FORCESPRO: MATHWORKS MPC TOOLBOX PLUGIN

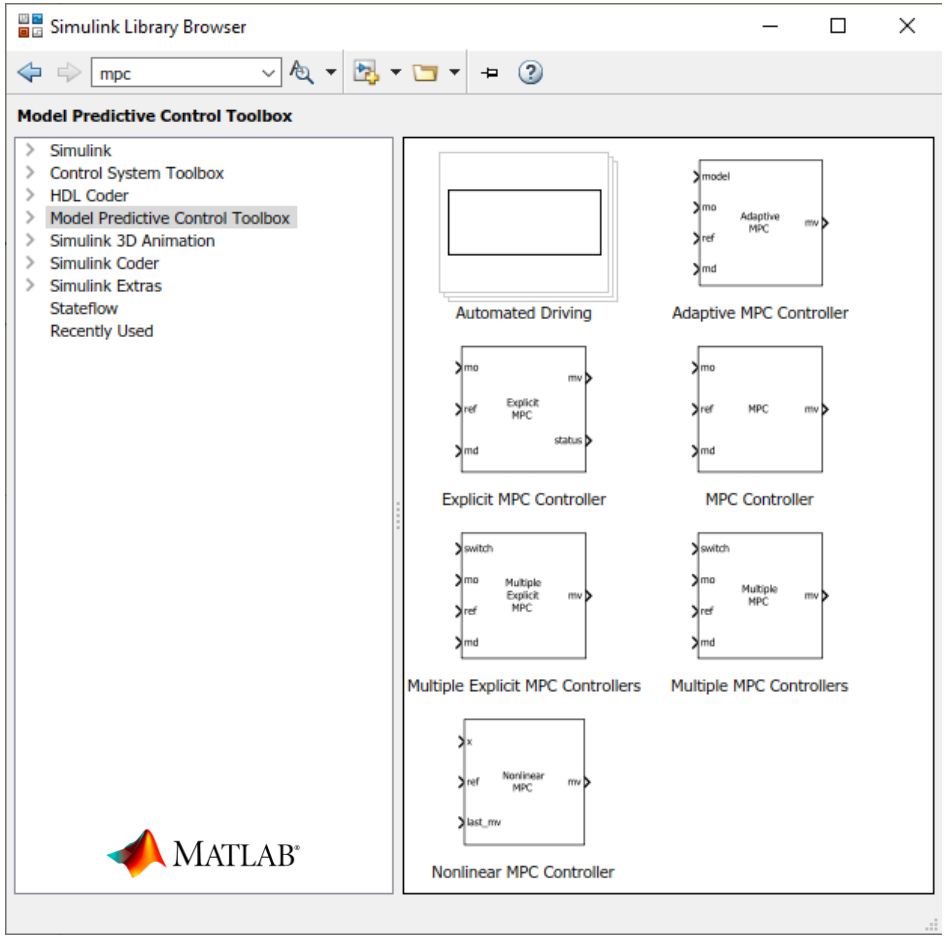
Uros Markovic
Optimization Specialist

2022 American Control Conference
Atlanta, GA, USA / June 2022

embotech

MATHWORKS MPC TOOLBOX PLUGIN

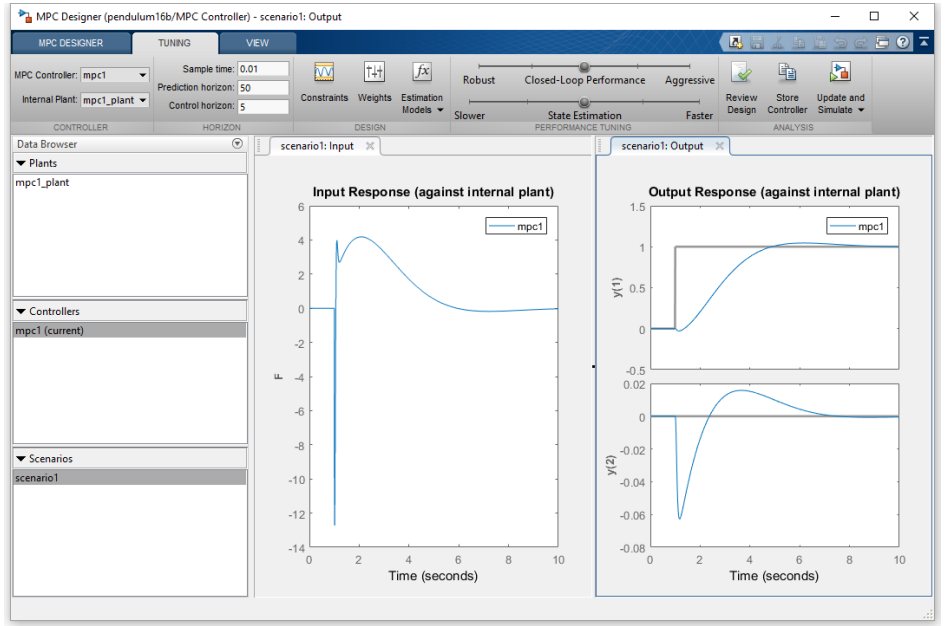
- Use of **FORCES**PRO solver in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Enables code deployment of the **FORCES**PRO solver on real-time hardware from within MATLAB and Simulink
- Leverages the **powerful design capabilities** of MPC Toolbox and the **computational performance** of **FORCES**PRO
- Additional features such as Simulink blocks that can generate code runnable on embedded targets such as dSpace
- Generated code is **highly optimized** for **fast computations** and **low memory footprint**
- Both **linear** and **nonlinear** MPC Toolbox plugins available

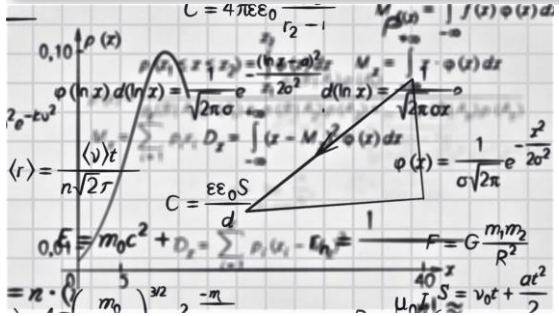


embotech

MATHWORKS MPC TOOLBOX PLUGIN

- Use of **FORCES**PRO solver in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Enables code deployment of the **FORCES**PRO solver on real-time hardware from within MATLAB and Simulink
- Leverages the **powerful design capabilities** of MPC Toolbox and the **computational performance** of **FORCES**PRO
- Additional features such as Simulink blocks that can generate code runnable on embedded targets such as dSpace
- Generated code is **highly optimized** for **fast computations** and **low memory footprint**
- Both **linear** and **nonlinear** MPC Toolbox plugins available





FORCESPRO

embotech

LINEAR MPC TOOLBOX PLUGIN

embotech

LINEAR MPC PLUGIN

- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* - for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* - for calling the generated solver on a specific MPC problem instance
  - *mpcCustomSolver* - for using the **FORCES**PRO dense QP solver as a custom solver
  - *mpcToForcesOptions* - an auxiliary file exposed to the users for generating different solver options

embotech

LINEAR MPC PLUGIN

- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - ***mpcToForces*** - for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - ***mpcmoveForces*** - for calling the generated solver on a specific MPC problem instance
  - *mpcCustomSolver* - for using the **FORCES**PRO dense QP solver as a custom solver
  - *mpcToForcesOptions* - an auxiliary file exposed to the users for generating different solver options

```
% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);

% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);
```

embotech

LINEAR MPC PLUGIN

- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* - for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* - for calling the generated solver on a specific MPC problem instance
  - *mpcCustomSolver* - for using the **FORCES**PRO dense QP solver as a custom solver
  - *mpcToForcesOptions* - an auxiliary file exposed to the users for generating different solver options

LTI MPC output of *mpc(...)*

```
% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);

% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);
```

embotech

LINEAR MPC PLUGIN

- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* - for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* - for calling the generated solver on a specific MPC problem instance
  - *mpcCustomSolver* - for using the **FORCES**PRO dense QP solver as a custom solver
  - *mpcToForcesOptions* - an auxiliary file exposed to the users for generating different solver options

output of *mpcToForcesOptions(...)*

```
% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);

% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);
```

embotech

LINEAR MPC PLUGIN

- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* - for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* - for calling the generated solver on a specific MPC problem instance
  - *mpcCustomSolver* - for using the **FORCES**PRO dense QP solver as a custom solver
  - *mpcToForcesOptions* - an auxiliary file exposed to the users for generating different solver options

output of *mpcToForcesOptions(...)*

```
% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);

% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);
```

Two types of **QP solvers** can be generated via *mpcToForces* (provided via *mpcToForcesOptions* arguments):

- **"sparse"** solver - corresponds to a multi-stage formulation as in low-level interface (*customForcesSparseQP.m*)
- **"dense"** solver - corresponds to a single-stage QP with inequality constraints (*customForcesDenseQP.m*)

embotech

LINEAR MPC PLUGIN

- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* - for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* - for calling the generated solver on a specific MPC problem instance
  - *mpcCustomSolver* - for using the **FORCES**PRO dense QP solver as a custom solver
  - *mpcToForcesOptions* - an auxiliary file exposed to the users for generating different solver options

```
% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);

% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);
```

constant data needed to
construct QP at run-time

LINEAR MPC PLUGIN

- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* - for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* - for calling the generated solver on a specific MPC problem instance
  - *mpcCustomSolver* - for using the **FORCES**PRO dense QP solver as a custom solver
  - *mpcToForcesOptions* - an auxiliary file exposed to the users for generating different solver options

```
% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);

% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);
```

prediction model states
and last optimal MV

LINEAR MPC PLUGIN

- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* - for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* - for calling the generated solver on a specific MPC problem instance
  - *mpcCustomSolver* - for using the **FORCES**PRO dense QP solver as a custom solver
  - *mpcToForcesOptions* - an auxiliary file exposed to the users for generating different solver options

```
% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);

% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);
```

online signals
(*signals*, *weights*, *constraints*)

embotech

LINEAR MPC PLUGIN

- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* - for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* - for calling the generated solver on a specific MPC problem instance
  - *mpcCustomSolver* - for using the **FORCES**PRO dense QP solver as a custom solver
  - *mpcToForcesOptions* - an auxiliary file exposed to the users for generating different solver options

```
% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);

% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);
```

optimal manipulated variables
at current solve time instant

LINEAR MPC PLUGIN

- Currently, only **convex quadratic programs** are supported by the Linear MPC plugin
- The plugin mainly consists of the following MATLAB commands:
  - *mpcToForces* - for generating a **FORCES**PRO solver from an MPC object designed by the MPC Toolbox
  - *mpcmoveForces* - for calling the generated solver on a specific MPC problem instance
  - *mpcCustomSolver* - for using the **FORCES**PRO dense QP solver as a custom solver
  - *mpcToForcesOptions* - an auxiliary file exposed to the users for generating different solver options

```
% Generating a QP solver from an MPC object
[coredata,statedata,onlinedata] = mpcToForces(mpcobj,options);

% Solving a QP from MPC online data
[mv,statedata,info] = mpcmoveForces(coredata,statedata,onlinedata);
```

information about the
FORCESPRO solve

embotech

SIMULINK INTERFACE

- Both **sparse** and **dense** solvers **can be used inside Simulink**

DENSE QP SIMULINK INTERFACE

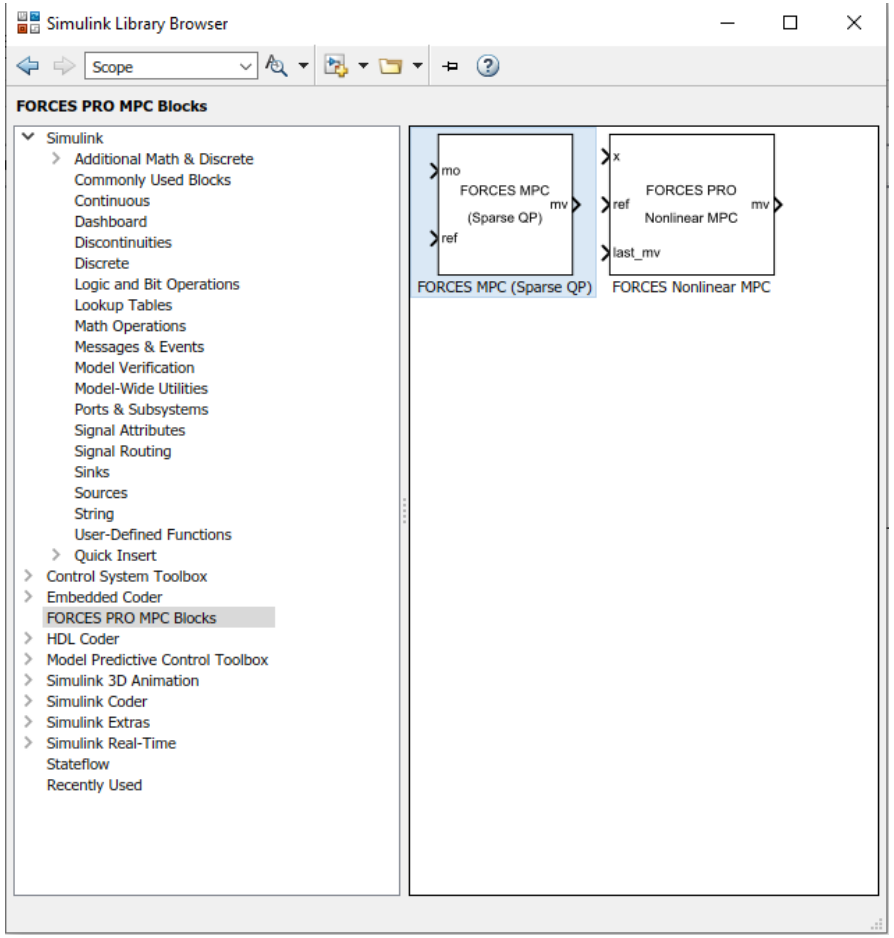
- Both **sparse** and **dense** solvers **can be used inside Simulink**
- **Dense QP** formulation usable from the shipped **Simulink MPC controller** block directly

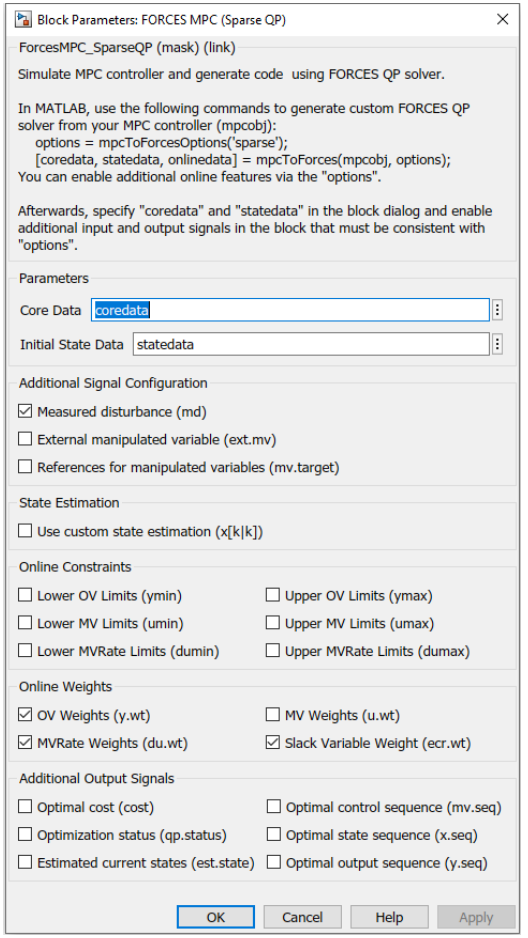
```
% Generate a custom dense FORCESPRO solver
options = mpcToForcesOptions('dense');
mpcToForces(mpcobj, options);

% Enable the settings in the MPC object
mpcobj.Optimizer.CustomSolver = true;
mpcobj.Optimizer.CustomSolverCodeGen = true;
```

embotech

SPARSE QP SIMULINK INTERFACE

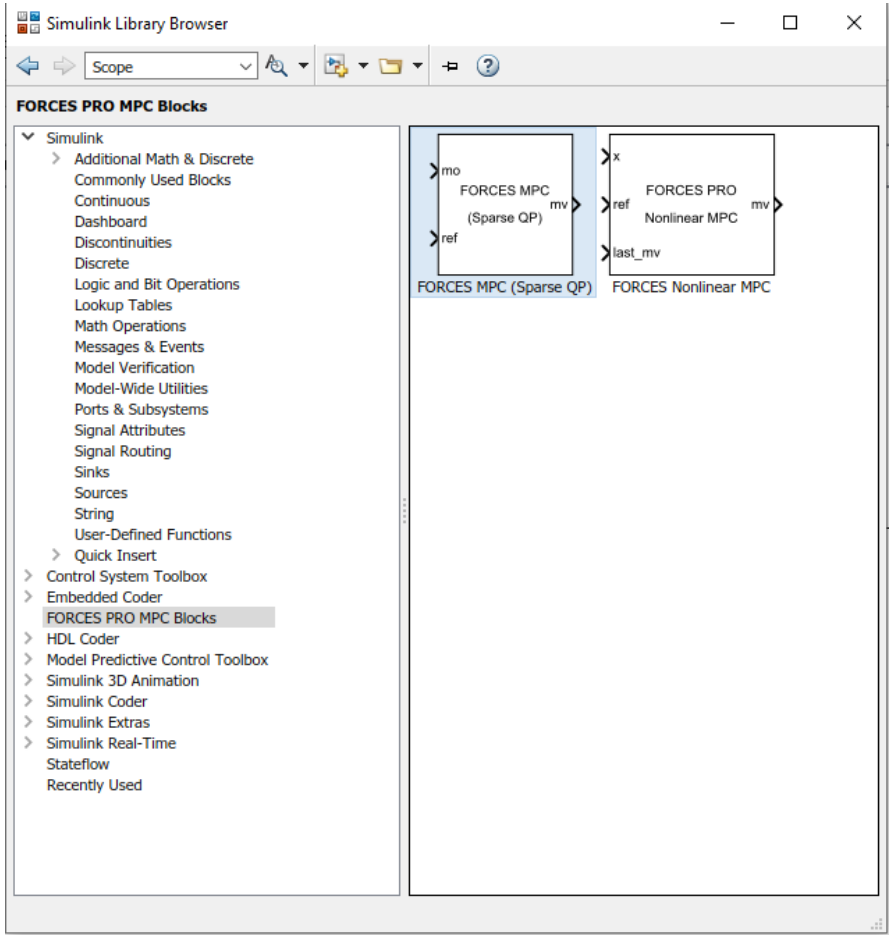
- Both **sparse** and **dense** solvers **can be used inside Simulink**
- **Sparse QP** solver also available via the **MPC Toolbox** and **FORCES**PRO **MPC block** in the Library browser
- Configuration done via Simulink UI
- All MATLAB plugin features are **available** through this Simulink block

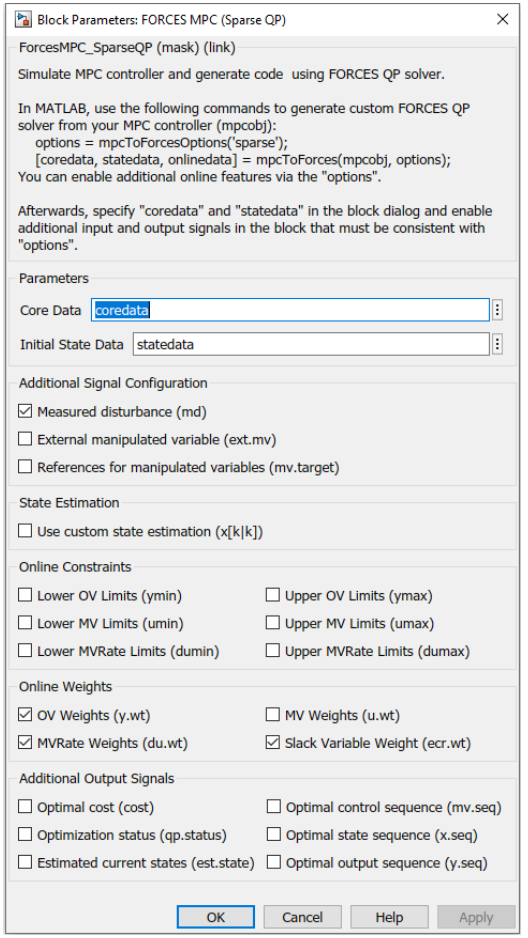




SPARSE QP SIMULINK INTERFACE

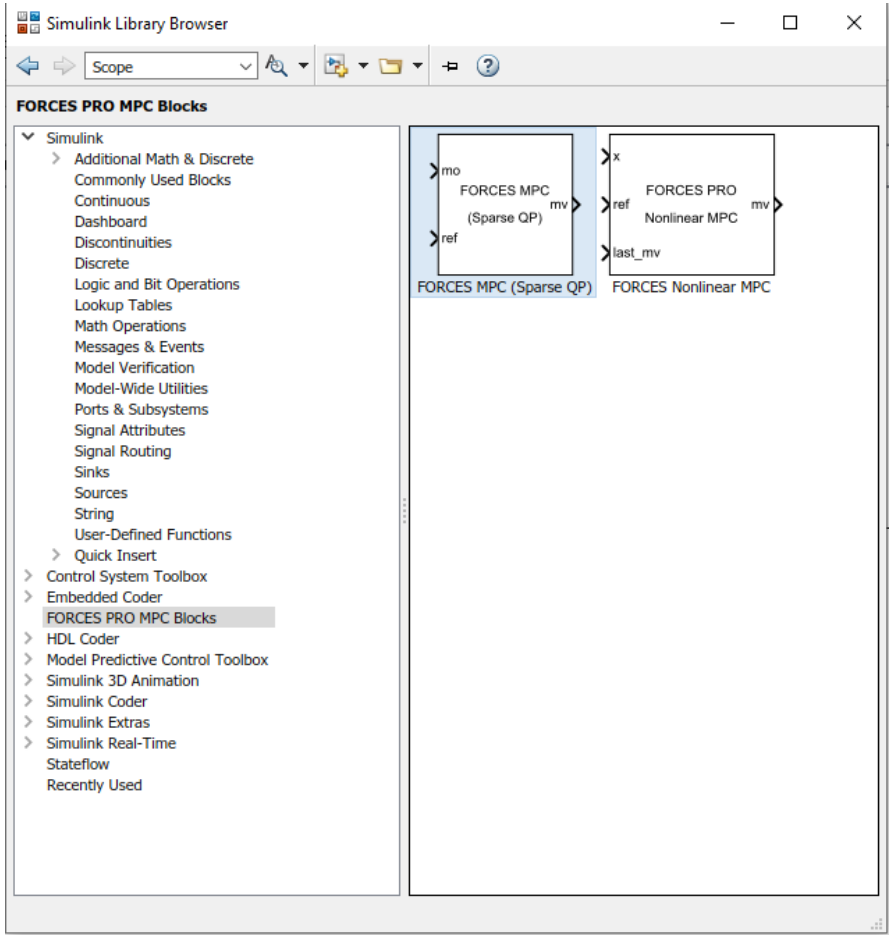
- Both **sparse** and **dense** solvers **can be used inside Simulink**
- **Sparse QP** solver also available via the **MPC Toolbox** and **FORCES**PRO **MPC block** in the Library browser
- Configuration done via Simulink UI
- All MATLAB plugin features are **available** through this Simulink block
  - measured disturbances (*md*)

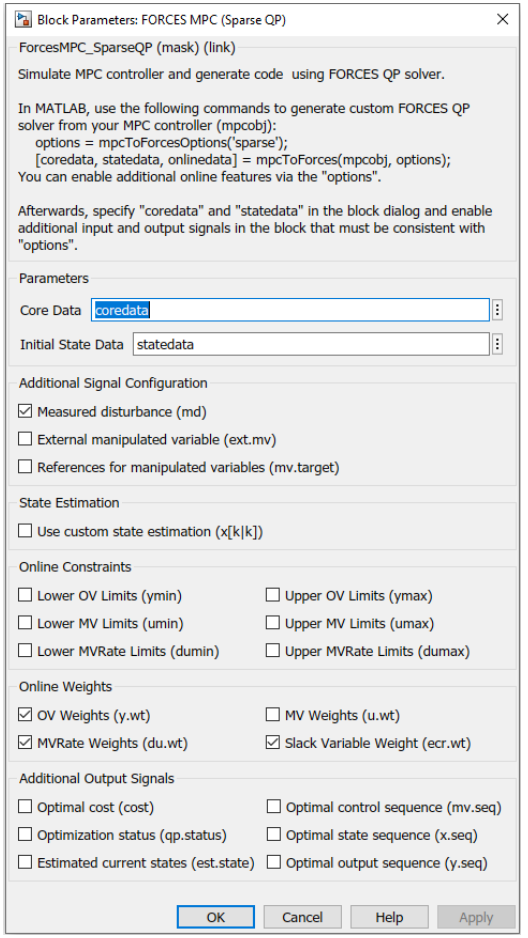




SPARSE QP SIMULINK INTERFACE

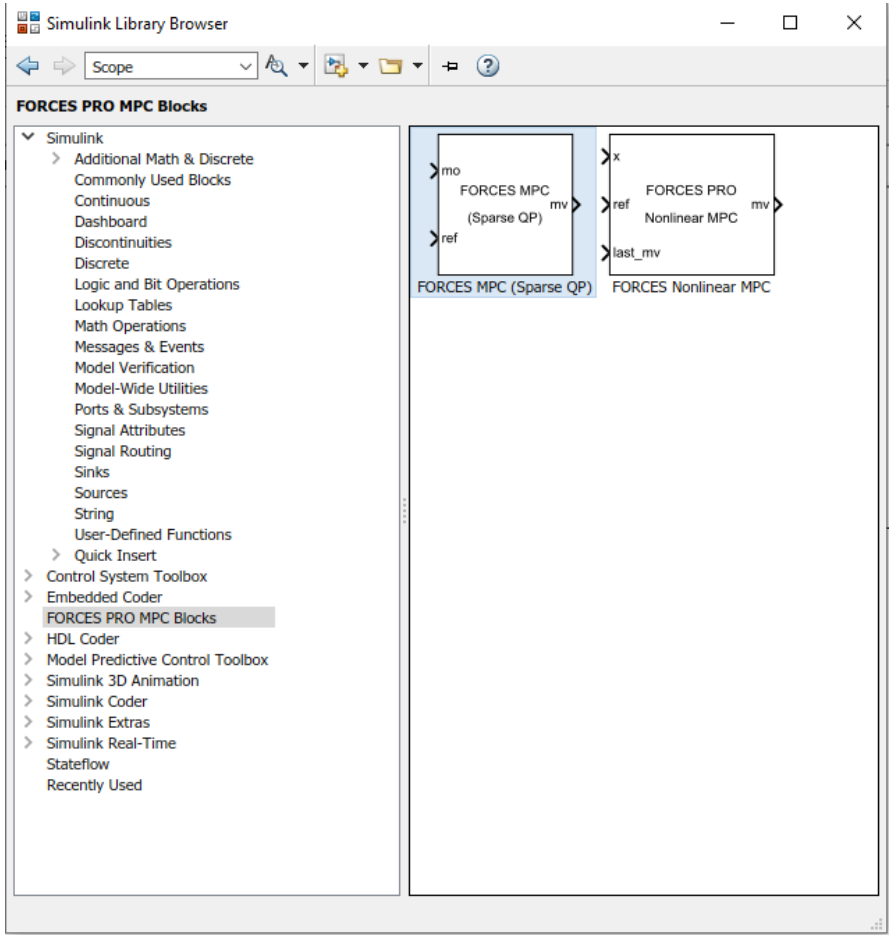
- Both **sparse** and **dense** solvers **can be used inside Simulink**
- **Sparse QP** solver also available via the **MPC Toolbox** and **FORCES**PRO **MPC block** in the Library browser
- Configuration done via Simulink UI
- All MATLAB plugin features are **available** through this Simulink block
  - measured disturbances (*md*)
  - external manipulated vars (*ext.mv*)

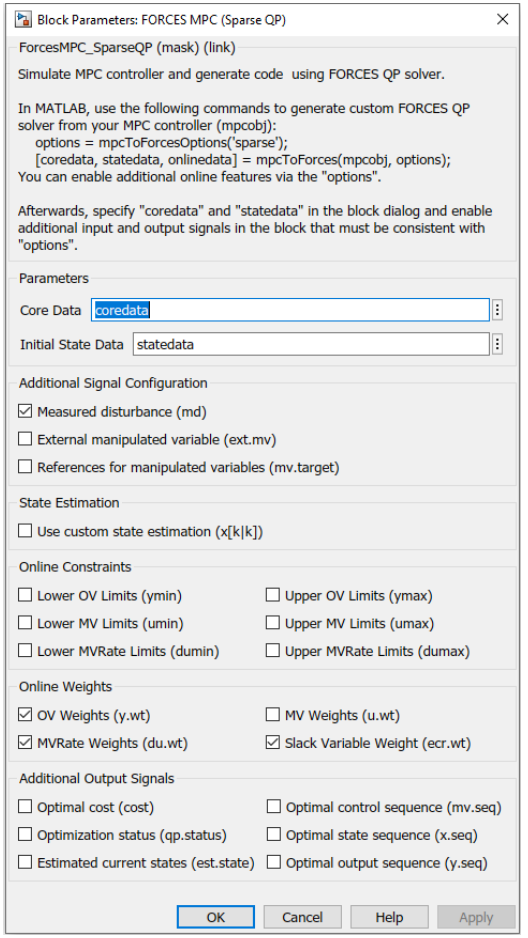




SPARSE QP SIMULINK INTERFACE

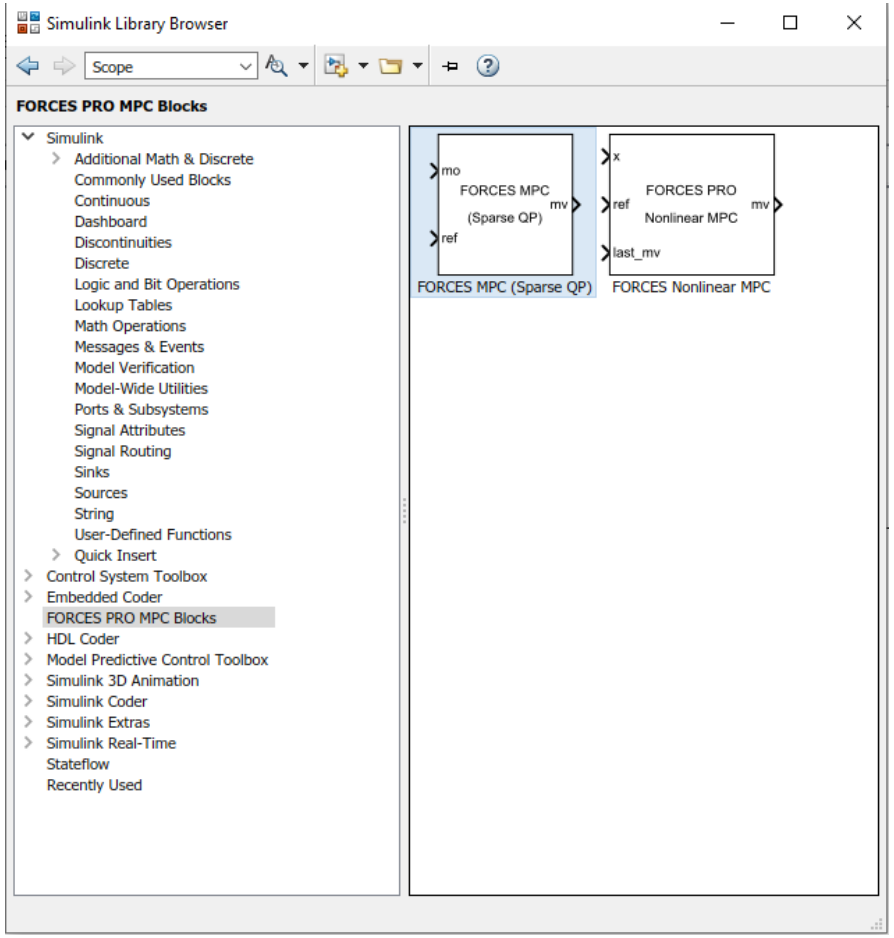
- Both **sparse** and **dense** solvers **can be used inside Simulink**
- **Sparse QP** solver also available via the **MPC Toolbox** and **FORCES**PRO **MPC block** in the Library browser
- Configuration done via Simulink UI
- All MATLAB plugin features are **available** through this Simulink block
  - measured disturbances (*md*)
  - external manipulated vars (*ext.mv*)
  - ref. for manipulated vars (*mv.target*)

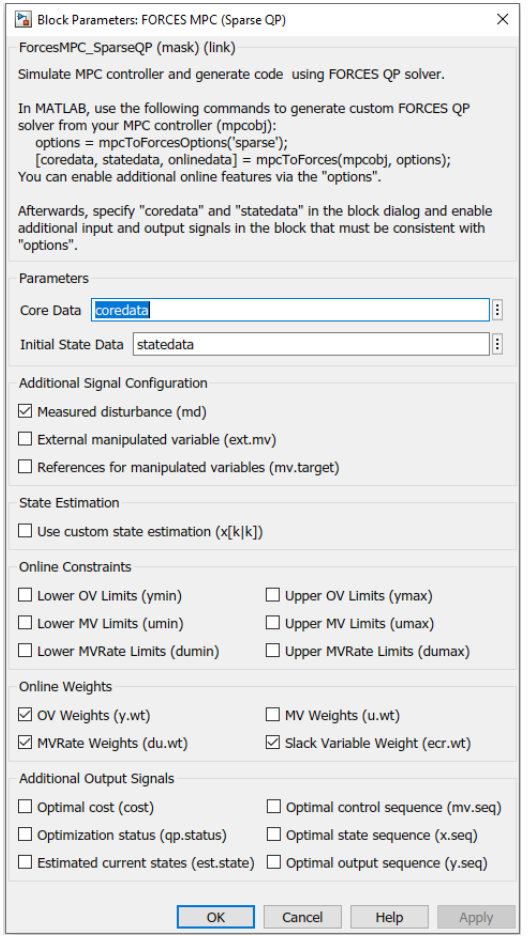




SPARSE QP SIMULINK INTERFACE

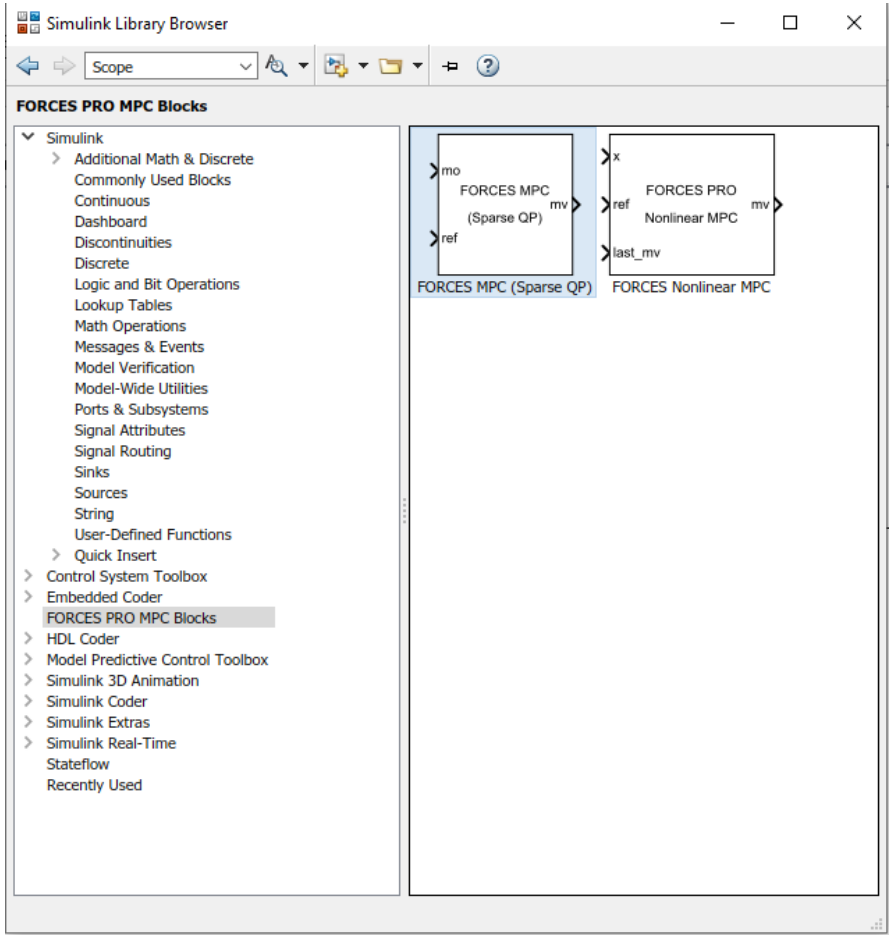
- Both **sparse** and **dense** solvers **can be used inside Simulink**
- **Sparse QP** solver also available via the **MPC Toolbox** and **FORCES**PRO **MPC block** in the Library browser
- Configuration done via Simulink UI
- All MATLAB plugin features are **available** through this Simulink block
  - measured disturbances (*md*)
  - external manipulated vars (*ext.mv*)
  - ref. for manipulated vars (*mv.target*)
  - custom state estimation

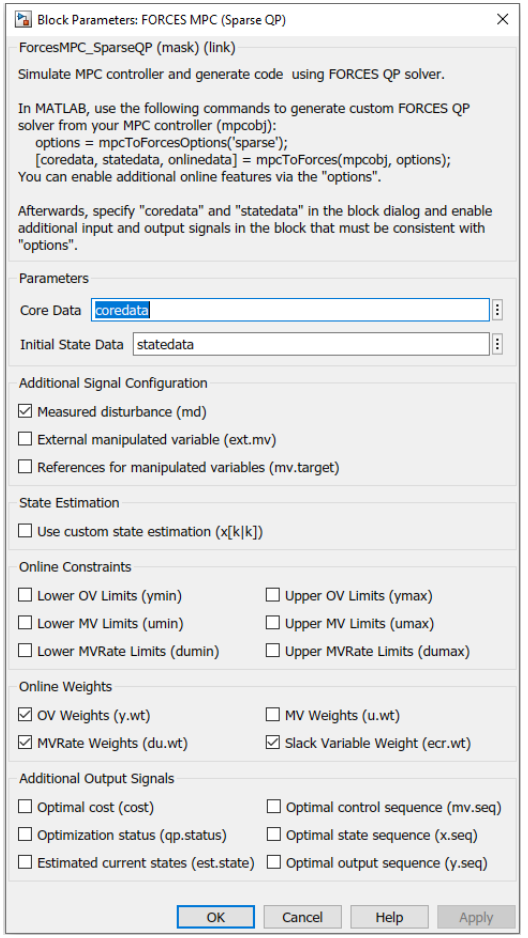




SPARSE QP SIMULINK INTERFACE

- Both **sparse** and **dense** solvers **can be used inside Simulink**
- **Sparse QP** solver also available via the **MPC Toolbox** and **FORCESPRO MPC block** in the Library browser
- Configuration done via Simulink UI
- All MATLAB plugin features are **available** through this Simulink block
  - measured disturbances (*md*)
  - external manipulated vars (*ext.mv*)
  - ref. for manipulated vars (*mv.target*)
  - custom state estimation
  - online weights and constraints

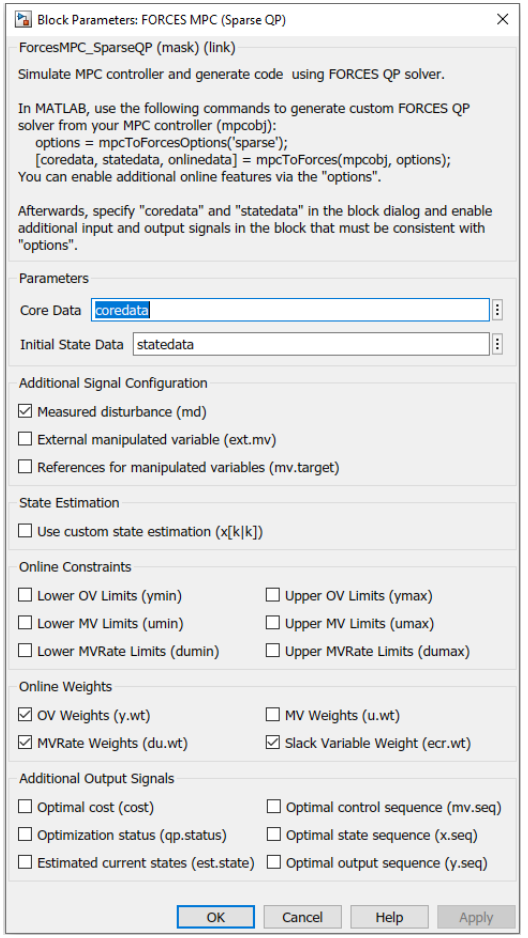




SPARSE QP SIMULATION WORKFLOW

- Generate **FORCES**PRO **sparse QP** solver

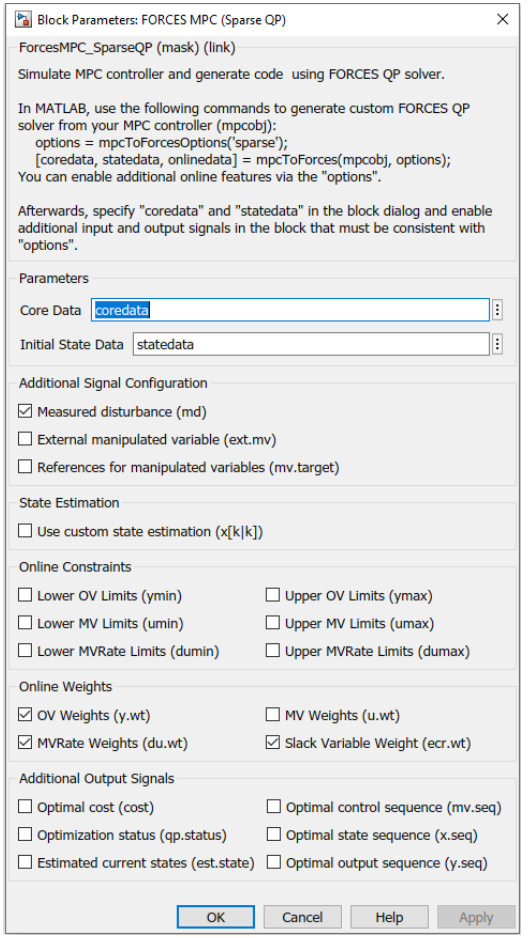
```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');

% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```



SPARSE QP SIMULATION WORKFLOW

- Generate **FORCES**PRO **sparse QP** solver
- Provide *coredata*, *statedata* and *onlinedata* to the Simulink block

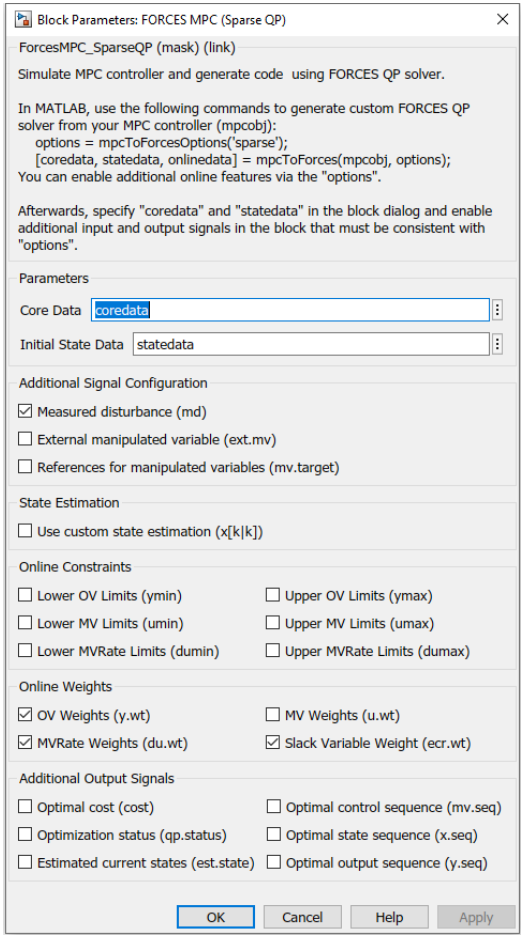
```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');

% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```



SPARSE QP SIMULATION WORKFLOW

- Generate **FORCES**PRO **sparse QP** solver
- Provide *coredata*, *statedata* and *onlinedata* to the Simulink block
- Select md checkbox if MD channels exist in the MPC object

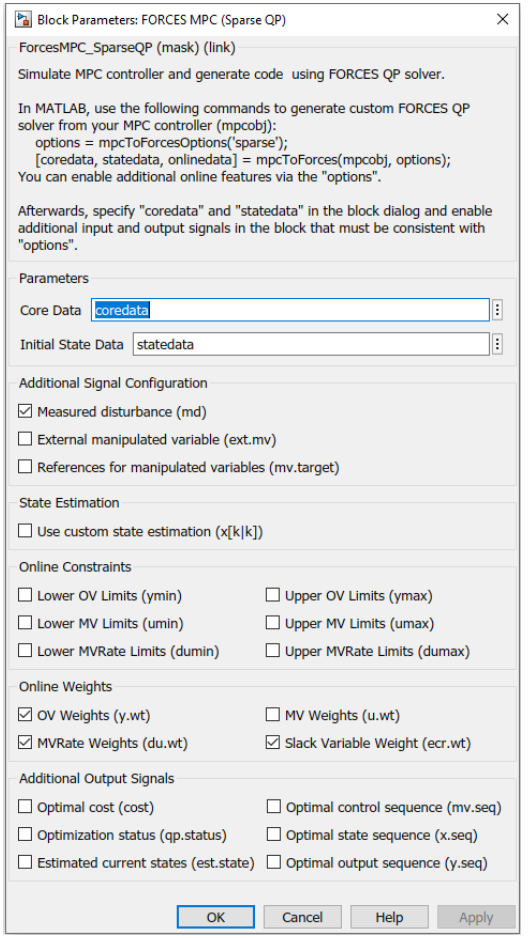
```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');

% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```



SPARSE QP SIMULATION WORKFLOW

- Generate **FORCES**PRO **sparse QP** solver
- Provide *coredata*, *statedata* and *onlinedata* to the Simulink block
- Select md checkbox if MD channels exist in the MPC object
- Select x[k|k] checkbox for using a custom state estimator

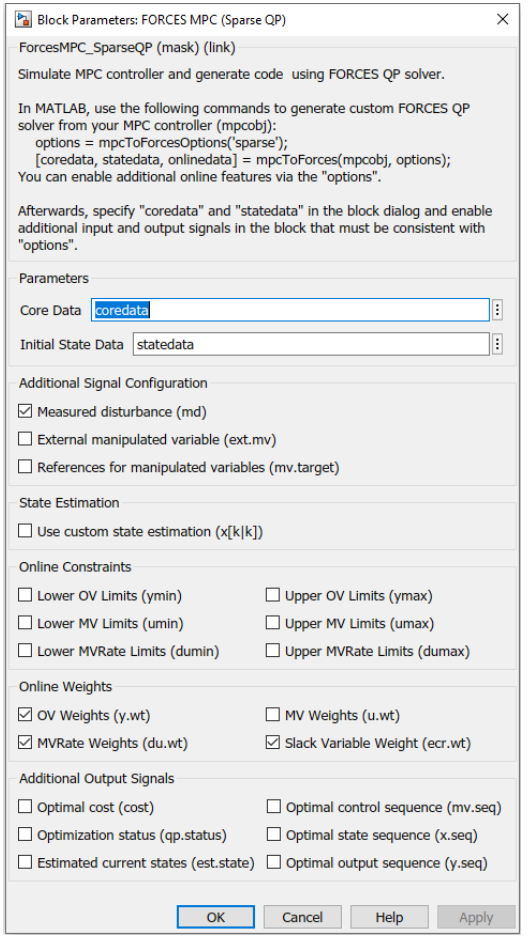
```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');

% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```



SPARSE QP SIMULATION WORKFLOW

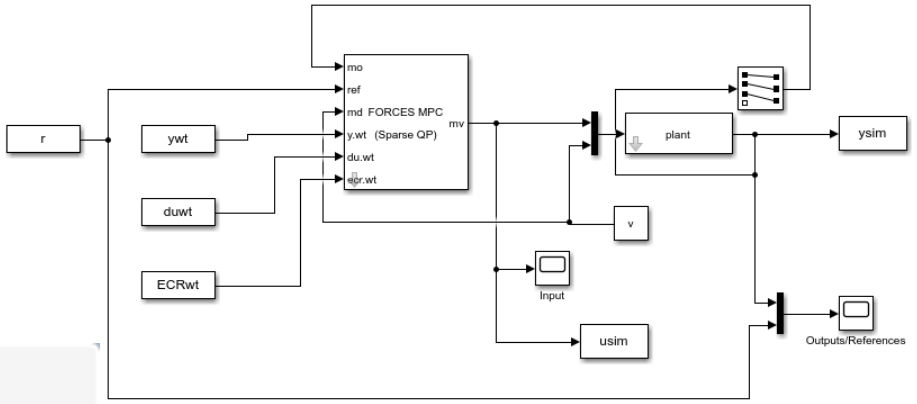
- Generate **FORCES**PRO **sparse QP** solver
- Provide *coredata*, *statedata* and *onlinedata* to the Simulink block
- Select md checkbox if MD channels exist in the MPC object
- Select x[k|k] checkbox for using a custom state estimator
- Monitor qp.status output for feasibility of the MPC block solution

```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');

% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```



SPARSE QP SIMULATION WORKFLOW

- Generate **FORCES**PRO **sparse QP** solver
- Provide *coredata*, *statedata* and *onlinedata* to the Simulink block
- Select md checkbox if MD channels exist in the MPC object
- Select x[k|k] checkbox for using a custom state estimator
- Monitor qp.status output for feasibility of the MPC block solution

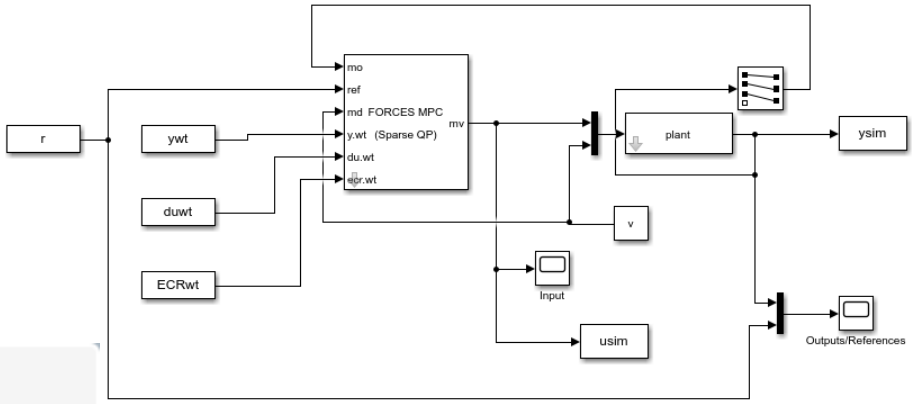



```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');

% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);

% Simulation
open_system('forcesmpc_onlinetuning'); % Open Simulink model
sim(mdl); % Start simulation
```

embotech

DEPLOYMENT TO DSPACE MICROAUTOBOX II

- Generate **FORCES**PRO **sparse QP** solver for the **targeted platform**

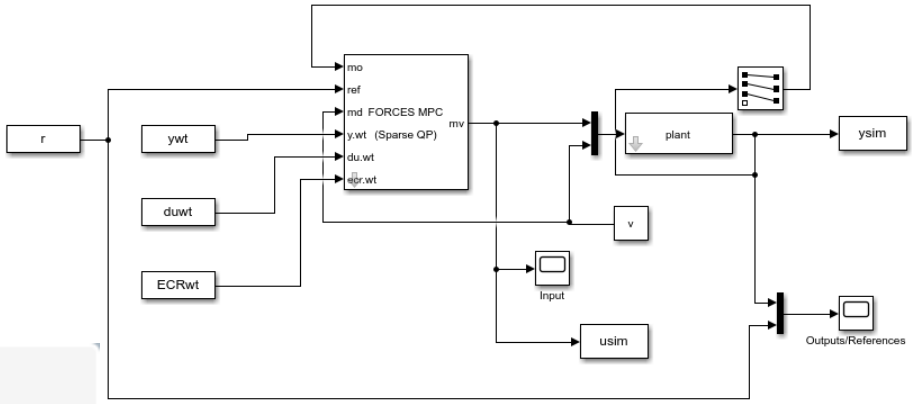



```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');

% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
options.ForcesTargetPlatform = 'dSPACE-MABII';
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```

embotech

DEPLOYMENT TO DSPACE MICROAUTOBOX II

- Generate **FORCES**PRO **sparse QP** solver for the **targeted platform**
- *options.ForcesTargetPlatform* needs to be specified

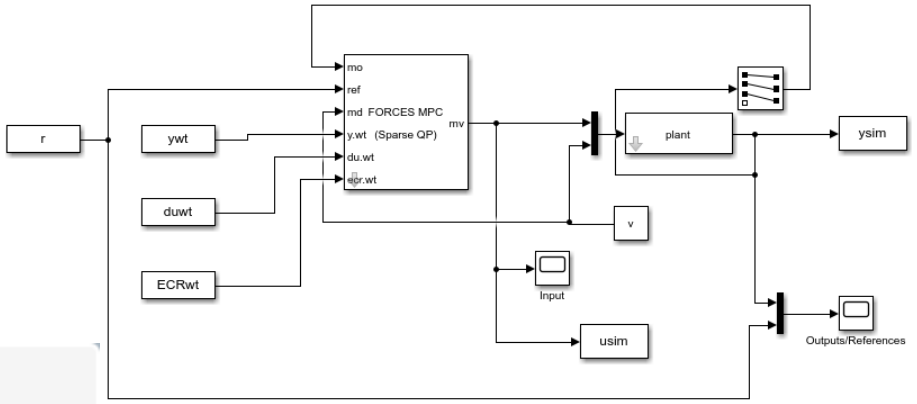



```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');

% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
options.ForcesTargetPlatform = 'dSPACE-MABII';
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```

embotech

DEPLOYMENT TO DSPACE MICROAUTOBOX II

- Generate **FORCES**PRO **sparse QP** solver for the **targeted platform**
- *options.ForcesTargetPlatform* needs to be specified
- Provide *coredata*, *statedata* and *onlinedata* to the Simulink block



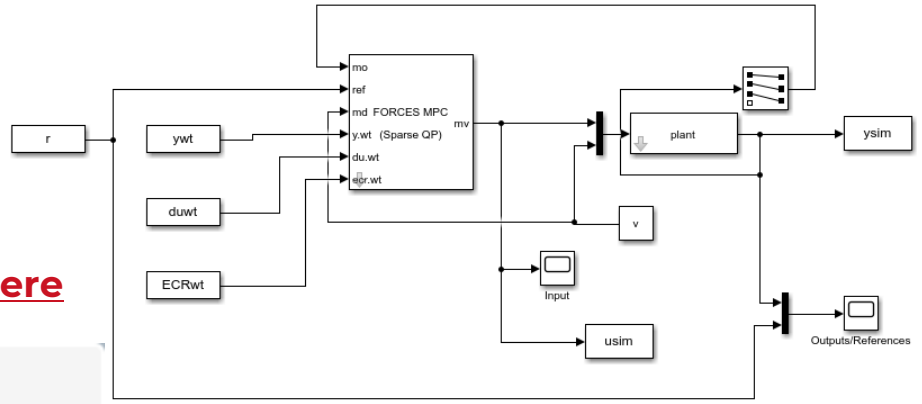

```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');

% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
options.ForcesTargetPlatform = 'dSPACE-MABII';
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```

embotech

DEPLOYMENT TO DSPACE MICROAUTOBOX II

- Generate **FORCES**PRO **sparse QP** solver for the **targeted platform**
- *options.ForcesTargetPlatform* needs to be specified
- Provide *coredata*, *statedata* and *onlinedata* to the Simulink block

More details on deployment to dSPACE MicroAutobox II **can be found here**




```
% Generate FORCESPRO sparse QP solver
options = mpcToForcesOptions('sparse');

% Specify online weights on outputs, input rates and ECR slacks
options.UseOnlineWeightOV = true;
options.UseOnlineWeightMVRate = true;
options.UseOnlineWeightECR = true;
options.ForcesTargetPlatform = 'dSPACE-MABII';
[coredata, statedata, onlinedata] = mpcToForces(mpcobj, options);
```

embotech

NONLINEAR MPC TOOLBOX PLUGIN

embotech

NONLINEAR MPC PLUGIN

- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- **FORCES**PRO automatically **generates jacobian functions** (via automatic differentiation tool CasADi)
- **State** and **output jacobian** functions that may be provided within the MPC prediction model ***will be ignored***
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin **can also use the MATLAB Symbolic Math Toolbox**

embotech

NONLINEAR MPC PLUGIN

- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- **FORCES**PRO automatically **generates jacobian functions** (via automatic differentiation tool CasADi)
- **State** and **output jacobian** functions that may be provided within the MPC prediction model ***will be ignored***
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin **can also use the MATLAB Symbolic Math Toolbox**

Depending on the chosen object, the nonlinear MPC plugin consists of two API methods:

- *nlmpcToForces* / *nlmpcMultistageToForces* - generates a **FORCES**PRO nonlinear solver from a nonlinear MPC (*nlmpc*) or a nonlinear MPC multistage (*nlmpcMultistage*) object designed by the MPC Toolbox
- *nlmpcmoveForces* / *nlmpcmoveForcesMultistage* - calls the generated solver to calculate optimal control actions

embotech

NONLINEAR MPC PLUGIN

- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- **FORCES**PRO automatically **generates jacobian functions** (via automatic differentiation tool CasADi)
- **State** and **output jacobian** functions that may be provided within the MPC prediction model ***will be ignored***
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin **can also use the MATLAB Symbolic Math Toolbox**

```
% Generating the NLP solver
[coredata, onlinedata] = nlmpcToForces(nlobj, options);
```

embotech

NONLINEAR MPC PLUGIN

- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- **FORCES**PRO automatically **generates jacobian functions** (via automatic differentiation tool CasADi)
- **State** and **output jacobian** functions that may be provided within the MPC prediction model ***will be ignored***
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin **can also use the MATLAB Symbolic Math Toolbox**

nlmpc / *nlmpcMultistage* object

NONLINEAR MPC PLUGIN

- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- **FORCES**PRO automatically **generates jacobian functions** (via automatic differentiation tool CasADi)
- **State** and **output jacobian** functions that may be provided within the MPC prediction model ***will be ignored***
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin **can also use the MATLAB Symbolic Math Toolbox**

```
% Generating the NLP solver
[coredata, onlinedata] = nlmpcToForces(nlobj, options);

% Calling the solver
[mv, onlinedata, info] = nlmpcmoveForces(coredata, x, mv, onlinedata); % MPC Toolbox interface
```

embotech

NONLINEAR MPC PLUGIN

- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- **FORCES**PRO automatically **generates jacobian functions** (via automatic differentiation tool CasADi)
- **State** and **output jacobian** functions that may be provided within the MPC prediction model ***will be ignored***
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin **can also use the MATLAB Symbolic Math Toolbox**

```
% Generating the NLP solver
[coredata, onlinedata] = nlmpcToForces(nlobj, options);

% Calling the solver
[mv, onlinedata, info] = nlmpcmoveForces(coredata, x, mv, onlinedata); % MPC Toolbox interface
[mv, onlinedata, info] = nlmpcmove_CstrSolver(x, mv, onlinedata); % MEX function (speed up)
```

embotech

NONLINEAR MPC PLUGIN

- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- **FORCES**PRO automatically **generates jacobian functions** (via automatic differentiation tool CasADi)
- **State** and **output jacobian** functions that may be provided within the MPC prediction model ***will be ignored***
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin **can also use the MATLAB Symbolic Math Toolbox**

```
% Generating the NLP solver
[coredata, onlinedata] = nlmpcToForces(nlobj, options);

% Calling the solver
[mv, onlinedata, info] = nlmpcmoveForces(coredata, x, mv, onlinedata); % MPC Toolbox interface
[mv, onlinedata, info] = nlmpcmove_CstrSolver(x, mv, onlinedata); % MEX function (speed up)
```

current prediction model states

embotech

NONLINEAR MPC PLUGIN

- Use of **FORCES**PRO **nonlinear interior-point (IP)** and **sequential quadratic programming (SQP)** solvers in MATLAB and Simulink from within the MATLAB Model Predictive Control Toolbox
- Code-generated IP and SQP solvers are **not based** on finite-difference derivatives computation, resulting in **faster convergence**
- **FORCES**PRO automatically **generates jacobian functions** (via automatic differentiation tool CasADi)
- **State** and **output jacobian** functions that may be provided within the MPC prediction model ***will be ignored***
- Comes with Simulink libraries that enable users to run the **FORCES**PRO solvers from within Simulink models
- Nonlinear MPC plugin **can also use the MATLAB Symbolic Math Toolbox**

```
% Generating the NLP solver
[coredata, onlinedata] = nlmpcToForces(nlobj, options);

% Calling the solver
[mv, onlinedata, info] = nlmpcmoveForces(coredata, x, mv, onlinedata); % MPC Toolbox interface
[mv, onlinedata, info] = nlmpcmove_CstrSolver(x, mv, onlinedata); % MEX function (speed up)
```

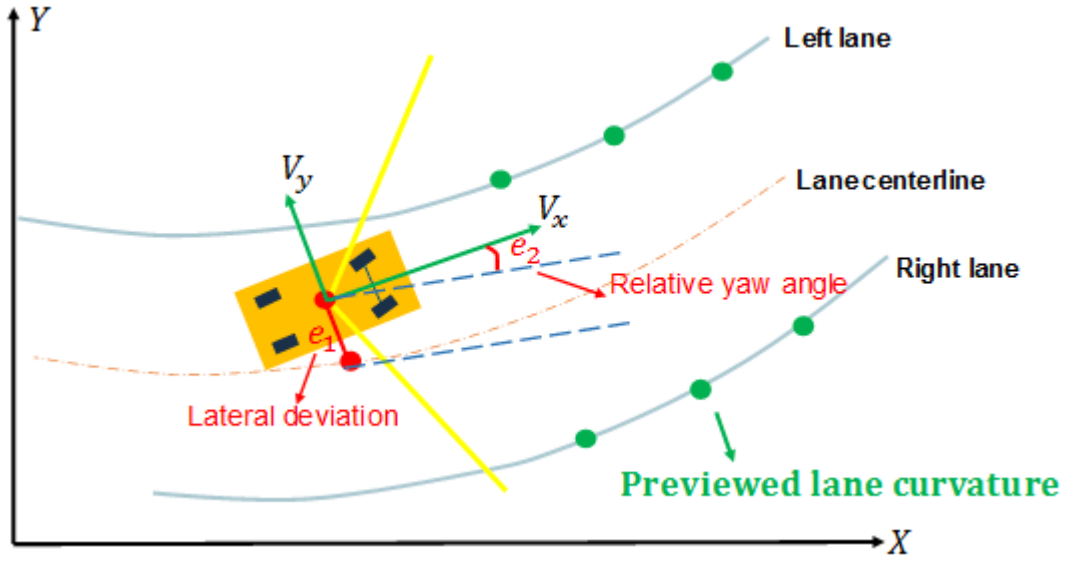
last optimal MV

embotech

EXAMPLE: LANE FOLLOWING

- Lane-following system **keeps the vehicle** traveling **along the centerline** of a highway lane, while **maintaining a user-set velocity**

- Manipulates both the **longitudinal acceleration** and the **front steering angle** of the vehicle in order to:
  - keep the **lateral deviation e_1** and **relative yaw angle e_2 small**
  - keep the **longitudinal velocity V_x** close to a driver-set velocity
  - **balance the above two goals** when they cannot be met simultaneously



Detailed **example description** and **MATLAB code** are **available for download here**

embotech

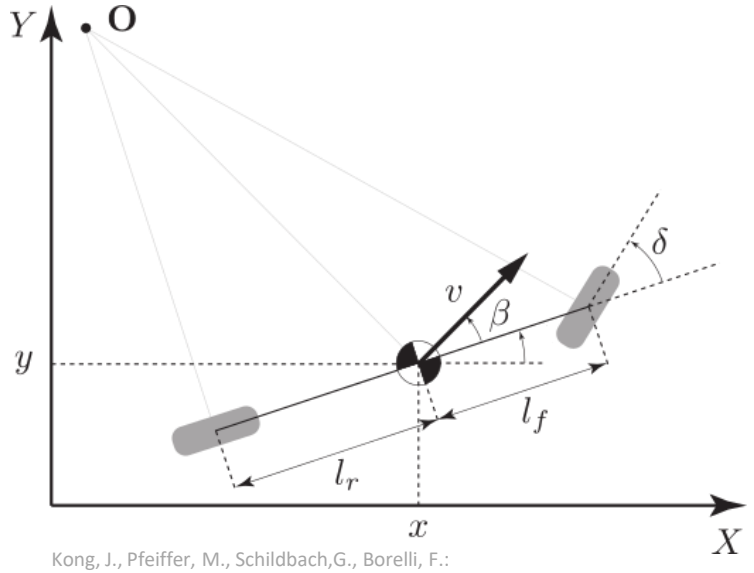
SYSTEM COMPONENTS

Model contains **four** main components:

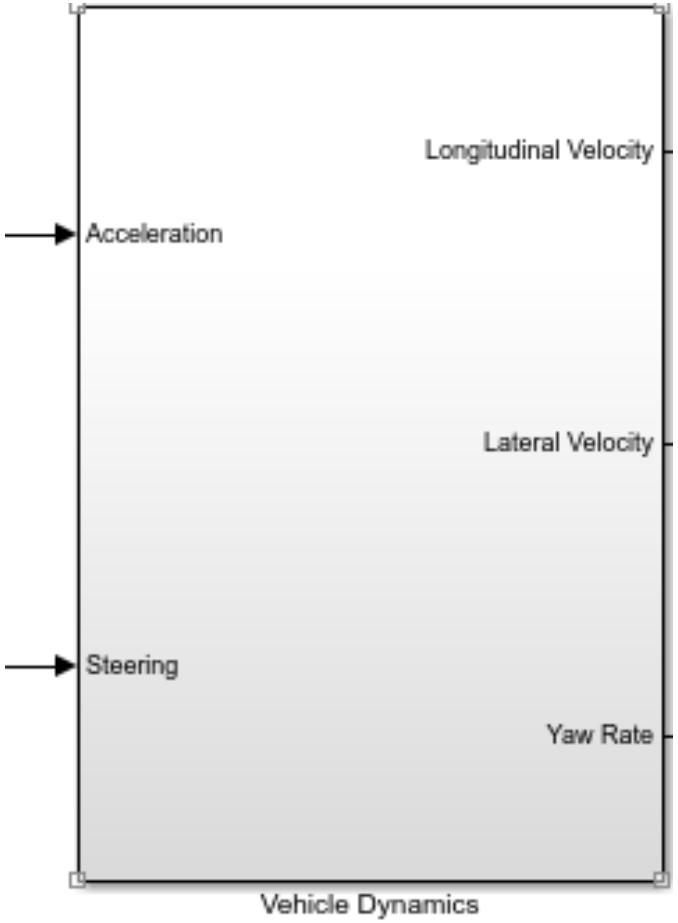
SYSTEM COMPONENTS

Model contains **four** main components:

- **Vehicle dynamics:** applies **3rd-order bicycle model** of **lateral vehicle dynamics** and approximates longitudinal dynamics using a time constant τ



Kong, J., Pfeiffer, M., Schildbach,G., Borelli, F.:
''Kinematic and dynamic models for autonomous driving control design.''
2015 IEEE Intelligent Vehices Symposium (IV), pp.1094-1099, 2015.

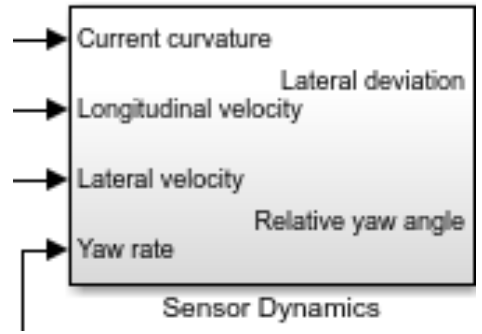


SYSTEM COMPONENTS

Model contains **four** main components:

- **Vehicle dynamics:** applies **3rd-order bicycle model** of **lateral vehicle dynamics** and approximates longitudinal dynamics using a time constant τ

- **Sensor dynamics:** approximates a sensor such as a camera to calculate the **lateral deviation** and **relative yaw angle**



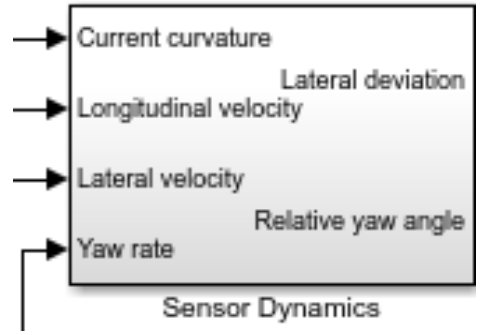
SYSTEM COMPONENTS

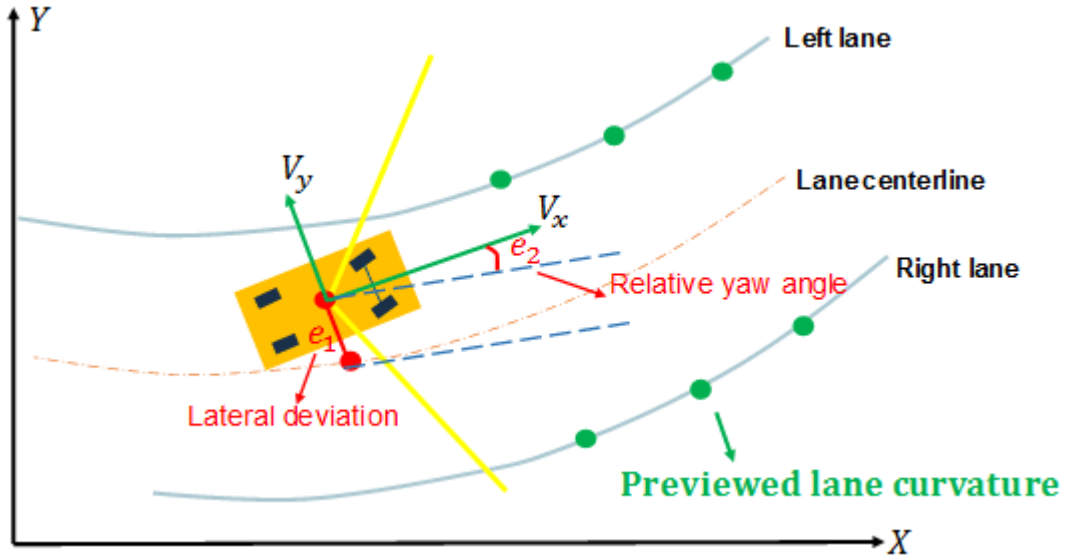
Model contains **four** main components:

- **Vehicle dynamics:** applies **3rd-order bicycle model** of **lateral vehicle dynamics** and approximates longitudinal dynamics using a time constant τ

- **Sensor dynamics:** approximates a sensor such as a camera to calculate the **lateral deviation** and **relative yaw angle**

$$\dot{e}_1 = \text{longitudinal velocity} * e_2 + \text{lateral velocity}$$
$$\dot{e}_2 = \text{yaw rate - longitudinal velocity} * \text{curvature}$$





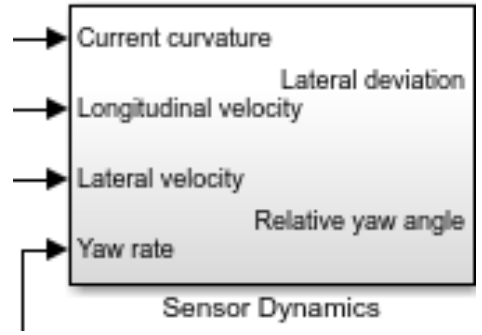
SYSTEM COMPONENTS

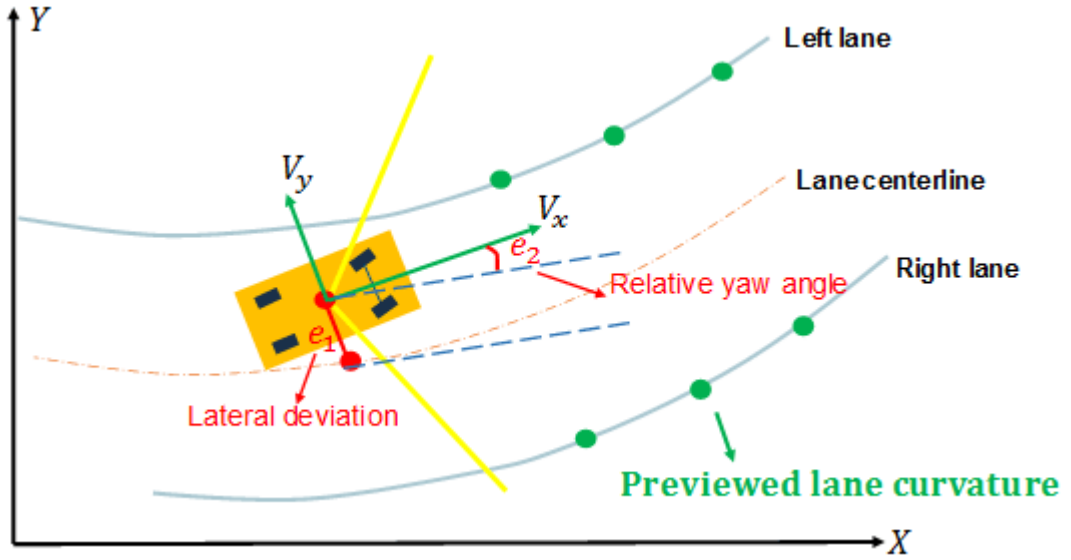
Model contains **four** main components:

- **Vehicle dynamics:** applies **3rd-order bicycle model** of **lateral vehicle dynamics** and approximates longitudinal dynamics using a time constant τ

- **Sensor dynamics:** approximates a sensor such as a camera to calculate the **lateral deviation** and **relative yaw angle**

$$\dot{e}_1 = V_x * e_2 + V_y$$
$$\dot{e}_2 = \dot{\psi} - u_3$$





embotech

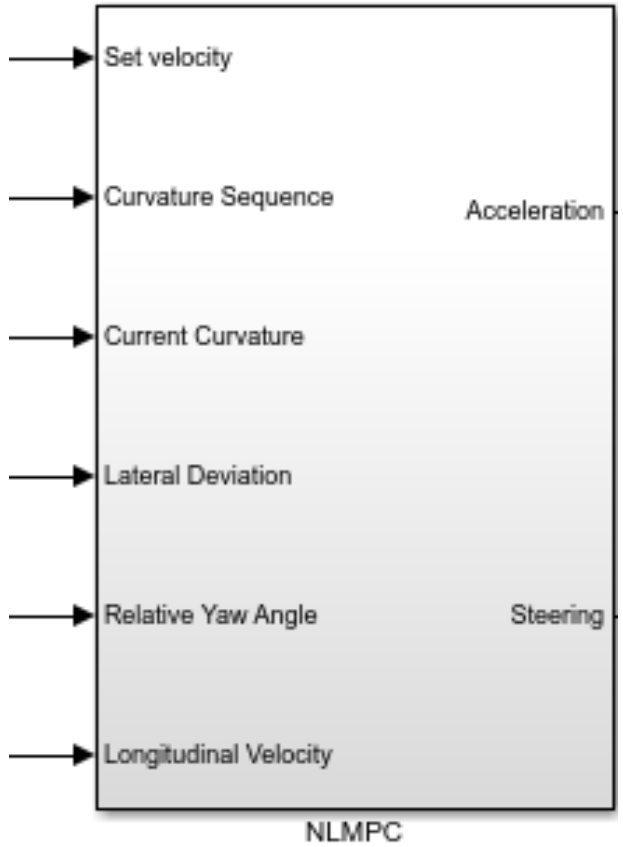
SYSTEM COMPONENTS

Model contains **four** main components:

- **Vehicle dynamics:** applies **3rd-order bicycle model** of **lateral vehicle dynamics** and approximates longitudinal dynamics using a time constant τ

- **Sensor dynamics:** approximates a sensor such as a camera to calculate the **lateral deviation** and **relative yaw angle**

- **Lane-following controller:** simulates **nonlinear MPC**



SYSTEM COMPONENTS

Model contains **four** main components:

- **Vehicle dynamics:** applies **3rd-order bicycle model** of **lateral vehicle dynamics** and approximates longitudinal dynamics using a time constant τ

- **Sensor dynamics:** approximates a sensor such as a camera to calculate the **lateral deviation** and **relative yaw angle**

- **Lane-following controller:** simulates **nonlinear MPC**
  - Comprises **seven states**, **three outputs**, and **two inputs**

```
function y = LaneFollowingOutputFcn(x,u)
    % Output
    y = [x(3); ...
         x(5); ...
         x(6)+x(7)];
end
```

```
% States x = [  lateral velocity (Vy)
%               yaw rate (psi_dot)
%               longitudinal velocity (Vx)
%               longitudinal acceleration (Vx_dot)
%               lateral deviation (e1)
%               relative yaw angle (e2)
%               output disturbance of relative yaw angle (xOD)];
%
% Inputs u = [  acceleration
%               steering angle
%               road curvature * Vx (measured disturbance)
%               white noise (unmeasured disturbance)];
```

embotech

SYSTEM COMPONENTS

Model contains **four** main components:

- **Vehicle dynamics:** applies **3rd-order bicycle model** of **lateral vehicle dynamics** and approximates longitudinal dynamics using a time constant τ

- **Sensor dynamics:** approximates a sensor such as a camera to calculate the **lateral deviation** and **relative yaw angle**

- **Lane-following controller:** simulates **nonlinear MPC**
  - Comprises **seven states**, **three outputs**, and **two inputs**
    - Model has two **MV** signals: **acceleration** and **steering**

```
% States x = [  lateral velocity (Vy)
%              yaw rate (psi_dot)
%              longitudinal velocity (Vx)
%              longitudinal acceleration (Vx_dot)
%              lateral deviation (e1)
%              relative yaw angle (e2)
%              output disturbance of relative yaw angle (xOD)];
%
% Inputs u = [  acceleration
%              steering angle
%              road curvature * Vx (measured disturbance)
%              white noise (unmeasured disturbance)];
```

embotech

SYSTEM COMPONENTS

Model contains **four** main components:

- **Vehicle dynamics:** applies **3rd-order bicycle model** of **lateral vehicle dynamics** and approximates longitudinal dynamics using a time constant τ

- **Sensor dynamics:** approximates a sensor such as a camera to calculate the **lateral deviation** and **relative yaw angle**

- **Lane-following controller:** simulates **nonlinear MPC**
  - Comprises **seven states**, **three outputs**, and **two inputs**
    - Model has two **MV** signals: **acceleration** and **steering**
    - **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
    - **Unmeasured disturbance** modeled by white noise

```
% States x = [  lateral velocity (Vy)
%              yaw rate (psi_dot)
%              longitudinal velocity (Vx)
%              longitudinal acceleration (Vx_dot)
%              lateral deviation (e1)
%              relative yaw angle (e2)
%              output disturbance of relative yaw angle (xOD)];
%
% Inputs u = [  acceleration
%              steering angle
%              road curvature * Vx (measured disturbance)
%              white noise (unmeasured disturbance)];
```

embotech

SYSTEM COMPONENTS

```
% States x = [  lateral velocity (Vy)
%              yaw rate (psi_dot)
%              longitudinal velocity (Vx)
%              longitudinal acceleration (Vx_dot)
%              lateral deviation (e1)
%              relative yaw angle (e2)
%              output disturbance of relative yaw angle (xOD)];
%
% Inputs u = [  acceleration
%              steering angle
%              road curvature * Vx (measured disturbance)
%              white noise (unmeasured disturbance)];
```

- **Lane-following controller:** simulates **nonlinear MPC**
  - Comprises **seven states**, **three outputs**, and **two inputs**
    - Model has two **MV** signals: **acceleration** and **steering**
    - **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
    - **Unmeasured disturbance** modeled by white noise

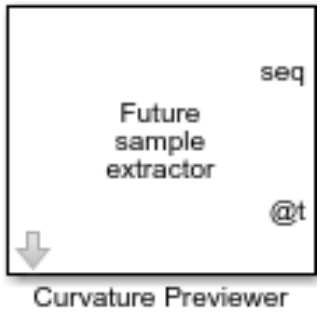
```
function dxdt = LaneFollowingStateFcn(x,u)
  % Vehicle Parameters
  m = 1575;   % Mass of car
  Iz = 2875;  % Moment of inertia about Z axis
  lf = 1.2;   % Distance between Center of Gravity and Front axle
  lr = 1.6;   % Distance between Center of Gravity and Rear axle
  Cf = 19000; % Cornering stiffness of the front tires (N/rad)
  Cr = 33000; % Cornering stiffness of the rear tires (N/rad).
  tau = 0.2;  % Time constant

  % State Equations
  a1 = -(2*Cf+2*Cr)/m/x(3);
  a2 = -(2*Cf*lf-2*Cr*lr)/m/x(3) - x(3);
  a3 = -(2*Cf*lf-2*Cr*lr)/Iz/x(3);
  a4 = -(2*Cf*lf^2+2*Cr*lr^2)/Iz/x(3);
  b1 = 2*Cf/m;
  b2 = 2*Cf*lf/Iz;
  dxdt = [a1*x(1) + a2*x(2) + b1*u(2);  % Vy
          a3*x(1) + a4*x(2) + b2*u(2);  % psi_dot
          x(2)*x(1) + x(4);             % Vx
          (1/tau)*(-x(4) + u(1));       % Vx_dot
          x(1) + x(3)*x(6);             % e1
          x(2) - u(3);                  % e2
          u(4)];                        % xOD
end
```

embotech

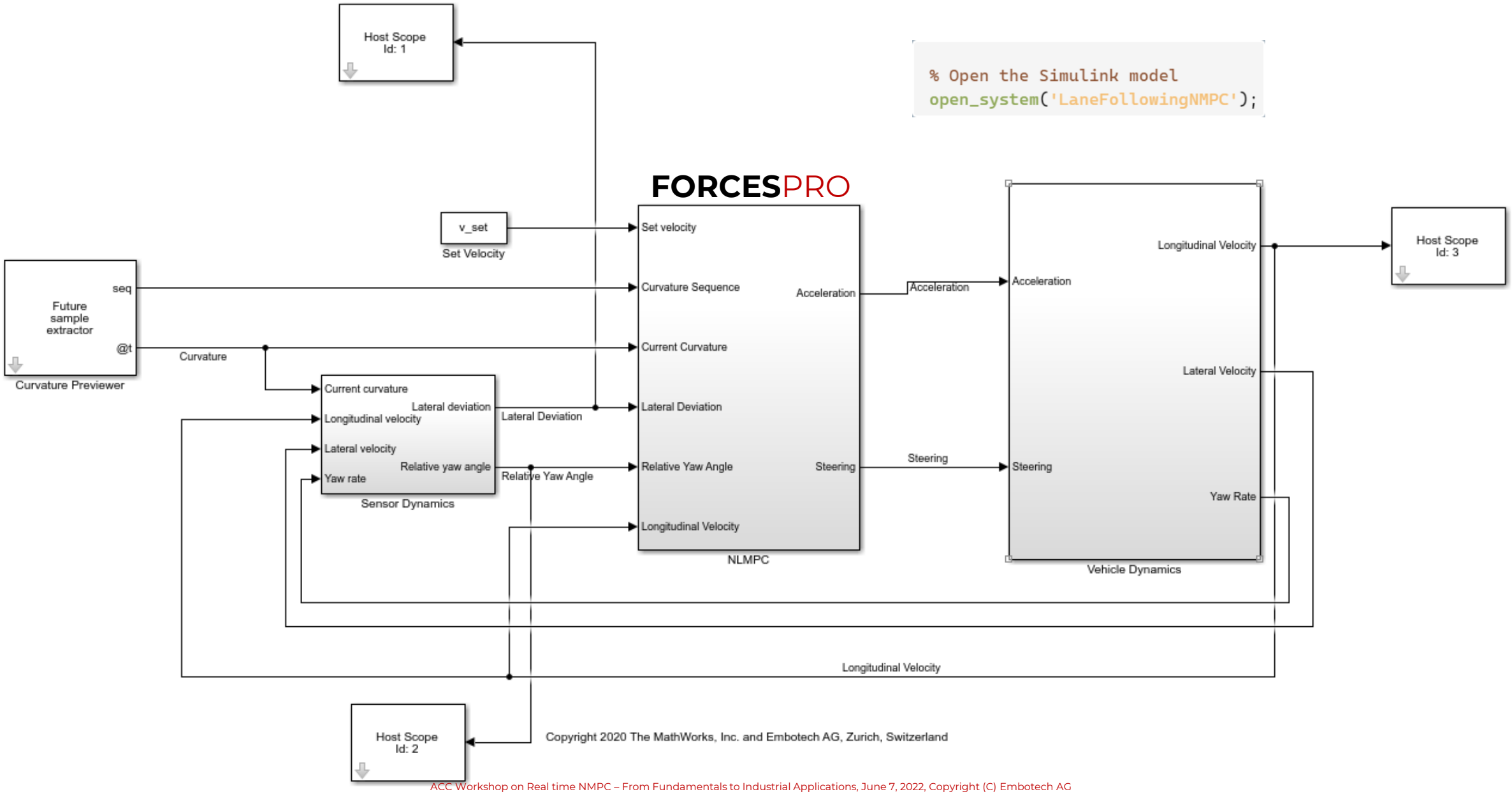
SYSTEM COMPONENTS

Model contains **four** main components:

- **Vehicle dynamics:** applies **3rd-order bicycle model** of **lateral vehicle dynamics** and approximates longitudinal dynamics using a time constant τ
- **Sensor dynamics:** approximates a sensor such as a camera to calculate the **lateral deviation** and **relative yaw angle**
- **Lane-following controller:** simulates **nonlinear MPC**
- **Curvature previewer:** detects the curvature at the current time step and the **curvature sequence** over the prediction horizon of the MPC controller



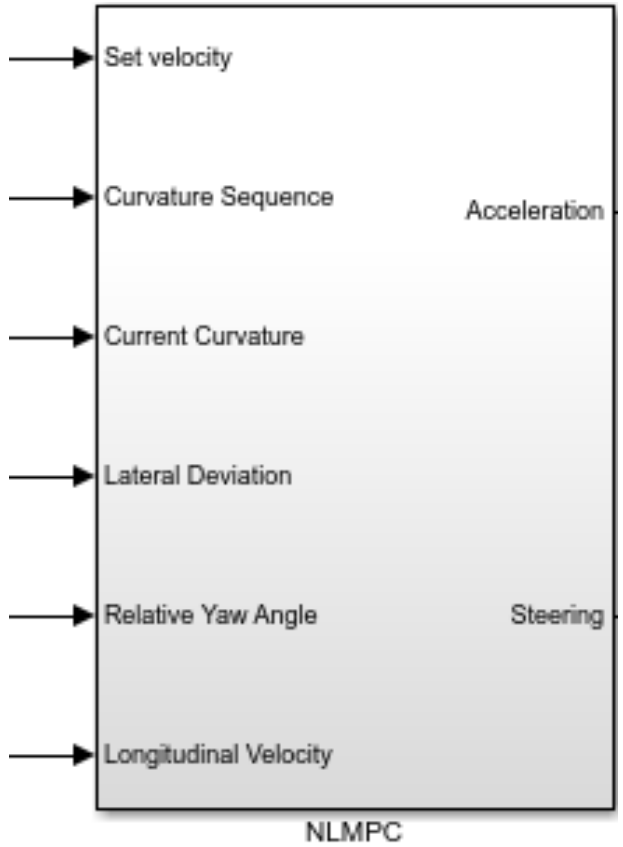
SYSTEM OVERVIEW



FORCESPRO CONTROLLER SETUP

Lane-following **controller specification**:

- Comprises **seven states**, **three outputs**, and **two inputs**
- Model has two **MV** signals: **acceleration** and **steering**
- **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
- **Unmeasured disturbance** modeled by white noise



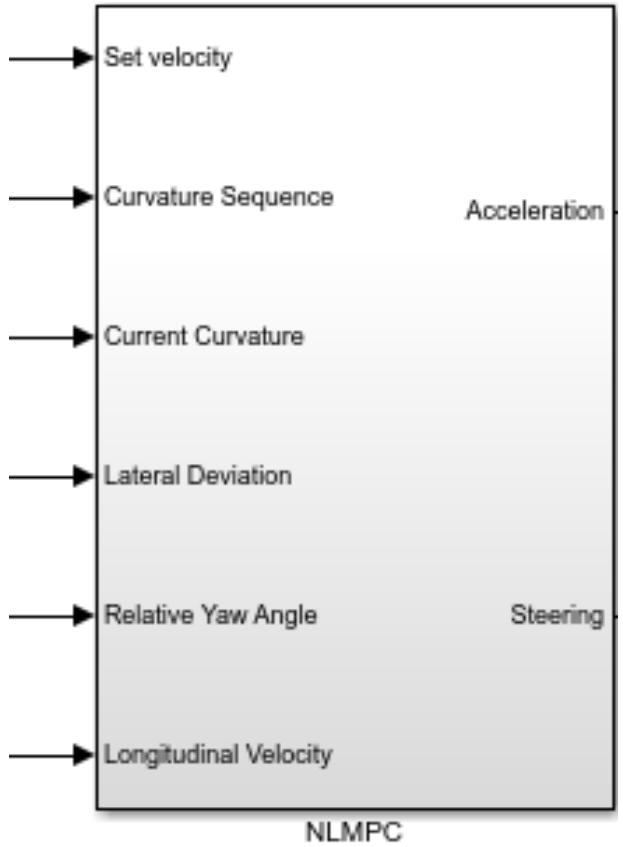
embotech

FORCESPRO CONTROLLER SETUP

Lane-following **controller specification**:

- Comprises **seven states**, **three outputs**, and **two inputs**
- Model has two **MV** signals: **acceleration** and **steering**
- **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
- **Unmeasured disturbance** modeled by white noise

```
% Create nonlinear MPC controller
nlobj = nlmpc(7,3,'MV',[1 2],'MD',3,'UD',4);

% Specify sample time, prediction and control horizon
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 10;
nlobj.ControlHorizon = 2;

% Specify state and output functions for the plant model
nlobj.Model.StateFcn = 'LaneFollowingStateFcn';
nlobj.Model.OutputFcn = 'LaneFollowingOutputFcn';
```



FORCESPRO CONTROLLER SETUP

Lane-following **controller specification**:

- Comprises **seven states**, **three outputs**, and **two inputs**
- Model has two **MV** signals: **acceleration** and **steering**
- **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
- **Unmeasured disturbance** modeled by white noise

```
% Create nonlinear MPC controller
nlobj = nlmpc(7,3,'MV',[1 2],'MD',3,'UD',4);

% Specify sample time, prediction and control horizon
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 10;
nlobj.ControlHorizon = 2;

% Specify state and output functions for the plant model
nlobj.Model.StateFcn = 'LaneFollowingStateFcn';
nlobj.Model.OutputFcn = 'LaneFollowingOutputFcn';
```

```
% Set constraints for manipulated variables
nlobj.MV(1).Min = -3;      % Maximum acceleration 3 m/s^2
nlobj.MV(1).Max = 3;       % Minimum acceleration -3 m/s^2
nlobj.MV(2).Min = -1.13;   % Minimum steering angle -65 deg
nlobj.MV(2).Max = 1.13;    % Maximum steering angle 65 deg
```

embotech

FORCESPRO CONTROLLER SETUP

Lane-following **controller specification**:

- Comprises **seven states**, **three outputs**, and **two inputs**
- Model has two **MV** signals: **acceleration** and **steering**
- **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
- **Unmeasured disturbance** modeled by white noise

```
% Set constraints for manipulated variables
nlobj.MV(1).Min = -3;      % Maximum acceleration 3 m/s^2
nlobj.MV(1).Max = 3;       % Minimum acceleration -3 m/s^2
nlobj.MV(2).Min = -1.13;   % Minimum steering angle -65 deg
nlobj.MV(2).Max = 1.13;    % Maximum steering angle 65 deg

% Set scale factors
nlobj.OV(1).ScaleFactor = 15;   % Typical value of long. velocity
nlobj.OV(2).ScaleFactor = 0.5;  % Range for lateral deviation
nlobj.OV(3).ScaleFactor = 0.5;  % Range for relative yaw angle
nlobj.MV(1).ScaleFactor = 6;    % Range of steering angle
nlobj.MV(2).ScaleFactor = 2.26; % Range of acceleration
nlobj.MD(1).ScaleFactor = 0.2;  % Range of curvature
```

```
% Create nonlinear MPC controller
nlobj = nlmpc(7,3,'MV',[1 2],'MD',3,'UD',4);

% Specify sample time, prediction and control horizon
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 10;
nlobj.ControlHorizon = 2;

% Specify state and output functions for the plant model
nlobj.Model.StateFcn = 'LaneFollowingStateFcn';
nlobj.Model.OutputFcn = 'LaneFollowingOutputFcn';
```

embotech

FORCESPRO CONTROLLER SETUP

Lane-following **controller specification**:

- Comprises **seven states**, **three outputs**, and **two inputs**
- Model has two **MV** signals: **acceleration** and **steering**
- **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
- **Unmeasured disturbance** modeled by white noise

```
% Create nonlinear MPC controller
nlobj = nlmpc(7,3,'MV',[1 2],'MD',3,'UD',4);

% Specify sample time, prediction and control horizon
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 10;
nlobj.ControlHorizon = 2;

% Specify state and output functions for the plant model
nlobj.Model.StateFcn = 'LaneFollowingStateFcn';
nlobj.Model.OutputFcn = 'LaneFollowingOutputFcn';
```

```
% Set constraints for manipulated variables
nlobj.MV(1).Min = -3;      % Maximum acceleration 3 m/s^2
nlobj.MV(1).Max = 3;       % Minimum acceleration -3 m/s^2
nlobj.MV(2).Min = -1.13;   % Minimum steering angle -65 deg
nlobj.MV(2).Max = 1.13;    % Maximum steering angle 65 deg

% Set scale factors
nlobj.OV(1).ScaleFactor = 15;   % Typical value of long. velocity
nlobj.OV(2).ScaleFactor = 0.5;  % Range for lateral deviation
nlobj.OV(3).ScaleFactor = 0.5;  % Range for relative yaw angle
nlobj.MV(1).ScaleFactor = 6;    % Range of steering angle
nlobj.MV(2).ScaleFactor = 2.26; % Range of acceleration
nlobj.MD(1).ScaleFactor = 0.2;  % Range of curvature

% Specify the weights in the standard MPC cost function
nlobj.Weights.OutputVariables = [1 1 0];
```

embotech

FORCESPRO CONTROLLER SETUP

Lane-following **controller specification**:

- Comprises **seven states**, **three outputs**, and **two inputs**
- Model has two **MV** signals: **acceleration** and **steering**
- **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
- **Unmeasured disturbance** modeled by white noise

```
% Create nonlinear MPC controller
nlobj = nlmpc(7,3,'MV',[1 2],'MD',3,'UD',4);

% Specify sample time, prediction and control horizon
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 10;
nlobj.ControlHorizon = 2;

% Specify state and output functions for the plant model
nlobj.Model.StateFcn = 'LaneFollowingStateFcn';
nlobj.Model.OutputFcn = 'LaneFollowingOutputFcn';
```

```
% Set constraints for manipulated variables
nlobj.MV(1).Min = -3;      % Maximum acceleration 3 m/s^2
nlobj.MV(1).Max = 3;       % Minimum acceleration -3 m/s^2
nlobj.MV(2).Min = -1.13;   % Minimum steering angle -65 deg
nlobj.MV(2).Max = 1.13;    % Maximum steering angle 65 deg

% Set scale factors
nlobj.OV(1).ScaleFactor = 15;   % Typical value of long. velocity
nlobj.OV(2).ScaleFactor = 0.5;  % Range for lateral deviation
nlobj.OV(3).ScaleFactor = 0.5;  % Range for relative yaw angle
nlobj.MV(1).ScaleFactor = 6;    % Range of steering angle
nlobj.MV(2).ScaleFactor = 2.26; % Range of acceleration
nlobj.MD(1).ScaleFactor = 0.2;  % Range of curvature

% Specify the weights in the standard MPC cost function
nlobj.Weights.OutputVariables = [1 1 0];

% Penalize acceleration change more for smooth driving experience
nlobj.Weights.ManipulatedVariablesRate = [0.3 0.1];
```

embotech

FORCESPRO CONTROLLER SETUP

Lane-following **controller specification**:

- Comprises **seven states**, **three outputs**, and **two inputs**
- Model has two **MV** signals: **acceleration** and **steering**
- **Measured disturbance** modeled as a product of the road curvature and the longitudinal velocity
- **Unmeasured disturbance** modeled by white noise

```
% Create nonlinear MPC controller
nlobj = nlmpc(7,3,'MV',[1 2],'MD',3,'UD',4);

% Specify sample time, prediction and control horizon
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 10;
nlobj.ControlHorizon = 2;

% Specify state and output functions for the plant model
nlobj.Model.StateFcn = 'LaneFollowingStateFcn';
nlobj.Model.OutputFcn = 'LaneFollowingOutputFcn';
```

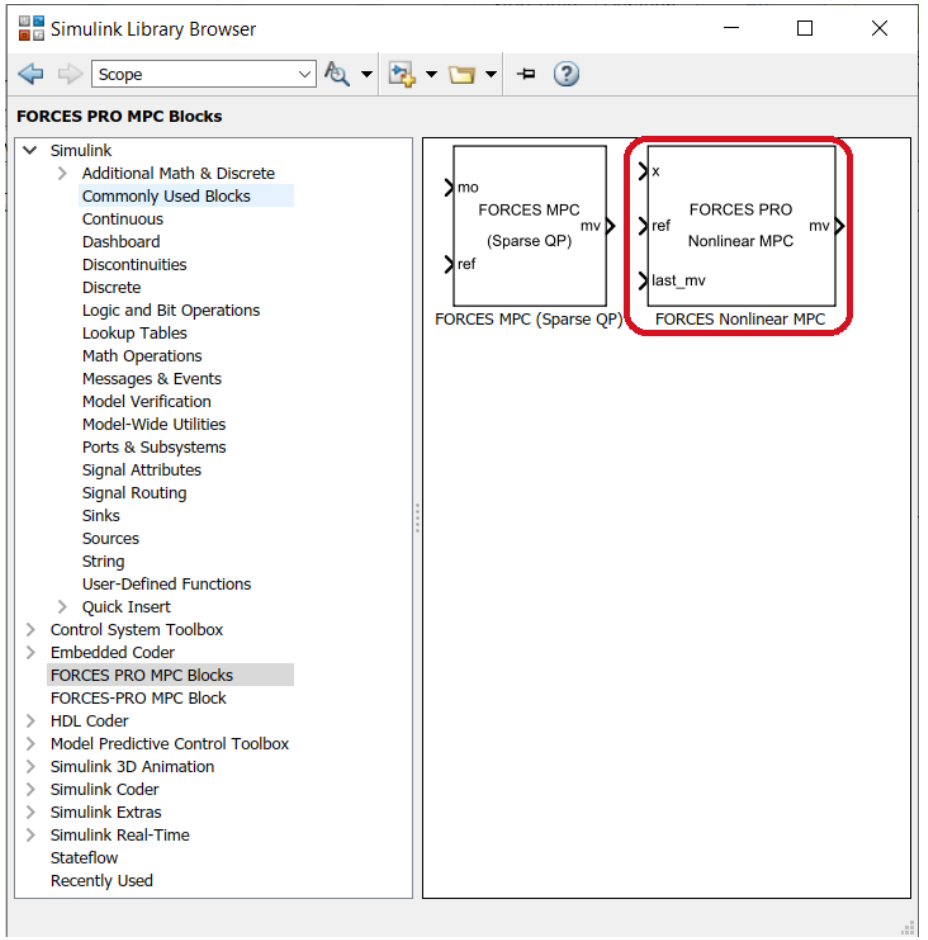
```
% Set constraints for manipulated variables
nlobj.MV(1).Min = -3;      % Maximum acceleration 3 m/s^2
nlobj.MV(1).Max = 3;       % Minimum acceleration -3 m/s^2
nlobj.MV(2).Min = -1.13;   % Minimum steering angle -65 deg
nlobj.MV(2).Max = 1.13;    % Maximum steering angle 65 deg

% Set scale factors
nlobj.OV(1).ScaleFactor = 15;   % Typical value of long. velocity
nlobj.OV(2).ScaleFactor = 0.5;  % Range for lateral deviation
nlobj.OV(3).ScaleFactor = 0.5;  % Range for relative yaw angle
nlobj.MV(1).ScaleFactor = 6;    % Range of steering angle
nlobj.MV(2).ScaleFactor = 2.26; % Range of acceleration
nlobj.MD(1).ScaleFactor = 0.2;  % Range of curvature

% Specify the weights in the standard MPC cost function
nlobj.Weights.OutputVariables = [1 1 0];

% Penalize acceleration change more for smooth driving experience
nlobj.Weights.ManipulatedVariablesRate = [0.3 0.1];

% Generate FORCESPRO nonlinear solver
options = nlmpcToForcesOptions();
options.SolverName = 'LaneFollowSolver';
options.SolverType = 'InteriorPoint'; % 'InteriorPoint' or 'SQP'
options.x0 = [0.1 0.5 25 0.1 0.1 0.001 0.5];
options.mv0 = [0.125 0.4];
[coredata, onlinedata] = nlmpcToForces(nlobj,options);
```

embotech

SIMULINK INTEGRATION

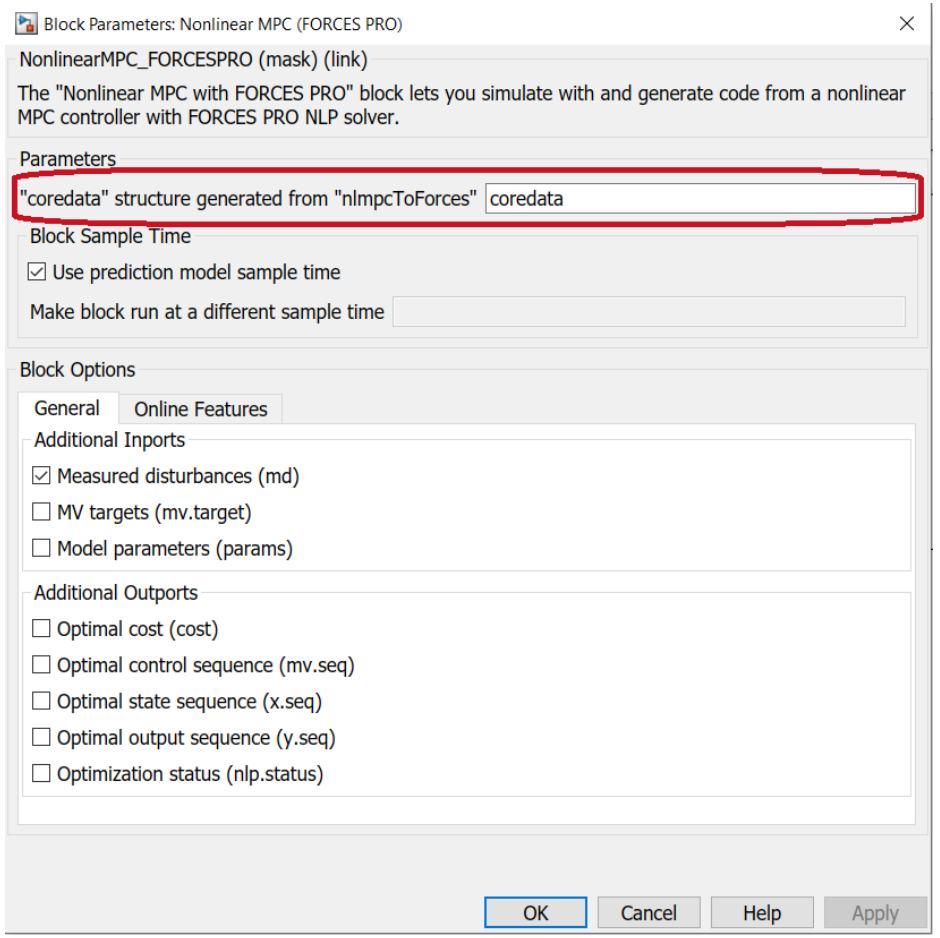
- Generated **FORCES**PRO **NLMPC Simulink block** can now be used seamlessly with the model

- Available in the **Model Predictive Control Toolbox section** of the Simulink Library Browser



SIMULINK INTEGRATION

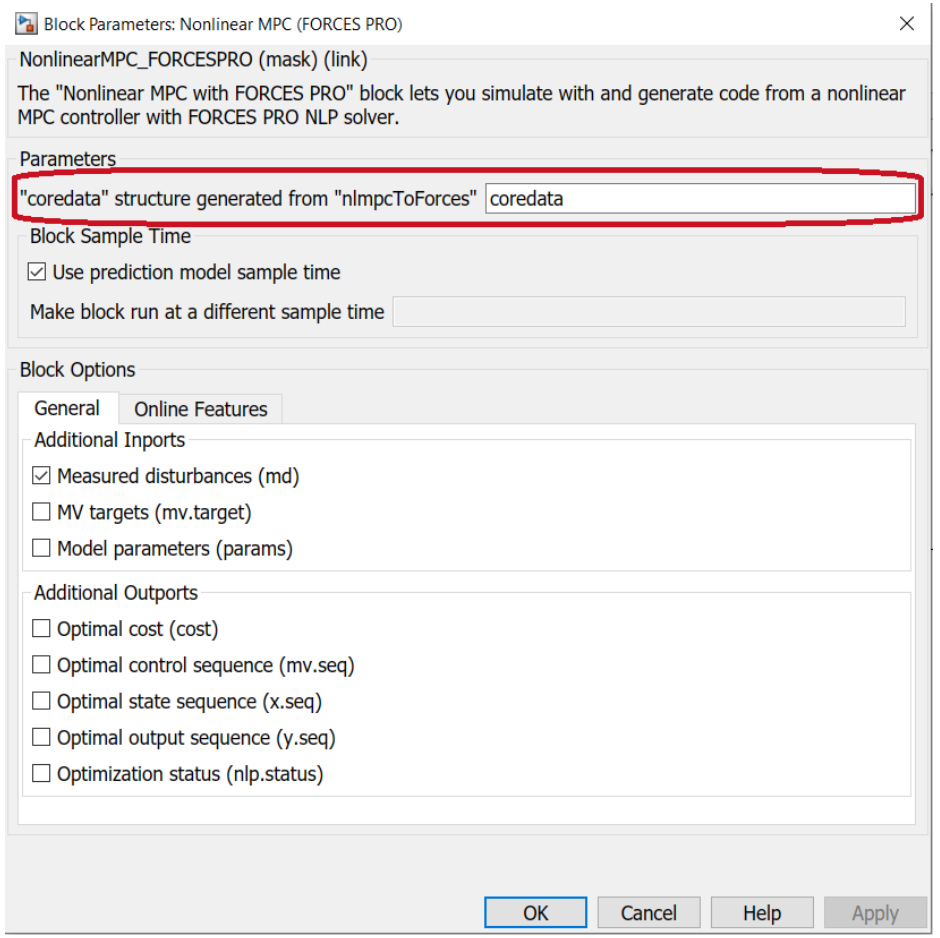
- Generated **FORCES**PRO **NLMPC Simulink block** can now be used seamlessly with the model
- Available in the **Model Predictive Control Toolbox section** of the Simulink Library Browser
- In order to run the nonlinear interior-point solver, the ***coredata*** structure returned by *nlmpcToForces* must be provided in the block mask



SIMULINK INTEGRATION

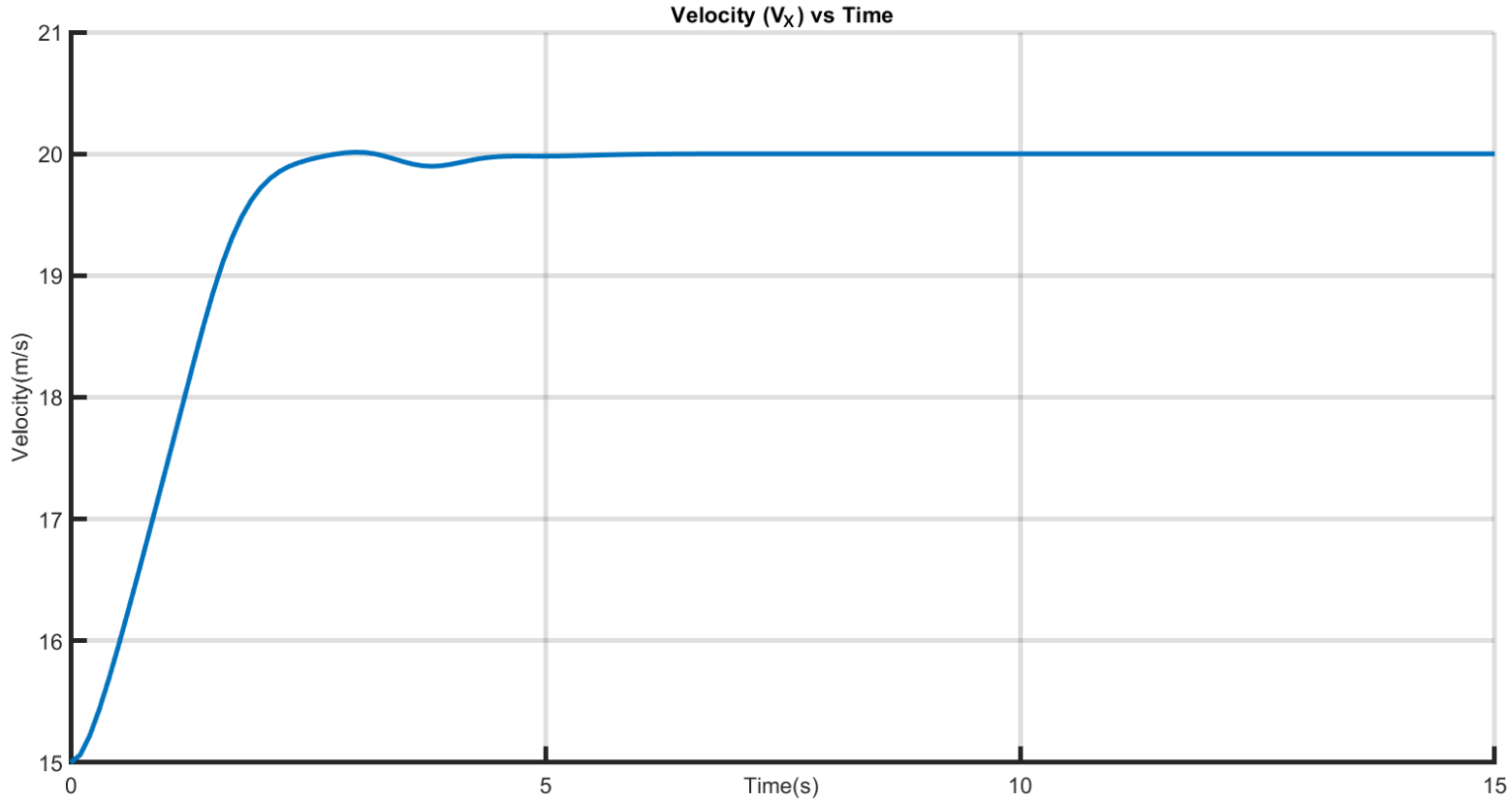
- Generated **FORCES**PRO **NLMPC Simulink block** can now be used seamlessly with the model
- Available in the **Model Predictive Control Toolbox section** of the Simulink Library Browser
- In order to run the nonlinear interior-point solver, the ***coredata*** structure returned by *nlmpcToForces* must be provided in the block mask
- **Simulate** the model and obtain results

```
% Simulate the model using nonlinear MPC
sim('LaneFollowingNMPC')
```



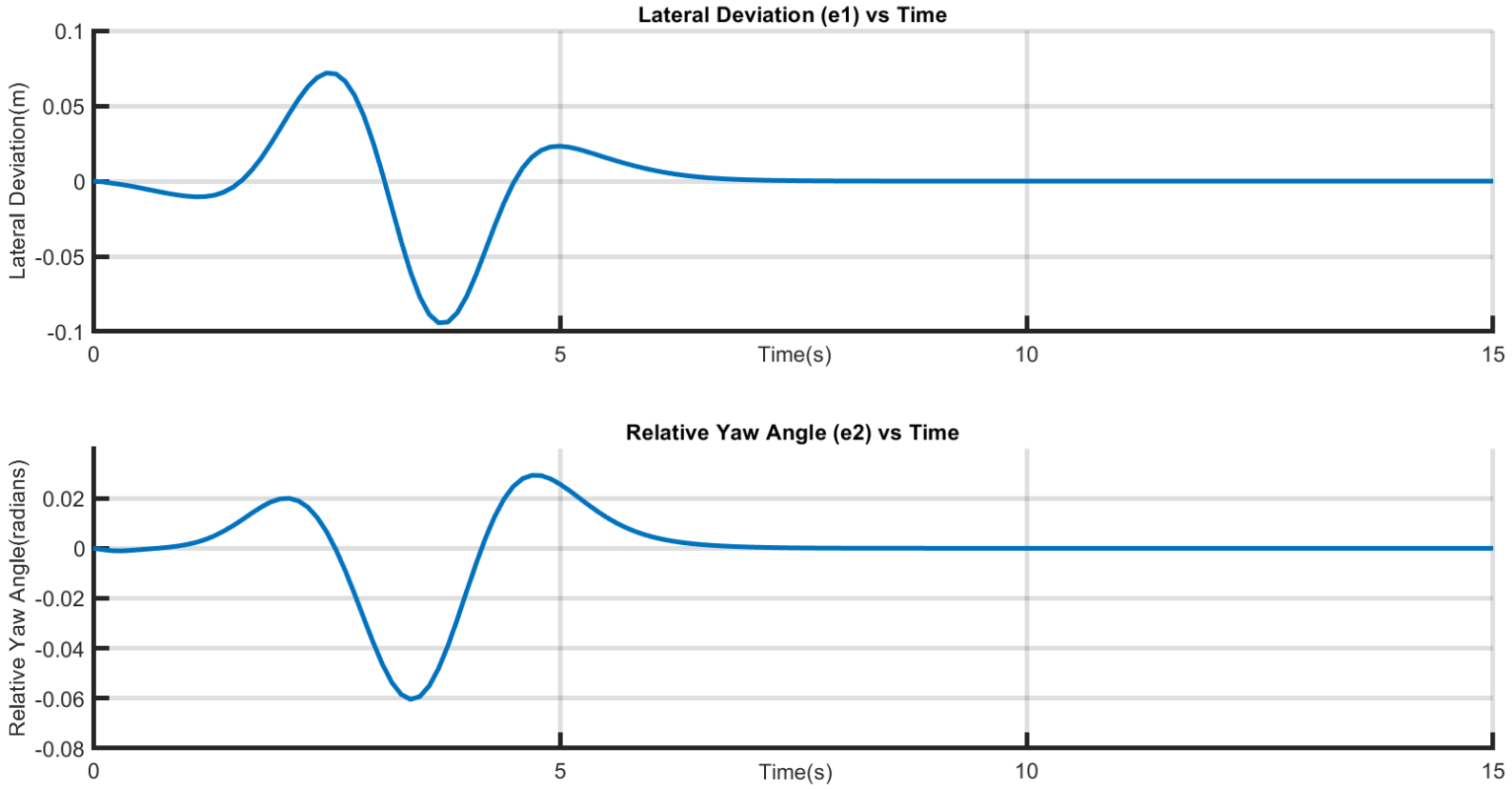
CONTROL PERFORMANCE

Longitudinal vehicle velocity is kept close to a **driver-set velocity**



CONTROL PERFORMANCE

Lateral deviation e_1 and **relative yaw angle e_2** are kept **small**



DEPLOYMENT TO SPEEDGOAT X86

- Generate **FORCES**PRO **nonlinear solver** for the **targeted platform**

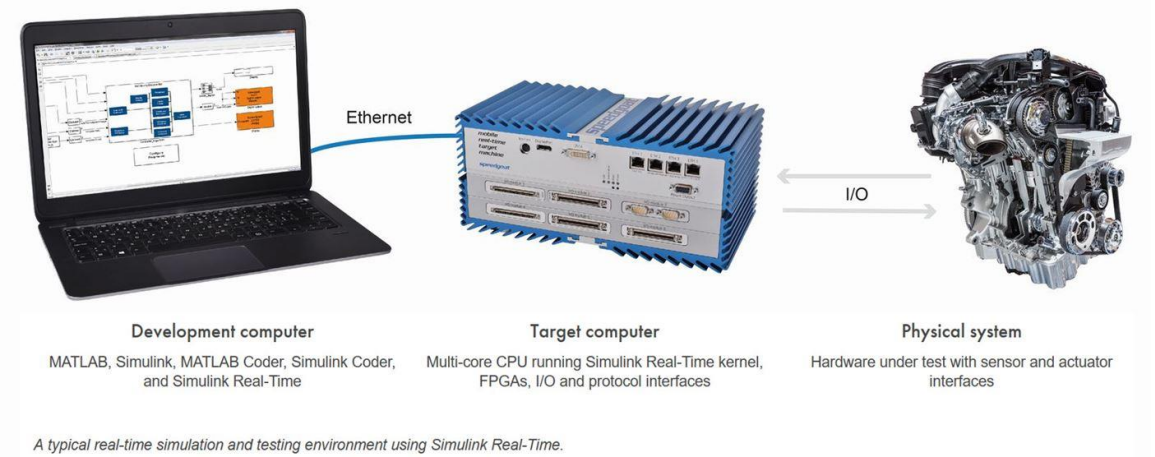
```
% Generate FORCESPRO nonlinear solver to run on Sppedgoat x86
options = nlmpcToForcesOptions();
options.SolverName = 'LaneFollowSolver';
options.SolverType = 'InteriorPoint'; % 'InteriorPoint' or 'SQP'
options.ForcesTargetPlatform = 'Speedgoat-x86';
options.x0 = [0.1 0.5 25 0.1 0.1 0.001 0.5];
options.mv0 = [0.125 0.4];
```

A typical real-time simulation and testing environment using Simulink Real-Time.

DEPLOYMENT TO SPEEDGOAT X86

- Generate **FORCES**PRO **nonlinear solver** for the **targeted platform** (via *options.ForcesTargetPlatform*)

```
% Generate FORCESPRO nonlinear solver to run on Sppedgoat x86
options = nlmpcToForcesOptions();
options.SolverName = 'LaneFollowSolver';
options.SolverType = 'InteriorPoint'; % 'InteriorPoint' or 'SQP'
options.ForcesTargetPlatform = 'Speedgoat-x86';
options.x0 = [0.1 0.5 25 0.1 0.1 0.001 0.5];
options.mv0 = [0.125 0.4];
```

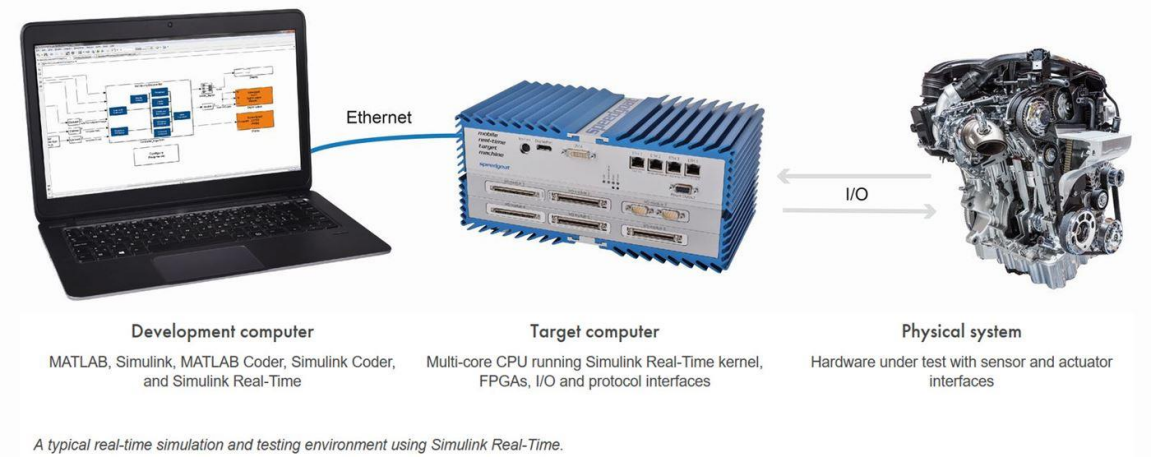


A typical real-time simulation and testing environment using Simulink Real-Time.

DEPLOYMENT TO SPEEDGOAT X86

- Generate **FORCES**PRO **nonlinear solver** for the **targeted platform** (via *options.ForcesTargetPlatform*)

```
% Generate FORCESPRO nonlinear solver to run on Sppedgoat x86
options = nlmpcToForcesOptions();
options.SolverName = 'LaneFollowSolver';
options.SolverType = 'InteriorPoint'; % 'InteriorPoint' or 'SQP'
options.ForcesTargetPlatform = 'Speedgoat-x86';
options.x0 = [0.1 0.5 25 0.1 0.1 0.001 0.5];
options.mv0 = [0.125 0.4];

% Start code generation for Speedgoat x86
mdl = 'LaneFollowingNMPC_Speedgoat_x86';
open_system(mdl);   % Open Simulink model
load_system(mdl);   % Load Simulink model
rtwbuild(mdl);      % Start code generation

% Deploy application from the start
tg = slrt;
if(~strcmpi(tg.Application, 'loader'))
    tg.unload();
end
tg.load(mdl);

% Execute application
tg.start();
```



A typical real-time simulation and testing environment using Simulink Real-Time.

DEPLOYMENT TO SPEEDGOAT X86

- Generate **FORCES**PRO **nonlinear solver** for the **targeted platform** (via *options.ForcesTargetPlatform*)

```
% Generate FORCESPRO nonlinear solver to run on Sppedgoat x86
options = nlmpcToForcesOptions();
options.SolverName = 'LaneFollowSolver';
options.SolverType = 'InteriorPoint'; % 'InteriorPoint' or 'SQP'
options.ForcesTargetPlatform = 'Speedgoat-x86';
options.x0 = [0.1 0.5 25 0.1 0.1 0.001 0.5];
options.mv0 = [0.125 0.4];

% Start code generation for Speedgoat x86
mdl = 'LaneFollowingNMPC_Speedgoat_x86';
open_system(mdl);   % Open Simulink model
load_system(mdl);   % Load Simulink model
rtwbuild(mdl);      % Start code generation

% Deploy application from the start
tg = slrt;
if(~strcmpi(tg.Application, 'loader'))
    tg.unload();
end
tg.load(mdl);

% Execute application
tg.start();
```

More details on **YouTube**

"Embotech: Model based Rapid Prototyping of NMPC"

embotech

embotech.com