# 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*