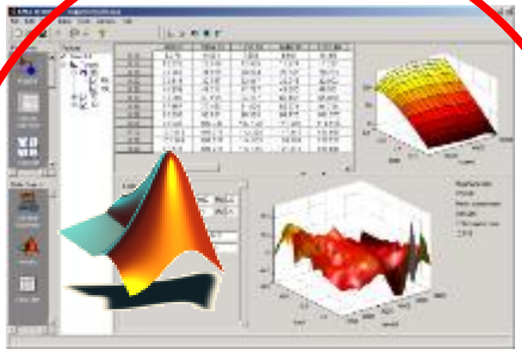
Model-Based Calibration Workflow



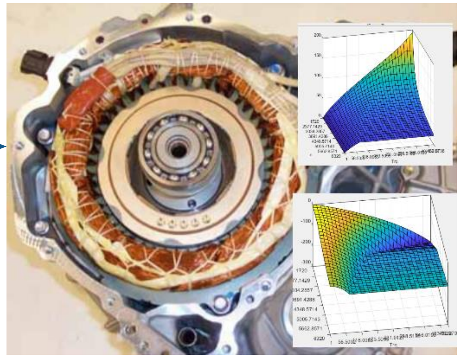
DoE



Data Modeling



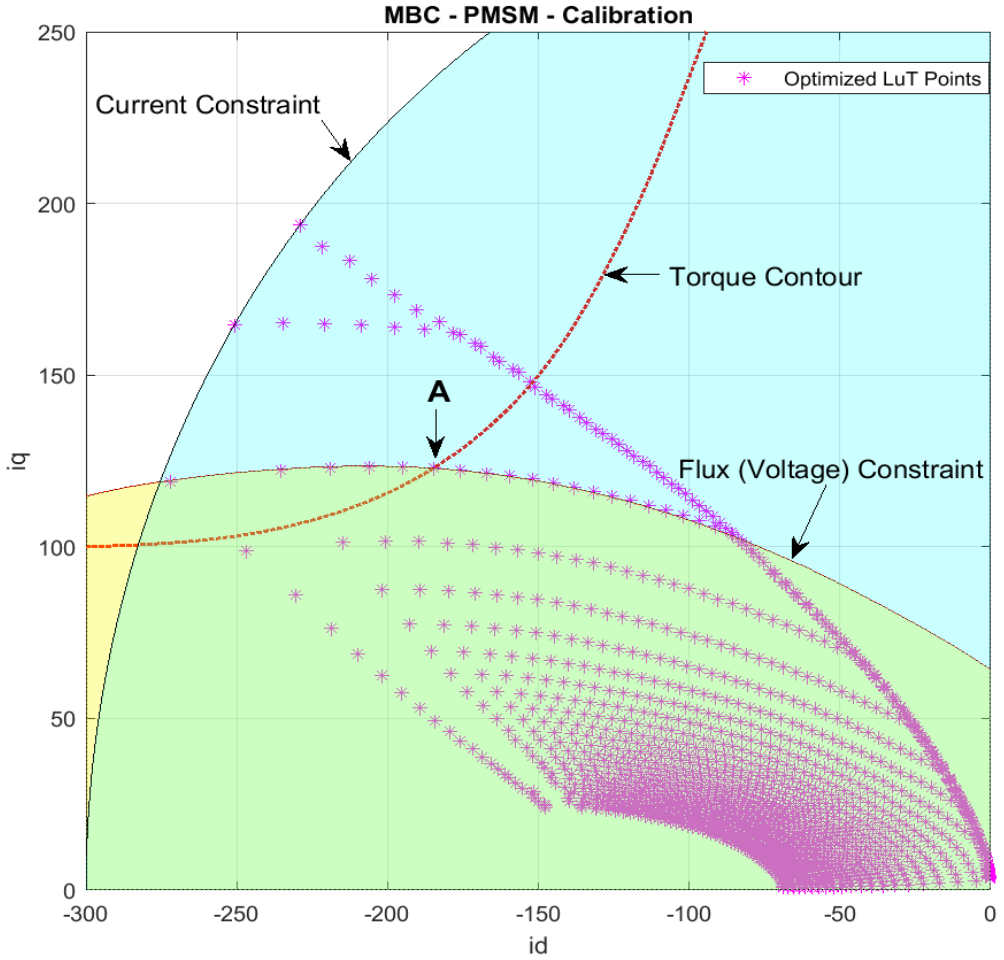
Calibration

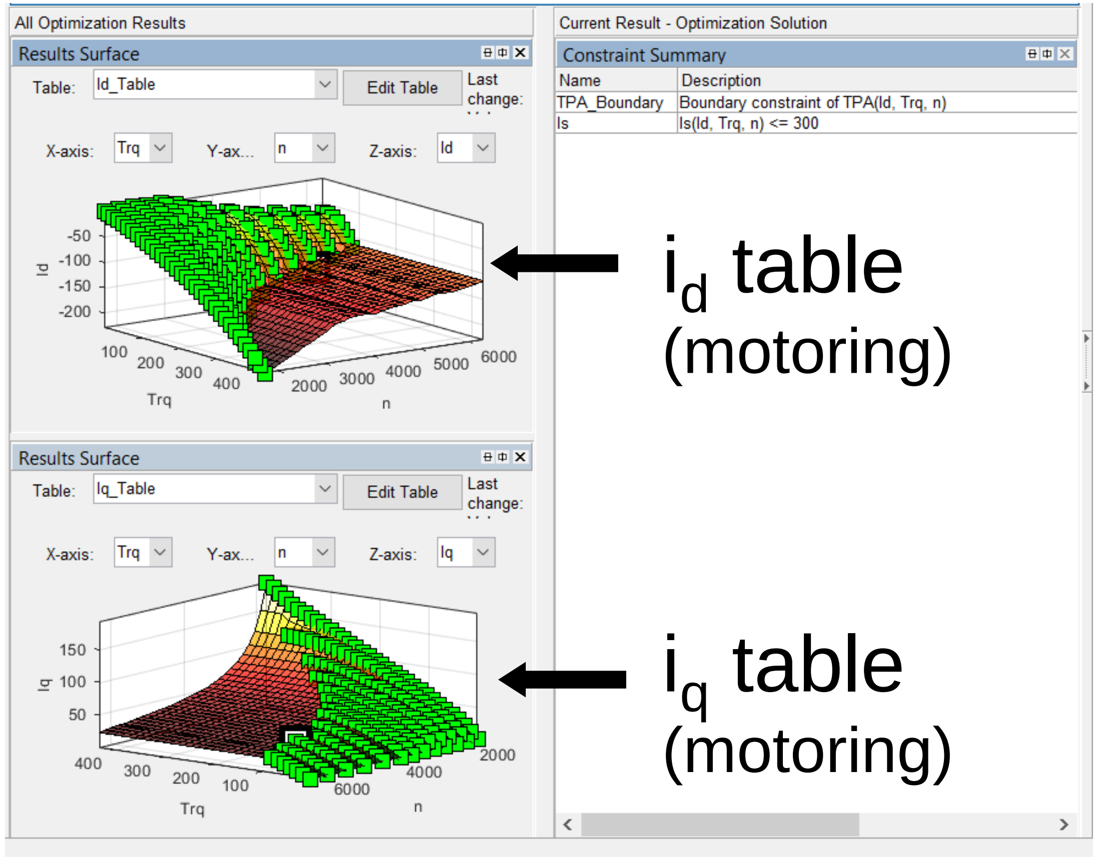


Implementation

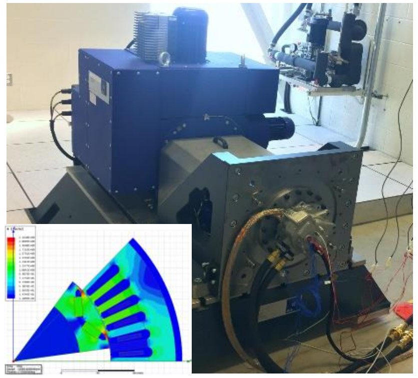
Calibration

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

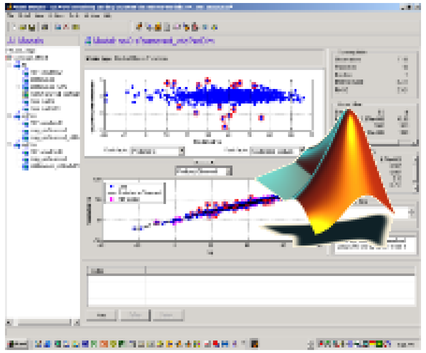




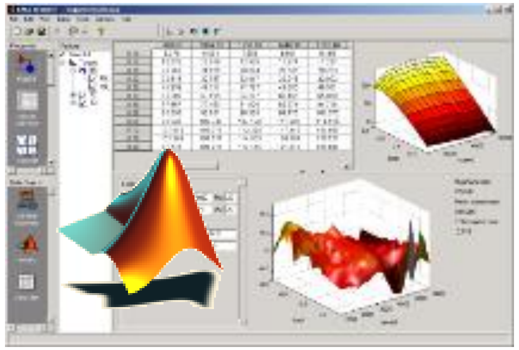
Model-Based Calibration Workflow



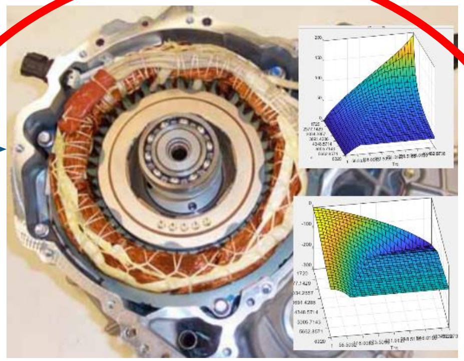
DoE



Data Modeling

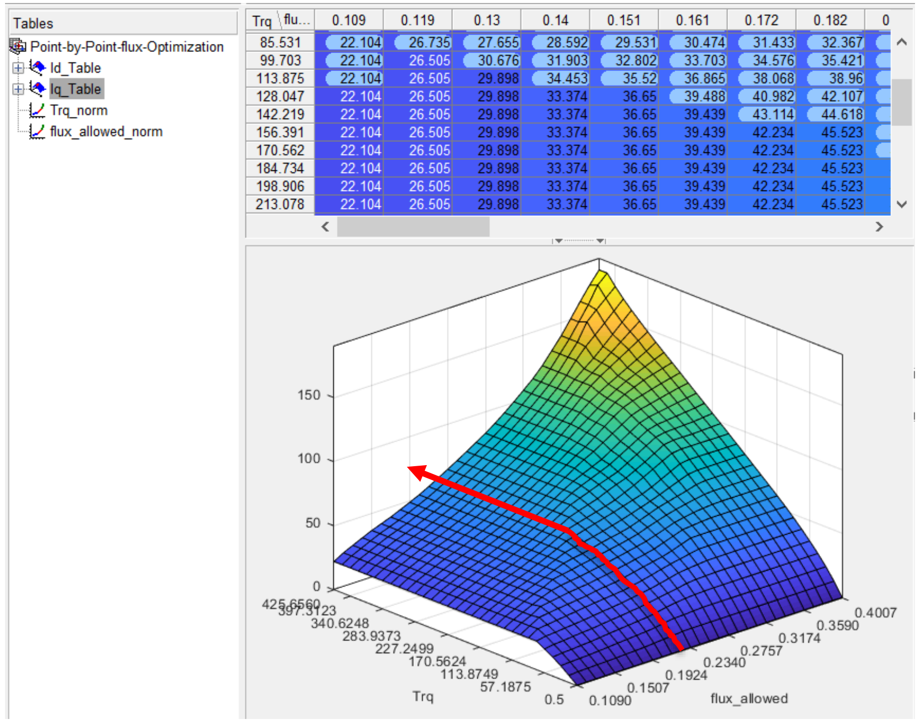


Calibration



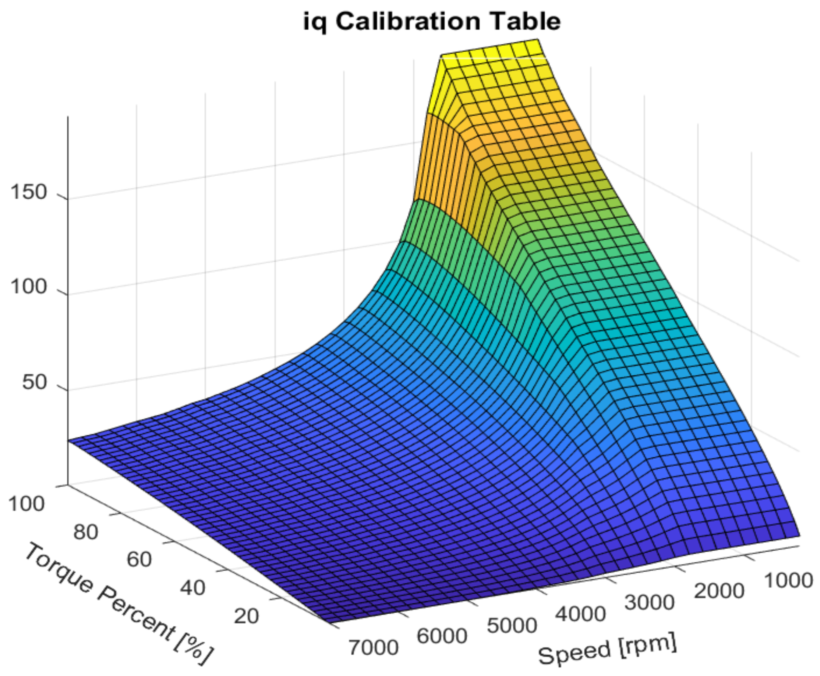
Implementation

Calibration Results – Fill Calibration Tables



Trq \ flu...	0.109	0.119	0.13	0.14	0.151	0.161	0.172	0.182	0
85.531	22.104	26.735	27.655	28.592	29.531	30.474	31.433	32.367	
99.703	22.104	26.505	30.676	31.903	32.802	33.703	34.576	35.421	
113.875	22.104	26.505	29.898	34.453	35.52	36.865	38.068	38.96	
128.047	22.104	26.505	29.898	33.374	36.65	39.488	40.982	42.107	
142.219	22.104	26.505	29.898	33.374	36.65	39.439	43.114	44.618	
156.391	22.104	26.505	29.898	33.374	36.65	39.439	42.234	45.523	
170.562	22.104	26.505	29.898	33.374	36.65	39.439	42.234	45.523	
184.734	22.104	26.505	29.898	33.374	36.65	39.439	42.234	45.523	
198.906	22.104	26.505	29.898	33.374	36.65	39.439	42.234	45.523	
213.078	22.104	26.505	29.898	33.374	36.65	39.439	42.234	45.523	

i_q table (after clipping max torque)

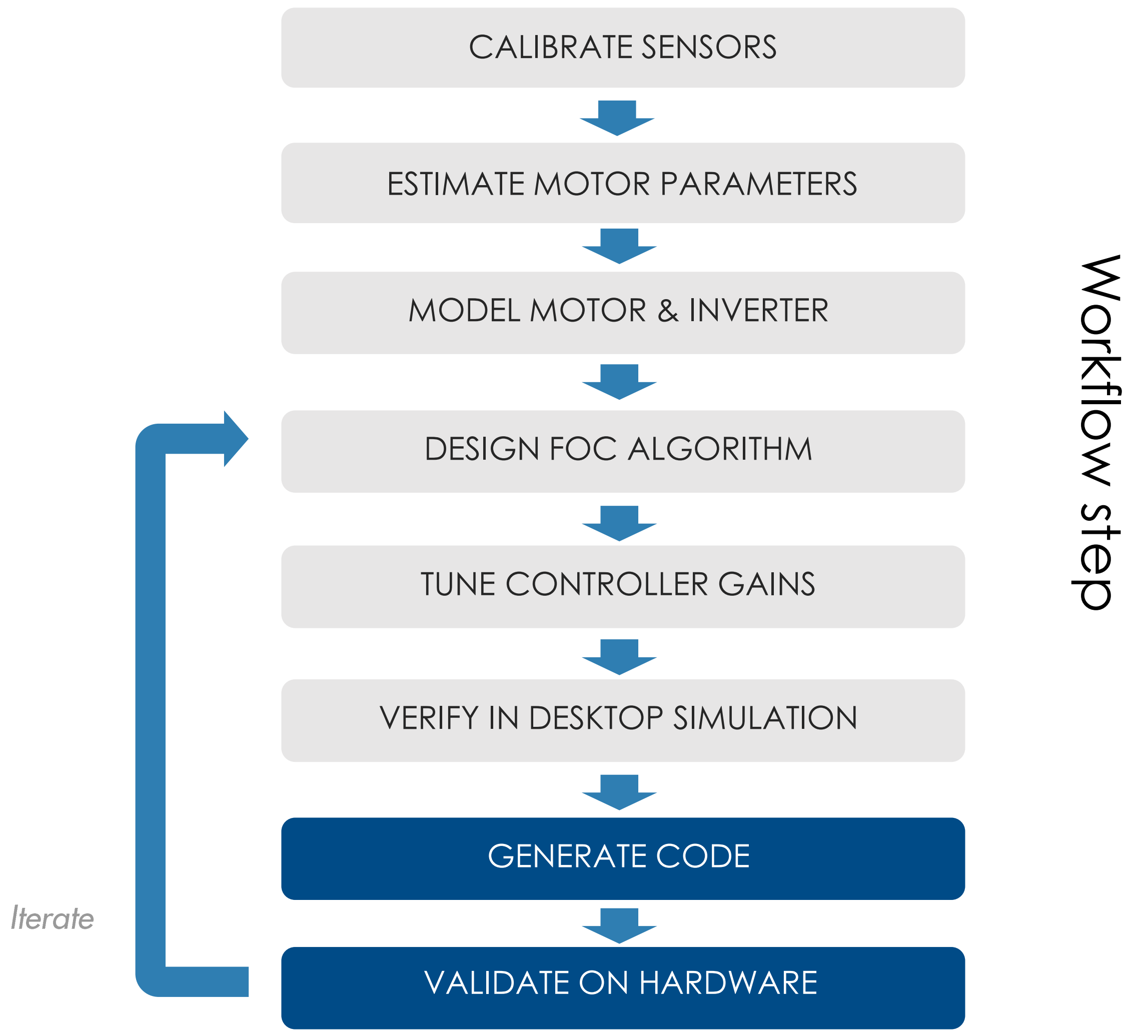


i_q table (based on percentage of max torque)

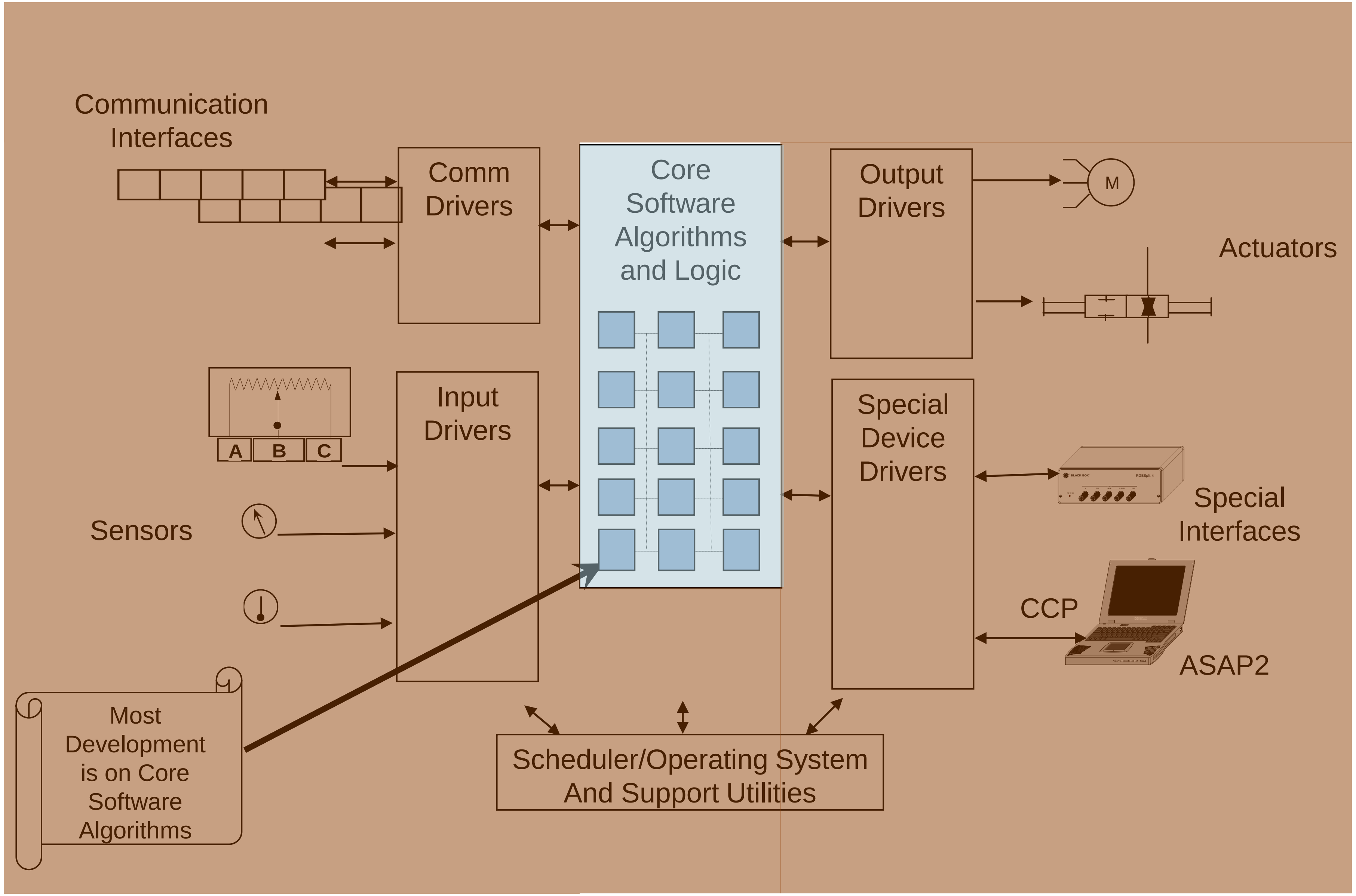
Agenda

From Desktop Simulation to Software Deployment

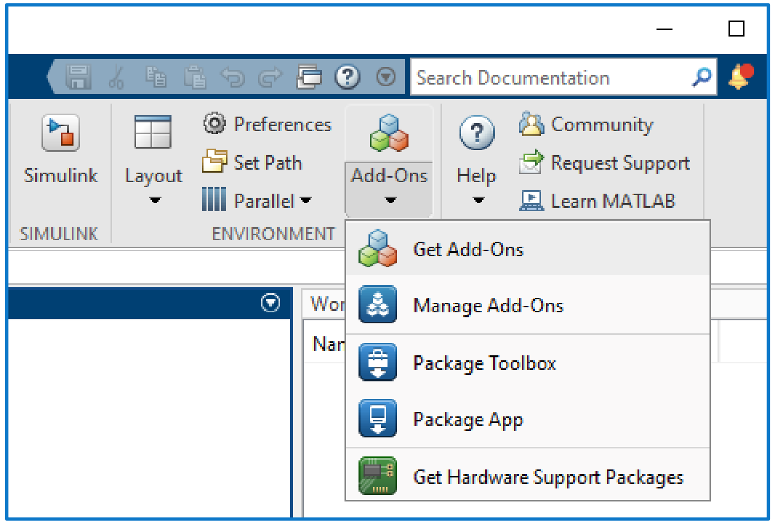
- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model
- Algorithm design with simulation
  - Field-Oriented control
  - Controller tuning
  - Calibration
- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test

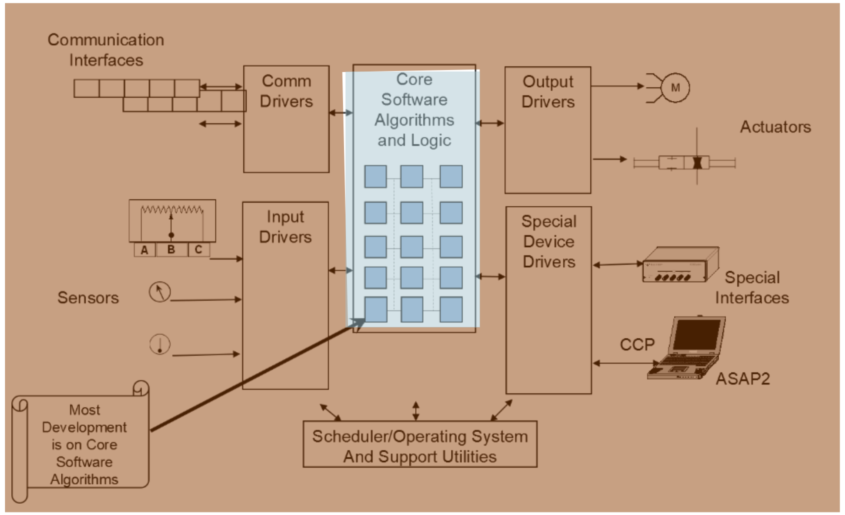


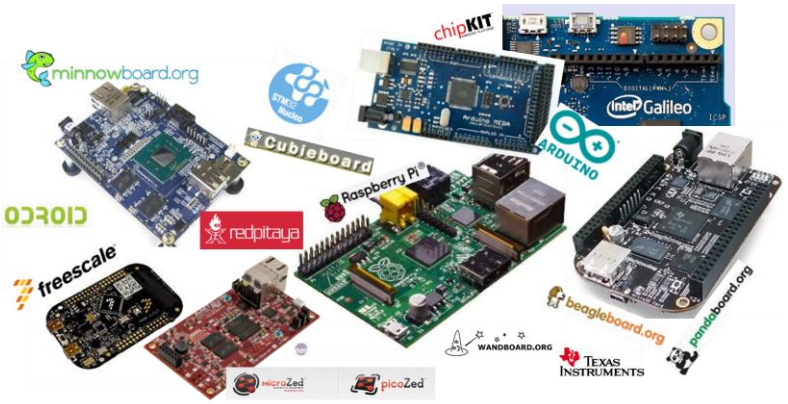
Simple Embedded Software Architecture

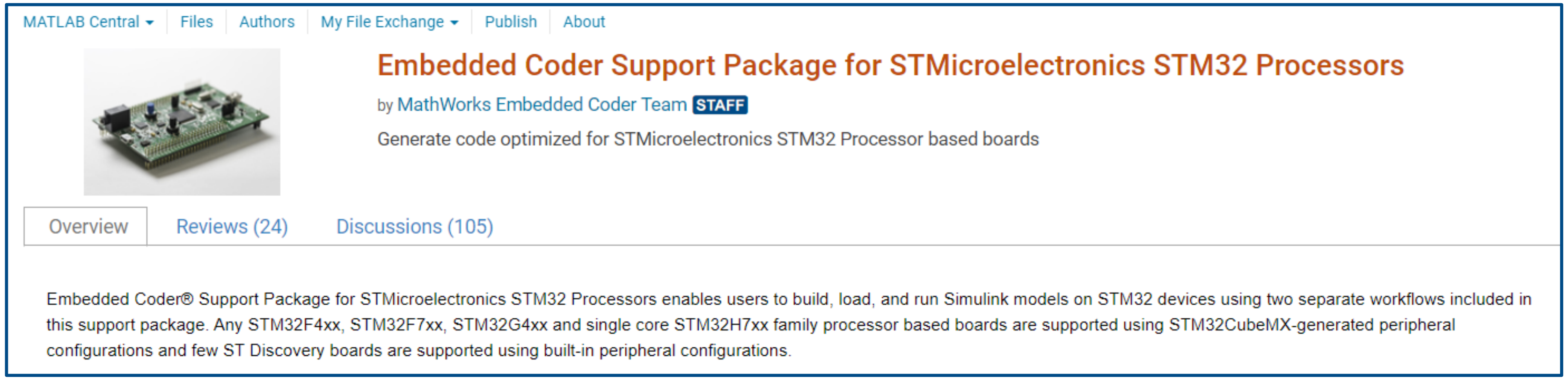


Custom Target, Hardware Support Package or Specialized toolbox









Hardware Support Packages: 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

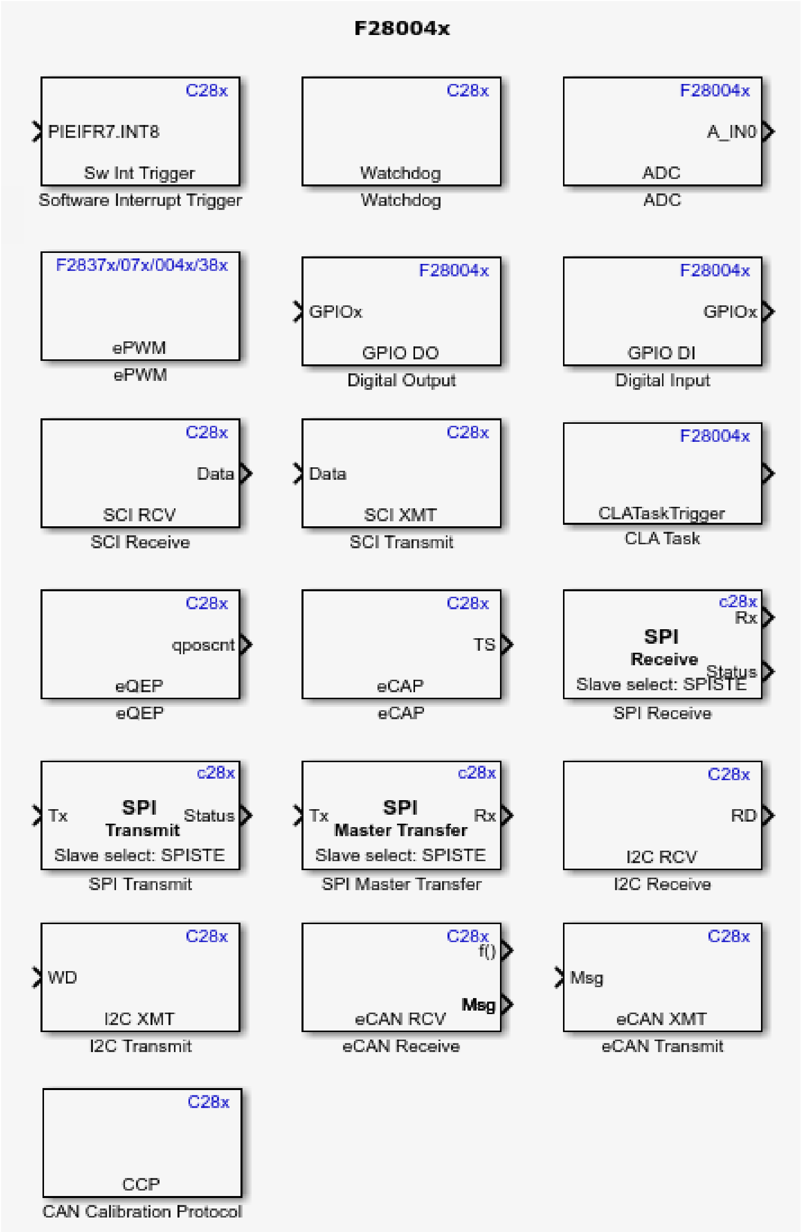
Scheduling the generated code:

- 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

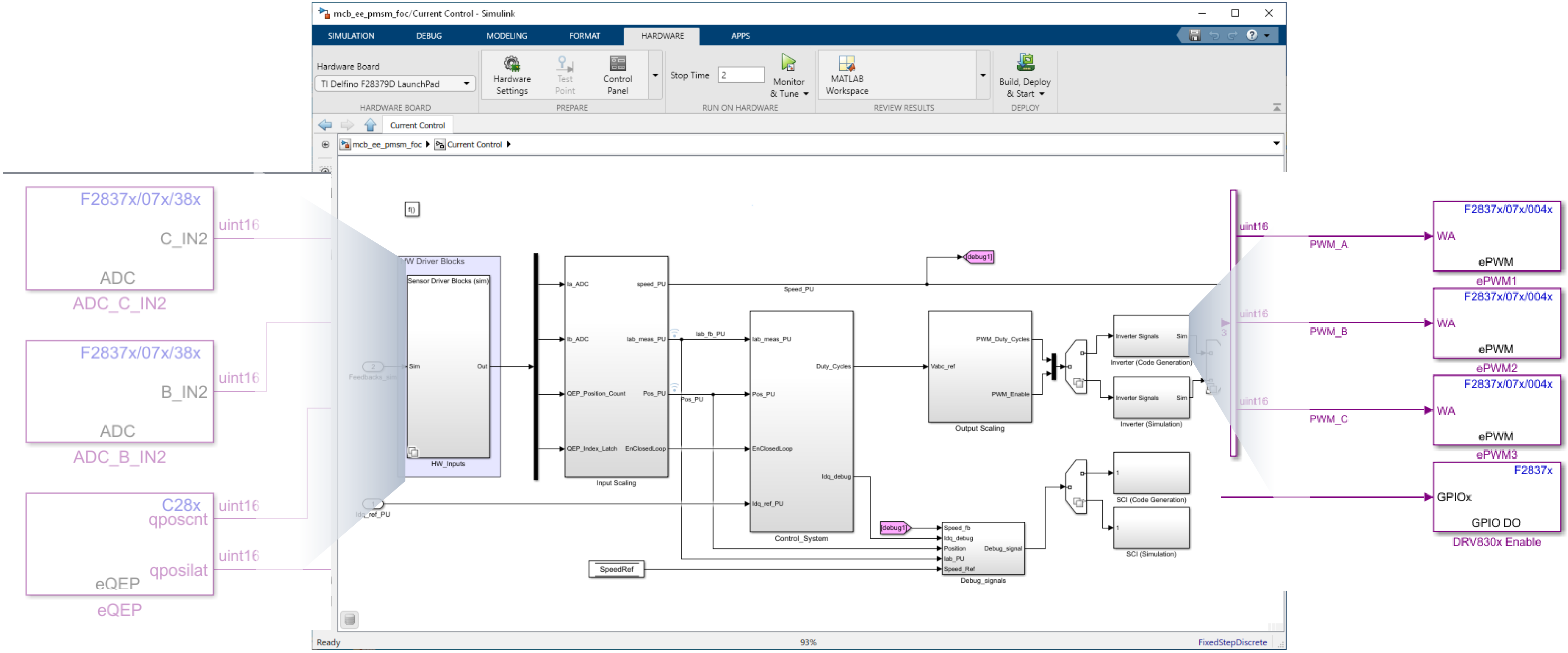
Supported TI C2000 drivers

- 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)

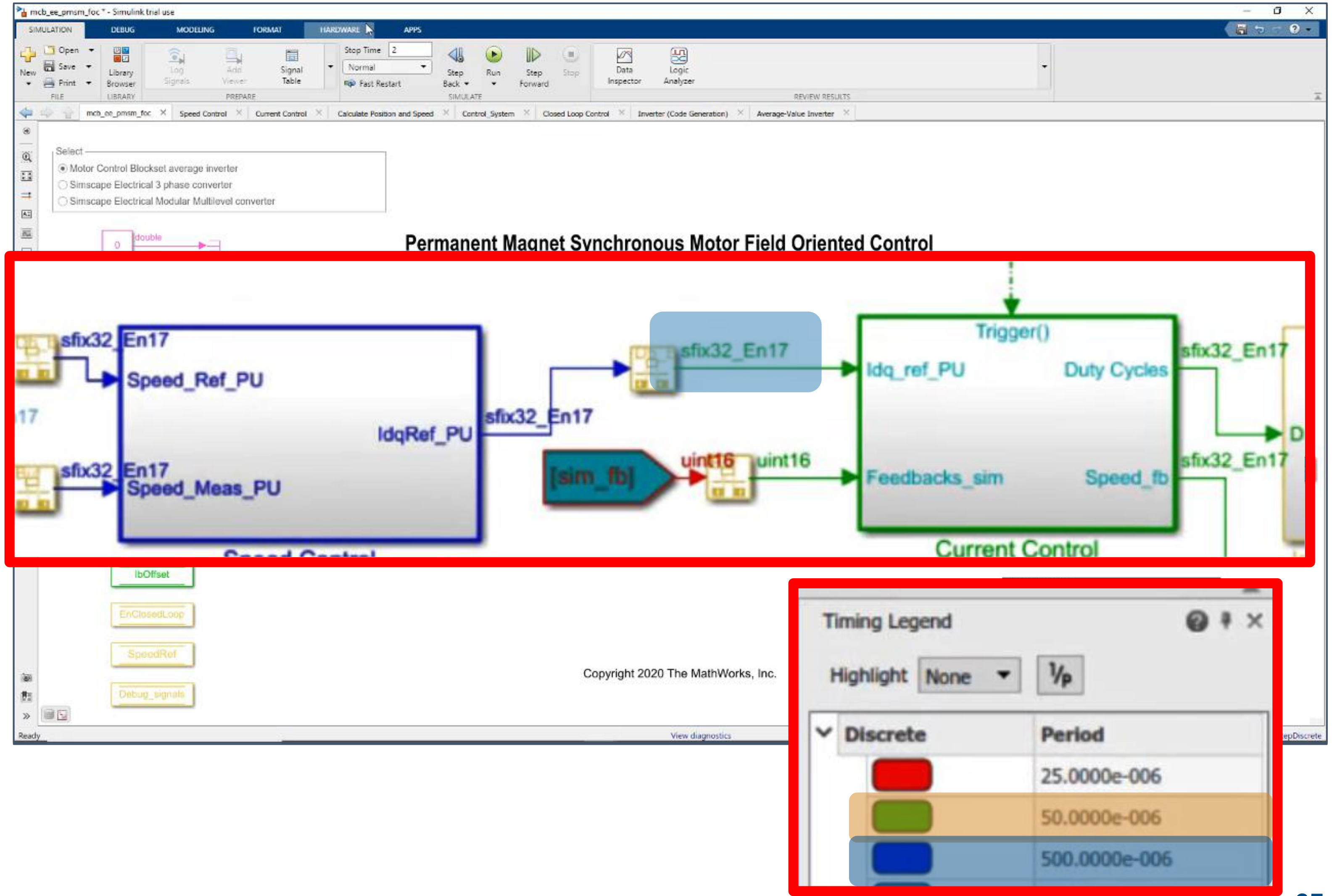


Prepare the Model for Code Generation Using Supported TI C2000 Drivers Blocks



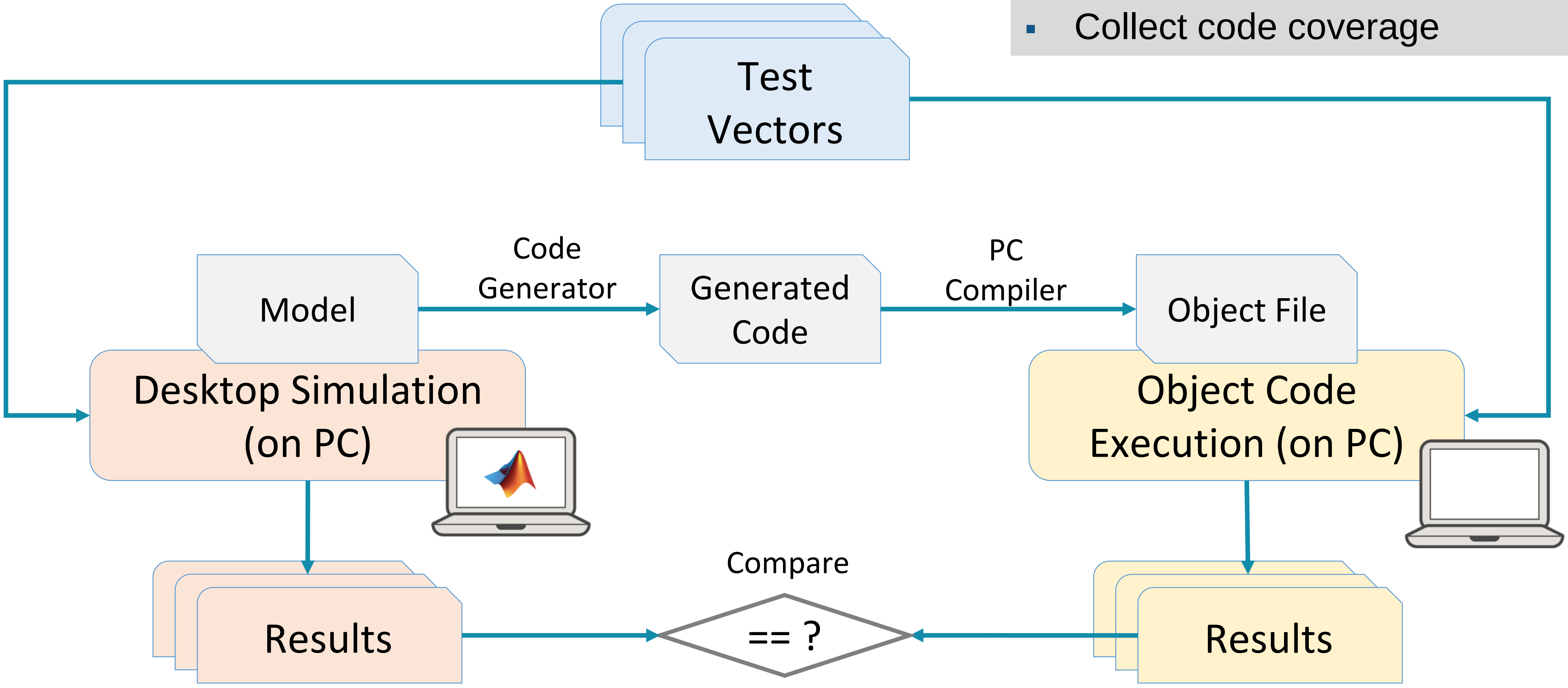
Deployment on the Target

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



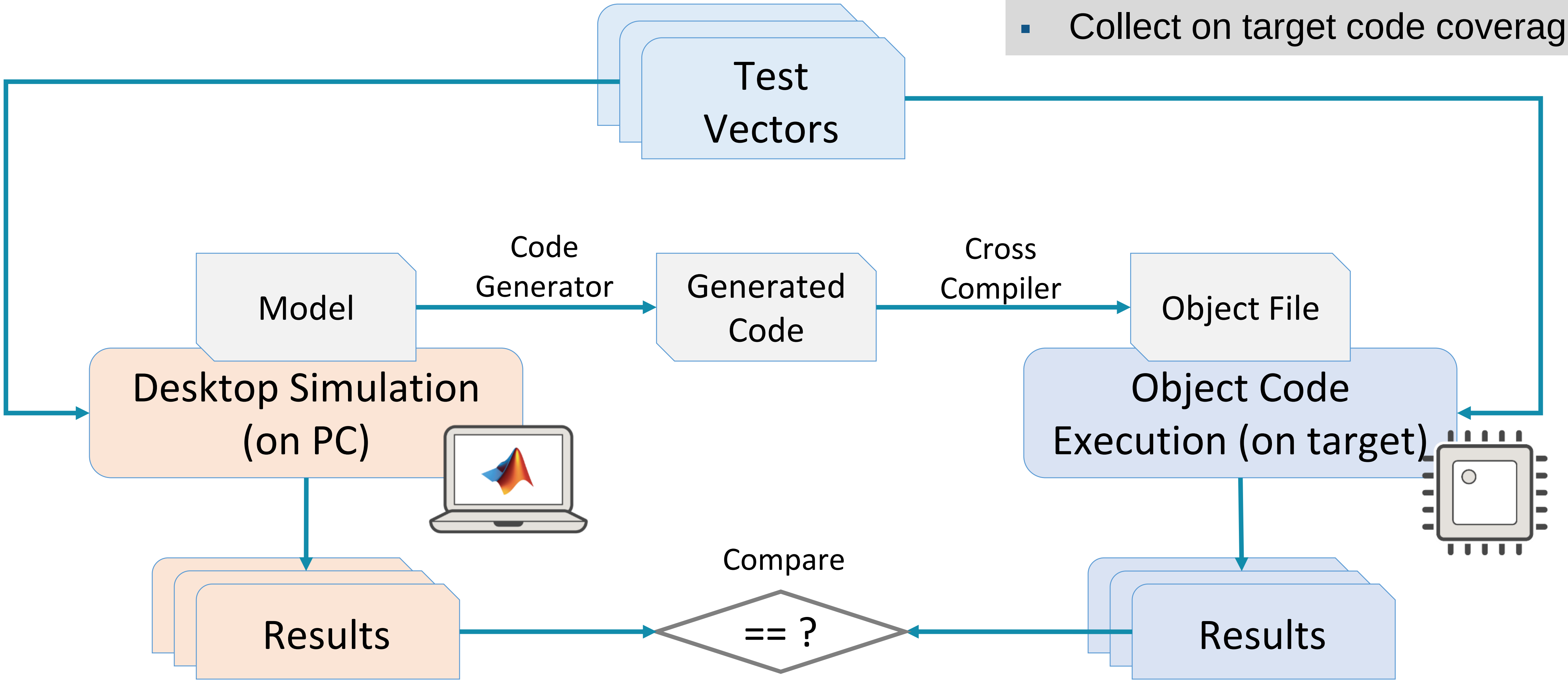
Software-In-the-Loop (SIL) Testing

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



Processor-In-the-Loop (PIL) Testing

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



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

Total time	50681790
Unit of time	ns
Command	report(executionProfile, 'Units', 'seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	2e+08
Profiling data created	16-Jan-2020 18:09:48

2. Profiled Sections of Code

Section	Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls	
[+] Current_initialize	2260	2260	1365	1365	1	
Current_step [5e-05 0]	5135	5067	5135	5067	10001	
Current_terminate	540	540	540	540	1	

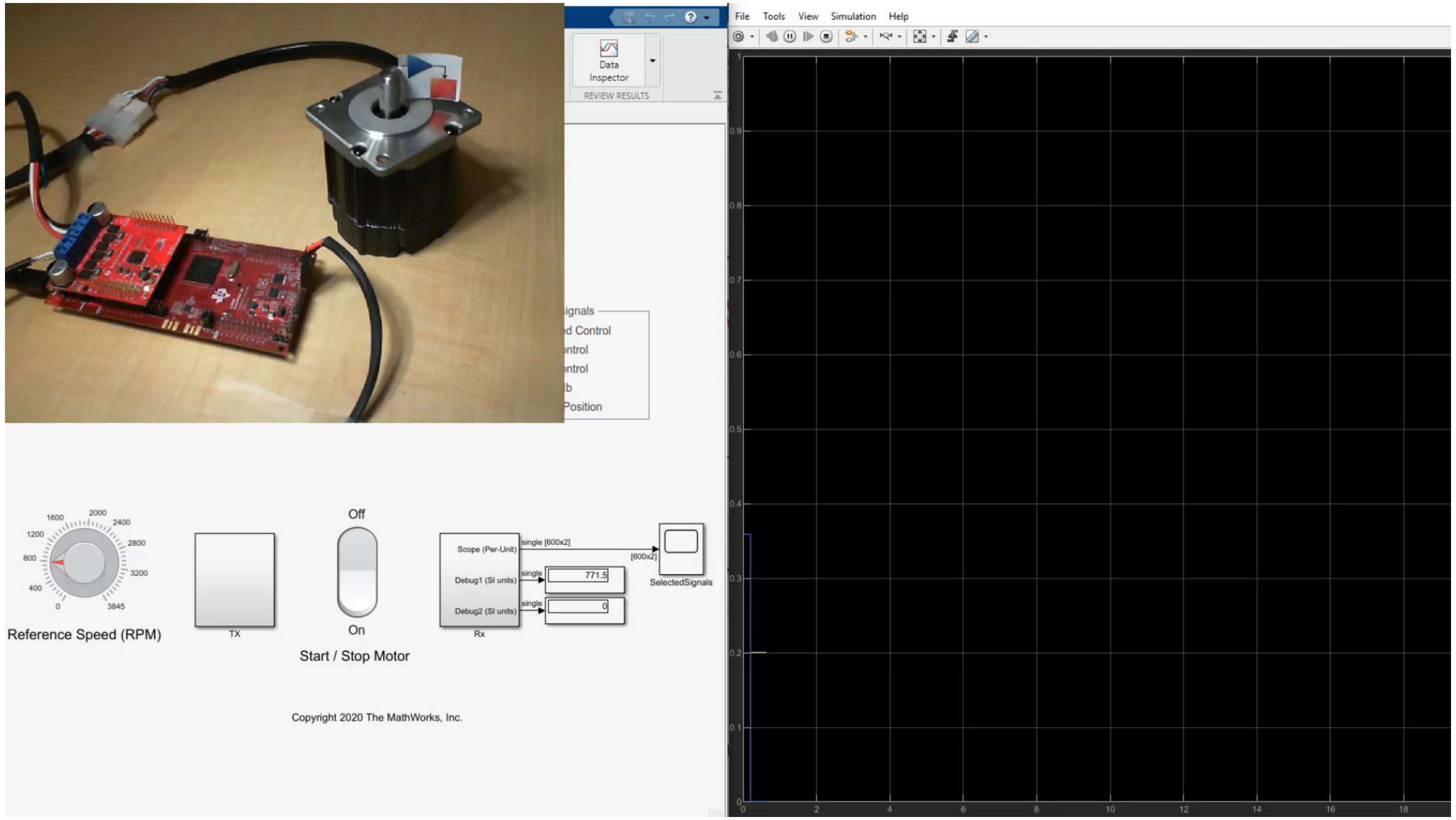
3. CPU Utilization

Task	Average CPU Utilization	Maximum CPU Utilization
Current_step [5e-05 0]	10.13%	10.27%
Overall CPU Utilization	10.13%	10.27%

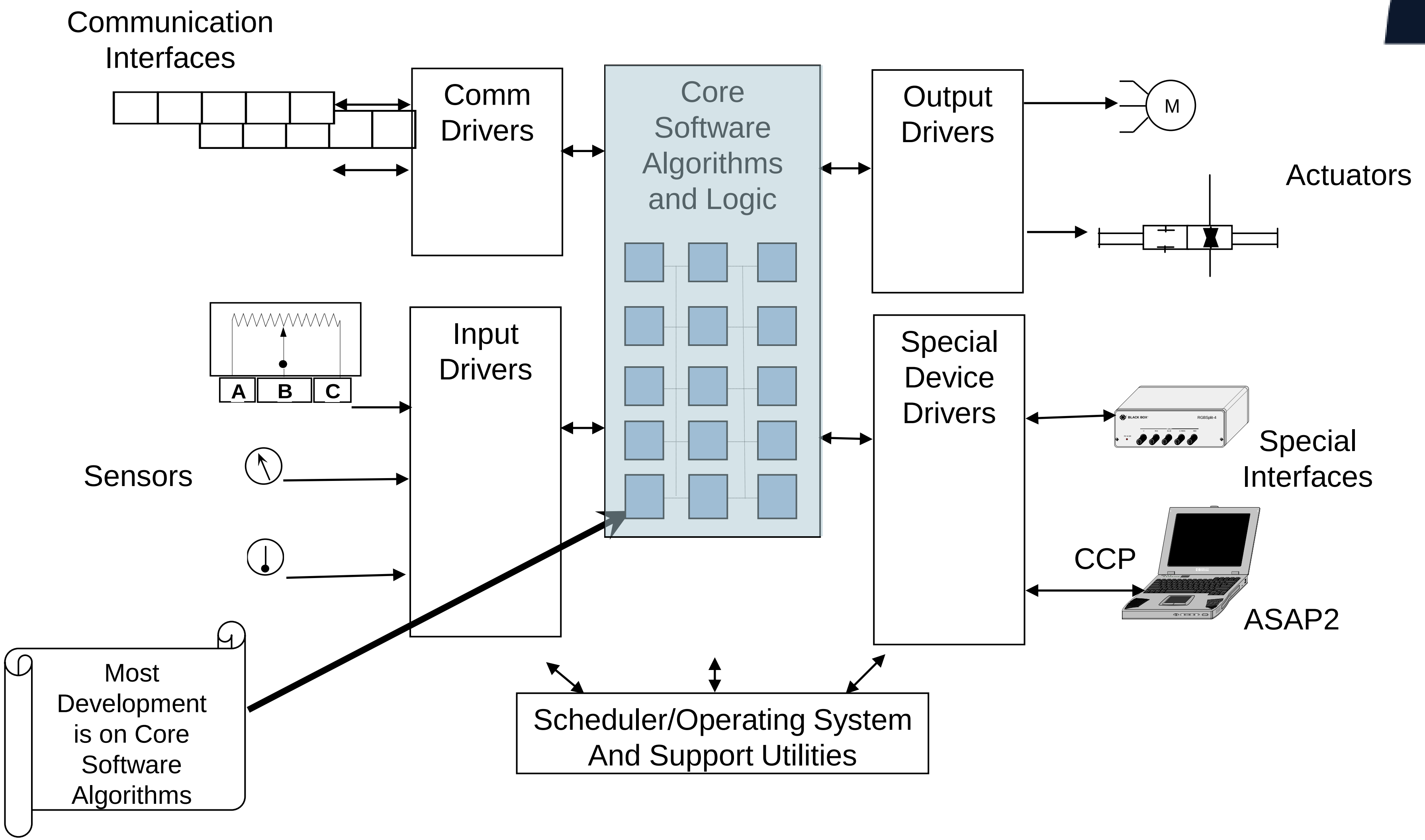
Profiling in real time execution

Deployment on the Target

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

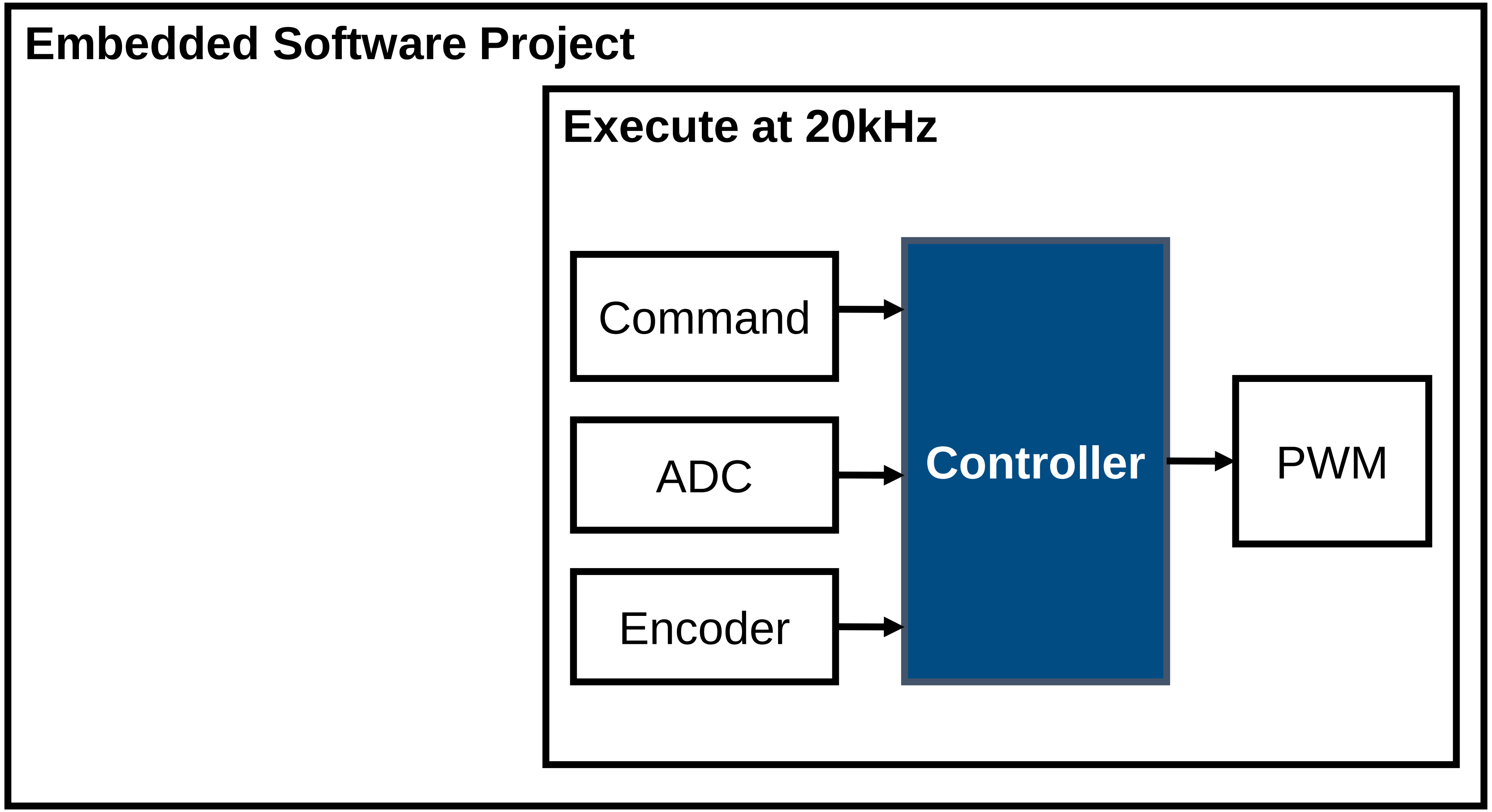


Simple Embedded Software Architecture



Integrating Generated Controller Code with an Embedded Software Project

Model **Hand**



Integrate Generated Controller Code with Your Hand-Coded Software Project

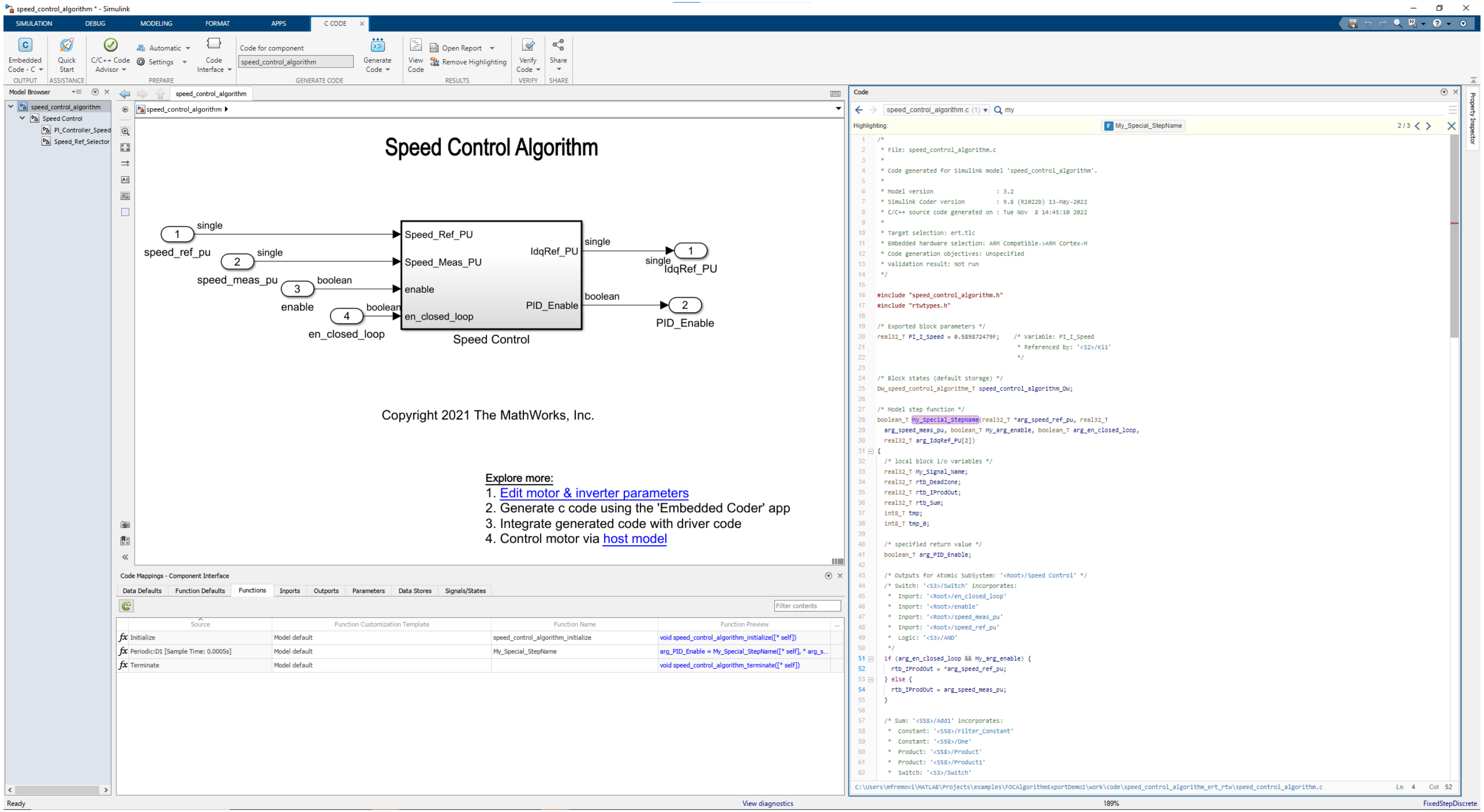
Model **Hand**

Embedded Software Project Pseudo-Code

```
main()
{
    adcInit();
    encoderInit();
    pwmInit();

    controllerInit();

    while(1) {
    }
}
```

```
interruptServiceRountine()
{
    AdcStruct = readAdcCountFromDriver();
    EncoderStruct = readEncoderCountFromDriver();

    PwmStruct = controllerStep(AdcStruct, EncoderStruct);

    writePwmCountToDriver(PwmStruct);
}
```

Customize Generated Code



Basic generated code architecture & interface

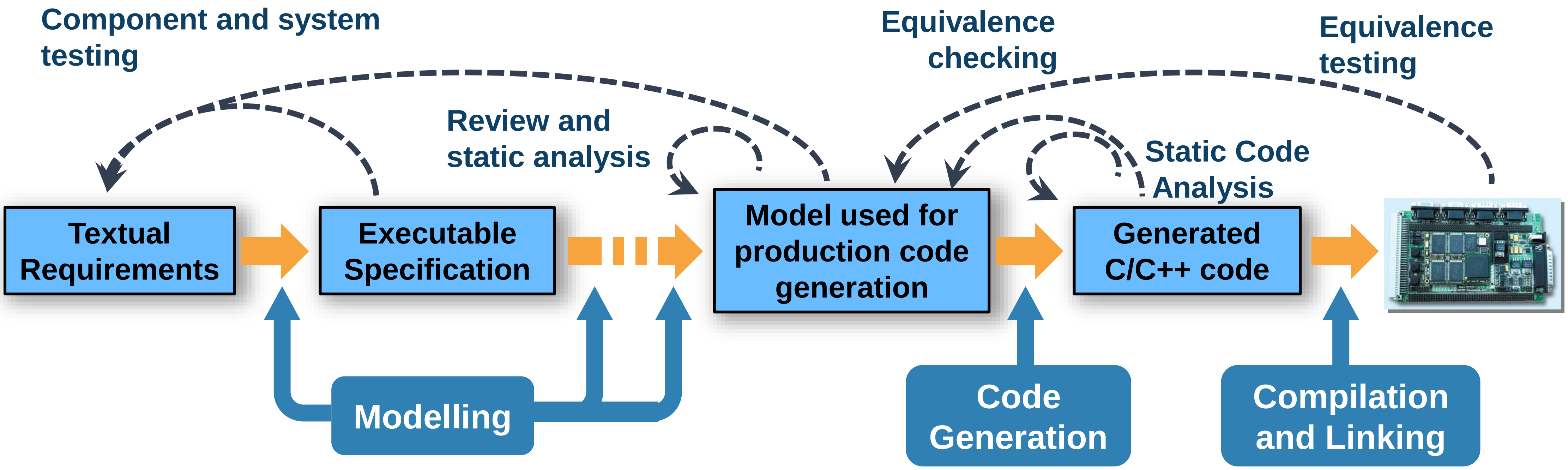
- Basic generated functions format:
  - void modelname_initialize(void) : to call at system initialization
  - Void modelname_step(void) : to call each processing step (on timer or interrupt)
  - Void modelname_terminate(void) : to call at system shutdown.

- 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)

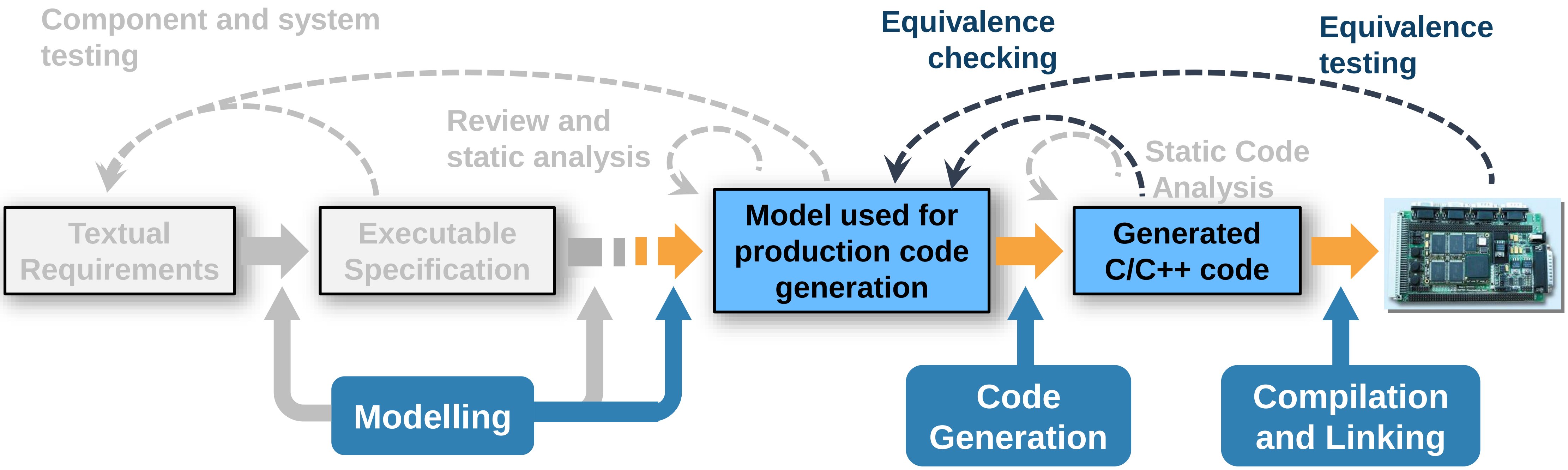
Complete Model-Based Design Workflow

Get the complete confidence in your design



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