Robust Control of the RRBot Robotic Arm

4.2.1 Initial Setup

a) 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:

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

```
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)

b) Consider the equations of motion derived for the robot in Programming Assignment 1. As discussed in Programming Assignment 3, the equations of motion can be transformed into the standard Manipulator form:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau$$

For the purpose of this assignment, the robot's equations of motion in the manipulator from are provided in the Appendix. Copy and paste the provided equations of motion into a MATLAB script (with appropriate symbols defined) to obtain symbolic expressions for the robot's dynamics.

```
Mmat =

[I1 + I2 + m1*r1^2 + m2*(l1^2 + r2^2) + 2*l1*m2*r2*cos(theta2), m2*r2^2 + l1*m2*cos(theta2)*r2 + I2]
[                          m2*r2^2 + l1*m2*cos(theta2)*r2 + I2,                        m2*r2^2 + I2]


Cmat =

[-l1*m2*r2*theta2_dot*sin(theta2), -l1*m2*r2*sin(theta2)*(theta1_dot + theta2_dot)]
[ l1*m2*r2*theta1_dot*sin(theta2),                                               0]


Gmat =

- g*m2*(r2*sin(theta1 + theta2) + l1*sin(theta1)) - g*m1*r1*sin(theta1)
                                        -g*m2*r2*sin(theta1 + theta2)
```

(Manipulator Equation Form - Symbolic)

c) ***(7 points)*** Design a Robust Inverse Dynamics control law for trajectory tracking by the robot using the method described in Lecture 19. 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 Lyapunov matrix $P = P^T > 0 \in \mathbb{R}^{4\times 4}$ for the robust control term v_r.

$$\hat{m}_1 = \hat{m}_2 = 0.75\ (kg), \qquad \hat{I}_1 = \hat{I}_2 = 0.063\ (kg \cdot m^2)$$

- Use state-feedback control design to determine the control gains K_p and K_d to place the eigenvalues at $\{-1, -1, -2, -2\}$. 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.

Moreover, initialize a constant value for the uncertainty upper bound ρ, which is used to compute the robust control term v_r. This bound can be later tuned in simulation by trial and error. For the purpose of this assignment, you can use a *constant* ρ value.

```
% ------- Robust 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 = [-1,-1,-2,-2];
K = place(A, B, desiredEigenValues);

Acl = A - B*K;
Q = eye(4);
P = lyap(Acl',Q);
phi = 0;                  %boundary layer - tunable Parameter
rho = 10;                 %upper bound - tunable Parameter
```

→ The tunable parameters are tuned further.

$\Rightarrow$ M, C & G calculated on the nominal values.

```
Mmat =

[(27*cos(theta2))/40 + 4719/4000, (27*cos(theta2))/80 + 1719/8000]
[(27*cos(theta2))/80 + 1719/8000,                       1719/8000]


Cmat =

[-(27*theta2_dot*sin(theta2))/80, -(27*sin(theta2)*(theta1_dot + theta2_dot))/80]
[ (27*theta1_dot*sin(theta2))/80,                                              0]


Gmat =

- (1323*sin(theta1 + theta2))/400 - (4263*sin(theta1))/400
                          -(1323*sin(theta1 + theta2))/400
```

(Manipulator Equation Form - Based on the nominal values)

$\Rightarrow$ The Gain Matrix :

```
K =

    2.0000         0    3.0000         0
         0    2.0000         0    3.0000
```

$\rightarrow K \in R^{2\times4}$

where,

$K = [k_p \quad k_d]$

$\Rightarrow$ The Matrix P :

```
P =

    1.2500         0    0.2500         0
         0    1.2500         0    0.2500
    0.2500         0    0.2500         0
         0    0.2500         0    0.2500
```

$\rightarrow$ Here, $P = P^T > 0 \in R^{4\times4}$

d) ***(2 points)*** Update the `ode` function developed in Programming Assignment 3 to implement the robust inverse dynamics control law designed in part (c). 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.

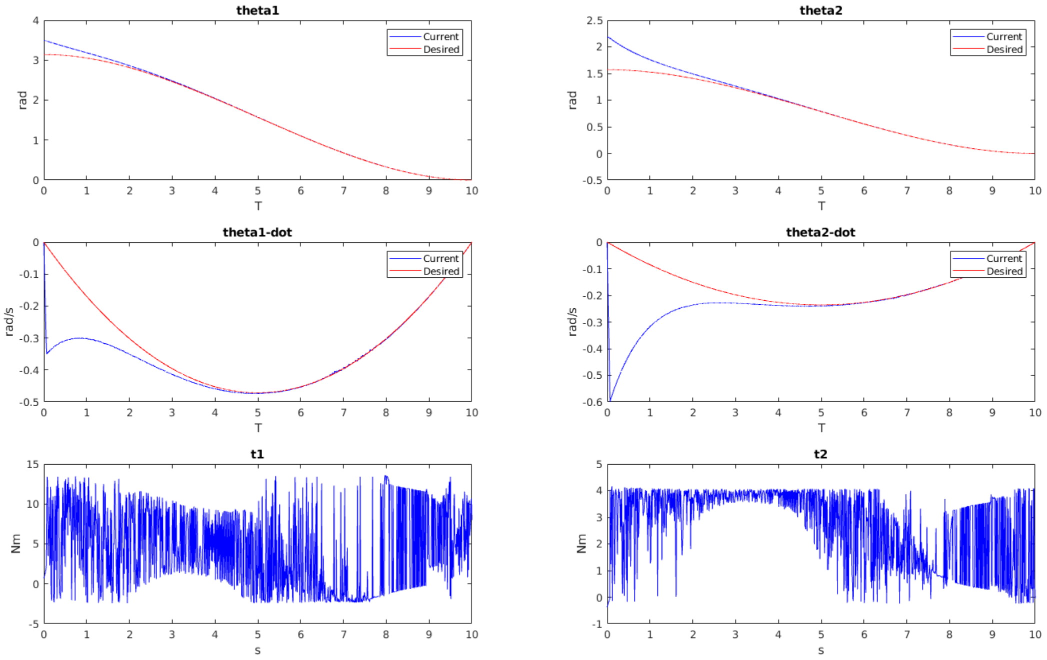
```
%--- Generate cubic polynomial trajectories for both the joints ------%

q1 = (pi*t^3)/500 - (3*pi*t^2)/100 - (6189958033024885*t)/10141204801825835211973625643008 + pi;
q2 = (pi*t^3)/1000 - (3*pi*t^2)/200 - (6189958033024885*t)/20282409603651670423947251286016 + pi/2;
q1_dot = (3*pi*t^2)/500 - (3*pi*t)/50 - 6189958033024885/10141204801825835211973625643008;
q2_dot = (3*pi*t^2)/1000 - (3*pi*t)/100 - 6189958033024885/20282409603651670423947251286016;
q1_ddot = (3*pi*t)/250 - (3*pi)/50;
q2_ddot = (3*pi*t)/500 - (3*pi)/100;

q_desired = [q1;q2];
qdot_desired = [q1_dot;q2_dot];
qddot_desired = [q1_ddot;q2_ddot];
Q_desired = [q_desired; qdot_desired];

%--------- Manipulator Equation Form ---------%
m1_hat = 0.75; m2_hat = 0.75;
I1_hat = 0.063; I2_hat = 0.063;

a = I1_hat + I2_hat + m1_hat*r1^2 + m2_hat*(l1^2 + r2^2);
b = m2_hat*l1*r2;
d = I2_hat + m2_hat*r2^2;

Mmat= [a+2*b* cos(theta2), d+b* cos(theta2); d+b* cos(theta2), d];
Cmat= [-b* sin(theta2)*theta2_dot, -b* sin(theta2)*(theta1_dot+theta2_dot); b* sin(theta2)*theta1_dot,0];
Gmat= [-m1_hat*g*r1* sin(theta1)-m2_hat*g*(l1* sin(theta1)+r2* sin(theta1+theta2)); -m2_hat*g*r2* sin(theta1+theta2)];
% ------- Robust Inverse Dynamics Control --------%
B = [0,0;
    0,0;
    1,0;
    0,1];

K = [2.0000,0, 3.0000, 0;
     0, 2.0000, 0, 3.0000];

P = [1.2500,         0,    0.2500,         0;
          0,    1.2500,         0,    0.2500;
     0.2500,         0,    0.2500,         0;
          0,    0.2500,         0,    0.2500];

phi = 0;              %boundary layer - tunable Parameter
rho = 10;             %upper bound - tunable Parameter

error = jointValues - Q_desired;
nr = error'*P*B;
norm_nr = norm(nr);
if phi > 0
    if phi >= norm_nr
        vr = -rho*(nr/phi);
    elseif phi < norm_nr
        vr = -rho*(nr/norm_nr);
    end
else
    if norm_nr ~= 0
        vr = -rho*(nr/norm_nr);
    else
        vr = [0,0];
    end
end
%-------------- Virtual control input ---------------------%
V = -K*error + qddot_desired + vr';

%------------- The overall control law --------------%
Tau = Mmat*V + Cmat*[theta1_dot; theta2_dot] + Gmat;
```

→ The tunable parameters are tuned going further.

→ Tau is calculated using the given nominal values only.

e) ***(3 points)*** Use `ode45` and the `ode` function developed in part (d) to construct a simulation of the system in MATLAB with the time span of $[0, 10]$ sec and initial conditions of:

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



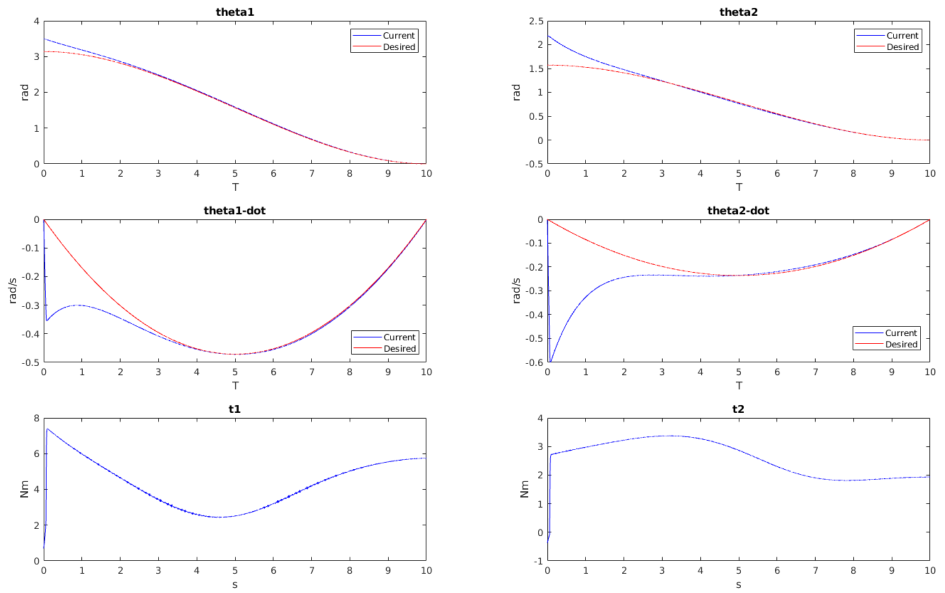
(ODE45 simulation : rho = 10; no boundary layer)

⇒ As explained during the class, the system converges efficiently, though the chattering problem is observed for the robust controller without boundary layer ($\phi = 0$).

→ The control input stays within given bounds.

$$-20 \le u_1 \le 20 \ (Nm), \qquad -10 \le u_2 \le 10 \ (Nm)$$

f) ***(3 points)*** Include a boundary layer in the robust control term v_r to reduce the chattering effect observed in the control input in part (e). Perform the same simulation as part (e), and plot the state trajectories, the control inputs trajectories, and the associated desired trajectories to evaluate the performance. Discuss your findings in your final report.



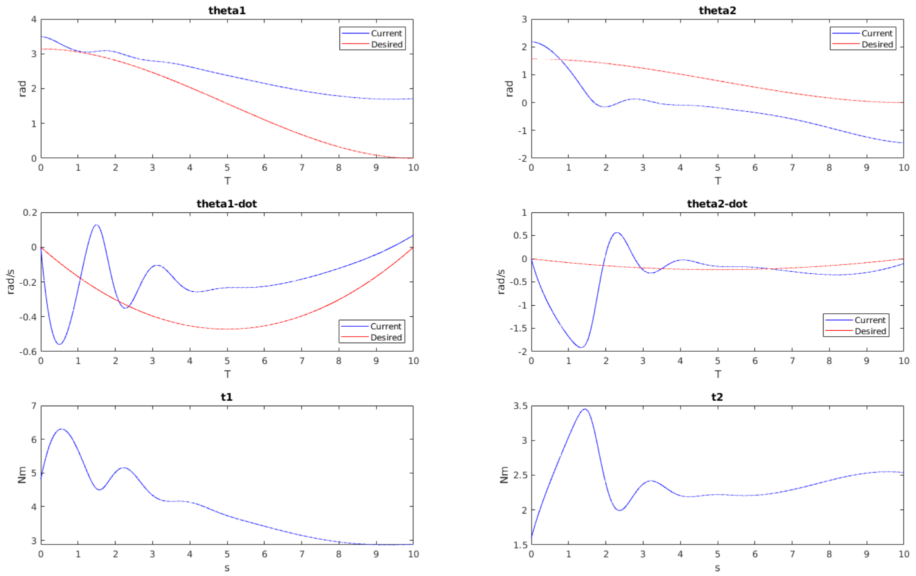
(ODE45 simulation: rho = 10; phi = 0.01)

→ Now, the boundary layer is added to deal with the chattering observed in the control input.

→ with $\phi = 0.01$; the control input is significantly improved as compared to $\phi = 0$ & it stays within bounds.

$$-20 \leq u_1 \leq 20\ (Nm), \qquad -10 \leq u_2 \leq 10\ (Nm)$$

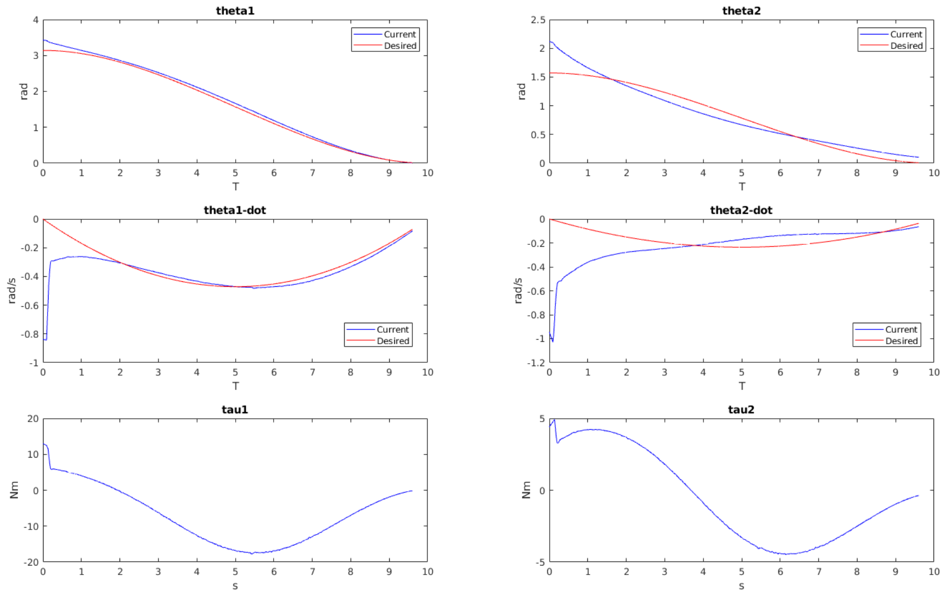
g) ***(1 points)*** To evaluate the performance of the robot without the *robust* inverse dynamics control, construct a simulation of the system in MATLAB with the same control law but with the v_r term set to zero (do not change other design parameters such as K_p and K_d). Again, plot the state trajectories, the control inputs trajectories and the associated desired trajectories to compare the resulting performance with the performance obtained in part (e). Discuss the results in your final report.



(ODE45 simulation: no robust term)

→ As the control law is designed with the nominal values, in absence of the robust term, the system can't account for "lumped Uncertainties" & hence, we could see the bad performance of the controller in such scenario.

h) ***(4 points)*** Create a new copy of the `rrbot_control.m` file provided in Programming Assignment 2, and rename the new file to `rrbot_robust_control.m`.



(Gazebo simulation: rho = 10; phi = 0.1)

$\Rightarrow$ As compared to ODE45, Gazebo simulation produces the similar results with some noteable differences in terms of convergence.

→ The chattering problem is taken care by taking $\phi = 0.1$. However, it affected the system states.

→ The control input stays within the bounds.

$$-20 \leq u_1 \leq 20 \; (Nm), \qquad -10 \leq u_2 \leq 10 \; (Nm)$$