PD with Gravity Compensation Controller

Table of Contents

RBE 502 Fall 2018

Homework 5

November 21, 2018

```
function dx = PDControlGravity(t,x,measurements,params)

% Getting K_P and K_D as a positive definite, symmetric, diagonal
 matrix
% from the parameters:
K_P = params{1};
K_D = params{2};

% Getting the final desired state
xf = params{3};

% Measurements of the 2-link arm
I1 = measurements{1};
I2 = measurements{2};
m1 = measurements{3};
m2 = measurements{4};
l1 = measurements{5};
l2 = measurements{6};
r1 = measurements{7};
r2 = measurements{8};
g = measurements{9};

a = I1+I2+m1*r1^2+ m2*(l1^2+ r2^2);
b = m2*l1*r2;
d = I2+ m2*r2^2;
% the actual dynamic model of the system:
Mmat = [a+2*b*cos(x(2)), d+b*cos(x(2));  d+b*cos(x(2)), d];
Cmat = [-b*sin(x(2))*x(4), -b*sin(x(2))*(x(3)+x(4));
 b*sin(x(2))*x(3),0];
Gmat =  [m1*g*r1*cos(x(1))+m2*g*(l1*cos(x(1))+r2*cos(x(1)+x(2)));
m2*g*r2*cos(x(1)+x(2))];
```

```
invM = inv(Mmat);
invMC = invM*Cmat;


% Setting the K gain matrix
K = [-1*K_P -1*K_D];


% Error vector is the difference between current state and desired
 final
% state. So, e = [(q - q_desired) (q_dot - q_dot_desired)]^T
e = x' - xf;


% Input controller u for a PD controller for set point tracking and
 gravity compensation is
% u = -K_P(q - q_desired)-K_D(q_dot - q_dot_desired) + N(q)
%   = [-K_P -K_D]*e + N(q)
u = (K*e') + Gmat;


% From state-space representation, dx = [q_dot q_double_dot]^T
dx = zeros(4,1);

% q_dot can be taken from the current state variable x
q_dot = x(3:4);
dx(1:2) = q_dot;

% q_double_dot comes from the dynamics of the arm and setting the
 torque as
% the input controller u for PD Control with gravity compensation
q_double_dot = invM * (u - Cmat*q_dot - Gmat);
dx(3:4) = q_double_dot;

end
```

Published with MATLAB® R2018a

Iterative Learning Controller

Table of Contents

RBE 502 Fall 2018

Homework 5

November 21, 2018

```
function dxdt = ILControl(t,x)
    % *************** VARIABLE DEFINITION ***************
    persistent Compensation % Variable to store the iteratively
 estimated gravity term
    if isempty(Compensation) % Fill with zeros initially, this
 happends one time per program execution
        Compensation = zeros(2,1);
    end

    % System properties
    I1=10;  I2 = 10;
    m1=5; r1=.5; m2=5; r2=.5; l1=1; l2=1;
    g = 9.8;

    % we compute the parameters in the dynamic model
    a = I1+I2+m1*r1^2+ m2*(l1^2+ r2^2);
    b = m2*l1*r2;
    d = I2+ m2*r2^2;

    symx= sym('symx',[4,1]);

    M = [a+2*b*cos(x(2)), d+b*cos(x(2));
        d+b*cos(x(2)), d];
    C = [-b*sin(x(2))*x(4), -b*sin(x(2))*(x(3)+x(4));
 b*sin(x(2))*x(3),0];

    invM = inv(M);
    invMC= inv(M)*C;

    % PD controller gains
    Kp1 = 30.4; Kp2 = 30; Kd1 = 145; Kd2 = 145; % works ok

    % *************** COMPUTATIONS ***************
```

```
    A_Mat = [0 0 1 0;
             0 0 0 1;
             0 0 0 0;
             0 0 0 0];

    A_Mat(3:4,3:4) = -double(invMC);
    %A_Mat(3:4,3:4) = double(invMC);

    Gain_Mat = [Kp1  0   Kd1  0;
                 0   Kp2  0   Kd2];

    % Joint torques from PD controller
    U_Mat = -Gain_Mat * x;

    B_Mat = [0 0;
             0 0;
             invM];

    % The torque resulting from the gravity
    G = [m1*g*r1*cos(x(1))+m2*g*(l1*cos(x(1))+r2*cos(x(1)+x(2)));
         m2*g*r2*cos(x(1)+x(2))];

    tresh = 0.0001; % using to check joint velocities
    beta = 0.10; % Condition: 0 < beta < 0.5

    U_Net_Mat = (1/beta)*U_Mat + Compensation - G; % Include
 compensation and gravity

    % Update the known gravity term if joint velocities are zero
    if(abs(x(3)) < tresh && abs(x(4)) < tresh)
        Compensation =  -(1/beta)*[Kp1; Kp2].*[x(1); x(2)] +
 Compensation;
        U_Net_Mat = Compensation - G;
    end

    dxdt = zeros(4,1);
    dxdt = A_Mat*x + B_Mat*U_Net_Mat; % update new states
end
```

Published with MATLAB® R2018a

Inverse Dynamics Controller

Table of Contents

RBE 502 Fall 2018

Homework 5

November 21, 2018

```
function dxdt = inverseDC(t,x,system,params_inverseDC)

%params should include: a1, a2 (trajectory parameters), kp and kd,
%m1,m2,I1,I2,l1,l2,r1,r2 (system parameters);
%This function is taking the system and the controller parameters as
 input
%and computes the inverse dynamics controller that should be fed to
 the
%system and based on that it gets the first order state vector to be
 sloved
%by the ode45 Function.
```

Parameters of the trajectory and the controller

a1 is the coefficients of the trajectory generated for theta1

```
a1 = params_inverseDC{1};
% a2 is the coefficients of the trajectory generated for theta2
a2 = params_inverseDC{2};

% Defining Lambda as a positive definite, symmetric, diagonal matrix
% with arbitrarily chosen values along the diagonal:
kp = params_inverseDC{3};

% Defining Kd as a positive definite, symmetric, diagonal matrix
% with arbitrarily chosen values along the diagonal:
kd = params_inverseDC{4};
```

Desired Trajectory setting up

```
vec_t = [1; t; t^2; t^3];          % cubic polynomials
theta_d= [a1'*vec_t; a2'*vec_t]    % Desired Position



% compute the coffecients velocity and acceleration in both theta 1
 and theta2.
 a1_vel = [a1(2), 2*a1(3), 3*a1(4), 0];
 a1_acc = [2*a1(3), 6*a1(4),0,0 ];
 a2_vel = [a2(2), 2*a2(3), 3*a2(4), 0];
 a2_acc = [2*a2(3), 6*a2(4),0,0 ];


 % compute the desired trajectory (assuming 3rd order polynomials for
 trajectories)
 dtheta_d =[a1_vel*vec_t; a2_vel* vec_t];   %Desired Velocity
 ddtheta_d =[a1_acc*vec_t; a2_acc* vec_t];  %Desired Acceleration
```

Two Link Manipulator System Dynamic Model

```
 % Measurements of the 2-link arm
I1 = system{1};
I2 = system{2};
m1 = system{3};
m2 = system{4};
l1 = system{5};
l2 = system{6};
r1 = system{7};
r2 = system{8};
g =  system{9};

a = I1+I2+m1*r1^2+ m2*(l1^2+ r2^2);
b = m2*l1*r2;
d = I2+ m2*r2^2;

 % the actual dynamic model of the system:
 M = [a+2*b*cos(x(2)), d+b*cos(x(2)); d+b*cos(x(2)), d] %Inertia
 Matrix
 C = [-b*sin(x(2))*x(4), -b*sin(x(2))*(x(3)+x(4));
 b*sin(x(2))*x(3),0] %Coriolos Materix
 G = [m1*g*r1*cos(x(1))+m2*g*(l1*cos(x(1))+r2*cos(x(1)+x(2)));
        m2*g*r2*cos(x(1)+x(2))];

invM = inv(M);
invMC= inv(M)*C;
```

defining the controller

```
 e=[x(1)-theta_d(1);x(2)-theta_d(2)];         %position error vector
 e_dot=[x(3)-dtheta_d(1);x(4)-dtheta_d(2)];   %Velocity error vector
```

```
 aq=ddtheta_d-kp*e-kd*e_dot;                       %Virtual Controller
 u=M*aq+C*[x(3);x(4)];                            %Original Controller
 xdd=invM*u-invM*C*[x(3);x(4)];

% Defining the first order state vector.
dxdt=zeros(4,1);
dxdt(1)=x(3);
dxdt(2)=x(4);
dxdt(3)= xdd(1,:);
dxdt(4)= xdd(2,:);

end
```

Published with MATLAB® R2018a

Lyapanuv Based Controller

Table of Contents

RBE 502 Fall 2018

Homework 5

November 21, 2018

```
function dydt = LyapanuvBased(t,y,system,params_LyapanuvCtrl)
```

```
%params should include: a1, a2 (trajectory parameters), lambda and Kd
%m1,m2,I1,I2,l1,l2,r1,r2 (system parameters);
%This function is taking the system and the controller parameters as
 input
%and computes the Lyapanuv based controller that should be fed to the
%system and based on that it gets the first order state vector to be
 sloved
%by the ode45 Function.
```

Parameters of the trajectory and the controller

a1 is the coefficients of the trajectory generated for theta1

```
a1 = params_LyapanuvCtrl{1};
% a2 is the coefficients of the trajectory generated for theta2
a2 = params_LyapanuvCtrl{2};

% Defining Lambda as a positive definite, symmetric, diagonal matrix
% with arbitrarily chosen values along the diagonal:
lambda_L = params_LyapanuvCtrl{3};

% Defining Kd as a positive definite, symmetric, diagonal matrix
% with arbitrarily chosen values along the diagonal:
Kd = params_LyapanuvCtrl{4};
```

Desired Trajectory setting up

```
vec_t = [1; t; t^2; t^3];          % cubic polynomials
```

```
theta_d= [a1'*vec_t; a2'*vec_t]    % Desired Position



% compute the coffecients velocity and acceleration in both theta 1
 and theta2.
 a1_vel = [a1(2), 2*a1(3), 3*a1(4), 0];
 a1_acc = [2*a1(3), 6*a1(4),0,0 ];
 a2_vel = [a2(2), 2*a2(3), 3*a2(4), 0];
 a2_acc = [2*a2(3), 6*a2(4),0,0 ];


 % compute the desired trajectory (assuming 3rd order polynomials for
 trajectories)
 dtheta_d =[a1_vel*vec_t; a2_vel* vec_t];   %Desired Velocity
 ddtheta_d =[a1_acc*vec_t; a2_acc* vec_t];  %Desired Acceleration
```

Two Link Manipulator System Dynamic Model

```
 % Measurements of the 2-link arm
I1 = system{1};
I2 = system{2};
m1 = system{3};
m2 = system{4};
l1 = system{5};
l2 = system{6};
r1 = system{7};
r2 = system{8};
g = system{9};

a = I1+I2+m1*r1^2+ m2*(l1^2+ r2^2);
b = m2*l1*r2;
d = I2+ m2*r2^2;

 % the actual dynamic model of the system:
 M = [a+2*b*cos(y(2)), d+b*cos(y(2)); d+b*cos(y(2)), d] %Inertia
 Matrix
 C = [-b*sin(y(2))*y(4), -b*sin(y(2))*(y(3)+y(4));
 b*sin(y(2))*y(3),0] %Coriolos Materix
 G = [m1*g*r1*cos(y(1))+m2*g*(l1*cos(y(1))+r2*cos(y(1)+y(2)));
        m2*g*r2*cos(y(1)+y(2))];

invM = inv(M);
invMC= inv(M)*C;
```

Defining the controller

```
e=[y(1)-theta_d(1);y(2)-theta_d(2)];         %position error vector
e_dot=[y(3)-dtheta_d(1);y(4)-dtheta_d(2)];   %Velocity error vector

ksi_d=dtheta_d - lambda_L*(e)                %ksi dot parameter
ksi_dd = ddtheta_d-(lambda_L*e_dot)          %ksi double dot parameter
sigma = e_dot+(lambda_L*e);                  %Sigma Paramter
```

```
u=M*(ksi_dd)+C*(ksi_d)-Kd*(sigma)+G          %Controller Definition

ydd=invM*u-invMC*[y(3);y(4)]-invM*G          %Feeding the controller
 into the 2nd order system/

 % Defining the first order state vector.

dydt=zeros(4,1);   %initialization of first order ode vector%
dydt(1)=y(3);
dydt(2)=y(4);
dydt(3)= ydd(1,:);
dydt(4)= ydd(2,:);

end
```

Published with MATLAB® R2018a

Passivity Based Controller

Table of Contents

RBE 502 Fall 2018

Homework 5

November 21, 2018

```
function [ dx ] = passivityCtrl( t,x, system, params  )
%params should include: a1, a2 (trajectory parameters), lambda and kv,
%m1,m2,I1,I2,l1,l2,r1,r2 (system parameters);
%This function is taking the system and the controller parameters as
 input
%and computes the Passivity based controller that should be fed to the
%system and based on that it gets the first order state vector to be
 sloved
%by the ode45 Function.

% Getting parameters of the 2-link manipulator system
% a1 is the coefficients of the trajectory generated for theta1
a1 = params{1};
% a2 is the coefficients of the trajectory generated for theta2
a2 = params{2};

% Defining Lambda as a positive definite, symmetric, diagonal matrix
% with arbitrarily chosen values along the diagonal:
Lambda = params{3};

% Defining K_v as a positive definite, symmetric, diagonal matrix
% with arbitrarily chosen values along the diagonal:
K_v = params{4};

% The previous q_double_dot is saved as a parameter
q_double_dot_previous = params{5};


% q_dot comes from the current state variable
q_dot = x(3:4);

vec_t = [1; t; t^2; t^3]; % cubic polynomials
theta_d= [a1'*vec_t; a2'*vec_t];
```

```
%ref = [ref,theta_d];
% compute the velocity and acceleration in both theta 1 and theta2.
a1_vel = [a1(2), 2*a1(3), 3*a1(4), 0];
a1_acc = [2*a1(3), 6*a1(4),0,0 ];
a2_vel = [a2(2), 2*a2(3), 3*a2(4), 0];
a2_acc = [2*a2(3), 6*a2(4),0,0 ];

% compute the desired trajectory (assuming 3rd order polynomials for
 trajectories)
dtheta_d =[a1_vel*vec_t; a2_vel* vec_t];
ddtheta_d =[a1_acc*vec_t; a2_acc* vec_t];
theta= x(1:2,1);
dtheta= x(3:4,1);

% Measurements of the 2-link arm
I1 = system{1};
I2 = system{2};
m1 = system{3};
m2 = system{4};
l1 = system{5};
l2 = system{6};
r1 = system{7};
r2 = system{8};
g = system{9};

a = I1+I2+m1*r1^2+ m2*(l1^2+ r2^2);
b = m2*l1*r2;
d = I2+ m2*r2^2;
% the actual dynamic model of the system:
Mmat = [a+2*b*cos(x(2)), d+b*cos(x(2));  d+b*cos(x(2)), d];
Cmat = [-b*sin(x(2))*x(4), -b*sin(x(2))*(x(3)+x(4));
 b*sin(x(2))*x(3),0];
Gmat =  [m1*g*r1*cos(x(1))+m2*g*(l1*cos(x(1))+r2*cos(x(1)+x(2)));
m2*g*r2*cos(x(1)+x(2))];
invM = inv(Mmat);
invMC = invM*Cmat;

% TODO: compute the control input for the system, which
% should provide the torques

% use the computed torque and state space model to compute
% the increment in state vector.
%TODO: compute dx = f(x,u) hint dx(1)=x(3); dx(2)= x(4); the rest
%of which depends on the dynamic model of the robot.

% Computing the error between the current position, velocity, and
% acceleration with the desired position, velocity, and acceleration
e = theta - theta_d;
e_dot = dtheta - dtheta_d;
e_double_dot = q_double_dot_previous - ddtheta_d;

% Set r = sigma = e_dot + Lambda*e
r = e_dot + Lambda*e;
r_dot = e_double_dot + Lambda*e_dot;
```

```
% Setting inputs to a and v for the controller
a = q_double_dot_previous - r_dot;
v = q_dot - r;

% Input controller for Passivity-based controller
u = Mmat*a + Cmat*v + Gmat - K_v*r;

% From state-space representation, dx = [q_dot q_double_dot]^T
dx = zeros(4,1);

% q_dot can be taken from the current state variable x
dx(1:2) = q_dot;

% q_double_dot comes from the dynamics of the arm and setting the
 torque as
% the input controller u for Passivity-based controller
q_double_dot = invM * (u - Cmat*q_dot - Gmat);
dx(3:4) = q_double_dot;

% Saving the current value of q_double_dot as a parameter to be used
 again
% the the next call of this ode function for passivity control
params{5} = q_double_dot;

%disp("Previous q_double_dot: ");
%disp(params{3});

end
```

Published with MATLAB® R2018a