MECH 6V29 - MPC - HW 4

Jonas Wagner

Abstract

In this homework assignment an active suspension system was analyzed and controlled using both an LQR and an MPC controller.

I. Problem Definition

The system to be controlled was provided in the MATLAB file as

$$\begin{aligned} \dot{x}(t) &= \mathbf{A}_c x(t) + \mathbf{B}_c u(t) + \mathbf{V}_c d(t) \\ y(t) &= \mathbf{C} x(t) + \mathbf{D} u(t) + \mathbf{W} d(t) \end{aligned} \tag{1}$$

with states $x = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T \in \mathbb{R}^{n_x=4}$ (tire-deflection, unsprung mass velocity, suspension deflection, and sprung mass velocity), input $u = u_1 \in \mathbb{R}^{n_u=1}$ (active suspension force), disturbances $d = \begin{bmatrix} d_1 & d_2 \end{bmatrix}^T \in \mathbb{R}^{n_d=2}$ (road height change and road height), and outputs $y = \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix}^T \in \mathbb{R}^{n_y=3}$ (unsprung mass height, sprung mass height, and spring mass acceleration). The matrices are defined as

$$\mathbf{A}_c = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{K_t}{M_u} & -\frac{B_s+B_t}{M_u} & \frac{K_s}{M_u} & \frac{B_s}{M_u} \\ 0 & -1 & 0 & 1 \\ 0 & \frac{B_s}{M_s} & -\frac{K_s}{M_s} & -\frac{B_s}{M_s} \end{bmatrix} \mathbf{B}_c = \begin{bmatrix} 0 \\ \frac{-1}{M_u} \\ 0 \\ \frac{1}{M_s} \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & \frac{B_s}{M_s} & -\frac{K_s}{M_s} & -\frac{B_s}{M_s} \end{bmatrix} D = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{M_s} \end{bmatrix}$$

$$\mathbf{V}_c = \begin{bmatrix} -1 & 0 \\ \frac{B_t}{M_u} & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} W = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

with parameters $K_s = 900$ [N/m], $K_t = 2500$ [N/m], $M_s = 2.45$ [kg], $M_u = 1$ [kg], $B_s = 7.5$ [s/m], and $B_t = 5$ [s/m].

The code derives the DT-LTI system at different discretization time steps depending on the selected velocity and the disturbance data.

In addition, bounds on the states, inputs, and disturbances were provided as the intervals:

$$x \in \mathcal{X} = \left\{ x \in \mathbb{R}^{n_x} \mid |x| \leq \begin{bmatrix} 0.01 \\ 1 \\ 0.03 \\ 1 \end{bmatrix} \right\}$$

$$u \in \mathcal{U} = \{ u \in \mathbb{R}^{n_u} \mid |u| \leq 30 \}$$

$$d \in \mathcal{D} = \left\{ d \in \mathbb{R}^{n_d} \mid |d| \leq \begin{bmatrix} 0.5 \\ 0.02 \end{bmatrix} \right\}$$

II. Open-Loop Dynamics

Initial open-loop dynamics are simulated with lsim() and the provided disturbance data resulting in the poor response dynamics shown in Fig. 1 for the realistic road profile (IRI_737b) along with the synthetic roadBumpHole road profiles. (See appendix for additional results) From these dynamics the high oscillatory (and poor dampening) effects can easily be observed. The first state, tire deflection (x_1), was often exceeding the acceptable state bounds.

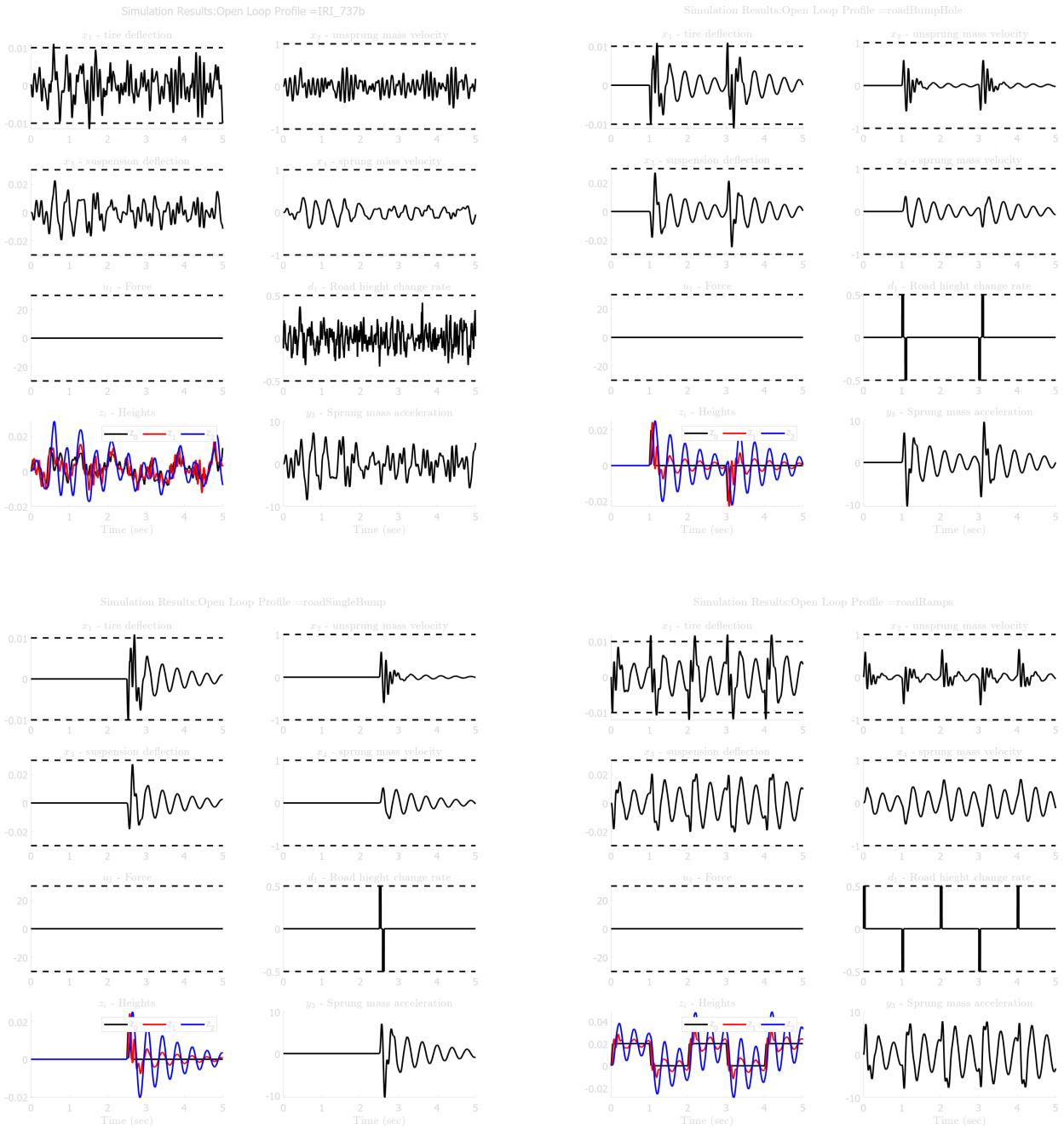

Fig. 1. Open-loop dynamics for the IRI_737b, roadBumpHole, roadRamps, and roadSingleBump road profiles.

III. LQR Controller Design

Initial LQR controller designs were implemented and tested to provide a baseline optimization function for the MPC controller itself. The MATLAB lqr() function was used to design a controller with designed Q and R matrices.

Two primary cost matrices for state-optimization were developed: direct acceleration minimization (Q_{accel}) and bound-based optimization (Q_{bounds}).

$$Q_{accel} = \frac{10^5}{\|C^T C\|_\infty} \cdot C^T \begin{bmatrix} 0 & & \\ & 0 & \\ & & 1 \end{bmatrix} C \tag{2}$$

$$Q_{bounds} = 10^3 \cdot \begin{bmatrix} \frac{1}{0.01} & & & \\ & 1 & & \\ & & \frac{1}{0.03} & \\ & & & 1 \end{bmatrix} \tag{3}$$

These were tested individually and combined as $Q_{both} = Q_{accel} + Q_{bounds}$.

For the input cost, a simple inverse bound cost was used:

$$R = \begin{bmatrix} \frac{1}{0.5} & \\ & \frac{1}{0.02} \end{bmatrix} \tag{4}$$

The simulations are provided in Fig. 2, Fig. 3, and Fig. 4 respectively.

Each of these lqr control schemes had a noticeable improvement in dampening transient response. The demonstrative Q matrices are weighted against the R matrices by multiple orders of magnitude as iteratively testing yielded these weights to have a reasonable balance.

Although it does have a noticeable improvement in response, it does not meet the ideal specs; specifically, the state bounds are still often exceeded and the oscillations still don't seem like an enjoyable ride.

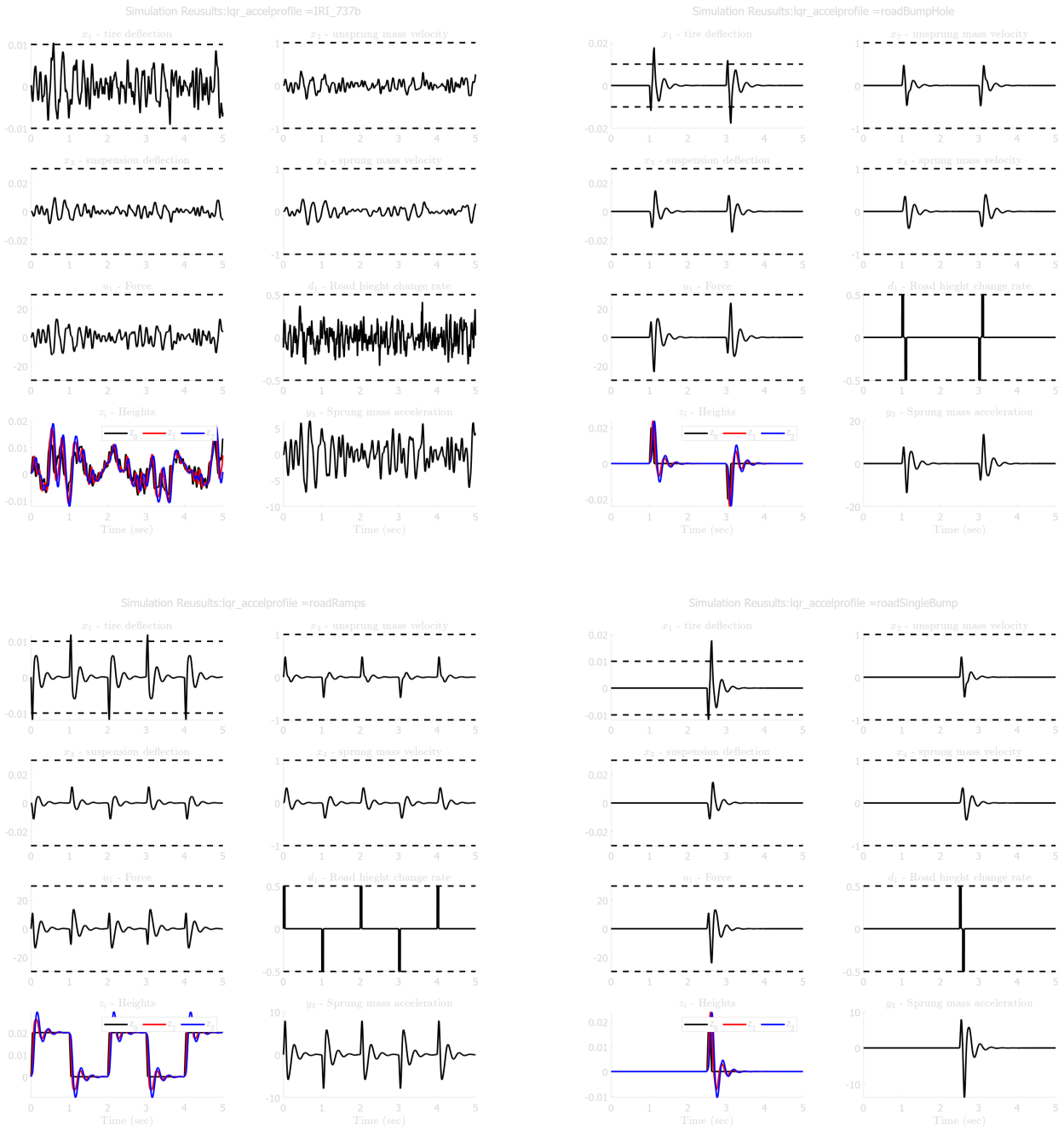

Fig. 2. LQR closed-loop dynamics with Q_{accel} and R for the IRI_737b, roadBumpHole, roadRamps, and roadSingleBump road profiles.

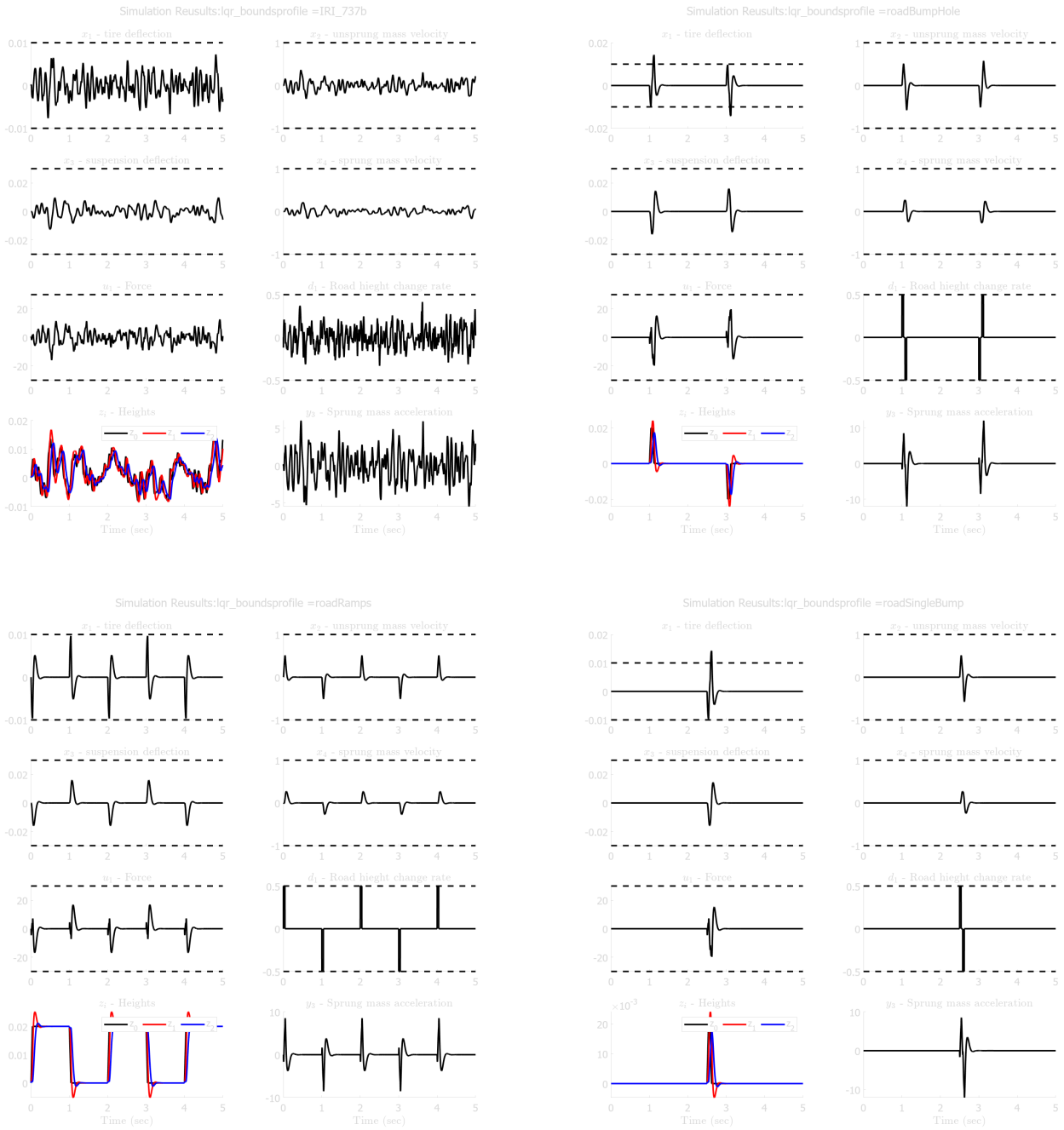

Fig. 3. LQR closed-loop dynamics with Q_bounds and R for the IRI_737b, roadBumpHole, roadRamps, and roadSingleBump road profiles.

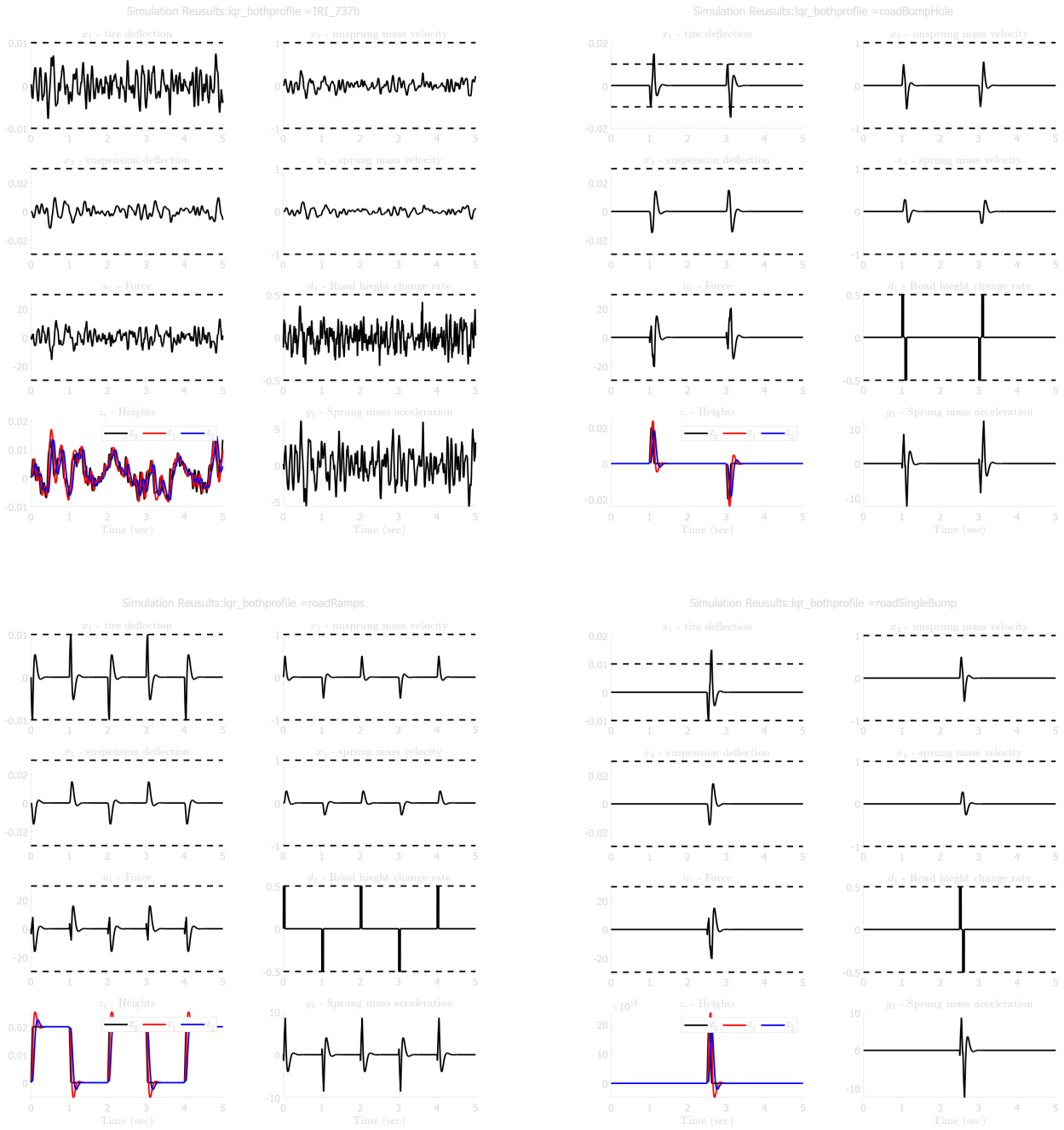

Fig. 4. LQR closed-loop dynamics with Q_{both} and R for the IRI_737b, roadBumpHole, roadRamps, and roadSingleBump road profiles.

IV. Simple MPC Controller Formulation

An initial implementation of the MPC controller was constructed using the provided code. The same Q and R matrices from before were used and tested iteratively to attempt good responses. (See MATLAB code with rFlag = 0 for the simple formulation). For demonstrative purposes only the Q_{both} version of cost are provided in this report.

A. Update Parameter Selection

The time-horizon and time-step selection were tested for multiple values and ultimately the decision to select a specific value for this was pretty arbitrary. The demonstrative plots are using $N = 15$, but this selection was mostly just an initial guess and since no explicit hardware was selected the computational limit is pretty arbitrary. The integer multiplier was selected mainly by comparing a few different values. As can be seen in Fig. 5, the case where the prediction forward is done every-single time-step the response becomes very oscillatory and the bounds are greatly exceeded.

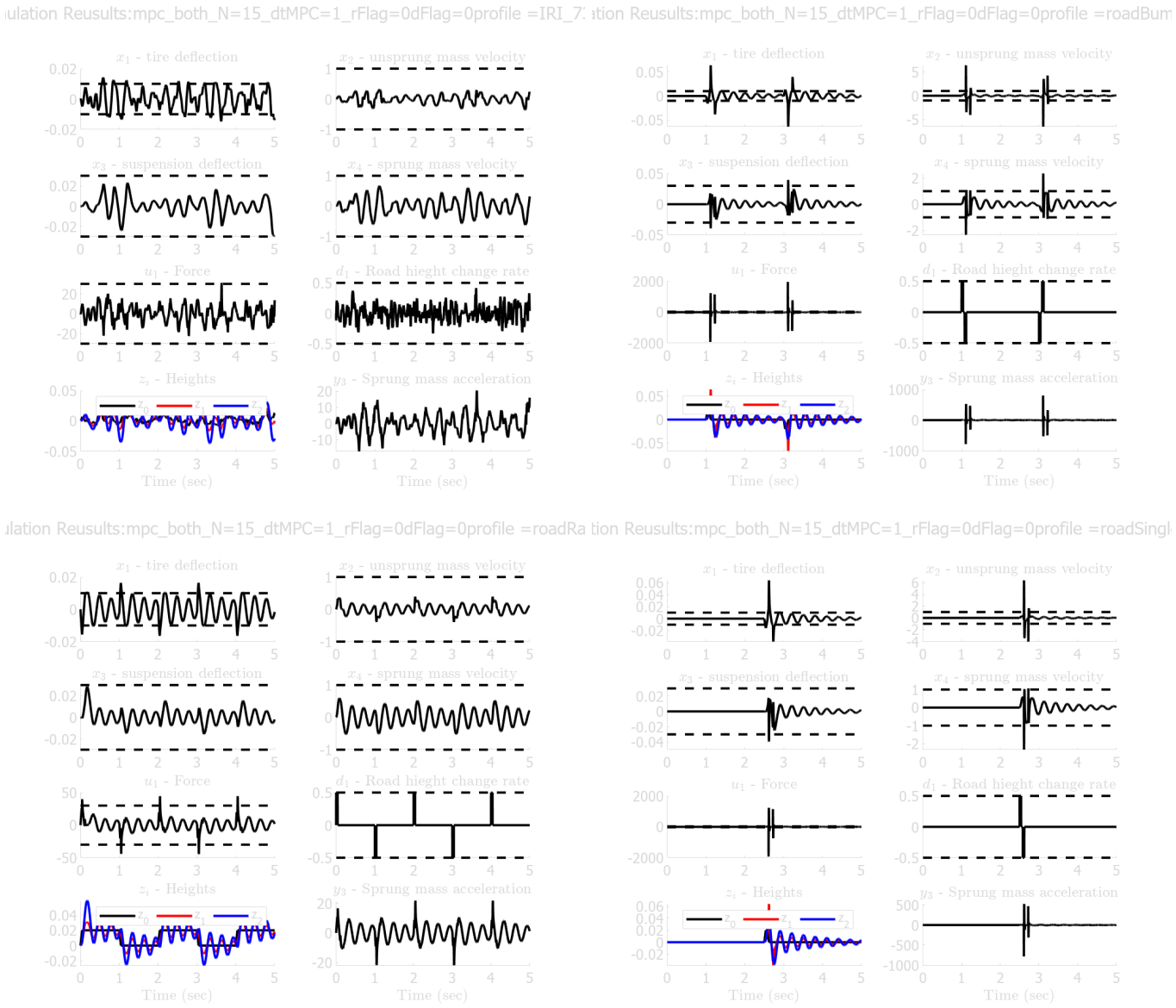

Fig. 5. Closed-loop dynamics with no disturbance information for standard MPC using Q_{both} and R updating every time-step with $N = 15$ for the IRI_737b, roadBumpHole, roadRamps, and roadSingleBump road profiles.

By comparison, as seen in Fig. 6, predicting forward in increments of 3 time-steps is far less oscillatory; however this is also dependent on what N is.

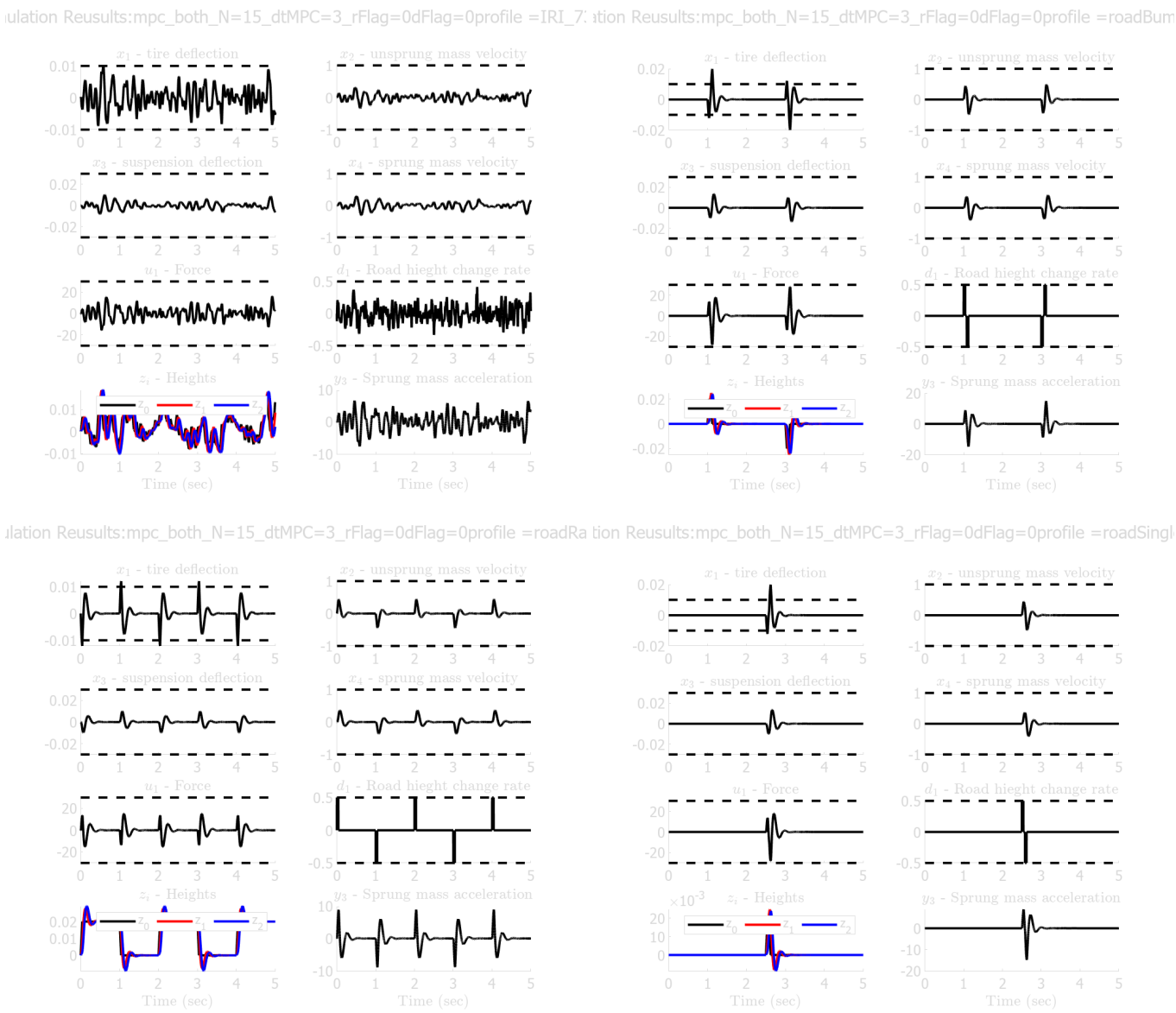

Fig. 6. Closed-loop dynamics with no disturbance information for standard MPC using Q_{both} and R updating every 3 time-steps with $N = 15$ for the IRI_737b, roadBumpHole, roadRamps, and roadSingleBump road profiles.

B. Disturbance knowledge Comparison

The closed-loop dynamics were also tested for when the MPC is given different levels of disturbance information:

1) The nominal case (dFlag = 0) sets $d_1 = d_2 = 0$ for the entire time-horizon.
2) The current state-known case (dFlag = 1) assumes that d_1 and d_2 are known at k. Although also tested when the rest of the testing-horizon is fully nominal ($d_1 = d_2 = 0$) and $d_1 = 0$ for the rest of the horizon, the better approach was for d_2 to be derived from having d_1 held constant for the time-horizon.
3) The entire knowledge case (dFlag = 2) has both d_1 and d_2 directly inputed into the MPC problem.

A direct comparison can be seen in Fig. 7,Fig. 7, and Fig. 8.

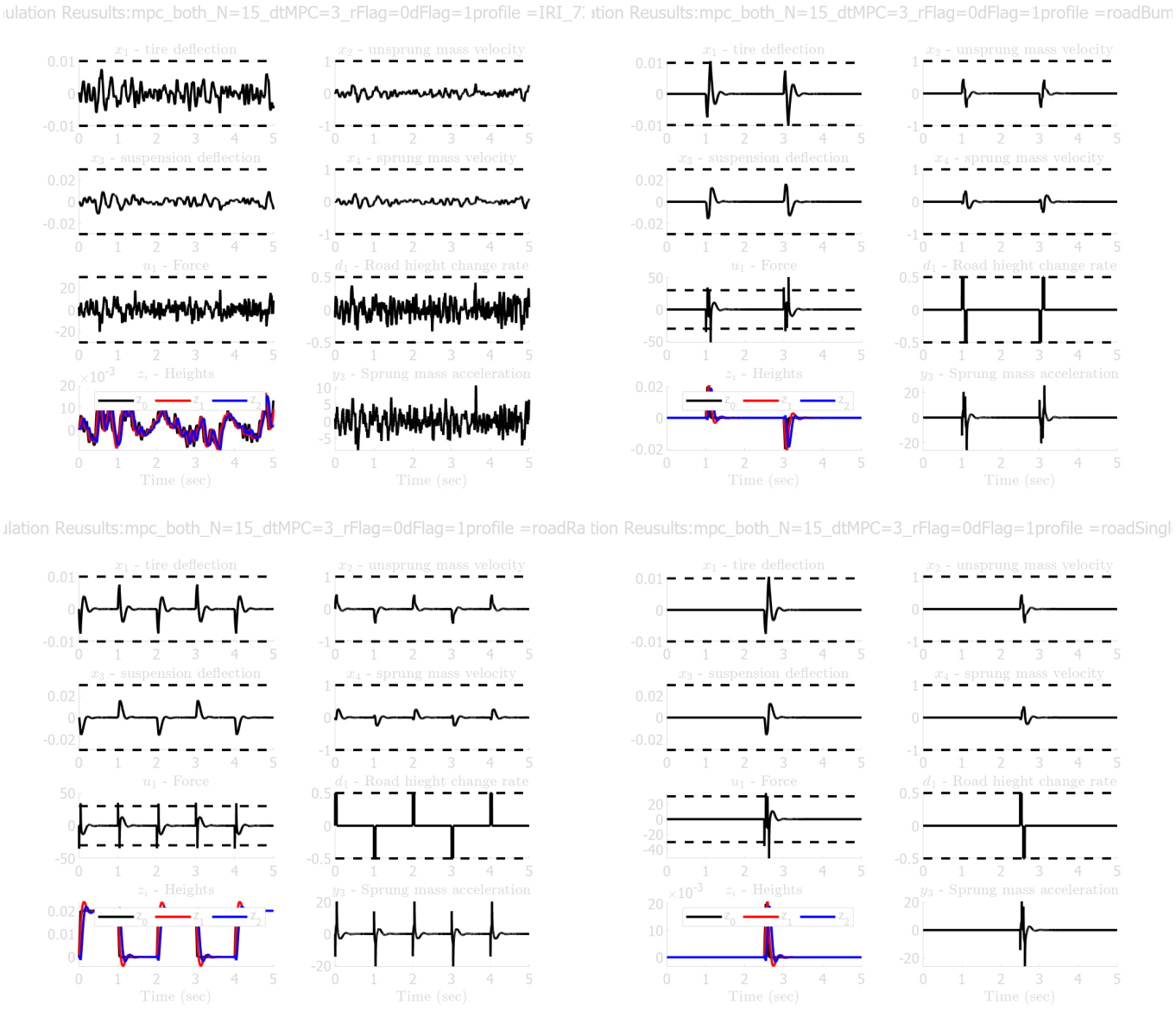

Fig. 7. Closed-loop dynamics with partial disturbance information for standard MPC using Q_{both} and R updating every 3 time-steps with $N = 15$ for the IRI_737b, roadBumpHole, roadRamps, and roadSingleBump road profiles.

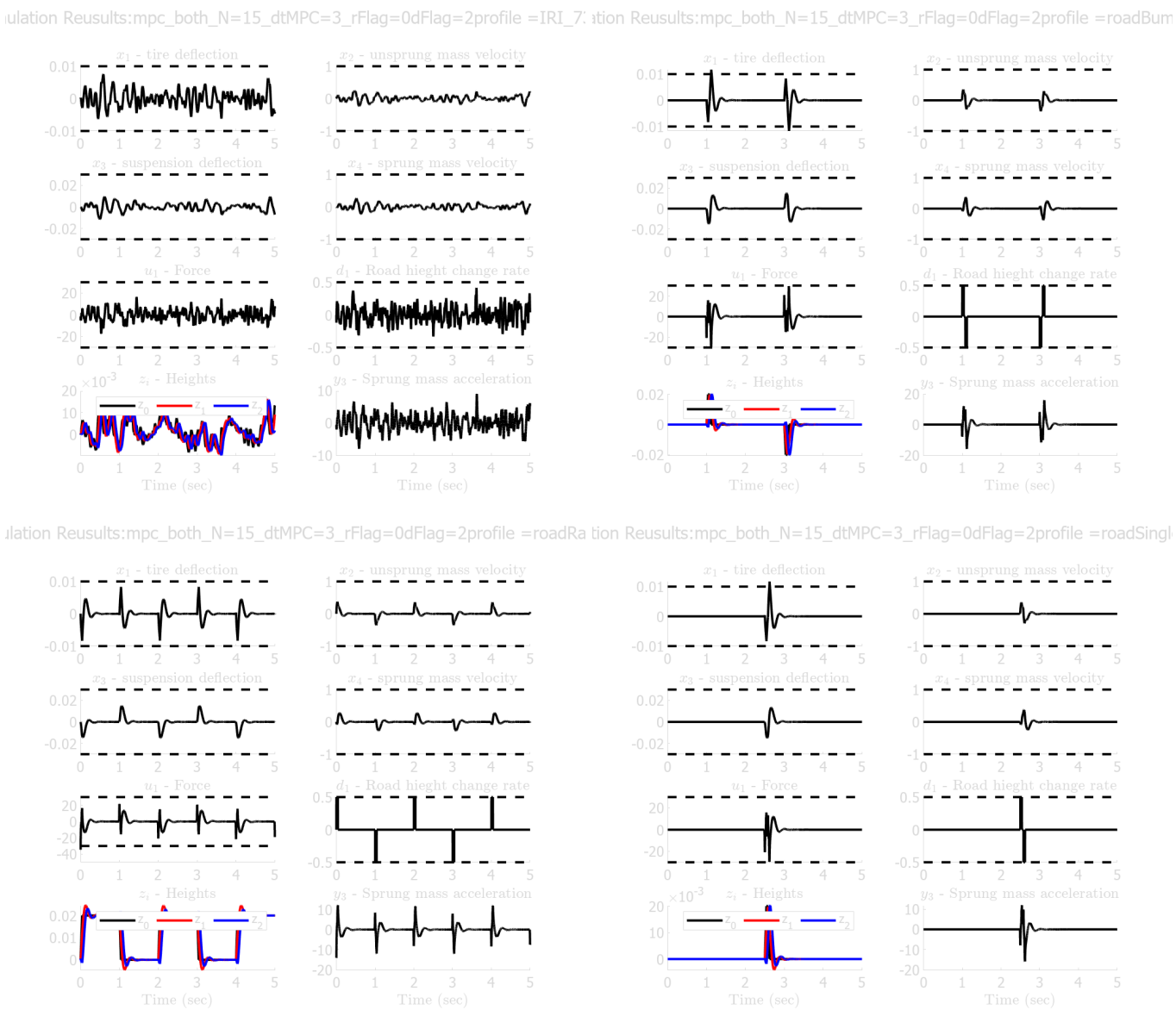

Fig. 8. Closed-loop dynamics with full disturbance information for standard MPC using Q_{both} and R updating every 3 time-steps with $N = 15$ for the IRI_737b, roadBumpHole, roadRamps, and roadSingleBump road profiles.

V. Robust MPC Controller Formulation

The biggest issue with all of the previous implementations has been that the bounds (specifically the tire deflection and input; and even more-so when no disturbance information is known). In order to address this, additional constraints can be implimented to fix this.

A naive approach (something I have done prior to taking this course) is to directly attach these as hard constraints in the optimization problem (see appendix... well really the associated ones in the folder on my github) for these results; but essentially this causes an abundance of feasibility issues.

To fix this issue (something I did do prior to taking this class) would be to use those constraints instead as what I referred to as "soft" constraints within the objective function. This was also tested, although disregarded, since we had already learned better solutions (namely the constraint tightening and tube MPC approaches).

Although I intuitively understand the constraint tightening approach more, the implementation for the RPI sets is simpler to implemented[1], thus this was implemented.

When implemented there initially was a problem that although it was stable, the control input was now no longer robust as designed. This is likely an issue with the implimentation itself; however, in order to test how this designed system would perform, a saturation was added. This is shown in Fig. 9 and Fig. 10.

VI. Conclusion

As a result of completing this entire design, I would first continue to evaluate the implimentation of the robust MPC as I can only assume that it is failing on a part of my own mistakes. I do think that the best solution I've obtained so far was the simple MPC using $N = 15$ updating every 3 time-steps (along with providing as much disturbance information that is available) and the balenced cost funciton values I decided on (Q_{both} and R, with $P = Q_{both}$). That selection seemed to strike a good balance of transient dynamics for all of the cases observed, and didn't have the issues faced with the shorter time-horizons or fast updating steps.

[1] not a great reason, but it's why I made the choice I did

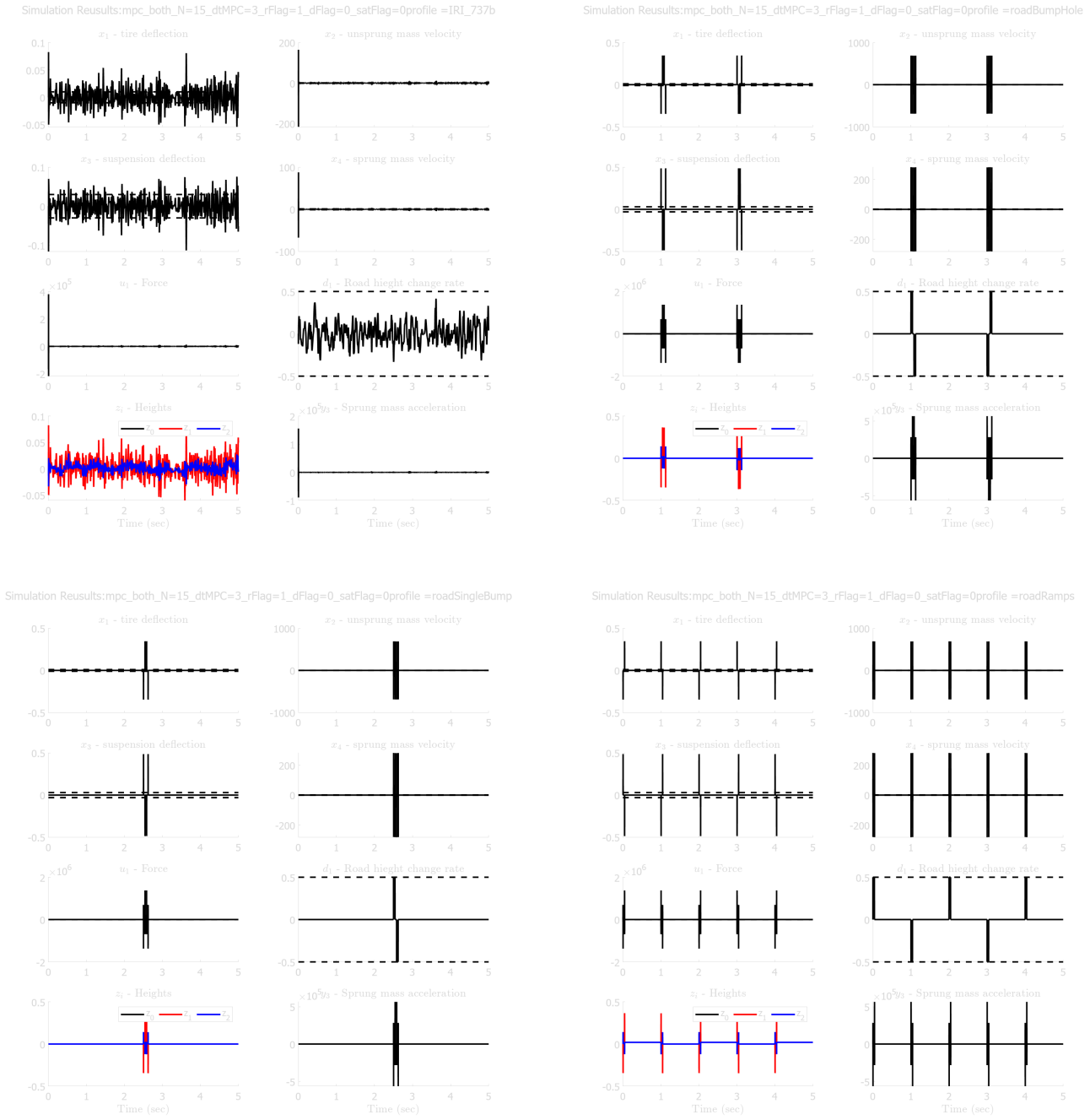

Fig. 9. Closed-loop dynamics for the unsaturated robust MPC using Q_{both} and R updating every 3 time-steps with $N = 15$ for the IRI_737b, roadBumpHole, roadRamps, and roadSingleBump road profiles.

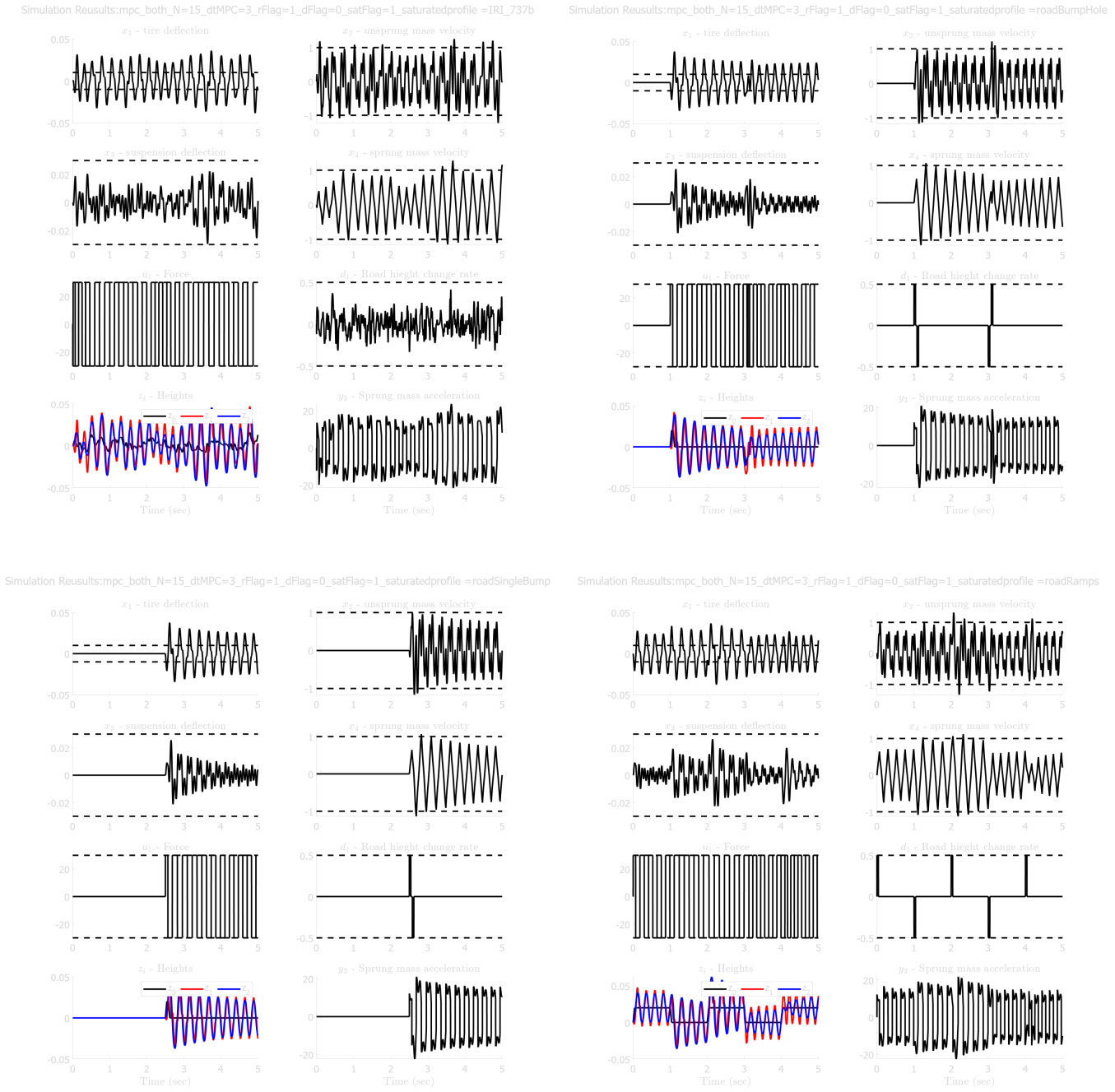

Fig. 10. Closed-loop dynamics for the saturated robust MPC using Q_{both} and R updating every 3 time-steps with $N = 15$ for the IRI_737b, roadBumpHole, roadRamps, and roadSingleBump road profiles.

APPENDIX

A. Github

See my github repo for all my course related materials: https://github.com/jonaswagner2826/MECH6v29_MPC

B. Matlab results

MATLAB code and results are attached.

Table of Contents


```
clear; close all;
addpath(genpath(pwd));
[filepath,~,~] = fileparts(mfilename('fullpath'));
fig_subfolder = strcat(filepath,filesep,'figs');
data_subfolder = strcat(filepath,filesep,'data');
```

System Definition

System dimensions

```
nx = 4; % Number of states
nu = 1; % Number of inputs (controllable)
nd = 2; % Number of disturbances
ny = 3; % Number of outputs

% Model parameters
Ks = 900;   % Suspension Stiffness (N/m)
Kt = 2500;  % Tire stiffness (N/m)
Ms = 2.45;  % Sprung Mass (kg)
Mu = 1;     % Unsprung Mass (kg)
Bs = 7.5;   % Suspension Inherent Damping coefficient (sec/m)
Bt = 5;    % Tire Inhenrent Damping coefficient (sec/m)

% Continuous-time state-space model
% x1 = z1-z0 (length of tire spring)
% x2 = z1_dot (velocity of unsprung mass)
% x3 = z2-z1 (length of suspension spring)
% x4 = x2_dot (velocity of sprung mass)
% u1 = F (active suspension force)
% d1 = z0_dot (rate of change of road height)
% d2 = z0 (road height)
% y1 = z1 (unsprung mass height)
% y2 = z2 (sprung mass height)
% y3 = z2_dotdot (acceleration of sprung mass)
Ac = [0 1 0 0;...
     -Kt/Mu -(Bs+Bt)/Mu Ks/Mu Bs/Mu;...
```

```
      0 -1 0 1;...
      0 Bs/Ms -Ks/Ms -Bs/Ms];
Bc = [0; -1/Mu; 0; 1/Ms];
Vc = [-1 0; Bt/Mu 0; 0 0; 0 0];
C = [1 0 0 0;...
      1 0 1 0;...
      0 Bs/Ms -Ks/Ms -Bs/Ms];
D = [0; 0; 1/Ms];
W = [0 1; 0 1; 0 0];
% LTI system
sys_c = ss(Ac,[Bc Vc],C,[D W]);
```

Constraints

Maximum tire deflection +/- 0.01 meters Maximum unsprung mass velocity is +/- 1 m/s Maximum suspension deflection +/- 0.03 meters Maximum unsprung mass velocity is +/- 1 m/s Maximum force is +/- 30 N Maximum change in road profile velocity +/- 0.5 meters/second Maximum change in road height +/- 0.02 meters

```
bounds.x_ub = [0.01; 1; 0.03; 1];
bounds.x_lb = -bounds.x_ub;
bounds.u_ub = 30;
bounds.u_lb = -bounds.u_ub;
bounds.d_ub = [0.5; 0.02];
bounds.d_lb = -bounds.d_ub;

X_set = Polyhedron('lb',bounds.x_lb,'ub',bounds.x_ub); X_set.minHRep;
U_set = Polyhedron('lb',bounds.u_lb,'ub',bounds.u_ub); U_set.minHRep;
D_set = Polyhedron('lb',bounds.d_lb,'ub',bounds.d_ub); D_set.minHRep;
```

Open-loop Simulation (no active suspension force)

```
tf = 5;                                 % simulation time length (must be less
than 10 - will run out of road)
scale = 1;                                % Scaling of road height
x0 = zeros(nx,1);                         % initial condition
v = 10;                                   % vehicle velocity (m/s) (will
change the simulation time step)
% Chose your road profile
% load IRI_737b                           % road profile data (realistic road)
% load roadRamps                          % road profile data (multiple large
ramps)
% load roadSingleBump                     % road profile data (one bump)
% load roadBumpHole                       % road profile data (one bump and
one hole)

roadProfile = 'roadRamps';
% for roadProfile = ["IRI_737b","roadRamps","roadSingleBump","roadBumpHole"]

load(roadProfile);
dx = road_x(2) - road_x(1);               % spacial step for input data
dt = dx/v;                                % simulation time step
```

```
tspan = 0:dt:tf;                              % simulation time
z0 = road_z(1:tf*v/dx+1)*scale;          % road height
z0dot = [0 diff(z0)/dt];                      % road profile velocity
pos = v*tspan;                                    % position along the road
sys_d = c2d(sys_c,dt);                        % Discrete-time state-space model
for simulation

% Collect inputs/disturbances
u_OL = zeros(length(tspan),nu+nd);            % Zero force
u_OL(:,2) = z0dot;                            % Rate of change of road height
u_OL(:,3) = z0;                               % Road height

if false % don't re-run open loop if false
% Simulate open-loop system
[y_OL,~,x_OL] = lsim(sys_d,u_OL,tspan,x0);
% Plot all simulation data
plotActiveSuspension(tspan,u_OL,x_OL,y_OL,bounds)
% Animate simulation data (can comment out if you dont want to animate)
% animateActiveSuspension(t,u_OL,y_OL,road_x,road_z,pos,scale)
sgtitle(strcat('Simulation Results: ', ...
    'Open Loop ', ' Profile = ', roadProfile))
saveas(gcf,strcat(fig_subfolder,filesep,'results_','open_',roadProfile,'.png'
))
% close all
end
```

Discrete-time LQR Control

LQR cost function matrices (DESIGN: prioritizing state minimization)

```
C_accel = diag([0,0,1])*C;
Q_lqr_accel = C_accel'*C_accel; %<--- acceleration component
Q_lqr_accel = 1e5*Q_lqr_accel./max(abs(Q_lqr_accel),[],'all'); %<---
normalized
Q_lqr_bounds = 1e3*diag(1./bounds.x_ub); % <--- state bounds
R_lqr_bounds = diag(1./bounds.u_ub); % <--- input


for test = ["lqr_accel", "lqr_bounds","lqr_both"]
   switch test
       case 'lqr_accel'
           Q = Q_lqr_accel; R = R_lqr_bounds;
       case 'lqr_bounds'
           Q = Q_lqr_bounds; R = R_lqr_bounds;
       case 'lqr_both'
           Q = Q_lqr_accel + Q_lqr_bounds; R = R_lqr_bounds;
   end

% LQR design using only the controllable input (force)
[K_LQR,~,~] = dlqr(sys_d.A,sys_d.B(:,1),Q,R);
K_LQR = -K_LQR; % For positive u = Kx

% Closed-loop system under LQR
```

```
sys_LQR =
ss(sys_d.A+sys_d.B(:,1)*K_LQR,sys_d.B(:,2:end),sys_d.C+sys_d.D(:,1)*K_LQR,sys
_d.D(:,2:end),dt);

save(strcat(data_subfolder,filesep,test),"Q","R","K_LQR","sys_LQR");

% Simulate open-loop system
[y_LQR,~,x_LQR] = lsim(sys_LQR,u_OL(:,2:end),tspan,x0);
u_LQR = u_OL;
u_LQR(:,1) = (K_LQR*x_LQR')';

if false % don't run LQR again if false
% Plot all simulation data
plotActiveSuspension(tspan,u_LQR,x_LQR,y_LQR,bounds)
sgtitle(strcat('Simulation Reusults:', test, 'profile = ',roadProfile))
saveas(gcf,strcat(fig_subfolder,filesep,'results_',test,'_',roadProfile,'.png
'))
% Animate simulation data (can comment out if you dont want to animate)
% animateActiveSuspension(t,u_LQR,y_LQR,road_x,road_z,pos,scale)
% close all
end
end
```

MPC Controller Design

Controller Design

```
dt_MPC = 3*dt; % Discrete-time step for MPC (must be an integer multiple of
dt)
N = 15; % Prediction horizon
rFlag = 1; % 0 = nominal, 1 = ...
dFlag = 2;
satFlag = 1;

% for dt_MPC = [1, 2, 3, 4, 5, 10].*dt
% for N = [2, 3, 5, 10, 15]

% Choose what the MPC controller knows (0 - no disturbance, 1 - current
% distrubance, 2 - full disturbance preview)
% dFlag = 0; % 0 = nominal, 1 = next distrubance, 2 = entire horizon

% for dFlag = [0,1,2]

% for satFlag = [0,1]
```

Robust things:

Reachability Analysis (Optional)

```
% Invariant Set (Optional)
```

```
% Controller and setup
K_nipotent = -acker(sys_d.A,sys_d.B(:,1),zeros(nx,1));
A_K = sys_d.A + sys_d.B(:,1)*K_nipotent; %<--- only u_1
W_rpi = sys_d.B(:,2)*(D_set.projection(1)); %<--- only d_1 inputs into state
equation

epsilon = 1;
Z = Approx_RPI(A_K,W_rpi,epsilon); Z.minHRep;

X_bar = X_set - Z; X_bar.minHRep;
U_bar = U_set - K_nipotent*Z; U_bar.minHRep;
```

Controller Design

```
if true
% mpc_test = 'mpc_bounded';
for test = "mpc_both"%["mpc_accel","mpc_bounds","mpc_both"]

% From LQR Design:switch test
switch test
   case 'mpc_accel'
       Q = Q_lqr_accel; R = R_lqr_bounds;
   case 'mpc_bounds'
       Q = Q_lqr_bounds; R = R_lqr_bounds;
   case 'mpc_both'
       Q = Q_lqr_accel + Q_lqr_bounds; R = R_lqr_bounds;
end
% P=0; %<--- no terminal cost
P = Q; %<--- same terminal cost (no final-state constraint)


mpc_test =
sprintf('%s_N=%d_dtMPC=%d_rFlag=%d_dFlag=%d_satFlag=%d',test,N,round(dt_MPC/
dt),rFlag,dFlag,satFlag);

% Resample discrete-time model with MPC time step
sys_MPC = d2d(sys_d, dt_MPC);
```

Controller Setup

```
if ~rFlag
    % YALMIP variables
    yalmip('clear'); clear('controller');
    x_ = sdpvar(repmat(nx,1,N+1),ones(1,N+1));
    u_ = sdpvar(repmat(nu,1,N),ones(1,N));
    d_ = sdpvar(repmat(nd,1,N),ones(1,N));

    % Time-evolution
    constraints = []; objective = 0;
    for k = 1:N
        % Cost Function
        objective = objective + x_{k}'*Q*x_{k} + u_{k}'*R*u_{k};
```

```
        % System Time-step Constraints
        constraints = [constraints, x_{k+1} == sys_MPC.A*x_{k} +
sys_MPC.B*[u_{k};d_{k}]];
    end
    objective = objective + x_{k+1}'*P*x_{k+1};
    constraints = [constraints, X_set.A*x_{k+1} <= X_set.b];

    % controller def
    controller
= optimizer(constraints,objective,sdpsettings('solver','gurobi'),
[x_(1)',d_(:)'],[u_{1}]);

else

    % YALMIP vars
    yalmip('clear'); clear('controller');
    x_bar_ = sdpvar(repmat(nx,1,N+1),ones(1,N+1));
    u_bar_ = sdpvar(repmat(nu,1,N),ones(1,N));
    x_1 = sdpvar(nx,1);
    u_1 = sdpvar(nu,1);
    d_ = sdpvar(repmat(nd,1,N),ones(1,N));
    d_bar_ = sdpvar(repmat(nd,1,N),ones(1,N));

    constraints = []; objective = 0;
    constraints = [constraints, Z.A*(x_1 - x_bar_{1}) <= Z.b];
    % constraints = [constraints, Z.A*x_bar_{1} <= Z.b]; %<-- initial
condition constraint
    for k = 1:N
        objective = objective + x_bar_{k}'*Q*x_bar_{k} +
u_bar_{k}'*R*u_bar_{k};
        constraints = [constraints, X_bar.A*x_bar_{k} <= X_bar.b];
        constraints = [constraints, U_bar.A*u_bar_{k} <= U_bar.b];
        constraints = [constraints, x_bar_{k+1} == ...
            sys_MPC.A*x_bar_{k} + sys_MPC.B(:,1)*u_bar_{k} ...
            + sys_MPC.B(:,2)*d_bar_{k}(1)];

        switch dFlag
            case 0
                constraints = [constraints, D_set.A*d_bar_{k} <= D_set.b];
            case {1,2}
                constraints = [constraints, d_bar_{k} == d_{k}];
        end
    end
    % constraints = [constraints, Z.A*(x_bar_{k+1}+0)<= Z.b];
    constraints = [constraints, x_bar_{k+1} == zeros(size(x_bar_{k+1}))];
    objective = objective + x_bar_{k+1}'*P*x_bar_{k+1};


    opts = sdpsettings;
    % controller = optimizer(constraints,objective,opts,[{x_1},d_(:)'], u_1);
    controller = optimizer(constraints,objective,opts,[{x_1},d_(:)'],
{u_bar_{1},x_bar_{1}});
end
```

Closed-loop MPC Simulation

```
z_all = [z0dot;z0];

[X,Y,U] = run_sim(sys_d, z_all, controller, x0, tspan, N, dt_MPC, dFlag,
rFlag, satFlag,bounds);


% Plot all simulation data
plotActiveSuspension(tspan(1:end-1),U',X',Y',bounds)
sgtitle(strcat('Simulation Reusults:', mpc_test,...
    'profile = ',roadProfile))
saveas(gcf,strcat(fig_subfolder,filesep,'results_',mpc_test,...
    '_',roadProfile,'.png'))
close all

end
end
% end
% end
% end
% end
```

Local functions

```
function [X,Y,U] = run_sim(sys, d_all, controller, x0, tspan, N, dt_MPC,
dFlag, rFlag, satFlag, bounds)

    [A,B,C,D] = ssdata(sys);
    nd = size(d_all,1);
    nx = size(A,1); nu = size(B,2) - nd;
    dt = tspan(2)-tspan(1);
    d_all = [d_all, d_all(:,1:N)];%<--- assuming restart...

    if rFlag; K = - acker(A,B(:,1),zeros(nx,1)); end

    X_{1} = x0;
    for k = 1:length(tspan)-1
        if mod(k-1,dt_MPC/dt) < eps
            switch dFlag
                case 0
                    V = mat2cell(zeros(nd,N),nd,ones(1,N));
                case 1
                    d_k = d_all(:,k);
                    d(1,:) = repmat(d_k(1),1,N);
                    d(2,:) = d_k(2) + (0:N-1).*d_k(1).*dt_MPC;
                    V = mat2cell(d,nd,ones(1,N));
                case 2
                    V = mat2cell(d_all(:,k + (1:N)),nd,ones(1,N));
            end
            results = controller{X_{k},V{:}};
            switch rFlag
                case 0
```

```
                    u = results{1};
                case 1
                    u_bar = results{1}; x_bar = results{2};
            end

            % [u,diagnostics] = controller{X_{k},V{:}};
            % if diagnostics ~= 0; error('not feasible'); end
        end
        if rFlag; u = + K*(X_{k} - x_bar); end
        if satFlag; u = min(bounds.u_ub, max(bounds.u_lb,u)); end
        U_{k} = [u; d_all(:,k)];
        Y_{k} = C*X_{k} + D*U_{k};
        X_{k+1} = A*X_{k} + B*U_{k};
    end
    U = [U_{:}];
    Y = [Y_{:}];
    X = [X_{1:end-1}];

end



    %
    % N = length(V_) - N_sim;
    % nd = size(V_{1},1);
    % X_{N_sim+1} = []; U_{N_sim} = []; diagnostics_{N_sim} = [];
    % X_{1} = x0;
    % for k = 1:N_sim
    %     switch dFlag
    %         case 0
    %             V = mat2cell(zeros(nd,N),nd,ones(1,N));
    %         case 1
    %             V = mat2cell([V_{k},zeros(nd,N-1)],nd,ones(1,N));
    %         case 2
    %             V = V_(k+(1:N));
    %     end
    %     [U_{k},diagnostics_{k}] = controller{X_{k},V};
    %     U_{k} = 0;%<--- open-loop test
    %     X_{k+1} = sys.A*X_{k} + sys.B*[U_{k}; V_{k}];
    % end
    % X = [X_{1:N_sim}]; U = [U_{:};V_{1:N_sim}];
% end
```

Warning: Error updating Text.

String scalar or character vector must have valid interpreter syntax:
Simulation Reusults:mpc_both_N=15_dtMPC=3_rFlag=1_dFlag=2_satFlag=1profile=roadRamps

Published with MATLAB® R2023b