# TwoLinkArm

## Table of Contents



This script simulates a non-planar two-link arm tracking a cubic polynomial trajectory with three different tracking controllers: inverse dynamic control, Lyapunov-based Control, and passivity-based control.

# Controller Remarks

When an initial error was introduced, it can be concluded that while all three controllers still converge to the desired trajectory.

Both the Lyapunov-based controller and passivity-based controller initially converged quicker than the inverse dynamic control, but then took longer to eliminate the small remaining error.

The inverse dynamic controller did not overshoot the desired trajectory and the Lyapunov-based controller and passivity-based controller did overshoot the desired trajectory when there was some initital error.

Clean up before running anything:

```
clc, clear all, close all
```

Initial state:

```
x0 = [-0.5,0.2,0.1,0.1];
```

final state:

```
xf = [0, 0, 0, 0];
```

Final time:

```
tf = 10;
```

Option for no figure for the trajectory generator (true/false):

```
nofigure = 1;
```

Compute the cubic polynomial coefficients:

```
a1 = TwoLinkArmTraj(x0(1), x0(3), xf(1), xf(3), tf, nofigure);
a2 = TwoLinkArmTraj(x0(2), x0(4), xf(2), xf(4), tf, nofigure);
```

Set the initial error state:

```
x0_error  = [-0.6,0.4,0.15,0.05];
```

Options for plotting each controller (true/false):

```
plot_inverseDC = true;
plot_lyapunov = true;
plot_passivity = true;
```

# Trajectory Generation For Plotting

Set time matrix for plotting:

```
time = 0:0.001:tf;
```

Initialize the trajectory maxtrix:

```
trajectory = zeros(length(time),2);
length(time);
for i = 1:length(time)
```

Grab the time value to use for each iteration:

```
    t = time(1,i);
```

Note *x* is in the form of *q1*, *q2*, *q1_dot*, *q2_dot*:

Cubic polynomials:

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

Save trajectory joint angles for each iteration:

```
    trajectory(i,:) = theta_d';

end

if plot_inverseDC
```

# Inverse Dynamic Control

Set the tolerance options for ODE45 function:

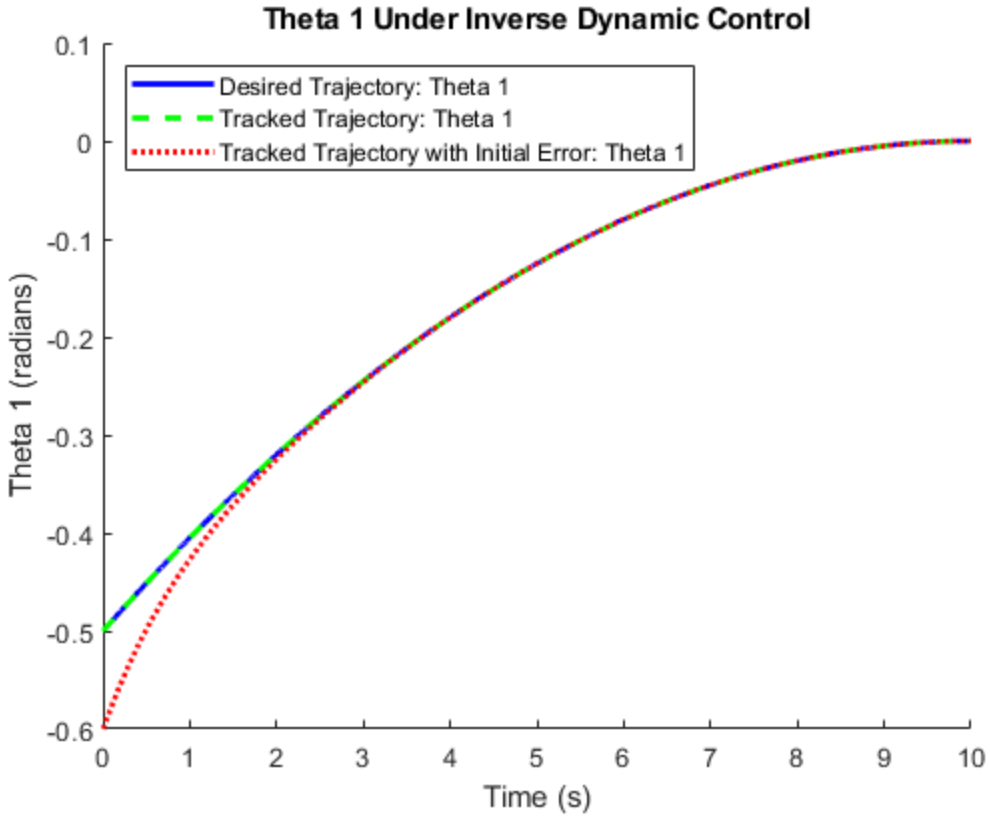
```
    options = odeset('RelTol',1e-4,'AbsTol',[1e-4, 1e-4, 1e-4, 1e-4]);
```

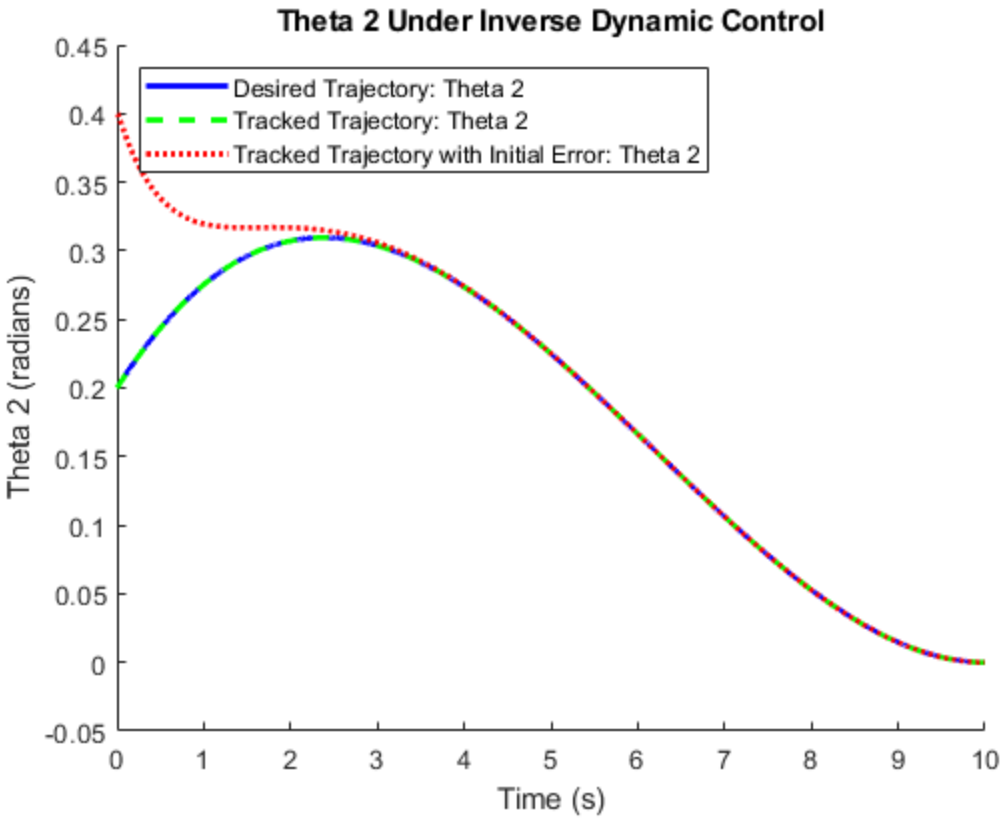
Simulate the tracking controller with no initial error:

```
    [T,X] = ode45(@(t,x) inverseDC(t, x, a1, a2), [0 tf], x0,
 options);
    [T_error,X_error] = ode45(@(t,x) inverseDC(t, x, a1, a2), [0 tf],
 x0_error, options);
```

Plot the results of the simulations:

```
   plotTrajectories(1, 'Inverse Dynamic', time, trajectory(:,1), T,
X(:,1), T_error, X_error(:,1));
   plotTrajectories(2, 'Inverse Dynamic', time, trajectory(:,2), T,
X(:,2), T_error, X_error(:,2));
```






```
end
if plot_lyapunov
```

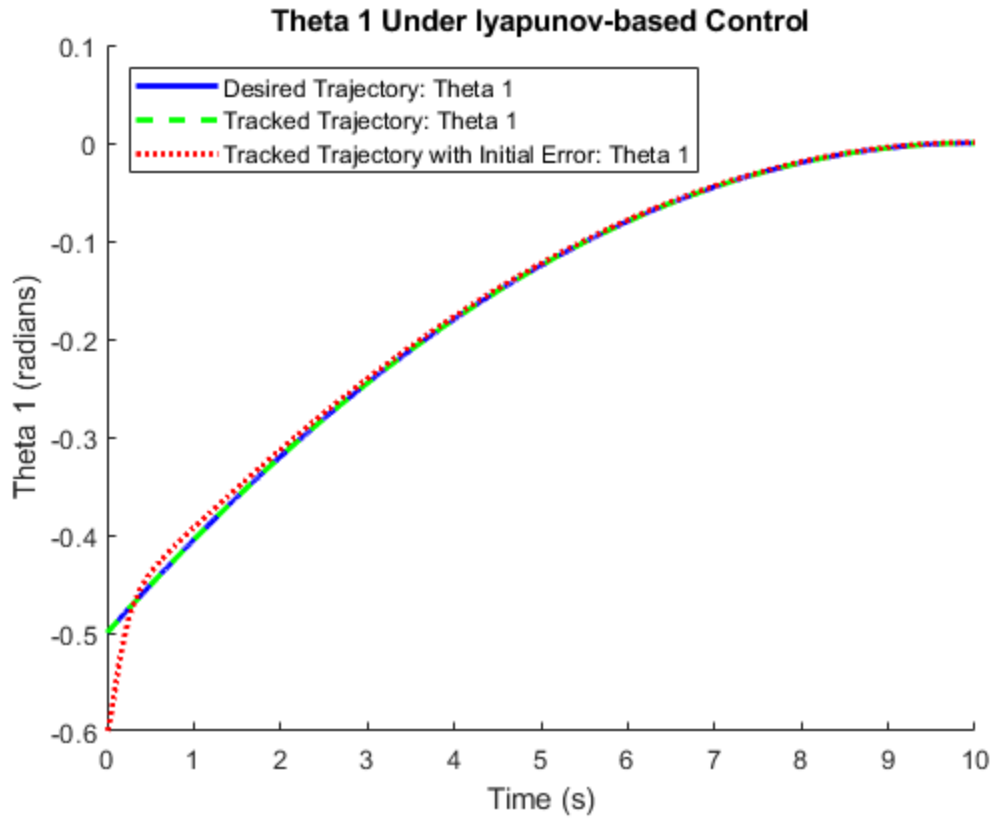
# Implement the lyapunov-based control

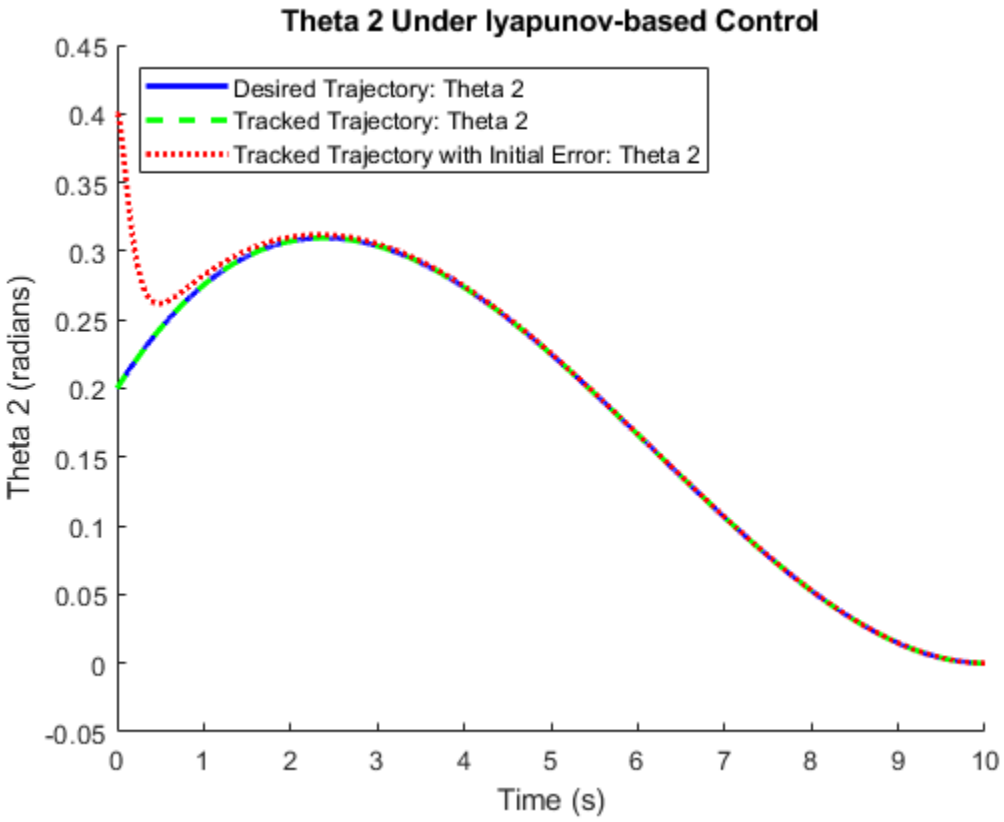
Set the tolerance options for ODE45 function:

```
    options = odeset('RelTol',1e-4,'AbsTol',[1e-4, 1e-4, 1e-4, 1e-4]);
    [T,X] = ode45(@(t,x) lyapunovCtrl(t, x, a1, a2),[0 tf],x0,
 options);
    [T_error,X_error] = ode45(@(t,x) lyapunovCtrl(t, x, a1, a2), [0
 tf], x0_error, options);
```

Plot the results of the simulations:

```
    plotTrajectories(1, 'lyapunov-based', time, trajectory(:,1), T,
 X(:,1), T_error, X_error(:,1));
    plotTrajectories(2, 'lyapunov-based', time, trajectory(:,2), T,
 X(:,2), T_error, X_error(:,2));
```


Theta 1 Under lyapunov-based Control
Desired Trajectory: Theta 1
Tracked Trajectory: Theta 1
Tracked Trajectory with Initial Error: Theta 1
Theta 1 (radians)
Time (s)
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
0
1
2
3
4
5
6
7
8
9
10


Theta 2 Under lyapunov-based Control
Desired Trajectory: Theta 2
Tracked Trajectory: Theta 2
Tracked Trajectory with Initial Error: Theta 2
Theta 2 (radians)
Time (s)
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
0
1
2
3
4
5
6
7
8
9
10

```
end

if plot_passivity
```

# Implement the passivity-based control

Initialize the A matrix (joint accelerations) as a global variable:

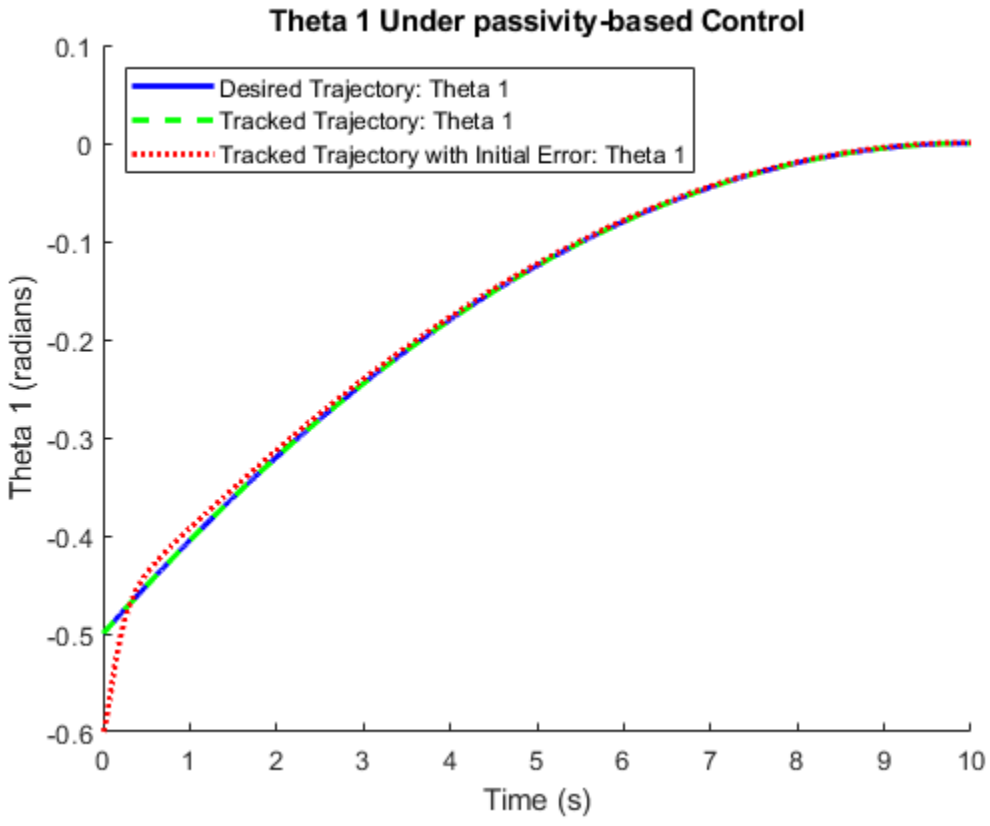
```
    global A
    A = [0;0];
```

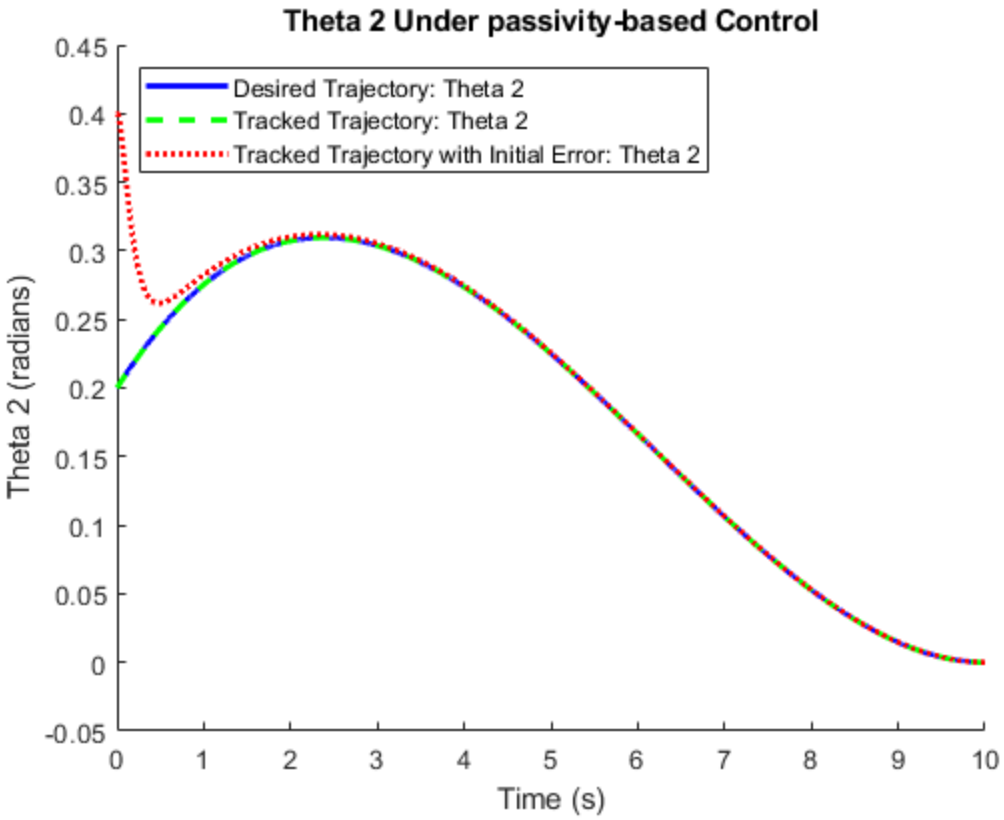
Set the tolerance options for ODE45 function:

```
    options = odeset('RelTol',1e-4,'AbsTol',[1e-4, 1e-4, 1e-4, 1e-4]);
    [T,X] = ode45(@(t,x) passivityCtrl(t,x, a1, a2),[0 tf],x0,
 options);
    [T_error,X_error] = ode45(@(t,x) passivityCtrl(t, x, a1, a2), [0
 tf], x0_error, options);
```

Plot the results of the simulations:

```
    plotTrajectories(1, 'passivity-based', time, trajectory(:,1), T,
 X(:,1), T_error, X_error(:,1));
    plotTrajectories(2, 'passivity-based', time, trajectory(:,2), T,
 X(:,2), T_error, X_error(:,2));
```






```
end
```

*Published with MATLAB® R2018a*

# Two Link Arm Trajectory Generater

This function takes in the initial and final joint angles and joint velocities as well as the final time. The function returns the coefficients for a cubic polynomial trajectory:

a(1) + a(2)t + a(3)t^2 + a(4)t^3;

```
function [a] = TwoLinkArmTraj(theta10, dtheta10, theta1f, dtheta1f,
 tf, nofigure)

M= [1 0 0 0;
    0 1 0 0;
    1 tf tf^2 tf^3;
    0 1 2*tf 3*tf^2];

b=[theta10; dtheta10;theta1f; dtheta1f];
a=M\b;
t=0:0.01:tf;

if nofigure==true
    return
else

    figure('Name','Position (degree)');
    plot(t,a(1)+a(2)*t+ a(3)*t.^2+a(4)*t.^3,'LineWidth',3);
    title('Position (degree)')
    grid

    figure('Name','Velocity (degree/s)');
    plot(t,a(2)*t+ 2*a(3)*t +3*a(4)*t.^2,'LineWidth',3);
    title('Velocity (degree/s)')
    grid

    figure('Name','Acceleration (degree/s^2)');
    plot(t, 2*a(3) +6*a(4)*t,'LineWidth',3);
    title('Acceleration (degree/s^2)')
    grid

end
end
```

*Published with MATLAB® R2018a*

## Table of Contents



# inverseDC

Creates the ODE function for a two link planar arm tracking a cubic polynomial trajectory by inverse dynamic control.

```
function [ dx ] = inverseDC( t, x, a1, a2)
```

# Constants and Variables:

Set the parameters for the arm:

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

Calculate 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;
```

# Trajectory Generation:

Note *x* is in the form of *q1*, *q2*, *q1_dot*, *q2_dot*:

Cubic polynomials:

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

Calculate the velocity and acceleration in both *theta 1* and *theta 2*:

```
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 ];
```

Calculate 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);
theta_dot = x(3:4,1);
```

# Planar Arm Dynamics:

Calculate the parameters in the dynamic model:

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

Calculate 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;
```

# Inverse Dynamic Ccontroller:

Set the *kp* and *kd* gain constants (positive definite diagonal matrices):

```
kp = [150 0
    0 150];
kd = [100 0,
    0  100];
```

Calculate the tracking errors, *e* and *e_dot*:

```
e = theta - theta_d;
e_dot = theta_dot - dtheta_d;
```

Calculate the *aq* matrix:

```
aq_desired = ddtheta_d;
aq = aq_desired - kp*e - kd*e_dot;
```

Calculate the controller, *u*:

```
u = zeros(2,1);
u = Mmat*aq + Cmat*theta_dot + Gmat;
```

Calculate the acceleration values:

```
theta_dot_dot = invM*( u - Cmat*theta_dot - Gmat);
```

# Outputs

Initialize the output of the function, *dx*:

```
dx = zeros(4,1);
```

Set the final outputs:

```
dx(1) = x(3,1);
```

```
dx(2) = x(4,1);
dx(3) = theta_dot_dot(1);
dx(4) = theta_dot_dot(2);

end
```

*Published with MATLAB® R2018a*

## Table of Contents



# lyapunovCtrl

Creates the ODE function for a two link planar arm tracking a cubic polynomial trajectory by lyapunov-based control.

```
function [ dx ] = lyapunovCtrl( t, x, a1, a2)
```

# Constants and Variables

Set the parameters for the arm:

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

Calculate 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;
```

# Trajectory Generation

Note *x* is in the form of *q1*, *q2*, *q1_dot*, *q2_dot*:

Cubic polynomials:

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

Calculate the velocity and acceleration in both *theta 1* and *theta 2*:

```
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 ];
```

Calculate 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);
theta_dot = x(3:4,1);
```

# Planar Arm Dynamics

Calculate the parameters in the dynamic model:

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

Calculate 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;
```

# Lyapunov-Based Controller

Set the *kv* gain constant (positive definite diagonal matrix):

```
kd = [25 0; ...
      0 25];
```

Set the *capital_lambda* constant (positive definite square matrix):

```
capital_lambda = [10 0; ...
                  0 10];
```

Calculate the tracking errors, *e* and *e_dot*:

```
e = theta - theta_d;
e_dot = theta_dot - dtheta_d;
```

Calculate *si_dot* and *si_dot_dot*:

```
si_dot = dtheta_d - capital_lambda*e;
si_dot_dot = ddtheta_d - capital_lambda*e_dot;
```

Calculate *sigma*:

```
I = eye(2,2);
% sigma = I*e_dot + capital_lambda*e;
sigma = theta_dot - si_dot;
```

Calculate the controller, *u*:

```
u = zeros(2,1);
u = Mmat*si_dot_dot + Cmat*si_dot + Gmat - kd*sigma;
```

Calculate the accleration values:

```
theta_dot_dot = zeros(2,1);
% theta_dot_dot = sigma_dot - si_dot_dot
```

```
theta_dot_dot = invM*( u - Cmat*theta_dot - Gmat);
```

# Outputs

Initialize the output of the function, *dx*:

```
dx = zeros(4,1);
```

Final outputs:

```
dx(1) = x(3,1);
dx(2) = x(4,1);
dx(3) = theta_dot_dot(1);
dx(4) = theta_dot_dot(2);

end
```

*Published with MATLAB® R2018a*

## Table of Contents



# passivityCtrl

Creates the ODE function for a two link planar arm tracking a cubic polynomial trajectory by passivity-based control.

```
function [ dx ] = passivityCtrl( t, x, a1, a2)
```

# Constants and Variables

Sets *A* as the global *A* variable (the past joint accelerations):

```
global A
```

Set the parameters for the arm:

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

Calculate 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;
```

# Trajectory Generation

Note *x* is in the form of *q1*, *q2*, *q1_dot*, *q2_dot*:

Cubic polynomials:

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

Calculate the velocity and acceleration in both *theta 1* and *theta 2*:

```
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 ];
```

Calculate 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];
```

Set the current joint values:

```
theta = x(1:2,1);
theta_dot = x(3:4,1);
theta_dot_dot = A;
```

# Planar Arm Dynamics

Calculate the parameters in the dynamic model:

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

Calculate 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;
```

# Passivity-Based Controller

Set the *kv* gain constant (positive definite matrix):

```
kv = [25 0; ...
      0 25];
```

Set the *capital_lambda* constant (positive definite square matrices):

```
capital_lambda = [10 0; ...
                  0 10];
```

Calculate the tracking errors, *e*, *e_dot*, and *e_dot_dot*:

```
e = theta - theta_d;
e_dot = theta_dot - dtheta_d;
e_dot_dot = theta_dot_dot - ddtheta_d;
```

Calculate *r* and *r_dot*:

```
I = eye(2,2);
r = e_dot + capital_lambda*e;
r_dot = e_dot_dot + capital_lambda*e_dot;
```

Calculate *v*:

```
% v = q_dot - r
v = theta_dot - r;
```

Calculate *a*:

```
%a = q_dot_dot - r
a = theta_dot_dot - r_dot;
```

Calculate the controller, *u*:

```
u = zeros(2,1);
u = Mmat*a + Cmat*v + Gmat - kv*r;
```

Calculate the acceleration values:

```
theta_dot_dot = zeros(2,1);
% theta_dot_dot = sigma_dot - si_dot_dot
theta_dot_dot = invM*(u - Cmat*theta_dot - Gmat);
```

Update the acceleration values:

```
A = theta_dot_dot;
```

# Outputs

Initialize the output of the function, *dx*:

```
dx = zeros(4,1);
```

Set the final outputs:

```
dx(1) = x(3,1);
dx(2) = x(4,1);
dx(3) = theta_dot_dot(1);
dx(4) = theta_dot_dot(2);

end
```

*Published with MATLAB® R2018a*

# plotTrajectories

This function plots two input trajectories with specific formatting based on the input angle number and the input control method.

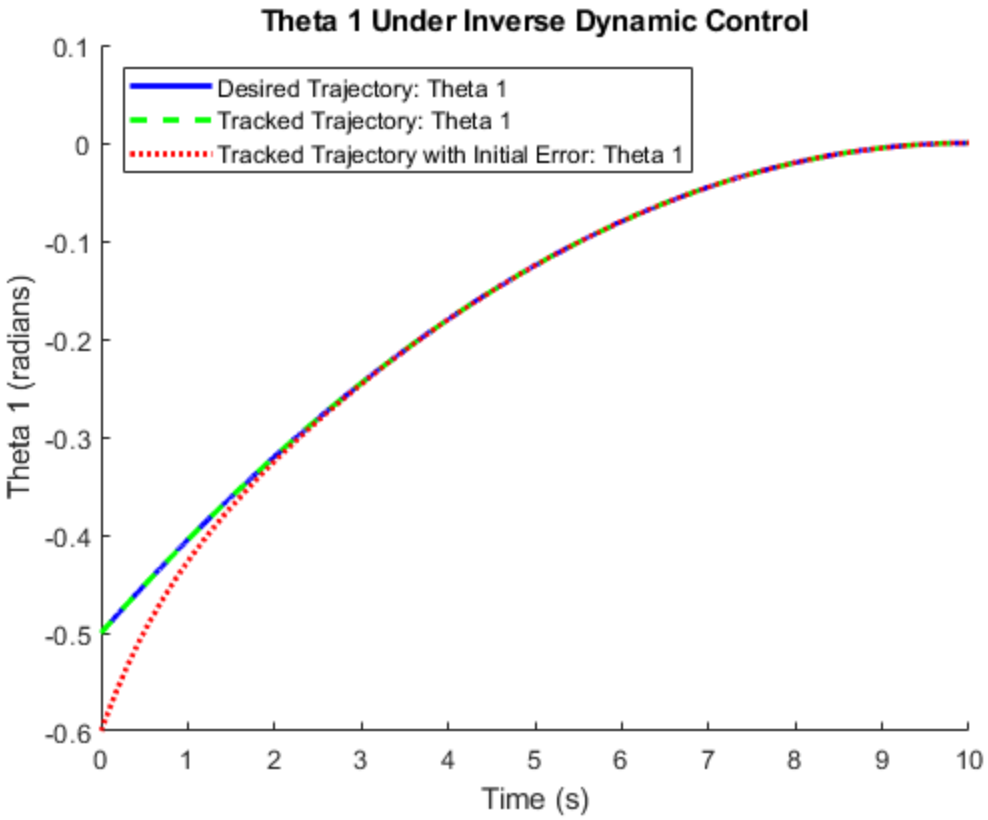
```
function plotTrajectories(theta, control, time, trajectory, T, X,
 T_error, X_error)
```

Creates name of the plot:

```
name = sprintf('Theta %d Under %s Control', theta, control);
```

Creates the figure:

```
figure('Name',name);
hold on
% Plot the trajectories:
plot(time, trajectory, ...
    'b', ...
    'LineWidth', 2, ...
    'DisplayName', sprintf('Desired Trajectory: Theta %d', theta));
plot(T, X, ...
    'g--', ...
    'LineWidth', 2, ...
    'DisplayName', sprintf('Tracked Trajectory: Theta %d', theta));
plot(T_error, X_error, ...
    'r:', ...
    'LineWidth', 2, ...
    'DisplayName', sprintf('Tracked Trajectory with Initial Error:
 Theta %d', theta));
% Sets lables:
title(name);
xlabel('Time (s)');
ylabel(sprintf('Theta %d (radians)', theta));
legend('location', 'northwest');
```




```
end
```

*Published with MATLAB® R2018a*