Adaptive Control of the RRBot Robotic Arm

5.2.1 Initial Setup

Use your code from Programming Assignment 3 to generate a cubic polynomial trajectory for the first and second joints of the robot. The time span and the desired initial and final joint angles and velocities are given by:

$$t_0 = 0, \quad t_f = 10\ sec$$
$$\theta_1(t_0) = 180^\circ, \quad \theta_1(t_f) = 0, \quad \theta_2(t_0) = 90^\circ, \quad \theta_2(t_f) = 0$$
$$\dot{\theta}_1(t_0) = \dot{\theta}_1(t_f) = \dot{\theta}_2(t_0) = \dot{\theta}_2(t_f) = 0$$

```
q_desired =

   (pi*t^3)/500 - (3*pi*t^2)/100 - (6189958033024885*t)/10141204801825835211973625643008 + pi
(pi*t^3)/1000 - (3*pi*t^2)/200 - (6189958033024885*t)/20282409603651670423947251286016 + pi/2


qdot_desired =

  (3*pi*t^2)/500 - (3*pi*t)/50 - 6189958033024885/10141204801825835211973625643008
(3*pi*t^2)/1000 - (3*pi*t)/100 - 6189958033024885/20282409603651670423947251286016


qddot_desired =

 (3*pi*t)/250 - (3*pi)/50
(3*pi*t)/500 - (3*pi)/100
```

(Cubic Trajectories)

5.2.2 Adaptive Control

a) ***(1 point)*** Consider the robot's equations of motion in the manipulator form derived in Programming Assignment 4. Re-write the dynamics of the robot in the *linear parametric form* of:

$$Y(q, \dot{q}, \ddot{q})\,\alpha = \tau$$

where $Y(q, \dot{q}, \ddot{q}) \in \mathbb{R}^{2\times 5}$ is the regressor and $\alpha \in \mathbb{R}^{5\times 1}$ is the parameter vector. Note that α will include lumped mass parameters from the original system.

```
%--------- Manipulator Equation Form ---------%
alpha_1 = I1 + I2 + m1*r1^2 + m2*(l1^2 + r2^2);
alpha_2 = m2*l1*r2;
alpha_3 = I2 + m2*r2^2;
alpha_4 = m1*r1 + m2*l1;
alpha_5 = m2*r2;

Mmat = [alpha_1+2*alpha_2* cos(theta2), alpha_3+alpha_2* cos(theta2); alpha_3+alpha_2* cos(theta2), alpha_3];
Cmat = [-alpha_2* sin(theta2)*theta2_dot, -alpha_2* sin(theta2)*(theta1_dot+theta2_dot); alpha_2* sin(theta2)*theta1_dot,0];
Gmat = [-alpha_4*g*sin(theta1)-alpha_5*g*sin(theta1+theta2); -alpha_5*g*sin(theta1+theta2)];

%--------- Linear Parametric Form -----------%
Y = [theta1_ddot, ...
    cos(theta2)*(2*theta1_ddot + theta2_ddot) - 2* sin(theta2)*theta1_dot*theta2_dot - sin(theta2)*theta2_dot^2, ...
    theta2_ddot, ...
    -sin(theta1)*g, ...
    -sin(theta1 + theta2)*g;
    0, ...
    sin(theta2)*theta1_dot^2 + cos(theta2)*theta1_ddot, ...
    theta1_ddot + theta2_ddot, ...
    0, ...
    -sin(theta1+theta2)*g];

alpha = [alpha_1;
        alpha_2;
        alpha_3;
        alpha_4;
        alpha_5];|
```

(From MATLAB)

$\Rightarrow$ The dynamics can be represented as,

$$\tau = \text{Mmat}\cdot\begin{bmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix} + \text{Cmat}\cdot\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix} + \text{Gmat} \quad \text{—(i)}$$

$\rightarrow$ In linear parametric equation form, the same can be represented as,

$$\tau = Y(\theta_1, \dot{\theta}_1, \ddot{\theta}_1, \theta_2, \dot{\theta}_2, \ddot{\theta}_2)\cdot\alpha \quad \text{—(ii)}$$

$\Rightarrow$ Equations (i) & (ii) are equivalent to each other as verified below.

```
%----------- Verification -------------%
Tau_MEF = Mmat*[theta1_ddot; theta2_ddot] + Cmat*[theta1_dot; theta2_dot] + Gmat;
Tau_LPE = Y * alpha;

Tau_MEF = subs(Tau_MEF, [theta1, theta2, theta1_dot, theta2_dot, theta1_ddot, theta2_ddot], [1,1,1,1,1,1]);
Tau_LPE = subs(Tau_LPE, [theta1, theta2, theta1_dot, theta2_dot, theta1_ddot, theta2_ddot], [1,1,1,1,1,1]);
res = isequal(Tau_MEF, Tau_LPE); %retuns 1 if equal, 0 otherwise.
if(res)
    disp("Both the forms are equivalent to each other.");
end
```

(Verification snippet from MATLAB)

```
Both the forms are equivalent to each other.
```

(Verification result)

b) ***(5 points)*** Design an Adaptive Inverse Dynamics control law along with the adaptation update law for trajectory tracking by the robot using the method described in Lecture 23. The control gains to be designed are $K_p \in \mathbb{R}^{2\times 2}$ and $K_d \in \mathbb{R}^{2\times 2}$ (for the virtual control input v), and the $P = P^T > 0 \in \mathbb{R}^{4\times 4}$ and $\Gamma = \Gamma^T > 0 \in \mathbb{R}^{5\times 5}$ for the adaptation update law $\dot{\hat{\alpha}}$.

- Use state-feedback control design to determine the control gains K_p and K_d to place the eigenvalues at $\{-2, -2, -3, -3\}$. You can use the `place` function in MATLAB to design the control gain matrix $K \in \mathbb{R}^{2\times 4}$, where $K = \begin{bmatrix} K_p & K_d \end{bmatrix}$.
- The matrix P is the solution to the Lyapunov equation $A^T P + PA = -Q$. You can use the `lyap` function in MATLAB to solve for P.
- The matrix Γ can be tuned by trial and error. An identity matrix could be used as the initial guess.

```
% ------- Adaptive Inverse Dynamics Control --------%
A = [0,0,1,0;
     0,0,0,1;
     0,0,0,0;
     0,0,0,0];

B = [0,0;
     0,0;
     1,0;
     0,1];

desiredEigenValues = [-2,-2,-3,-3];
K = place(A, B, desiredEigenValues);

Acl = A - B*K;
Q = eye(4);
P = lyap(Acl',Q);
gamma = eye(5)*0.03;
```

$\Gamma = \Gamma^T > 0 \in R^{5\times 5}$

(Implementation)

```
K =

    6.0000         0    5.0000         0
         0    6.0000         0    5.0000


P =

    1.1167         0    0.0833         0
         0    1.1167         0    0.0833
    0.0833         0    0.1167         0
         0    0.0833         0    0.1167
```

Here,

$K = [k_p \quad k_d] \in R^{2\times 4}$

$P = P^T > 0 \in R^{4\times 4}$

(Calculated control gain matrix and P matrix)

c) *(4 points)* Update the `ode` function developed in Programming Assignment 3 to implement the adaptive inverse dynamics control law and adaptation law designed in part (b). Note that you will need to evaluate the cubic polynomial trajectories inside the `ode` function to obtain the desired states at each point in time.

```
%--------------- Get the current state values ---------------%
dX = zeros(9,1);
X = num2cell(X);
[theta1, theta2, theta1_dot, theta2_dot,...
    alpha_1_hat, alpha_2_hat, alpha_3_hat, alpha_4_hat, alpha_5_hat] = deal(X{:});
jointValues = [theta1; theta2; theta1_dot; theta2_dot];
```

(Alpha hat integrated with the system dynamics)

```
% ------- Adaptive Inverse Dynamics Control --------%
B = [0,0;
    0,0;
    1,0;
    0,1];

K = [6.0000,0, 5.0000, 0;
     0, 6.0000, 0, 5.0000];

P = [1.1167,        0,    0.0833,         0;
          0,   1.1167,         0,    0.0833;
     0.0833,        0,    0.1167,         0;
          0,   0.0833,         0,    0.1167];

gamma = eye(5)*0.03;

error = jointValues - Q_desired;

%-------------- Virtual control input ---------------------%
V = -K*error + qddot_desired;

%--------- Manipulator Equation Form ---------%
alpha_1 = alpha_1_hat;
alpha_2 = alpha_2_hat;
alpha_3 = alpha_3_hat;
alpha_4 = alpha_4_hat;
alpha_5 = alpha_5_hat;

Mmat_hat = [alpha_1+2*alpha_2* cos(theta2), alpha_3+alpha_2* cos(theta2); alpha_3+alpha_2* cos(theta2), alpha_3];
Cmat_hat = [-alpha_2* sin(theta2)*theta2_dot, -alpha_2* sin(theta2)*(theta1_dot+theta2_dot); alpha_2* sin(theta2)*theta1_dot,0];
Gmat_hat = [-alpha_4*g*sin(theta1)-alpha_5*g*sin(theta1+theta2); -alpha_5*g*sin(theta1+theta2)];

%------------- The overall control law --------------%
Tau = Mmat_hat*V + Cmat_hat*[theta1_dot; theta2_dot] + Gmat_hat;
t1 = Tau(1);
t2 = Tau(2);
```

(Implementation of the adaptive inverse dynamics control law)

```
%------- Integrating the adaption law -------------%
theta1_ddot = dX(3);
theta2_ddot = dX(4);

Y = [theta1_ddot, ...
    cos(theta2)*(2*theta1_ddot + theta2_ddot) - 2* sin(theta2)*theta1_dot*theta2_dot - sin(theta2)*theta2_dot^2, ...
    theta2_ddot, ...
    -sin(theta1)*g, ...
    -sin(theta1 + theta2)*g;
    0, ...
    sin(theta2)*theta1_dot^2 + cos(theta2)*theta1_ddot, ...
    theta1_ddot + theta2_ddot, ...
    0, ...
    -sin(theta1+theta2)*g];

phi = Mmat_hat \ Y;
alpha_hat_dot = -gamma \(phi'*B'*P*error);

dX(5) = alpha_hat_dot(1);
dX(6) = alpha_hat_dot(2);
dX(7) = alpha_hat_dot(3);
dX(8) = alpha_hat_dot(4);
dX(9) = alpha_hat_dot(5);
```

(Implementation of the adaptation law)

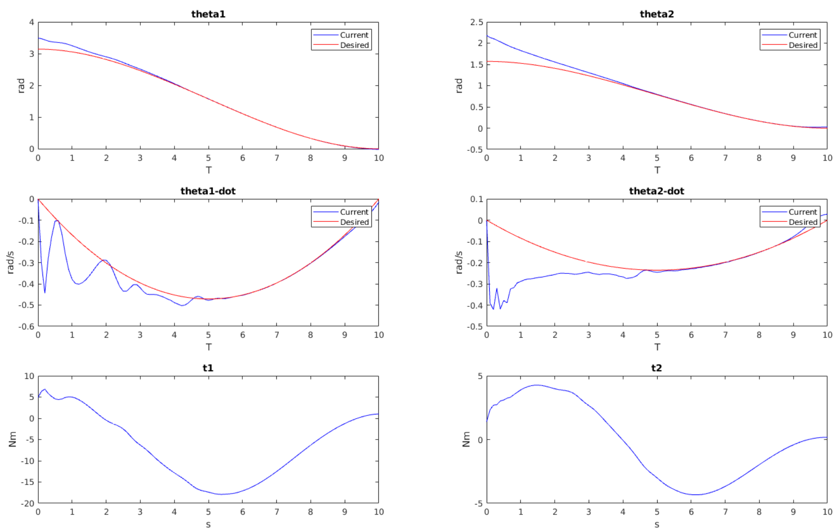
d) ***(6 points)*** Use `ode45` and the `ode` function developed in part (c) to construct a simulation of the system in MATLAB with the time span of $[0, 10]$ sec and initial conditions of:

$$\theta_1(0) = 200^\circ, \quad \theta_2(0) = 125^\circ, \quad \dot{\theta}_1 = 0, \quad \dot{\theta}_2 = 0$$

Set the initial values of the unknown parameter vector α to 75% of the actual values:

$$\hat{\alpha}(0) = 0.75\alpha$$

```
% simulate the system for 10 sec for given control inputs using ODE45
T = 10;
alpha_hat0 = 0.75*alpha;
y0 = [deg2rad(200), deg2rad(125),0 ,0, ...
        alpha_hat0(1), alpha_hat0(2), alpha_hat0(3), alpha_hat0(4), alpha_hat0(5)];
[t,y] = ode45(@ode_rrbot, 0:0.1:T, y0);
```



(State Trajectories)

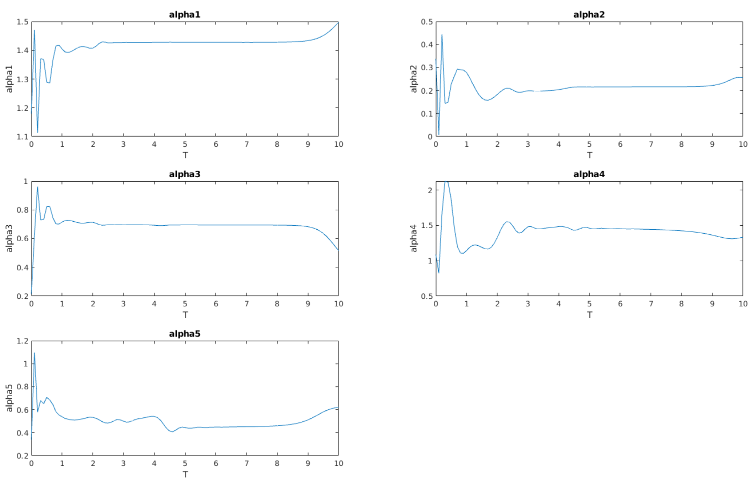
→ Despite the large-scale uncertainty associated with model parameters, the position and velocity tracking errors converge to zero, assuring the system's stability.

⇒ The control inputs stays within the given requirements.

```
Torque1: -17.896 < u1 < 6.881

Torque2: -4.334 < u2 < 4.268
```

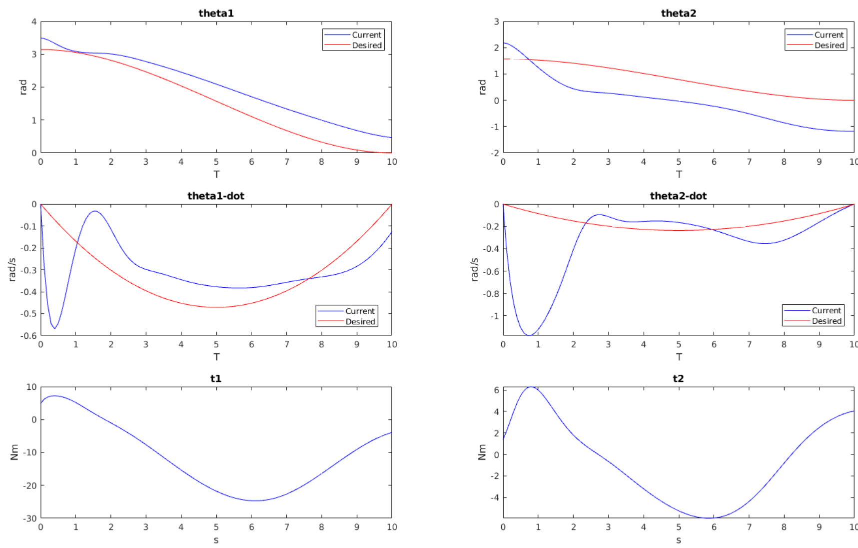
⇒



(Evolution of the parametric vector)

→ The parametric vector $\hat{\alpha}$ converges to constant values as states converge to zero trajectory errors.

e) ***(4 points)*** To evaluate the performance of the robot without the *adaptive* inverse dynamics control, construct a simulation of the system in MATLAB with the same control law but with the P matrix set to zero (do not change other design parameters such as K_p and K_d). Again, plot the state trajectories, the control inputs trajectories, and the associated desired trajectories to compare the resulting performance with the performance obtained in part (l). Discuss the results in your final report.

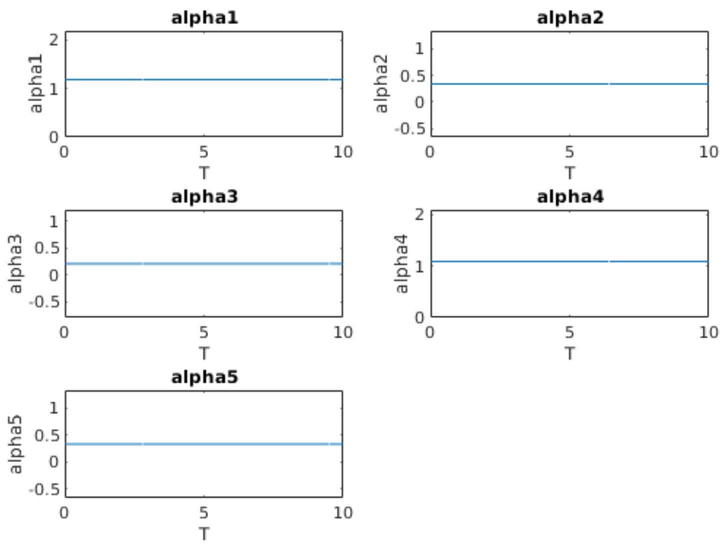


(State Trajectories- Without adaptive inverse dynamics control)

→ In absence of adaptive control, the control law continues with the initial values choosen for the parametric vector & couldn't update them over time.

→ As the initial values are not the actual values, with incorrect model parameters, the system didn't converge to zero tracking errors.

$\Rightarrow$



(No evolution of parametric vector)

→ without adaptive inverse dynamics law, the parametric vector couldn't update & uses the same initial values throughout the simulation period.