

Simulation Project 1

EML 6351

by

Aahlad Vanam

13681791

Filtered Torque:

Where u is the controller input.

In this case u =

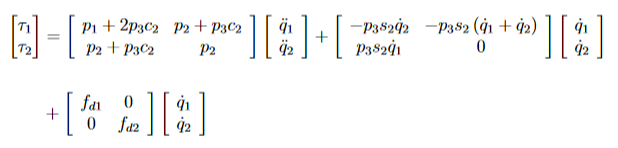
f ̇̇= f +

Solving numerically in MATLAB we get filtered Torque

Filtered regression matrix

Filtered desired regression matrix

3.

(a)Dynamic model: 

This is in the form of  = m(q)q̈ +Vm (q,q̇)q̇+Fdq̇ -----------------------(1)

(b) Problem definition and open-loop error system development:

Open Loop error system: e = q – qd

(c) Control design (including adaptive update law) and closed-loop error system development:

m(q)ṙ = m(q)e ̈ + m(q)αe ̇

= m(q)q̈ - m(q)q̈d +m(q) αe ̇

From (1)

m(q)ṙ =  - m(q)q̈d - Vm (q,q̇)q̇ - Fdq̇ + m(q) αe

r = e ̇+ αe

ṙ = e ̈ + αe ̇

* = -Kr - Y - e

Closed loop error system

r ̇ = m

Adaptive Update law for

Standard Adaptive controller:

Composite Adaptive controller(gradient):

Composite Adaptive controller (least squares):

(d) Stability analysis of each controller

Standard Adaptive Controller:

V = ½ rTm(q)r + ½ + ½ eTe

= rm(q)ṙ– +½ rTṁ(q)r

= m r) + ½ rTṁ(q)r

=

=

*Stability result:*

*Composite Adaptive Controller(gradient):*

V = ½ rTm(q)r + ½ + ½ eTe

= rm(q)ṙ– +½ rTṁ(q)r

*Composite Adaptive Controller (least squares):*

V = ½ rTm(q)r + ½ + ½ eTe

= rm(q)ṙ– +½ rTṁ(q)r

*(e)Simulation:*

*Standard Adaptive Controller:*

*Gains Used:*

*K =5*

*a=1.5*

function twoLinkRobotAdaptive

close all

%Set up parameters for sim

p1 = 3.473;

p2 = 0.196;

p3 = 0.242;

f1 = 5.3;

f2 = 1.1;

% Stacked parameter vector

theta = [p1;p2;p3;f1;f2];

% Simulation final time

tf = 100;

% Initial condition vector (X0 must be same size and "form" as X and XDot below)

% (i.e., in this sim, X0 = [e0;r0;thetahat0])

X0 = [4;10;3;2;1;1;1;1;1];

% Options for integration function

opts = odeset('RelTol',1e-3,'AbsTol',1e-3);

% Integrate (you can send the paramters theta to the dynamics as seen below)

[t,STATES] = ode45(@(t,X) twoLinkdynamics(t,X,theta),[0 tf],X0,opts);

% Set up desired trajectory data for plots (enter desired trajectory for your simulation)

qd = [cos(0.5\*t) 2\*cos(t)]';

% Parse integrated states (STATES is the same "form" as X0)

% (i.e., in this sim, STATES = [e r thetahat] over all time);

e = STATES(:,1:2)';

r = STATES(:,3:4)';

thetaHat = STATES(:,5:9)';

% Compute x from e and xd for plotting purposes

q = e + qd;

% Plot the actual vs desired trajectories

figure(1)

plot(t,qd,'-','LineWidth',2)

title('Actual and Desired Trajectories');

xlabel('time');

ylabel('Amplitude');

legend();

hold on

ax = gca;

ax.ColorOrderIndex = 1;

plot(t,q,':','LineWidth',2)

hold off

% Plot the filtered tracking error

figure(2)

plot(t,r,'--','LineWidth',2)

title('Filtered Tracking error')

xlabel('time');

ylabel('Amplitude');

% Plot the adaptive estimates vs actual parameters

figure(3)

plot(t,repmat(theta,1,length(t)),'-','LineWidth',2)

hold on

ax = gca;

ax.ColorOrderIndex = 1;

plot(t,thetaHat,':','LineWidth',2)

title('Adaptive Estimates & Actual Parameters')

xlabel('time');

ylabel('Amplitude');

hold off

function [XDot] = twoLinkdynamics(t,X,theta)

% Parse parameter vector

p1 = theta(1);

p2 = theta(2);

p3 = theta(3);

f1 = theta(4);

f2 = theta(5);

% Select gains for controller

K = 5; %Enter a number

a = 1.5; %Enter a number

% Desired trajectory and needed derivatives

qd = [cos(0.5\*t);2\*cos(t)]

qdDot = [-0.5\*sin(0.5\*t);-2\*sin(t)] %Enter the expression

qdDotDot = [-0.25\*cos(0.5\*t);-2\*cos(t)] %Enter the expression

% Parse current states (X is the same size and "form" as X0)

% (i.e., in this sim, X = [e;r;thetahat])

e = [X(1);X(2)];

r = [X(3);X(4)];

thetaHat = [X(5);X(6);X(7);X(8);X(9)];

% Compute current x and xDot for convenience

q = e + qd;

qDot = r - a\*e + qdDot;

% Compute cos(x2) and sin(x2) for convenience

c2 = cos(q(2));

s2 = sin(q(2));

% Compute current matrices for the dynamics

M = [p1 + 2\*p3\*c2 p2 + p3\*c2;p2 + p3\*c2 p2];

Vm = [-p3\*s2\*qDot(2) -p3\*s2\*(qDot(1) + qDot(2));p3\*s2\*qDot(1) 0];

fd = [f1 0;0 f2];

% Compute current regression matrix

y11 = -qdDotDot(1)+a\*(qDot(1)-qdDot(1)); %Enter the expression

y12 = -qdDotDot(2)+a\*(qDot(1)-qdDot(2)); %Enter the expression

y13 = s2\*qDot(2)\*qdDot(1) + s2\*qDot(1)\*qdDot(2) + s2\*qDot(2)\*qdDot(2) + a\*s2\*qDot(2)\*e(1) + a\*s2\*qDot(1)\*e(2) + a\*s2\*qDot(2)\*e(2) - 2\*c2\*qdDotDot(1) - c2\*qdDotDot(2) + 2\*a\*c2\*(qDot(1)-qdDot(1)) + a\*c2\*(qDot(2)-qdDot(2)); %Enter the expression

y14 = -qDot(1);

y15 = 0;

y21 = 0; %Enter the expression

y22 = -qdDotDot(1) - qdDotDot(2) + a\*(qDot(1)-qdDot(1)) + a\*(qDot(2)-qdDot(2)); %Enter the expression

y23 = -s2\*qDot(1)\*qdDot(1) - a\*s2\*qDot(1)\*e(1) - c2\*qdDotDot(1) + a\*c2\*(qDot(1)-qdDot(1)) ; %Enter the expression

y24 = 0;

y25 = -qDot(2);

Y = [y11 y12 y13 y14 y15;y21 y22 y23 y24 y25];

% Design controller

u = -K\*r-Y\*thetaHat-e; %Enter the expression

% Compute current closed-loop dynamics

eDot = qDot - qdDot;

rDot = M\(-Vm\*qDot-fd\*qDot+u-M\*qdDotDot+M\*a\*eDot); %Enter the expression

thetaHatDot = 10\*transpose(Y)\*r; %Enter the expression

% Stacked dynamics vector (XDot is the same size and "form" as X)

XDot = [eDot;rDot;thetaHatDot];



*Composite Adaptive Controller(gradient)*

*Gains used:*

K = 10

a = 2.5

gamma = 6

beta = 2

simulation:

function twoLinkRobotAdaptive

close all

%Set up parameters for sim

p1 = 3.473;

p2 = 0.196;

p3 = 0.242;

f1 = 5.3;

f2 = 1.1;

% Stacked parameter vector

theta = [p1;p2;p3;f1;f2];

% Simulation final time

tf = 100;

% Initial condition vector (X0 must be same size and "form" as X and XDot below)

% (i.e., in this sim, X0 = [e0;r0;thetahat0])

X0 = [4;10;3;2;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1];

% Options for integration function

opts = odeset('RelTol',1e-3,'AbsTol',1e-3);

% Integrate (you can send the paramters theta to the dynamics as seen below)

[t,STATES] = ode45(@(t,X) twoLinkdynamics(t,X,theta),[0 tf],X0,opts);

% Set up desired trajectory data for plots (enter desired trajectory for your simulation)

qd = [cos(0.5\*t) 2\*cos(t)]';

% Parse integrated states (STATES is the same "form" as X0)

% (i.e., in this sim, STATES = [e r thetahat] over all time);

e = STATES(:,1:2)';

r = STATES(:,3:4)';

thetaHat = STATES(:,5:9)';

uf = STATES(:,10:11)';

Ydf = STATES(:,12:13:14:15:16:17:18:19:20:21)';

% Compute x from e and xd for plotting purposes

q = e + qd;

% Plot the actual vs desired trajectories

figure(1)

plot(t,qd,'-','LineWidth',2)

title('Actual and Desired Trajectories');

hold on

ax = gca;

ax.ColorOrderIndex = 1;

plot(t,q,':','LineWidth',2)

hold off

% Plot the filtered tracking error

figure(2)

plot(t,r,'--','LineWidth',2)

title('Filtered Tracking error')

% Plot the adaptive estimates vs actual parameters

figure(3)

plot(t,repmat(theta,1,length(t)),'-','LineWidth',2)

title('Actual Parameters')

hold on

ax = gca;

ax.ColorOrderIndex = 1;

plot(t,thetaHat,':','LineWidth',2)

title('Adaptive Estimates')

hold off

function [XDot] = twoLinkdynamics(t,X,theta)

% Parse parameter vector

p1 = theta(1);

p2 = theta(2);

p3 = theta(3);

f1 = theta(4);

f2 = theta(5);

% Select gains for controller

K = 10; %Enter a number

a = 2.5; %Enter a number

gamma = 6;

beta = 2;

% Desired trajectory and needed derivatives

qd = [cos(0.5\*t);2\*cos(t)];

qdDot = [-0.5\*sin(0.5\*t);-2\*sin(t)]; %Enter the expression

qdDotDot = [-0.25\*cos(0.5\*t);-2\*cos(t)]; %Enter the expression

% Parse current states (X is the same size and "form" as X0)

% (i.e., in this sim, X = [e;r;thetahat])

e = [X(1);X(2)];

r = [X(3);X(4)];

thetaHat = [X(5);X(6);X(7);X(8);X(9)];

uf = [X(10);X(11)];

Ydf = [X(12) X(13) X(14) X(15) X(16);X(17) X(18) X(19) X(20) X(21)];

Ydf2 = [X(12);X(13);X(14);X(15);X(16);X(17);X(18);X(19);X(20);X(21)];

% Compute current x and xDot for convenience

q = e + qd;

qDot = r - a\*e + qdDot;

% Compute cos(x2) and sin(x2) for convenience

c2 = cos(q(2));

s2 = sin(q(2));

cd2 = cos(qd(2));

sd2 = sin(qd(2));

% Compute current matrices for the dynamics

M = [p1 + 2\*p3\*c2 p2 + p3\*c2;p2 + p3\*c2 p2];

Vm = [-p3\*s2\*qDot(2) -p3\*s2\*(qDot(1) + qDot(2));p3\*s2\*qDot(1) 0];

fd = [f1 0;0 f2];

% Compute current regression matrix

y11 = -qdDotDot(1)+a\*(qDot(1)-qdDot(1)); %Enter the expression

y12 = -qdDotDot(2)+a\*(qDot(1)-qdDot(2)); %Enter the expression

y13 = s2\*qDot(2)\*qdDot(1) + s2\*qDot(1)\*qdDot(2) + s2\*qDot(2)\*qdDot(2) + a\*s2\*qDot(2)\*e(1) + a\*s2\*qDot(1)\*e(2) + a\*s2\*qDot(2)\*e(2) - 2\*c2\*qdDotDot(1) - c2\*qdDotDot(2) + 2\*a\*c2\*(qDot(1)-qdDot(1)) + a\*c2\*(qDot(2)-qdDot(2)); %Enter the expression

y14 = -qDot(1);

y15 = 0;

y21 = 0; %Enter the expression

y22 = -qdDotDot(1) - qdDotDot(2) + a\*(qDot(1)-qdDot(1)) + a\*(qDot(2)-qdDot(2)); %Enter the expression

y23 = -s2\*qDot(1)\*qdDot(1) - a\*s2\*qDot(1)\*e(1) - c2\*qdDotDot(1) + a\*c2\*(qDot(1)-qdDot(1)) ; %Enter the expression

y24 = 0;

y25 = -qDot(2);

Y = [y11 y12 y13 y14 y15;y21 y22 y23 y24 y25];

%Design Yd Matrix

yd11 = qdDotDot(1);

yd12 = qdDotDot(2);

yd13 = 2\*cd2\*qdDotDot(1) + cd2\*qdDotDot(2) - sd2\*qdDot(2)\*qdDot(1) - sd2\*qdDot(1)\*qdDot(2) - sd2\*qdDot(2)\*qdDot(2);

yd14 = qdDot(1);

yd15 = 0;

yd21 = 0;

yd22 = qdDotDot(1)+qdDotDot(2);

yd23 = cd2\*qdDotDot(1) + sd2\*qdDot(1)\*qdDot(1);

yd24 = 0;

yd25 = qdDot(2);

Yd = [yd11 yd12 yd13 yd14 yd15;yd21 yd22 yd23 yd24 yd25];

Yd2 = [yd11; yd12; yd13; yd14; yd15; yd21; yd22; yd23; yd24; yd25];

%initialize the value

% Design controller

u = -K\*r-Y\*thetaHat-e; %Enter the expression

% Compute current closed-loop dynamics

eDot = qDot - qdDot;

epsilon = uf - Ydf\*thetaHat;

rDot = M\(-Vm\*qDot-fd\*qDot+u-M\*qdDotDot+M\*a\*eDot); %Enter the expression

thetaHatDot = (gamma\*r'\*Y)'+(gamma\*Ydf'\*epsilon); %Enter the expression

%Filtering

ufDot = -beta\*uf + beta\*u;

YdfDot = -beta\*Ydf2 + beta\*Yd2;

% Stacked dynamics vector (XDot is the same size and "form" as X)

XDot = [eDot;rDot;thetaHatDot;ufDot; YdfDot];





Composite Adaptive Controller ( Least Squares)

Gains used:

K = 10

a = 1.5

gamma = 12;

beta = 0.6;

function twoLinkRobotAdaptive\_2c\_version2

close all

%Set up parameters for sim

p1 = 3.473;

p2 = 0.196;

p3 = 0.242;

f1 = 5.3;

f2 = 1.1;

i1 = 11;

i2 = 0.1;

% Stacked parameter vector

theta = [p1;p2;p3;f1;f2];

% Simulation final time

tf = 100;

% Initial condition vector (X0 must be same size and "form" as X and XDot below)

% (i.e., in this sim, X0 = [e0;r0;thetahat0])

X0 = [4;10;3;2;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1

i1;i2;i2;i2;i2;

i2;i1;i2;i2;i2;

i2;i2;i1;i2;i2;

i2;i2;i2;i1;i2;

i2;i2;i2;i2;i1;];

% Options for integration function

opts = odeset('RelTol',1e-3,'AbsTol',1e-3);

% Integrate (you can send the paramters theta to the dynamics as seen below)

[t,STATES] = ode45(@(t,X) twoLinkdynamics(t,X,theta),[0 tf],X0,opts);

% Set up desired trajectory data for plots (enter desired trajectory for your simulation)

qd = [cos(0.5\*t) 2\*cos(t)]';

% Parse integrated states (STATES is the same "form" as X0)

% (i.e., in this sim, STATES = [e r thetahat] over all time);

e = STATES(:,1:2)';

r = STATES(:,3:4)';

thetaHat = STATES(:,5:9)';

uf = STATES(:,10:11)';

Ydf = STATES(:,12:13:14:15:16:17:18:19:20:21)';

P = STATES(:,22:46)';

% Compute x from e and xd for plotting purposes

q = e + qd;

% Plot the actual vs desired trajectories

figure(1)

plot(t,qd,'-','LineWidth',2)

title('Actual and Desired Trajectories');

hold on

ax = gca;

ax.ColorOrderIndex = 1;

plot(t,q,':','LineWidth',2)

xlabel('time')

hold off

% Plot the filtered tracking error

figure(2)

plot(t,r,'--','LineWidth',2)

title('Filtered Tracking error');

xlabel('time');

% Plot the adaptive estimates vs actual parameters

figure(3)

plot(t,repmat(theta,1,length(t)),'-','LineWidth',2)

hold on

ax = gca;

ax.ColorOrderIndex = 1;

plot(t,thetaHat,':','LineWidth',2)

title('Adaptive Estimates & Actual Parameters');

xlabel('time');

hold off

function [XDot] = twoLinkdynamics(t,X,theta)

% Parse parameter vector

p1 = theta(1);

p2 = theta(2);

p3 = theta(3);

f1 = theta(4);

f2 = theta(5);

% Select gains for controller

K = 10; %Enter a number

a = 1.5; %Enter a number

gamma = 12;

beta = 0.6;

% Desired trajectory and needed derivatives

qd = [cos(0.5\*t);2\*cos(t)];

qdDot = [-0.5\*sin(0.5\*t);-2\*sin(t)]; %Enter the expression

qdDotDot = [-0.25\*cos(0.5\*t);-2\*cos(t)]; %Enter the expression

% Parse current states (X is the same size and "form" as X0)

% (i.e., in this sim, X = [e;r;thetahat])

e = [X(1);X(2)];

r = [X(3);X(4)];

thetaHat = [X(5);X(6);X(7);X(8);X(9)];

uf = [X(10);X(11)];

Ydf = [X(12) X(13) X(14) X(15) X(16);X(17) X(18) X(19) X(20) X(21)];

Ydf2 = [X(12);X(13);X(14);X(15);X(16);X(17);X(18);X(19);X(20);X(21)];

P = [X(22) X(23) X(24) X(25) X(26);X(27) X(28) X(29) X(30) X(31); X(32) X(33) X(34) X(35) X(36);X(37) X(38) X(39) X(40) X(41); X(42) X(43) X(44) X(45) X(46)];

P2 = [X(22); X(23); X(24); X(25); X(26); X(27); X(28); X(29); X(30); X(31); X(32); X(33); X(34); X(35); X(36); X(37); X(38); X(39); X(40); X(41); X(42); X(43); X(44); X(45); X(46)];

% Compute current x and xDot for convenience

q = e + qd;

qDot = r - a\*e + qdDot;

% Compute cos(x2) and sin(x2) for convenience

c2 = cos(q(2));

s2 = sin(q(2));

cd2 = cos(qd(2));

sd2 = sin(qd(2));

% Compute current matrices for the dynamics

M = [p1 + 2\*p3\*c2 p2 + p3\*c2;p2 + p3\*c2 p2];

Vm = [-p3\*s2\*qDot(2) -p3\*s2\*(qDot(1) + qDot(2));p3\*s2\*qDot(1) 0];

fd = [f1 0;0 f2];

% Compute current regression matrix

y11 = -qdDotDot(1)+a\*(qDot(1)-qdDot(1)); %Enter the expression

y12 = -qdDotDot(2)+a\*(qDot(1)-qdDot(2)); %Enter the expression

y13 = s2\*qDot(2)\*qdDot(1) + s2\*qDot(1)\*qdDot(2) + s2\*qDot(2)\*qdDot(2) + a\*s2\*qDot(2)\*e(1) + a\*s2\*qDot(1)\*e(2) + a\*s2\*qDot(2)\*e(2) - 2\*c2\*qdDotDot(1) - c2\*qdDotDot(2) + 2\*a\*c2\*(qDot(1)-qdDot(1)) + a\*c2\*(qDot(2)-qdDot(2)); %Enter the expression

y14 = -qDot(1);

y15 = 0;

y21 = 0; %Enter the expression

y22 = -qdDotDot(1) - qdDotDot(2) + a\*(qDot(1)-qdDot(1)) + a\*(qDot(2)-qdDot(2)); %Enter the expression

y23 = -s2\*qDot(1)\*qdDot(1) - a\*s2\*qDot(1)\*e(1) - c2\*qdDotDot(1) + a\*c2\*(qDot(1)-qdDot(1)) ; %Enter the expression

y24 = 0;

y25 = -qDot(2);

Y = [y11 y12 y13 y14 y15;y21 y22 y23 y24 y25];

%Design Yd Matrix

yd11 = qdDotDot(1);

yd12 = qdDotDot(2);

yd13 = 2\*cd2\*qdDotDot(1) + cd2\*qdDotDot(2) - sd2\*qdDot(2)\*qdDot(1) - sd2\*qdDot(1)\*qdDot(2) - sd2\*qdDot(2)\*qdDot(2);

yd14 = qdDot(1);

yd15 = 0;

yd21 = 0;

yd22 = qdDotDot(1)+qdDotDot(2);

yd23 = cd2\*qdDotDot(1) + sd2\*qdDot(1)\*qdDot(1);

yd24 = 0;

yd25 = qdDot(2);

Yd = [yd11 yd12 yd13 yd14 yd15;yd21 yd22 yd23 yd24 yd25];

Yd2 = [yd11; yd12; yd13; yd14; yd15; yd21; yd22; yd23; yd24; yd25];

k = 6;

j = 0.6;

P2 = [k j j j j;...

j k j j j;...

j j k j j;...

j j j k j;...

j j j j k];

P = P2(:);

%initialize the value

% Design controller

u = -K\*r-Y\*thetaHat-e; %Enter the expression

% Compute current closed-loop dynamics

eDot = r - a\*e;

epsilon = uf - Ydf\*thetaHat;

PDot = P2\*Y'\*Y\*P2';

PDot2 = PDot(:);

rDot = M\(-Vm\*qDot-fd\*qDot+u-M\*qdDotDot+M\*a\*eDot); %Enter the expression

thetaHatDot = P2\*(r'\*Y)'+P2\*(Ydf'\*epsilon); %Enter the expression

%Filtering

ufDot = -beta\*uf + beta\*u;

YdfDot = -beta\*Ydf2 + beta\*Yd2;

% Stacked dynamics vector (XDot is the same size and "form" as X)

XDot = [eDot;rDot;thetaHatDot;ufDot; YdfDot;PDot2];





(f)

2.Tracking error is not changing much among the controllers.

3. Performance of Adaptation: Adaptive estimators converge faster as we go from standard adaptive controller to composite adaptive controller with least squares adaptative law.

4.