Design and Verify Control Systems using Simulink

Coorous Mohtadi
What are we doing today?
Problem Formulation

Higher Performance:

1. **Faster trajectory**
   - $V_{\text{max}}$: 150 → 250 rad/s
   - $A_{\text{max}}$: 2000 → 5000 rad/s$^2$

2. **Decreased error at standstill**
   - Error < 1 rad
Flexible Drive

Higher Performance:

1. **Faster trajectory**
   \[ V_{\text{max}}: 150 \rightarrow 250 \text{ rad/s} \]
   \[ A_{\text{max}}: 2000 \rightarrow 5000 \text{ rad/s}^2 \]

2. **Decreased error at standstill**
   \[ \text{error} < 1 \text{ rad} \]
Workflow

- **Modelling:**
  *understanding your design problem*

- **Control Design:**
  *solving your design problem*

- **Real-time Testing:**
  *verifying the design*
Agenda

- What is Simscape?
- Real-Time Simulation and Control
Workflow

Reference and Feedforward Rigid Body P 2P GUI

Tunable Controller

Plant Model

Measured Position

Vact

Pos

Ref

Tff

Vact

pos_L
## Challenges in System Design

Table 2: Top Six Challenges of Mechatronic Product Development

<table>
<thead>
<tr>
<th>Challenges</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty finding and hiring experienced system engineers / lack of cross-functional knowledge</td>
<td>50%</td>
</tr>
<tr>
<td>Early identification of system level problems</td>
<td>45%</td>
</tr>
<tr>
<td>Ensuring all design requirements are met in the final system</td>
<td>40%</td>
</tr>
<tr>
<td>Difficulty predicting / modeling system product behavior until physical prototypes exist</td>
<td>32%</td>
</tr>
<tr>
<td>Difficulty implementing an integrated product development solution for all disciplines involved in mechatronic product development</td>
<td>28%</td>
</tr>
<tr>
<td>Inability to understand the impact a design change will have across disciplines</td>
<td>18%</td>
</tr>
</tbody>
</table>

Challenges in System Design

- Tools & processes that support only bottom-up design
  - Design at a low-level
  - Provide a specification to the higher level
  - Examples: CAD

- Lack of top-down (system) design tools
  - No rigour – when is a design optimal for given requirements?
  - Some methods e.g. function-means, weighted objectives
  - Bond graphs – some use, but often too abstract
  - Simulink® - usefulness depends on system

- Disruptive technologies
  - Potentially a subsystem change forces a complete redesign for the whole system
  - CAD better suited to incremental design changes

- Communication between functional groups
Using Simulink® for System Modelling

- Fits some systems well
  - Airframes
  - Some aspects of vehicle components
  - **Systems defined by ODEs**

- Does not fit others so well
  - Mechanical
  - Hydraulic and flow-based systems
  - Electrical/electromechanical
  - **Systems defined by DAEs**

- Simscape™ Language and Simscape blocksets
  - Aimed at systems described by DAEs
  - Thermal, electrical, mechanical, hydraulic, pneumatic, custom…
DC Motor

\[ v = K_e \omega + i_m R_{\text{wind}} + L_{\text{wind}} \frac{di_m}{dt} \]

\[ T = K_t i_m - D \omega - J \frac{d\omega}{dt} \]
Electrical Systems in Simscape

**DC Motor**

- Simscape model advantages
  - Easier to read than equations
  - Quicker to create
  - More intuitive – easier to explain to other engineers
Electrical Systems in Simscape Language

- Simscape language advantages
  - For custom component, often easier to express equations
  - Flexibility & extensibility

DC Motor

\[ v = K_e \omega + i_m R_{\text{wind}} + L_{\text{wind}} \frac{di_m}{dt} \]

\[ T = K_t i_m - D \omega - J \frac{d\omega}{dt} \]
Modeling Dynamic Systems: two approaches

First-Principles Modeling

Use an understanding of the system’s physics to derive a mathematical representation

\[ \alpha = \int -L_2 \sin(\alpha) + nw_2 (-\sin(\alpha - \gamma)) \sin(\gamma) - ne(-\sin(\alpha - \gamma)) \cos(\alpha - \gamma) \alpha^2 - n \cos(\alpha - \gamma) \gamma^2 \frac{d\gamma}{1 - ne \sin^2(\alpha - \gamma)} \]
Modeling Dynamic Systems: two approaches

First-Principles Modeling

Use an understanding of the system's physics to derive a mathematical representation

\[ H(s) = \frac{s + 1}{s^3 + 3s^2 + 2} \]

\[ \alpha = \frac{s^2 L_2 \sin(\alpha)}{s^2 + s + 1} \]

Data-Driven Modeling

Use system test data to derive a mathematical representation

\[ \alpha = \beta \cos(\gamma) \]
Both have advantages & disadvantages

**Advantages:**
- Provides insight into the system’s underlying behavior
- Enables broad parameter sweeps to optimize design

**Disadvantages:**
- Effects like friction and turbulent flow are difficult to model
- Time consuming to develop

**Advantages:**
- Fast to develop a model
- Highly accurate

**Disadvantages:**
- Requires *plant* hardware to acquire test data
- Not a design tool for plant
Tools that span both modeling approaches

Complete Modelling Environment

First-Principles
- Simulink
- Stateflow
- Simscape
- SimPowersystems
- SimMechanics
- SimDriveline
- SimHydraulics
- SimElectronics

Data-Driven
- Simulink Design Optimization
- System Identification Toolbox
Modeling: *understanding your design problem*

- **Data Driven methods: the existing system**
  - Transfer function measurement using live data
  - System Identification for obtaining a model

- ‘First Principles’ modelling: the new design
  - Setting up a differential equation description
  - Using Physical Modeling tools

- **Parameter Estimation: the first prototype**
  - Physical parameters from measurements

![Diagram showing the transition between First Principles, Data Driven, and Parameter Estimation](image-url)
Data Driven Methods: the existing system

Transfer Function Measurement using live data
Data Driven Methods: the existing system

System Identification
Data Driven Methods: *the existing system*

Step response of identified system
Modeling: *understanding your design problem*

- **Data Driven methods: the existing system**
  - Transfer function measurement using live data
  - System Identification for obtaining a model

- **‘First Principles’ modeling: the new design**
  - Setting up a differential equation description
  - Using Physical Modeling tools

- **Parameter Estimation: the first prototype**
  - Physical parameters from measurements
Digital Motion Control Setup

Motor Encoder

Motor

Flexible Shaft

Load Encoder

I/O connectors

Power Amplifier
Digital Motion Control Setup

Actuator Inertia

Load Encoder

Load Inertia

Flexible Shaft

Demo
‘First Principles’ Modeling: the new design

Implementing with State-Space model

\[
\begin{align*}
J_1 & \quad x''_1 = -b_1 x'_1 - k (x'_1 - x'_2) - b_{12} (x'_1 - x'_2) + T \\
J_2 & \quad x''_2 = -b_2 x'_2 + k (x'_1 - x'_2) + b_{12} (x'_1 - x'_2)
\end{align*}
\]
‘First Principles’ Modeling: the new design

Implementing in block diagram form

\[
\begin{align*}
J_1 \ddot{x}_1 &= -b_1 \dot{x}_1 - k (x_1 - x_2) - b_{12} (\dot{x}_1 - \dot{x}_2) + T \\
J_2 \ddot{x}_2 &= -b_2 \dot{x}_2 + k (x_1 - x_2) + b_{12} (\dot{x}_1 - \dot{x}_2)
\end{align*}
\]
‘First Principles’ Modeling: *the new design*

Now, is there another way?.....

Suppose you designed your system in CAD!
CAD Translator for SimMechanics
SimMechanics: modeling of 3D mechanical systems

- Actuators
- Mechanical Device
- Sensors

- Plant

- 3D Multi-Body Dynamics
- Bodies and Joints
- CAD Translation
First-principles Modeling in Simulink

- Traditional process
  
  ![Diagram](image)

  Set up differential and algebraic equations

- Using physical modeling tools
  
  ![Diagram](image)

  Build model by combining components
‘First Principles’ Modeling: the new design

Simscape
Simscape: modeling in multiple domains

- Use Fundamental Analogy between Physical Domains

<table>
<thead>
<tr>
<th>Domain</th>
<th>Across Variable</th>
<th>Through Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mechanical</td>
<td>Position</td>
<td>Force</td>
</tr>
<tr>
<td>(translational/rotational)</td>
<td>Velocity</td>
<td>Force</td>
</tr>
<tr>
<td></td>
<td>Acceleration</td>
<td>Torque</td>
</tr>
<tr>
<td></td>
<td>Angular velocity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Angular acceleration</td>
<td></td>
</tr>
<tr>
<td>Hydraulic</td>
<td>Pressure</td>
<td>Volumetric flow rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mass flow rate, Volume</td>
</tr>
<tr>
<td>Electrical</td>
<td>Voltage</td>
<td>Current</td>
</tr>
<tr>
<td></td>
<td>Flux</td>
<td>Charge</td>
</tr>
<tr>
<td>Thermal</td>
<td>Temperature</td>
<td>Heat flow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enthalpy, Entropy</td>
</tr>
</tbody>
</table>
Simscape Language

- MATLAB-based
- Create new domains
- Custom components
- User defined libraries
- Object oriented
Physical Systems in Simulink

- SimMechanics: Mechanical dynamics (3-D)
- SimHydraulics: Fluid power and control
- SimPowerSystems: Electrical power systems
- Simscape: Multidomain physical systems
- SimDriveline: Drivetrain systems (1-D)
- SimElectronics: Electromechanical and electronic systems
Modeling: *understanding your design problem*

- **Data Driven methods: the existing system**
  - Transfer function measurement using live data
  - System Identification for obtaining a model

- ‘First Principles’ modeling: *the new design*
  - Setting up a differential equation description
  - Using Physical Modeling tools

- **Parameter Estimation: the first prototype**
  - Physical parameters from measurements
Parameter Estimation: the first prototype
Modeling Tools

Complete Modeling Environment

MATLAB
Simulink
Simscape

SimMechanics
SimDriveline
SimHydraulics
SimElectronics
SimPowerSystems
Aerospace Blockset
RF Blockset
Simulink Design Optimization
System Identification Toolbox
Optimization Toolbox
Agenda

- What is Simscape Language?

- Real-Time Simulation and Control
Real-time Testing: *verifying the design*

- Prepare Simulink model for Rapid Prototyping
- Automatic code generation with Real-Time Workshop
- Interfacing with your implemented design
Real-time Testing: *verifying the design*

Setup replaces plant model

Target PC implements controller
Preparation for Real-time Testing

Specify I/O

Signal Conditioning

Data Monitoring
Automatic Code Generation

```
1 /*
2  * DMC_system_demo.c
3  *
4  * Real-Time Workshop code generation for Simulink model "DMC_system_demo.mdl".
5  *
6  * Model Version : 1.374
7  * Real-Time Workshop version : 6.4 (R2006a) 03-Feb-2006
8  * C source code generated on : Mon Mar 27 11:14:28 2006
```
Interfacing with Your Implemented Design

Motion Control System
Parameters defined in MATLAB Workspace

FO_interface
Trajectory Specification
- Step Size: 100 rad
- Max Velocity: 250 rad/s
- Max Acceleration: 5000 rad/s^2
- Max Jerk: 5e5 rad/s^3
- Max dJerk: 1e8 rad/s^4

Feedforward
- 4th order
- Rigid Body
- Inactive

Status: Feedforward de-activated
1. Extending an Existing Simscape Library

- Nonlinear Rotational Spring
  - Torque varies as square of deflection \((\text{Torque} = -k*\theta^2)\)
  - All required elements are defined in a single file (parameters, equations, description, etc.)
  - The model is compatible with Simscape foundation library

- NonlinRotSpring.ssc
- NonlinRotSpring.jpg
Defining Components: Declaration

- Declaration section
  - Declare component members (node, input, variables, etc.)

Component Model

<table>
<thead>
<tr>
<th>Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
</tr>
<tr>
<td>Inputs, Outputs</td>
</tr>
<tr>
<td>Variables</td>
</tr>
<tr>
<td>(through, across, and internal)</td>
</tr>
<tr>
<td>Parameters</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Setup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter checking</td>
</tr>
<tr>
<td>Define relationship between component variables and nodes</td>
</tr>
<tr>
<td>Initial conditions</td>
</tr>
<tr>
<td>Derived parameters</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algebraic, discontinuous, differential</td>
</tr>
</tbody>
</table>
Defining Components: Declaration

Component Model

```matlab
component NonlinRotSpring

% Nonlinear Rotational Spring
% This block implements a nonlinear rotational spring. t = -k*theta^2, where k is the spring rate.
```

Block name and block description come directly from the Simscape file.
Component Model

\begin{verbatim}
component NonlinRotSpring
  \% Nonlinear Rotational Spring
  \% This block implements a nonlinear rotational spring. \( t\!=\!-k\theta^2 \), where \( k \) is the spring rate.

  nodes
    r = foundation.mechanical.rotational.rotational;
    \% r:left
    c = foundation.mechanical.rotational.rotational;
    \% c:right
  end
\end{verbatim}

Node declaration creates domain specific physical ports

Reuse Simscape physical domains to create compatible components

Define the port label and the side where the port appears

Nonlinear Rotational Spring
Defining Components: Declaration

Component Model

```matlab
component NonlinRotSpring
% Nonlinear Rotational Spring
% This block implements a nonlinear rotational spring. \( t = -k\theta^2 \), where \( k \) is the spring rate.

definition

nodes

\( r = \) foundation.mechanical.rotational.rotational; % r:LEFT
\( c = \) foundation.mechanical.rotational.rotational; % c:RIGHT

end

variables

\( t = \) \{ 0, 'N*m' \}; % torque through
\( w = \) \{ 0, 'rad/s' \}; % velocity across
\( \theta = \) \{ 0, 'rad' \};

end
```

Variable section is used to define across and through variables as well as other internal variables.
Defining Components: Declaration

Component Model

```matlab
component NonlinRotSpring
% Nonlinear Rotational Spring
% This block implements a nonlinear rotational spring. \( t = -k \theta^2 \), where \( k \) is the spring rate.

statements

nodes
\[ r = \text{foundation}.\text{mechanical}.\text{rotational}.\text{rotational} \]; % r:left
\[ c = \text{foundation}.\text{mechanical}.\text{rotational}.\text{rotational} \]; % c:right
end

variables
\[ t = \{ 0, 'N*m' \}; \] % torque through
\[ w = \{ 0, 'rad/s' \}; \] % velocity across
\[ \theta = \{ 0, 'rad' \}; \] % angle across
end

parameters
\[ k = \{ 10, 'N*m/rad^2' \}; \] % Spring rate
\[ \theta_0 = \{ 0, 'rad' \}; \] % Initial Deformation
end
```

Parameter label, default value, and units defined in the file
Defining Components: Setup

- Setup section used to:
  - Validate parameters
  - Compute derived parameters
  - Set initial conditions
  - Define relationship between nodes and variables

- Executed once per component instance during model compilation as regular MATLAB code
  - Use function syntax, not command syntax

Component Model

Declaration
- Nodes
- Inputs, Outputs
- Variables
  - (through, across, and internal)
- Parameters

Setup
- Parameter checking
- Define relationship between component variables and nodes
- Initial conditions
- Derived parameters

Equations
- Algebraic, discontinuous, differential
Defining Components: Setup

Setup
Parameter checking

function setup
    if k < 0
        error('Spring rate must be greater than 0');
    end

across( w, r.w, c.w );
% velocity variable w from node r to node c

through( t, r.t, c.t );
% torque variable t from node r to node c

theta = theta0;
end

Define relationship between component variables and nodes

Initial conditions

Use functions across and through to define relationship between component variables and nodes

Leverage MATLAB for tasks such as analyzing parameters, performing preliminary computations, and initializing system variables
Defining Components: Equation

- Establishes mathematical relationships among component variables, inputs, outputs, time, and time derivatives
  - Can be DAEs, ODEs, etc.
  - Specifying inputs and outputs not required
- Executed throughout simulation

**Component Model**

**Declaration**
- Nodes
- Inputs, Outputs
- Variables (through, across, and internal)
- Parameters

**Setup**
- Parameter checking
- Define relationship between component variables and nodes
- Initial conditions
- Derived parameters

**Equations**
- Algebraic, discontinuous, differential
Defining Components: Equation

<table>
<thead>
<tr>
<th>equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>t == k * theta * theta * sign(theta);</td>
</tr>
<tr>
<td>w == theta.der;</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

Equations section establishes relationships among component variables, inputs, outputs, time, and time derivatives.

Torque = -
Angular Velocity = $\frac{k\theta^2}{dt}$

\[
\text{Torque} = - \quad \text{Angular Velocity} = \frac{d\theta}{dt}
\]
Equations in the Simscape Language

- `==` is used to represent a symmetrical mathematical relationship, NOT assignment and NOT a boolean operation.

```matlab
equations
    a + b == c;
end
```

Equivalent

```matlab
equations
    0 == c - a - b;
end
```

- Conditional equations can be defined using IF statements.

```matlab
equations
    if abs(x) < xc
        f == k*x;          % Linear region
    else
        f == k*xc*sign(x) + k1*sign(x)*(abs(x)-xc)^3;
    end
end
```
Equations in the Simscape Language

- Equations expressions can be vectors or matrices
  - Van der Pol's equation: \( x'' - \mu (1-x^2)x' + x = 0 \)

\[
\begin{align*}
  y &= \begin{bmatrix} x \\ x' \end{bmatrix}, \\
  y' &= \begin{bmatrix} x' \\ x'' \end{bmatrix} = \begin{bmatrix} y_2 \\ \mu (1-y_1^2) y_2 + y_1 \end{bmatrix}
\end{align*}
\]

```matlab
functions
    equations
        y.der == [y(2); \\
                  mu * (1 - y(1)^2) * y(2) - y(1)] * [1,'1/s'];
        out1 == y(1);
        out2 == y(2);
    end
end
```

Building Simscape Libraries

- Use `ssc_build` to build Simscape libraries
- Generates `<package>_lib.mdl` in parent of top level directory
  - Package directory must begin with a `+`
  - Parent of top level directory must be on MATLAB path

Executed in `.\Example_1\+Mechanical` or any subdirectory of `+Mechanical` creates `.\Example_1\Mechanical_lib.mdl`

Executed in `.\Example_1` creates `.\Example_1\Mechanical_lib.mdl`
2. Defining Physical Domains

- Define pneumatic domain
  - Across variable is pressure
  - Through variable is mass flow rate
  - Domain-wide parameter specific heat capacity ($C_p$)

<table>
<thead>
<tr>
<th>Physical Domain</th>
<th>Across Variable</th>
<th>Through Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pneumatic</td>
<td>Pressure</td>
<td>Mass flow rate</td>
</tr>
</tbody>
</table>

Domain Model

- Declaration
  - Across Variables
  - Through Variables
  - Parameters

```
+Pneumatic
+MySimscape
```

```
Pneumatic.ssc
```
Defining Domains (Pneumatic.ssc)

Domain Model

```matlab
domain Pneumatic

Declaration

Across Variables

variables
    p = { 1, 'Pa' }; % pressure
end

Through Variables

variables (Balancing = true)
    mdot = { 1, 'kg/s' }; % mass flow
end

Parameters

parameters
    Cp = { 1005, 'J/(kg*K)' }; % specific heat
end

end
```

Use keywords `variables` to define the across and through variables for the new domain

Use keyword `parameters` to define network-wide parameters specific to this domain
The Value of Model-Based Design

Innovation
- Explore unique features through rapid design iterations
- Conduct cost-effective design trade-off studies

Quality
- Prevent errors from reaching the hardware stage of the design process
- Reduce rework

Cost
- Reduce expensive physical prototypes
- Reduce testing costs

Time-to-market
- Get it right the first time
- Accelerate the development process
Summary

- **Modelling:**
  
  *understanding your design problem*

- **Control Design:**
  
  *solving your design problem*

- **Real-time Testing:**
  
  *verifying the design*