FORCESPRO: LOW-LEVEL INTERFACE

Uros Markovic
Optimization Specialist

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

embotech

PROBLEM CLASS OVERVIEW

Low-level interface supports the class of **convex multistage quadratically constrained programs (QCQPs)**

- The most lightweight interface
- Provides full flexibility when designing custom solvers and MPC controllers based on non-standard formulations
- Supports all problem data to be parametric (unknown at code generation time)
- Matrices H_i and $Q_{i,k}$ should all be positive semidefinite

minimize	$\sum_{i=1}^{N} \frac{1}{2} z_i^\top H_i z_i + f_i^\top z_i$	(separable objective)
subject to	$D_1 z_1 = c_1$	(initial equality)
	$C_{i-1} z_{i-1} + D_i z_i = c_i, \;\; i = 2, \ldots, N$	(inter-stage equality)
	$\underline{z}_i \leq z_i$	(lower bound)
	$z_i \leq \bar{z}_i$	(upper bound)
	$A_i z_i \leq b_i$	(polytopic inequalities)
	$z_i^\top Q_{i,k} z_i + L_{i,k}^\top z_i \leq r_{i,k}$	(quadratic inequalities)

embotech

GENERAL PROBLEM FORMULATION

```
stages = MultistageProblem(N);

% Dimensions
stages(i).dims.n = ...; % length of stage variable zi
stages(i).dims.r = ...; % number of equality constraints
stages(i).dims.l = ...; % number of lower bounds
stages(i).dims.u = ...; % number of upper bounds
stages(i).dims.p = ...; % number of polytopic constraints
stages(i).dims.q = ...; % number of quadratic constraints
```

```
stages = forcespro.MultistagePoblem(N) # 0-indexed

# Dimensions
stages.dims[i]['n'] = ... # length of stage variable zi
stages.dims[i]['r'] = ... # number of equality constraints
stages.dims[i]['l'] = ... # number of lower bounds
stages.dims[i]['u'] = ... # number of upper bounds
stages.dims[i]['p'] = ... # number of polytopic constraints
stages.dims[i]['q'] = ... # number of quadratic constraints
```

VS

embotech

GENERAL PROBLEM FORMULATION

```
stages = MultistageProblem(N);

% Dimensions
stages(i).dims.n = ...; % length of stage variable zi
stages(i).dims.r = ...; % number of equality constraints
stages(i).dims.l = ...; % number of lower bounds
stages(i).dims.u = ...; % number of upper bounds
stages(i).dims.p = ...; % number of polytopic constraints
stages(i).dims.q = ...; % number of quadratic constraints

% Cost function
stages(i).cost.H = ...; % Hessian
stages(i).cost.f = ...; % linear term
```

VS

```
stages = forcespro.MultistagePoblem(N) # 0-indexed

# Dimensions
stages.dims[i]['n'] = ... # length of stage variable zi
stages.dims[i]['r'] = ... # number of equality constraints
stages.dims[i]['l'] = ... # number of lower bounds
stages.dims[i]['u'] = ... # number of upper bounds
stages.dims[i]['p'] = ... # number of polytopic constraints
stages.dims[i]['q'] = ... # number of quadratic constraints

# Cost function
stages.cost[i]['H'] = ... # Hessian
stages.cost[i]['f'] = ... # linear term
```

GENERAL PROBLEM FORMULATION

```
stages = MultistageProblem(N);

% Dimensions
stages(i).dims.n = ...; % length of stage variable zi
stages(i).dims.r = ...; % number of equality constraints
stages(i).dims.l = ...; % number of lower bounds
stages(i).dims.u = ...; % number of upper bounds
stages(i).dims.p = ...; % number of polytopic constraints
stages(i).dims.q = ...; % number of quadratic constraints

% Cost function
stages(i).cost.H = ...; % Hessian
stages(i).cost.f = ...; % linear term

% Equality constraints
stages(i).eq.C = ...;
stages(i).eq.c = ...;
stages(i).eq.D = ...;
```

VS

```
stages = forcespro.MultistagePoblem(N) # 0-indexed

# Dimensions
stages.dims[i]['n'] = ... # length of stage variable zi
stages.dims[i]['r'] = ... # number of equality constraints
stages.dims[i]['l'] = ... # number of lower bounds
stages.dims[i]['u'] = ... # number of upper bounds
stages.dims[i]['p'] = ... # number of polytopic constraints
stages.dims[i]['q'] = ... # number of quadratic constraints

# Cost function
stages.cost[i]['H'] = ... # Hessian
stages.cost[i]['f'] = ... # linear term

# Equality constraints
stages.eq[i]['C'] = ...
stages.eq[i]['c'] = ...
stages.eq[i]['D'] = ...
```

GENERAL PROBLEM FORMULATION

```
stages = MultistageProblem(N);

% Dimensions
stages(i).dims.n = ...; % length of stage variable zi
stages(i).dims.r = ...; % number of equality constraints
stages(i).dims.l = ...; % number of lower bounds
stages(i).dims.u = ...; % number of upper bounds
stages(i).dims.p = ...; % number of polytopic constraints
stages(i).dims.q = ...; % number of quadratic constraints

% Cost function
stages(i).cost.H = ...; % Hessian
stages(i).cost.f = ...; % linear term

% Equality constraints
stages(i).eq.C = ...;
stages(i).eq.c = ...;
stages(i).eq.D = ...;

% Lower and upper bounds
stages(i).ineq.b.lbidx = ...; % index vector for lower bounds
stages(i).ineq.b.lb = ...; % lower bounds
stages(i).ineq.b.ubidx = ...; % index vector for upper bounds
stages(i).ineq.b.ub = ...; % upper bounds
```

vs

```
stages = forcespro.MultistagePoblem(N) # 0-indexed

# Dimensions
stages.dims[i]['n'] = ... # length of stage variable zi
stages.dims[i]['r'] = ... # number of equality constraints
stages.dims[i]['l'] = ... # number of lower bounds
stages.dims[i]['u'] = ... # number of upper bounds
stages.dims[i]['p'] = ... # number of polytopic constraints
stages.dims[i]['q'] = ... # number of quadratic constraints

# Cost function
stages.cost[i]['H'] = ... # Hessian
stages.cost[i]['f'] = ... # linear term

# Equality constraints
stages.eq[i]['C'] = ...
stages.eq[i]['c'] = ...
stages.eq[i]['D'] = ...

# Lower and upper bounds
stages.ineq[ i ]['b']['lbidx'] = ... # index vector for lower bounds, 1-indexed
stages.ineq[ i ]['b']['lb'] = ... # lower bounds
stages.ineq[ i ]['b']['ubidx'] = ... # index vector for upper bounds, 1-indexed
stages.ineq[ i ]['b']['ub'] = ... # upper bounds
```

GENERAL PROBLEM FORMULATION

```
stages = MultistageProblem(N);

% Dimensions
stages(i).dims.n = ...; % length of stage variable zi
stages(i).dims.r = ...; % number of equality constraints
stages(i).dims.l = ...; % number of lower bounds
stages(i).dims.u = ...; % number of upper bounds
stages(i).dims.p = ...; % number of polytopic constraints
stages(i).dims.q = ...; % number of quadratic constraints

% Cost function
stages(i).cost.H = ...; % Hessian
stages(i).cost.f = ...; % linear term

% Equality constraints
stages(i).eq.C = ...;
stages(i).eq.c = ...;
stages(i).eq.D = ...;

% Lower and upper bounds
stages(i).ineq.b.lbidx = ...; % index vector for lower bounds
stages(i).ineq.b.lb = ...; % lower bounds
stages(i).ineq.b.ubidx = ...; % index vector for upper bounds
stages(i).ineq.b.ub = ...; % upper bounds

% Polytopic constraints
stages(i).ineq.p.A = ...; % Jacobian of linear inequality
stages(i).ineq.p.b = ...; % RHS of linear inequality
```

vs

```
stages = forcespro.MultistagePoblem(N) # 0-indexed

# Dimensions
stages.dims[i]['n'] = ... # length of stage variable zi
stages.dims[i]['r'] = ... # number of equality constraints
stages.dims[i]['l'] = ... # number of lower bounds
stages.dims[i]['u'] = ... # number of upper bounds
stages.dims[i]['p'] = ... # number of polytopic constraints
stages.dims[i]['q'] = ... # number of quadratic constraints

# Cost function
stages.cost[i]['H'] = ... # Hessian
stages.cost[i]['f'] = ... # linear term

# Equality constraints
stages.eq[i]['C'] = ...
stages.eq[i]['c'] = ...
stages.eq[i]['D'] = ...

# Lower and upper bounds
stages.ineq[ i ]['b']['lbidx'] = ... # index vector for lower bounds, 1-indexed
stages.ineq[ i ]['b']['lb'] = ... # lower bounds
stages.ineq[ i ]['b']['ubidx'] = ... # index vector for upper bounds, 1-indexed
stages.ineq[ i ]['b']['ub'] = ... # upper bounds

# Polytopic constraints
stages.ineq[i]['p']['A'] = ... # Jacobian of linear inequality
stages.ineq[i]['p']['b'] = ... # RHS of linear inequality
```

GENERAL PROBLEM FORMULATION

MATLAB

```
stages = MultistageProblem(N);

% Dimensions
stages(i).dims.n = ...; % length of stage variable zi
stages(i).dims.r = ...; % number of equality constraints
stages(i).dims.l = ...; % number of lower bounds
stages(i).dims.u = ...; % number of upper bounds
stages(i).dims.p = ...; % number of polytopic constraints
stages(i).dims.q = ...; % number of quadratic constraints

% Cost function
stages(i).cost.H = ...; % Hessian
stages(i).cost.f = ...; % linear term

% Equality constraints
stages(i).eq.C = ...;
stages(i).eq.c = ...;
stages(i).eq.D = ...;

% Lower and upper bounds
stages(i).ineq.b.lbidx = ...; % index vector for lower bounds
stages(i).ineq.b.lb = ...; % lower bounds
stages(i).ineq.b.ubidx = ...; % index vector for upper bounds
stages(i).ineq.b.ub = ...; % upper bounds

% Polytopic constraints
stages(i).ineq.p.A = ...; % Jacobian of linear inequality
stages(i).ineq.p.b = ...; % RHS of linear inequality

% Quadratic constraints
stages(i).ineq.q.idx = { idx1, idx2, ...}; % index vectors
stages(i).ineq.q.Q = { Q1, Q2, ...}; % Hessians
stages(i).ineq.q.l = { L1, L2, ...}; % linear terms
stages(i).ineq.q.r = [ r1; r2; ... ]; % RHSs
```

vs

python

```
stages = forcespro.MultistagePoblem(N) # 0-indexed

# Dimensions
stages.dims[i]['n'] = ... # length of stage variable zi
stages.dims[i]['r'] = ... # number of equality constraints
stages.dims[i]['l'] = ... # number of lower bounds
stages.dims[i]['u'] = ... # number of upper bounds
stages.dims[i]['p'] = ... # number of polytopic constraints
stages.dims[i]['q'] = ... # number of quadratic constraints

# Cost function
stages.cost[i]['H'] = ... # Hessian
stages.cost[i]['f'] = ... # linear term

# Equality constraints
stages.eq[i]['C'] = ...
stages.eq[i]['c'] = ...
stages.eq[i]['D'] = ...

# Lower and upper bounds
stages.ineq[ i ]['b']['lbidx'] = ... # index vector for lower bounds, 1-indexed
stages.ineq[ i ]['b']['lb'] = ... # lower bounds
stages.ineq[ i ]['b']['ubidx'] = ... # index vector for upper bounds, 1-indexed
stages.ineq[ i ]['b']['ub'] = ... # upper bounds

# Polytopic constraints
stages.ineq[i]['p']['A'] = ... # Jacobian of linear inequality
stages.ineq[i]['p']['b'] = ... # RHS of linear inequality

# Quadratic constraints
stages.ineq[i]['q']['idx'] = ... # index vectors, 1-indexed
stages.ineq[i]['q']['Q'] = ... # Hessians
stages.ineq[i]['q']['l'] = ... # linear terms
stages.ineq[i]['q']['r'] = ... # RHSs
```

GENERAL PROBLEM FORMULATION

```
stages = MultistageProblem(N);

% Binary constraints
stages(i).bidx = [ idx1, idx2,   ... ]; % index vector for binaries

% Declaring parameters
parameter = addParam('xinit', 1, 'eq.c')

% Declaring Solver Outputs
% Standard output as a slice of the primal solution vector
output = newOutput(name, maps2stage, idxWithinStage)
% Output as a slice of certain Lagrange multipliers
output = newOutput(name, maps2stage, idxWithinStage, maps2const)

% Dumping Models
% not implemented at the moment!
```

VS

```
stages = forcespro.MultistagePoblem(N) # 0-indexed

# Binary constraints
stages.bidx[i] = np.array([ ... ]) # index vector for binaries

# Declaring parameters
stages.addParam("xinit", [1], 'eq.c') # 1-indexed

# Declaring Solver Outputs
# Standard output as a slice of the primal solution vector
stages.newOutput(name, maps2stage, idxWithinStage)
# Output as a slice of certain Lagrange multipliers
stages.newOutput(name, maps2stage, idxWithinStage, maps2const)

# Dumping Models
save_all(stages)
```

GENERAL PROBLEM FORMULATION

```
stages = MultistageProblem(N);

% Binary constraints
stages(i).bidx = [ idx1, idx2,   ... ]; % index vector for binaries

% Declaring parameters
parameter = addParam('xinit', 1, 'eq.c')

% Declaring Solver Outputs
% Standard output as a slice of the primal solution vector
output = newOutput(name, maps2stage, idxWithinStage)
% Output as a slice of certain Lagrange multipliers
output = newOutput(name, maps2stage, idxWithinStage, maps2const)

% Dumping Models
% not implemented at the moment!

# Generating the solver
codeoptions = getOptions('solver name');
generateCode(stages, params, codeoptions, outputs);

# Calling the generated low-level solver
problem = {} % a struct of solver parameters
SOLVER_NAME(problem)
```

VS

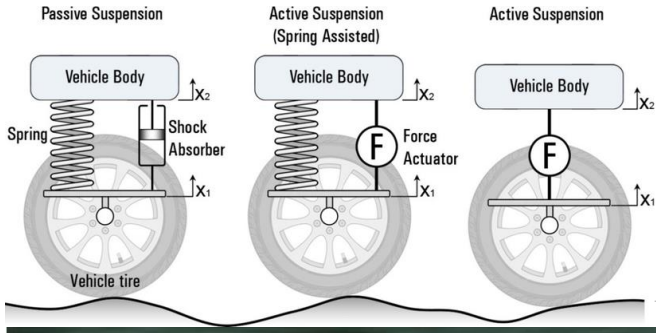
```
stages = forcespro.MultistagePoblem(N) # 0-indexed

# Binary constraints
stages.bidx[i] = np.array([ ... ]) # index vector for binaries

# Declaring parameters
stages.addParam("xinit", [1], 'eq.c') # 1-indexed

# Declaring Solver Outputs
# Standard output as a slice of the primal solution vector
stages.newOutput(name, maps2stage, idxWithinStage)
# Output as a slice of certain Lagrange multipliers
stages.newOutput(name, maps2stage, idxWithinStage, maps2const)

# Dumping Models
save_all(stages)

# Generating the solver
options = forcespro.CodeOptions('solver_name')
stages.codeoptions = options
stages.generateCode(get_userid.user_id)

# Calling the generated low-level solver
import SOLVER_NAME_py # notice the _py suffix
problem = {} # a dictionary of solver parameters
SOLVER_NAME_py.SOLVER_NAME_solve(problem)
```

Note: Don't give your solver the same name as the script you are calling it from. Doing so will **overwrite** your calling script with the solver interface!

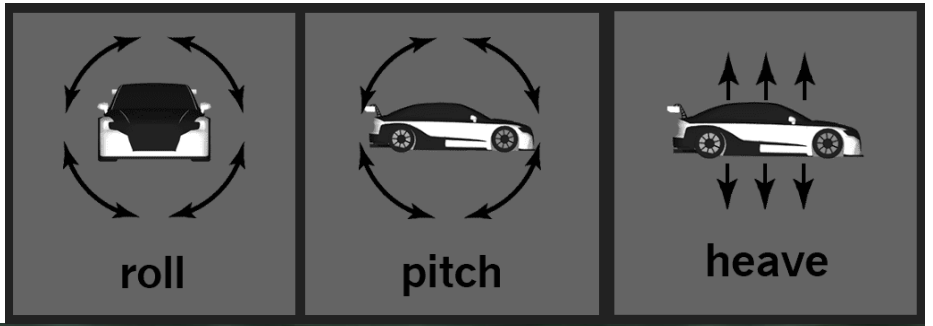
embotech

EXAMPLE: ACTIVE SUSPENSION CONTROL

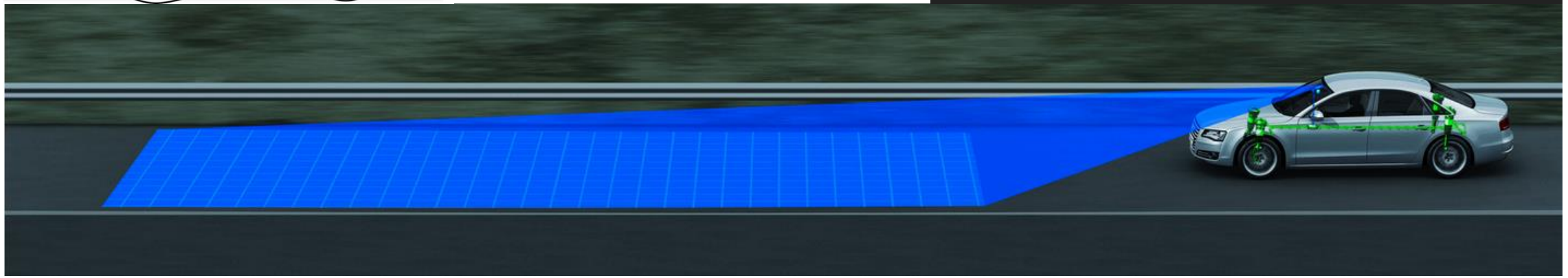
- Driving vehicle equipped with sensors that measure the **unevenness** of the road ahead
- **Preview information** used to **improve the riding comfort** by actively **controlling the suspension** of the vehicle
- **Objective**: Regulate suspension by **minimizing** the heave, pitch and roll acceleration



Elattar, Y.et al. "PDF vs PID Controller for Active Vehicle Suspension"



@SimCraft

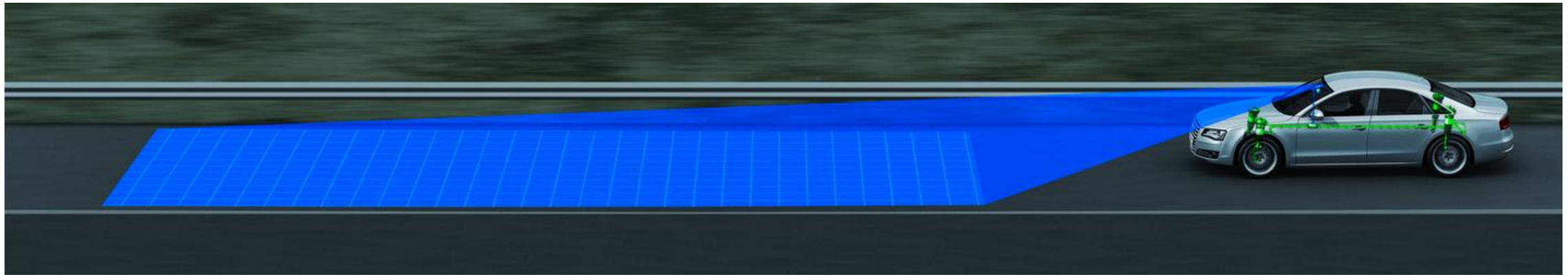


Göhrle, C.et al. "Design and Vehicle Implementation of Preview Active Suspension Controllers"

EXAMPLE: ACTIVE SUSPENSION CONTROL

- Driving vehicle equipped with sensors that measure the **unevenness** of the road ahead
- **Preview information** used to **improve the riding comfort** by actively **controlling the suspension** of the vehicle
- **Objective**: Regulate suspension by **minimizing** the heave, pitch and roll acceleration

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

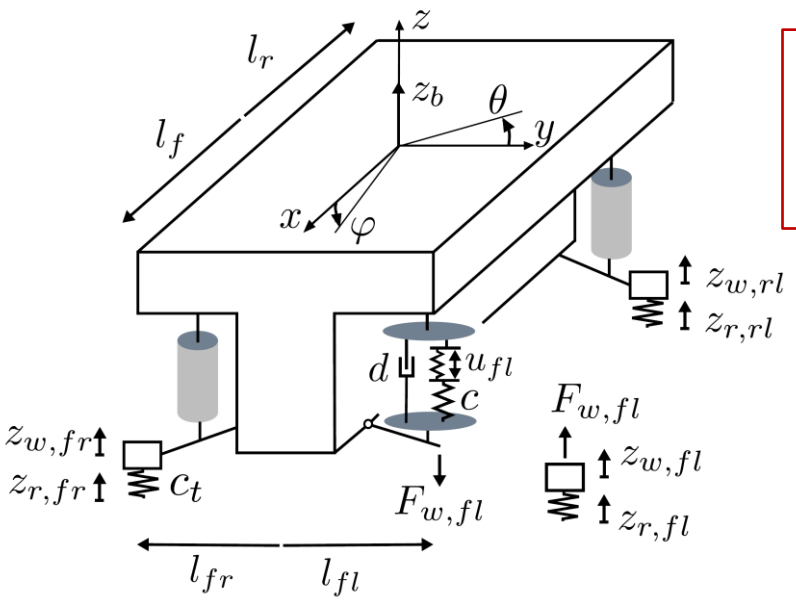


Göhrle, C.et al. “Design and Vehicle Implementation of Preview Active Suspension Controllers”

embotech

VEHICLE DYNAMICS

- Full car model **reduced to a 6th-order vehicle model** by neglecting wheel dynamics
- The **input** contains **active spring displacements** (u) and the **measurements of the height profile** of the upcoming road (ω) and its **first derivative** $(\dot{\omega})$.



$$\dot{x}(t) = Ax(t) + B_u u(t) + B_w \begin{pmatrix} w(t) \\ \dot{w}(t) \end{pmatrix}$$

$$y(t) = Cx(t) + Du(t)$$

reduced car model
(neglected wheel dynamics)

$$x := \begin{pmatrix} \text{heave displacement [m]} \\ \text{pitch angle [rads]} \\ \text{roll angle [rads]} \\ \text{heave velocity [m/s]} \\ \text{pitch rate [rads/s]} \\ \text{roll rate [rads/s]} \end{pmatrix}$$

$$u := (\text{active spring displacements [m]})$$

$$y := \begin{pmatrix} \text{heave acceleration [m/s}^2\text{]} \\ \text{pitch acceleration [rads/s}^2\text{]} \\ \text{roll acceleration [rads/s}^2\text{]} \end{pmatrix}$$

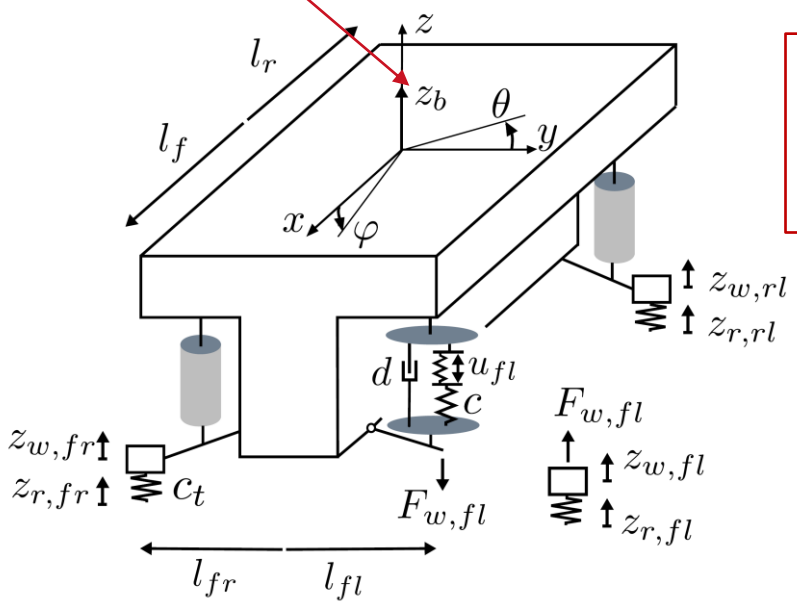
Göhrle, C.et al. "Design and Vehicle Implementation of Preview Active Suspension Controllers"

embotech

VEHICLE DYNAMICS

- Full car model **reduced to a 6th-order vehicle model** by neglecting wheel dynamics
- The **input** contains **active spring displacements** (u) and the **measurements of the height profile** of the upcoming road (ω) and its **first derivative** $(\dot{\omega})$.

heave displacement



$$\dot{x}(t) = Ax(t) + B_u u(t) + B_w \begin{pmatrix} w(t) \\ \dot{w}(t) \end{pmatrix}$$

$$y(t) = Cx(t) + Du(t)$$

reduced car model
(neglected wheel dynamics)

$$x := \begin{pmatrix} \text{heave displacement [m]} \\ \text{pitch angle [rads]} \\ \text{roll angle [rads]} \\ \text{heave velocity [m/s]} \\ \text{pitch rate [rads/s]} \\ \text{roll rate [rads/s]} \end{pmatrix}$$

$$u := (\text{active spring displacements [m]})$$

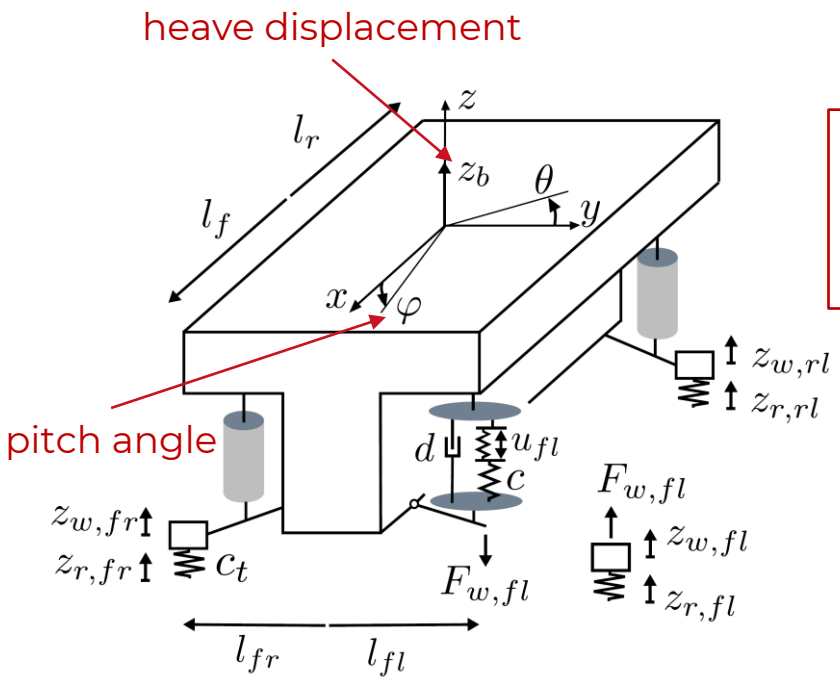
$$y := \begin{pmatrix} \text{heave acceleration [m/s}^2\text{]} \\ \text{pitch acceleration [rads/s}^2\text{]} \\ \text{roll acceleration [rads/s}^2\text{]} \end{pmatrix}$$

Göhrle, C.et al. "Design and Vehicle Implementation of Preview Active Suspension Controllers"

embotech

VEHICLE DYNAMICS

- Full car model **reduced to a 6th-order vehicle model** by neglecting wheel dynamics
- The **input** contains **active spring displacements** (u) and the **measurements of the height profile** of the upcoming road (ω) and its **first derivative** $(\dot{\omega})$.



$$\dot{x}(t) = Ax(t) + B_u u(t) + B_w \begin{pmatrix} w(t) \\ \dot{w}(t) \end{pmatrix}$$

$$y(t) = Cx(t) + Du(t)$$

reduced car model
(neglected wheel dynamics)

$$x := \begin{pmatrix} \text{heave displacement [m]} \\ \text{pitch angle [rads]} \\ \text{roll angle [rads]} \\ \text{heave velocity [m/s]} \\ \text{pitch rate [rads/s]} \\ \text{roll rate [rads/s]} \end{pmatrix}$$

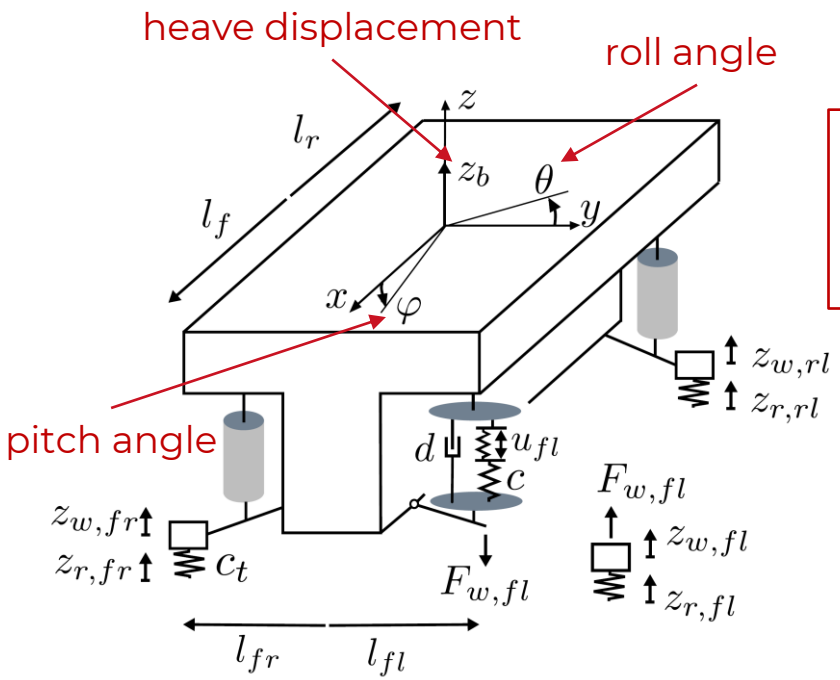
$$u := (\text{active spring displacements [m]})$$

$$y := \begin{pmatrix} \text{heave acceleration [m/s}^2\text{]} \\ \text{pitch acceleration [rads/s}^2\text{]} \\ \text{roll acceleration [rads/s}^2\text{]} \end{pmatrix}$$

Göhrle, C.et al. "Design and Vehicle Implementation of Preview Active Suspension Controllers"

VEHICLE DYNAMICS

- Full car model **reduced to a 6th-order vehicle model** by neglecting wheel dynamics
- The **input** contains **active spring displacements** (u) and the **measurements of the height profile** of the upcoming road (ω) and its **first derivative** $(\dot{\omega})$.



$$\dot{x}(t) = Ax(t) + B_u u(t) + B_w \begin{pmatrix} w(t) \\ \dot{w}(t) \end{pmatrix}$$

$$y(t) = Cx(t) + Du(t)$$

reduced car model
(neglected wheel dynamics)

$$x := \begin{pmatrix} \text{heave displacement [m]} \\ \text{pitch angle [rads]} \\ \text{roll angle [rads]} \\ \text{heave velocity [m/s]} \\ \text{pitch rate [rads/s]} \\ \text{roll rate [rads/s]} \end{pmatrix}$$

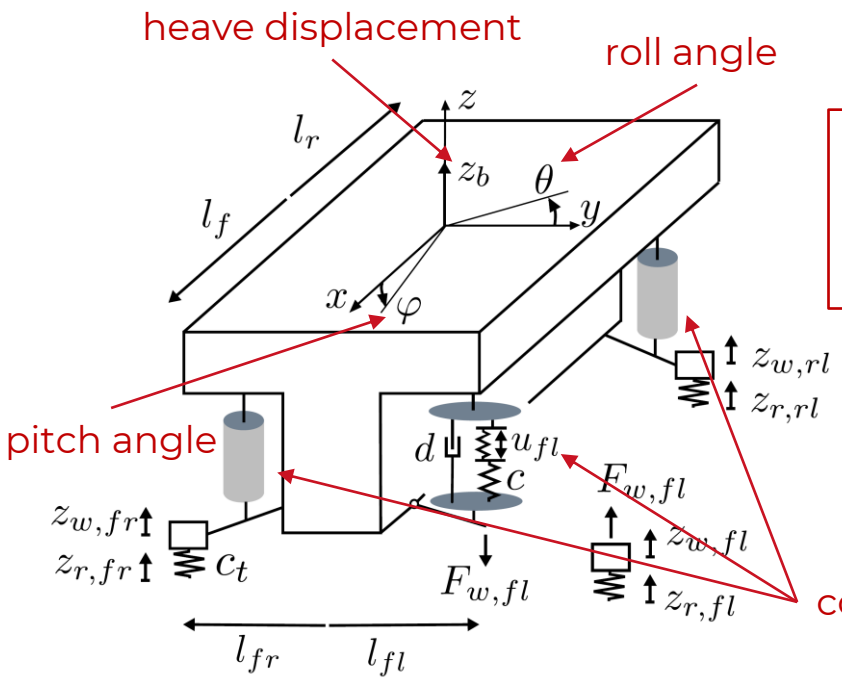
$$u := (\text{active spring displacements [m]})$$

$$y := \begin{pmatrix} \text{heave acceleration [m/s}^2\text{]} \\ \text{pitch acceleration [rads/s}^2\text{]} \\ \text{roll acceleration [rads/s}^2\text{]} \end{pmatrix}$$

Göhrle, C.et al. "Design and Vehicle Implementation of Preview Active Suspension Controllers"

VEHICLE DYNAMICS

- Full car model **reduced to a 6th-order vehicle model** by neglecting wheel dynamics
- The **input** contains **active spring displacements** (u) and the **measurements of the height profile** of the upcoming road (ω) and its **first derivative** $(\dot{\omega})$.



$$\dot{x}(t) = Ax(t) + B_u u(t) + B_w \begin{pmatrix} w(t) \\ \dot{w}(t) \end{pmatrix}$$

$$y(t) = Cx(t) + Du(t)$$

reduced car model
(neglected wheel dynamics)

$$x := \begin{pmatrix} \text{heave displacement [m]} \\ \text{pitch angle [rads]} \\ \text{roll angle [rads]} \\ \text{heave velocity [m/s]} \\ \text{pitch rate [rads/s]} \\ \text{roll rate [rads/s]} \end{pmatrix}$$

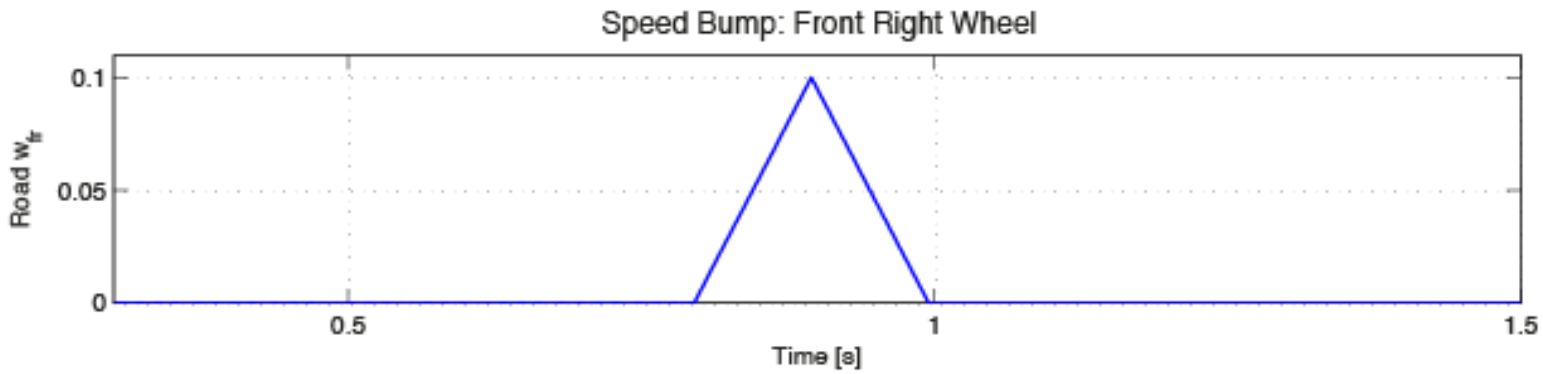
$$u := (\text{active spring displacements [m]})$$

$$y := \begin{pmatrix} \text{heave acceleration [m/s}^2\text{]} \\ \text{pitch acceleration [rads/s}^2\text{]} \\ \text{roll acceleration [rads/s}^2\text{]} \end{pmatrix}$$

Göhrle, C.et al. "Design and Vehicle Implementation of Preview Active Suspension Controllers"

DISTURBANCE MODEL: SPEED BUMP

- **Disturbance** modeled as a **speed bump** of 1m length and 0.1m height **hitting the right side of the car**
- Vehicle assumed to be driving at a **constant speed of 5m/s** over a speed bump
- Road bump only hits the front right wheel, while the front left wheel is unaffected
- The same bump will hit the rear right wheel 1.12s after it hits the front wheel



PROBLEM TYPE & INFORMATION PREVIEW

- **Linear MPC problem** with lower and upper bounds on inputs, and a terminal cost term
- **Initial state x** and **preview information ω** and $\dot{\omega}$ change at each sampling instant
- **Parametric additive terms g** containing preview information have to be defined at each stage
- '*pren_w*' represents the name of the **additive term g_n at stage n** of the multistage problem
- During runtime, the **preview information is mapped to these parameters**
- The length of the preview horizon is set to be equal to the prediction horizon ($N = 20$)

$$
\begin{aligned}
\text{minimize} \quad & x_N^T P x_N + \sum_{i=0}^{N-1} x_i^T Q x_i + u_i^T R u_i \\
\text{subject to} \quad & x_0 = x \\
& x_{i+1} = A x_i + B u_i + \underbrace{B_w w_i + B_w \dot{w}_i}_{g} \\
& \underline{u} \leq u_i \leq \overline{u}
\end{aligned}
$$

```
% Parameters: First Equation RHS
parameter(1) = newParam('minusA_times_x0_minusBw_times_w_pre',1,'eq.c');

% Paramteres: Preview Information
parameter(2) = newParam('pre2_w',2,'eq.c');
...
parameter(n) = newParam('pren_w',n,'eq.c');
...
parameter(N) = newParam('preN_w',N,'eq.c');
```

embotech

MPC PROBLEM DATA

$$\begin{aligned}
\text{minimize} \quad & x_N^T P x_N + \sum_{i=0}^{N-1} x_i^T Q x_i + u_i^T R u_i \\
\text{subject to} \quad & x_0 = x \\
& x_{i+1} = A x_i + B u_i + \underbrace{B_w w_i + B_w \dot{w}_i}_{g} \\
& \underline{u} \leq u_i \leq \overline{u}
\end{aligned}$$

```
% Dimensions
nx = 6;
nu = 4;

% Prediction (preview) horizon
N = 20;

% Cost matrices
Q = diag([50,50,50,50,50,50]);
R = eye(nu);
P = 20*Q;

% Constraints
umin = -.04; umax = .04;
```

```
% System matrices
Ad = [0.9785       0       0  0.0235       0       0;
           0  0.8978       0       0  0.0180       0;
           0       0  0.8524       0       0  0.0149;
     -1.6849       0       0  0.8781       0       0;
           0 -7.2761       0       0  0.4641       0;
           0       0 -9.8046       0       0  0.2680];

Bdu = [-0.0067 -0.0067 -0.0067 -0.0067;
        0.0114  0.0114 -0.0114 -0.0114;
       -0.0288  0.0288 -0.0288  0.0288;
       -0.5265 -0.5265 -0.5265 -0.5265;
        0.8121  0.8121 -0.8121 -0.8121;
       -1.9150  1.9150 -1.9150  1.9150];

Bdw = [-0.0054 -0.0054 -0.0054 -0.0054 -0.0003 -0.0003 -0.0003 -0.0003;
        0.0091  0.0091 -0.0091 -0.0091  0.0005  0.0005 -0.0005 -0.0005;
       -0.0231  0.0231 -0.0231  0.0231 -0.0014  0.0014 -0.0014  0.0014;
       -0.4212 -0.4212 -0.4212 -0.4212 -0.0251 -0.0251 -0.0251 -0.0251;
        0.6497  0.6497 -0.6497 -0.6497  0.0387  0.0387 -0.0387 -0.0387;
       -1.5320  1.5320 -1.5320  1.5320 -0.0913  0.0913 -0.0913  0.0913];

Cd = [ -71.5884         0         0 -4.2667         0         0;
            0 -404.1020         0       0 -24.0845         0;
            0         0 -659.7584       0         0 -39.3216];

Dd = [-22.37 -22.37  -22.37 -22.37  -17.89  -17.89  -17.89  -17.89 -1.06 -1.06 -1.06 -1.06;
       45.11  45.11  -45.11 -45.11   36.08   36.08  -36.08  -36.08  2.15  2.15 -2.15 -2.15;
     -128.86 128.86 -128.86 128.86 -103.09  103.09 -103.09  103.09 -6.14  6.14 -6.14  6.14];
```

embotech

MULTISTAGE MPC FORM

$$
\begin{aligned}
\text{minimize} \quad & x_N^T P x_N + \sum_{i=0}^{N-1} x_i^T Q x_i + u_i^T R u_i \\
\text{subject to} \quad & x_0 = x \\
& x_{i+1} = A x_i + B u_i + \underbrace{B_w w_i + B_w \dot{w}_i}_{g} \\
& \underline{u} \leq u_i \leq \overline{u}
\end{aligned}
$$

```
% Dimensions
nx = 6;
nu = 4;

% Prediction (preview) horizon
N = 20;

% Cost matrices
Q = diag([50,50,50,50,50,50]);
R = eye(nu);
P = 20*Q;

% Constraints
umin = -.04; umax = .04;
```

```
% Assume variable ordering zi = [ui; xi+1] for i=1 ... N-1
% Parameters: First Eq. RHS
parameter(1) = newParam('minusA_times_x0_minusBw_times_w_pre',1,'eq.c');
stages = MultistageProblem(N);
for i = 1:N
    % Dimensions
    stages(i).dims.n = nx+nu; % number of stage variables
    stages(i).dims.r = nx; % number of equality constraints
    stages(i).dims.l = nu; % number of lower bounds
    stages(i).dims.u = nu; % number of upper bounds
    % Cost
    if( i == N )
        stages(i).cost.H = blkdiag(R,P);
    else
        stages(i).cost.H = blkdiag(R,Q);
    end
    stages(i).cost.f = zeros(nx+nu,1);
    % Lower bounds
    stages(i).ineq.b.lbidx = 1:nu; % lower bound acts on these indices
    stages(i).ineq.b.lb = umin*ones(4,1); % lower bound for the input signal
    % Upper bounds
    stages(i).ineq.b.ubidx = 1:nu; % upper bound acts on these indices
    stages(i).ineq.b.ub = umax*ones(4,1); % upper bound for the input signal
    % Equality constraints
    if( i < N )
        stages(i).eq.C = [zeros(nx,nu), Ad];
    end
    stages(i).eq.D = [Bdu, -eye(nx)];
    % Parameters for preview
    if( i < N )
        parameter(i+1) = newParam(['pre',num2str(i+1),'_w'],i+1,'eq.c');
    end
end
% Define outputs of the solver
outputs(1) = newOutput('u0',1,1:nu);
% Solver settings
codeoptions = getOptions('VEHICLE_MPC_withPreview');
% Generate code
generateCode(stages,parameter,codeoptions,outputs);
```

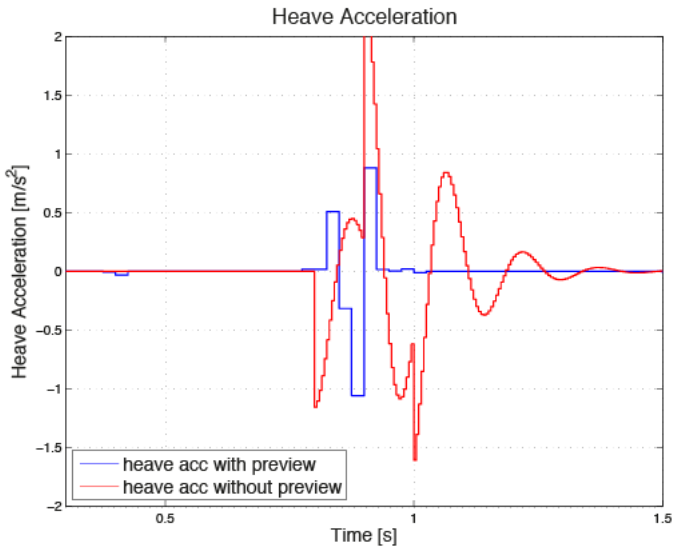
SIMULATION

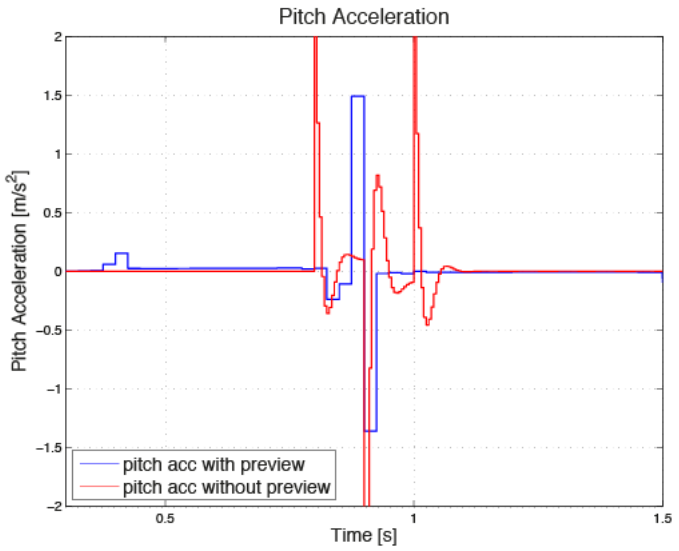
```
% Simulation
x1 = zeros(6,1);
X_wP = zeros(nx,kmax_pre+1); X_wP(:,1) = x1;
Y_wP = zeros(3,kmax_pre);
U_wP = zeros(nu,kmax_pre);
problem.z1 = zeros(2*nx,1);
for k = 1:kmax_pre
    problem.minusA_times_x0_minusBw_times_w_pre = -Ad*X_wP(:,k)-Bdw*[w_pre(:,k)];
    problem.pre2_w  = -Bdw*w_pre(:,k+1);  problem.pre3_w  = -Bdw*w_pre(:,k+2);
    problem.pre4_w  = -Bdw*w_pre(:,k+3);  problem.pre5_w  = -Bdw*w_pre(:,k+4);
    problem.pre6_w  = -Bdw*w_pre(:,k+5);  problem.pre7_w  = -Bdw*w_pre(:,k+6);
    problem.pre8_w  = -Bdw*w_pre(:,k+7);  problem.pre9_w  = -Bdw*w_pre(:,k+8);
    problem.pre10_w = -Bdw*w_pre(:,k+9);  problem.pre11_w = -Bdw*w_pre(:,k+10);
    problem.pre12_w = -Bdw*w_pre(:,k+11); problem.pre13_w = -Bdw*w_pre(:,k+12);
    problem.pre14_w = -Bdw*w_pre(:,k+13); problem.pre15_w = -Bdw*w_pre(:,k+14);
    problem.pre16_w = -Bdw*w_pre(:,k+15); problem.pre17_w = -Bdw*w_pre(:,k+16);
    problem.pre18_w = -Bdw*w_pre(:,k+17); problem.pre19_w = -Bdw*w_pre(:,k+18);
    problem.pre20_w = -Bdw*w_pre(:,k+19);
    [solverout,exitflag,info] = VEHICLE_MPC_withPreview(problem);
    if( exitflag == 1 )
        U_wP(:,k) = solverout.u0;
    else
        info
        error('Some problem in solver');
    end
    X_wP(:,k+1) = Ad*X_wP(:,k) + [Bdu, Bdw]*[U_wP(:,k); w_pre(:,k)];
    Y_wP(:,k) = Cd*X_wP(:,k) + Dd*[U_wP(:,k); w_pre(:,k)];
end
```

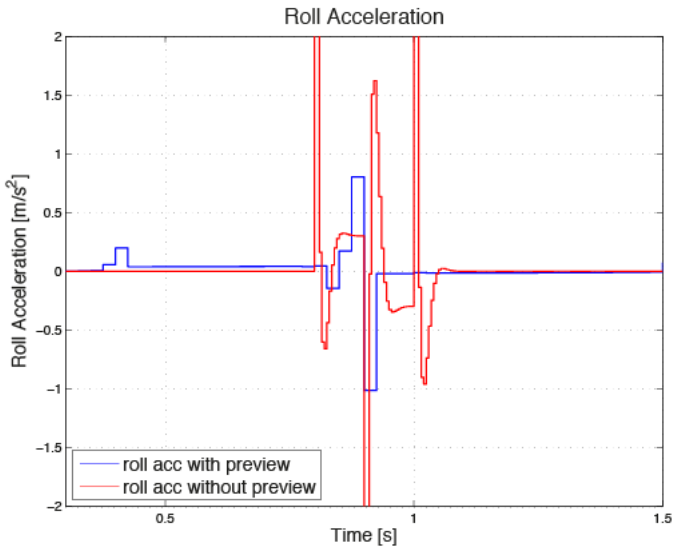
```
% Assume variable ordering zi = [ui; xi+1] for i=1 ... N-1
% Parameters: First Eq. RHS
parameter(1) = newParam('minusA_times_x0_minusBw_times_w_pre',1,'eq.c');
stages = MultistageProblem(N);
for i = 1:N
    % Dimensions
    stages(i).dims.n = nx+nu; % number of stage variables
    stages(i).dims.r = nx; % number of equality constraints
    stages(i).dims.l = nu; % number of lower bounds
    stages(i).dims.u = nu; % number of upper bounds
    % Cost
    if( i == N )
        stages(i).cost.H = blkdiag(R,P);
    else
        stages(i).cost.H = blkdiag(R,Q);
    end
    stages(i).cost.f = zeros(nx+nu,1);
    % Lower bounds
    stages(i).ineq.b.lbidx = 1:nu; % lower bound acts on these indices
    stages(i).ineq.b.lb = umin*ones(4,1); % lower bound for the input signal
    % Upper bounds
    stages(i).ineq.b.ubidx = 1:nu; % upper bound acts on these indices
    stages(i).ineq.b.ub = umax*ones(4,1); % upper bound for the input signal
    % Equality constraints
    if( i < N )
        stages(i).eq.C = [zeros(nx,nu), Ad];
    end
    stages(i).eq.D = [Bdu, -eye(nx)];
    % Parameters for preview
    if( i < N )
        parameter(i+1) = newParam(['pre',num2str(i+1),'_w'],i+1,'eq.c');
    end
end
% Define outputs of the solver
outputs(1) = newOutput('u0',1,1:nu);
% Solver settings
codeoptions = getOptions('VEHICLE_MPC_withPreview');
% Generate code
generateCode(stages,parameter,codeoptions,outputs);
```

PASSIVE VS ACTIVE VEHICLE SUSPENSION

- **Active** vehicle suspension **substantially reduces** vertical dynamics and **improves** riding comfort
- **Lower maximum acceleration** and **less time is required** **to regulate the acceleration** back to zero

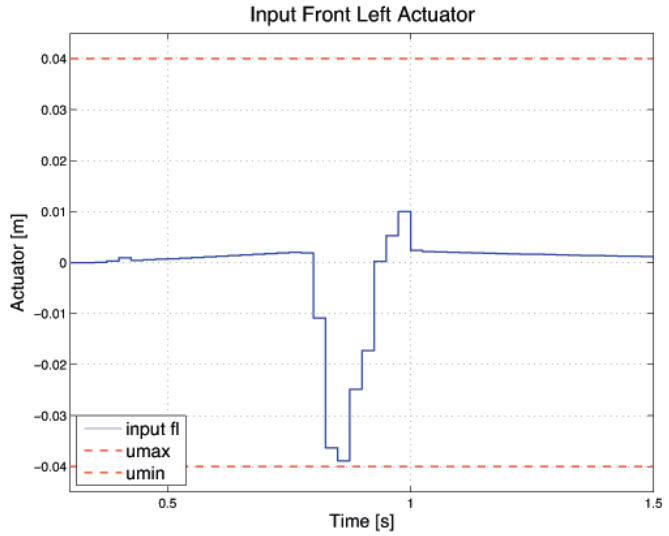


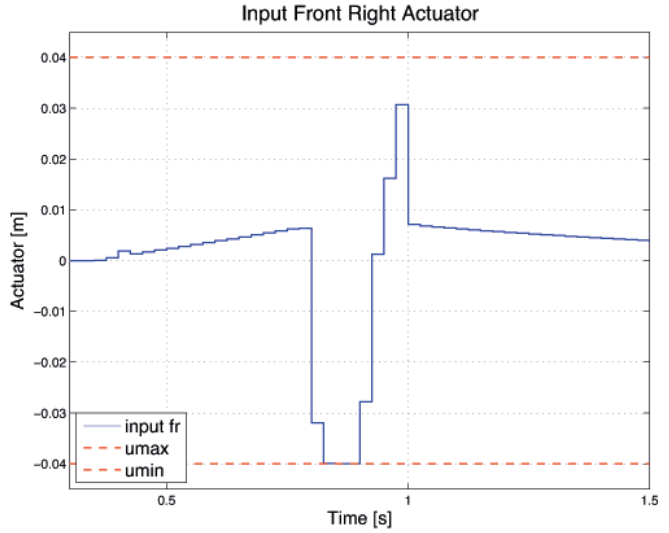


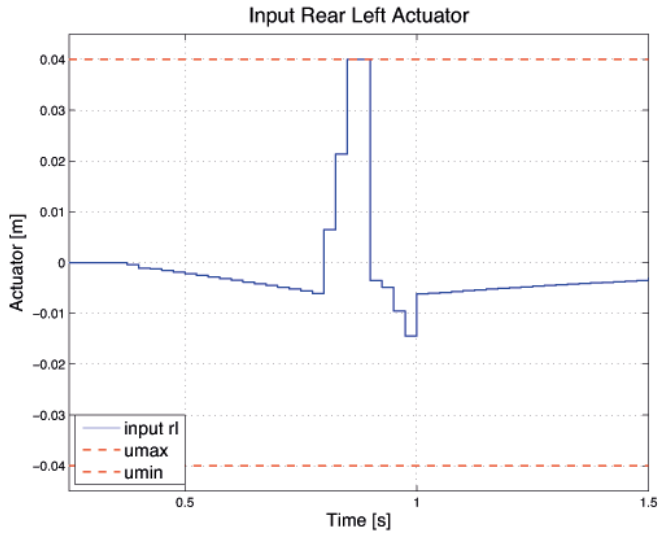


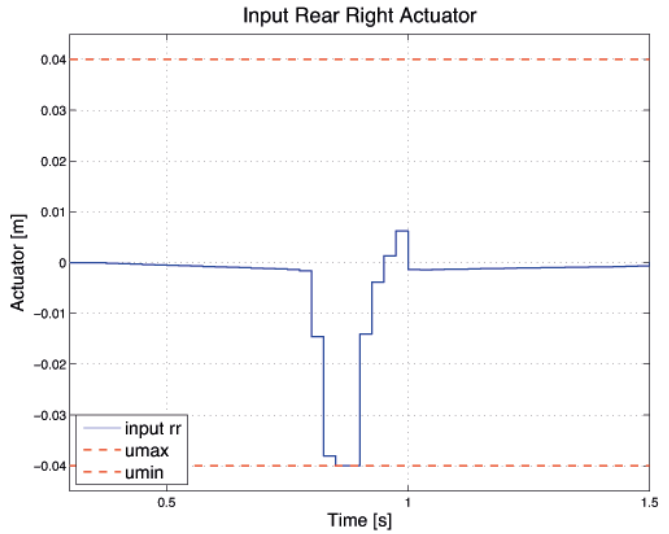
PASSIVE VS ACTIVE VEHICLE SUSPENSION

- The front right part of the vehicle body **lifts as soon as the bump is in sight** of the preview sensor (at time t = 0.3s)
- This is a full length of the preview horizon (0.5s) **before the front right wheel hits** the bump at time t = 0.8s
- Results in **better absorption** of the shock and therefore **reduced acceleration**









embotech

embotech.com