DNLR

Robotics and Computer Vision Master Program

Dynamics of Non-Linear Systems

Prepared by:
Ahmed Mohsen Ali

Supervised by:
Prof. Alexander Klimchik

Outline

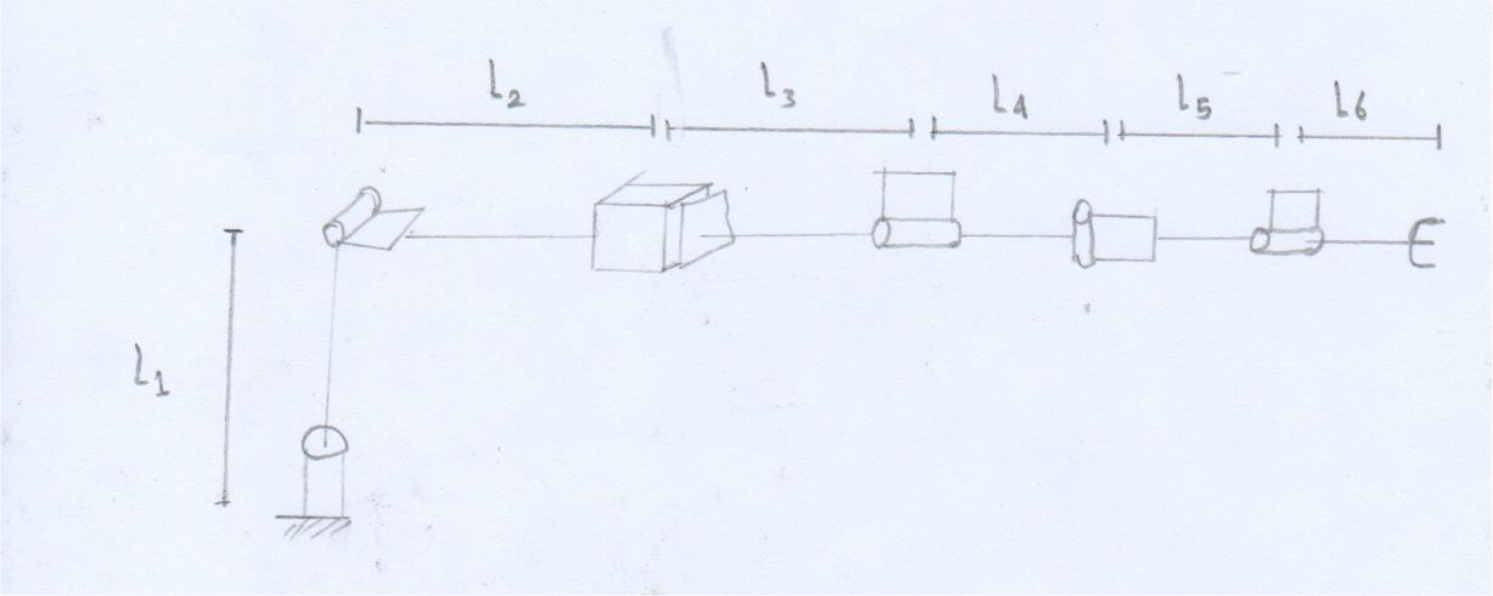
- Forward and Inverse Kinematics
- Differential Kinematics
- Trajectory Planning
- Dynamics

Part 1: Forward & Inverse kinematics

Robot Model

Spherical Robot

- Robot Model is Spherical Robot with spherical wrist.
- Robot Configuration is RRPRRR

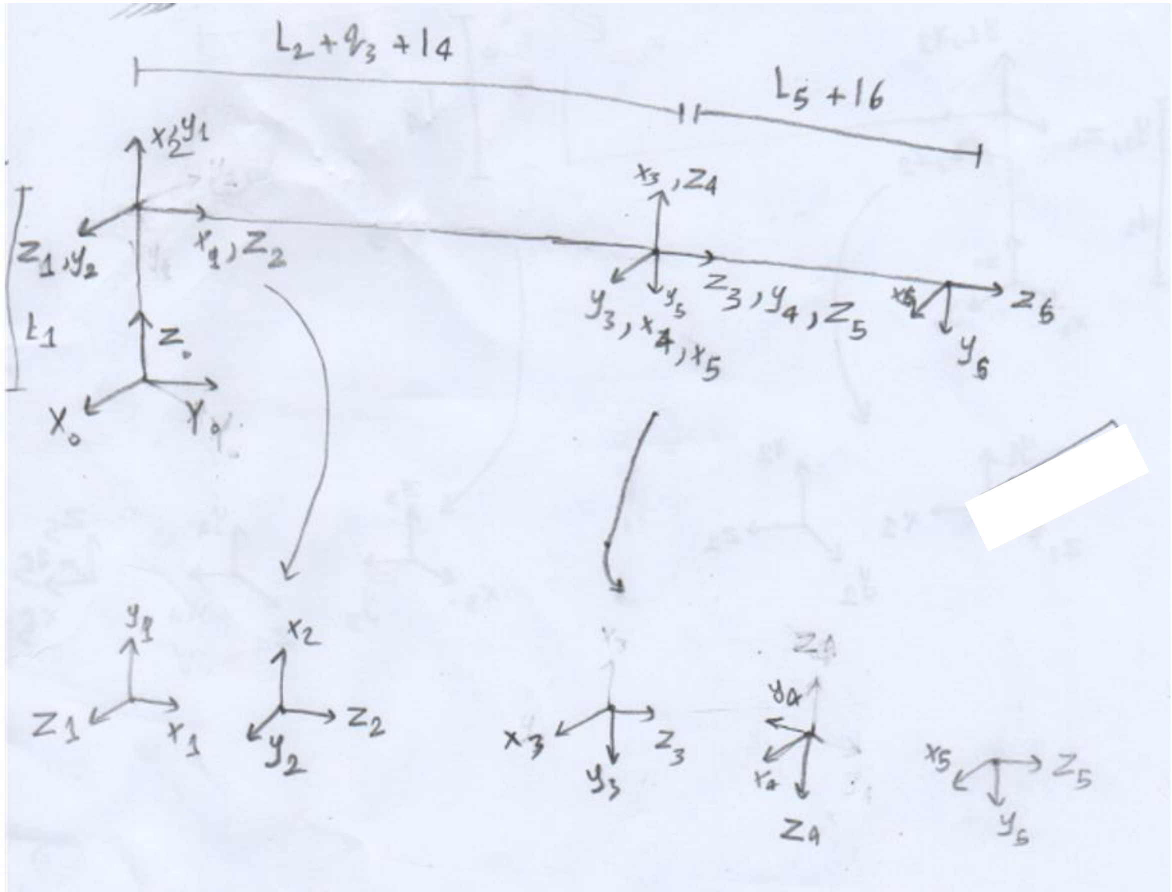


Hand Sketch drawing for the robot joints

Frames Assignment

Frames rules:

- All frames follow right hand rule
- All revolute and prismatic joint actuate around/along z axis
- Each x_i *is prependicular on* z_{i-1}



Hand sketch for Frames Assignment

DH Parameters

FK Equations:

```
a1 = rot_z1*rotm2tform(rotz(90))*trans_z1*rotm2tform(rotx(90));
a2 = rot_z2*rotm2tform(rotz(90))*rotm2tform(rotx(90));
a3 = rotm2tform(rotz(90))*trans_z3;
a4 = rot_z4*rotm2tform(rotx(-90));
a5 = rot_z5*rotm2tform(rotx(90));
a6 = rot_z6*trans_z6;
```

Link	θ	d	a	α
1	$\theta_1^* + 90°$	d_1	0	90°
2	$\theta_2^* +\ 90°$	0	0	90°
3	90	D_3^*	0	0
4	θ_4^*	0	0	-90°
5	θ_5^*	0	0	90°
6	θ_6^*	D_6	0	0

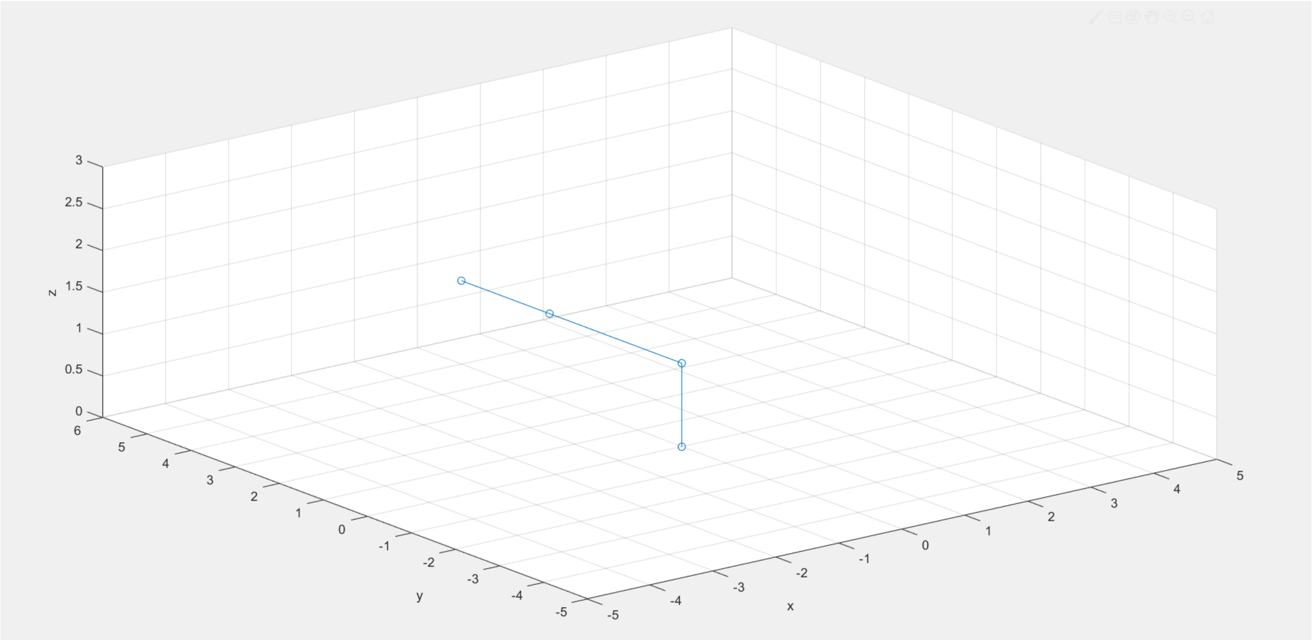
FK Results

DH Parameters=[0 0 1 0 0 0]

End Effector pose:

```
fk_val =

     1     0     0     0
     0     0     1     5
     0    -1     0     1
     0     0     0     1
```



Robot at initial pose

IK STEPS (Pieper's Solution)

1) Tool Pose Assumption:

```
T_06 =     [nx  sx ax x;
            ny  sy ay y;
            nz  sz az z;
            0   0  0  1];
```

2) Solve FK to obtain $\boldsymbol{A_0^3}$ and $\boldsymbol{A_3^6}$.

3) Solve for first three joint parameters.

4) Solve for last three joint parameters.

IK Results

IK Equations:

```
% IK Equations as derived in the report
q_1=atan2(-x_c,y_c);
q_2=atan2(cos(q_1)*(z_c-d_1),y_c);
q_3=(y_c/(cos(q_1)*cos(q_2)))-d_3;
q_4=atan2(ay*cos(q_1)*sin(q_2) - az*cos(q_2) - ax*sin(q_1)*sin(q_2), ax*cos(q_1) + ay*sin(q_1));
q_6=atan2(-sz*sin(q_2) + sy*cos(q_1)*cos(q_2) - sx*cos(q_2)*sin(q_1),nz*sin(q_2) + ny*cos(q_1)*cos(q_2) - nx*cos(q_2)*sin(q_1));
q_5=atan2(((ay*cos(q_1)*sin(q_2) - az*cos(q_2) - ax*sin(q_1)*sin(q_2))/sin(q_4)),az*sin(q_2) + ay*cos(q_1)*cos(q_2) - ax*cos(q_2)*sin(q_1));
```

IK Results

Tool Pose input:

```
fk_val =

     1     0     0     0
     0     0     1     5
     0    -1     0     1
     0     0     0     1
```

Joint Parmeters output:

q_1	0
q_2	0
q_3	1
q_4	0
q_5	NaN
q_6	0

Part 2: Differential Kinematics

Jacobin Matrix

- Jacobian of end effector was obtained with respect to each joint

- Third joint is different since it is prismatic

$$j_1 = \begin{pmatrix} Z_0 \times (O_6 - O_0) \\ Z_i \end{pmatrix}$$

$$j_2 = \begin{pmatrix} Z_1 \times (O_6 - O_1) \\ Z_1 \end{pmatrix}$$

$$j_3 = \begin{pmatrix} Z_2 \\ 0 \end{pmatrix}$$

$$j_4 = \begin{pmatrix} Z_3 \times (O_6 - O_3) \\ Z_3 \end{pmatrix}$$

$$j_5 = \begin{pmatrix} Z_4 \times (O_6 - O_4) \\ Z_4 \end{pmatrix}$$

$$j_6 = \begin{pmatrix} Z_5 \times (O_6 - O_5) \\ Z_5 \end{pmatrix}$$

Jacobin Code

- This matlab code calculates the jacobian for each joint
- The jacobian matrix for DH=[45 0 1 20 0 0] was as follow:

```
j =

   -2.7858    0.2418   -0.7071    0.2418    0.7071    0.2418
   -2.7858   -0.2418    0.7071    0.2418    0.7071   -0.2418
         0    3.9397         0         0         0    0.9397
         0    0.7071         0   -0.7071   -0.2418    0.7071
         0    0.7071         0    0.7071    0.2418    0.7071
    1.0000         0         0         0   -0.9397         0
```

```
for i=1:6
frame= frames(:,:,i);
pi_1 = frame(1:3,4);
zi_1 = frame(1:3,3);

if i==3        %jacobian for prismatic joint
        j(1:3,i)=zi_1;
        j(4:6,i)=[0 ;0 ;0];
else            %jacobian for revolute joint
        jp =cross(zi_1,pe-pi_1);

        j(1:3,i)=jp;
        j(4:6,i)=zi_1;

end
end
```

Jacobin Singularities

- Singularity occur when determinant of Jacobian matrix is zero

- This happens in three cases:
    1. Robot is fully extended
    2. End effector on first joint axis
    3. Axis 4,6 are collinear

```
%This condition checks if there is simgularity on a given configuration
if det(newj(1:3,1:3))*det(newj(4:6,4:6))==0
    fprintf('This is a signularity!')
```

Matlab condition to check for singularity

Jacobin Singularities

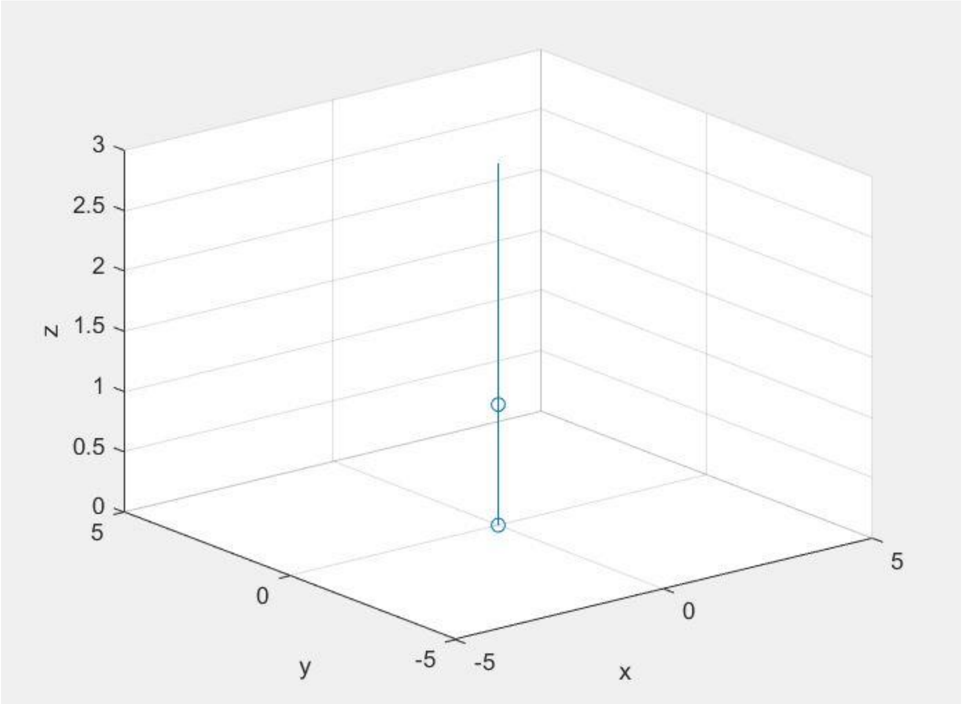
- In this robot model, Singulariy occur when theta_2 = 90 degrees which makes robot fully extended.

```
>> FK_with_Jacobian
This is a signularity!>> det(j)

ans =

     0
```

Code successfully detect the singularity with determinant = 0



Robot is fully extended (Case 1)

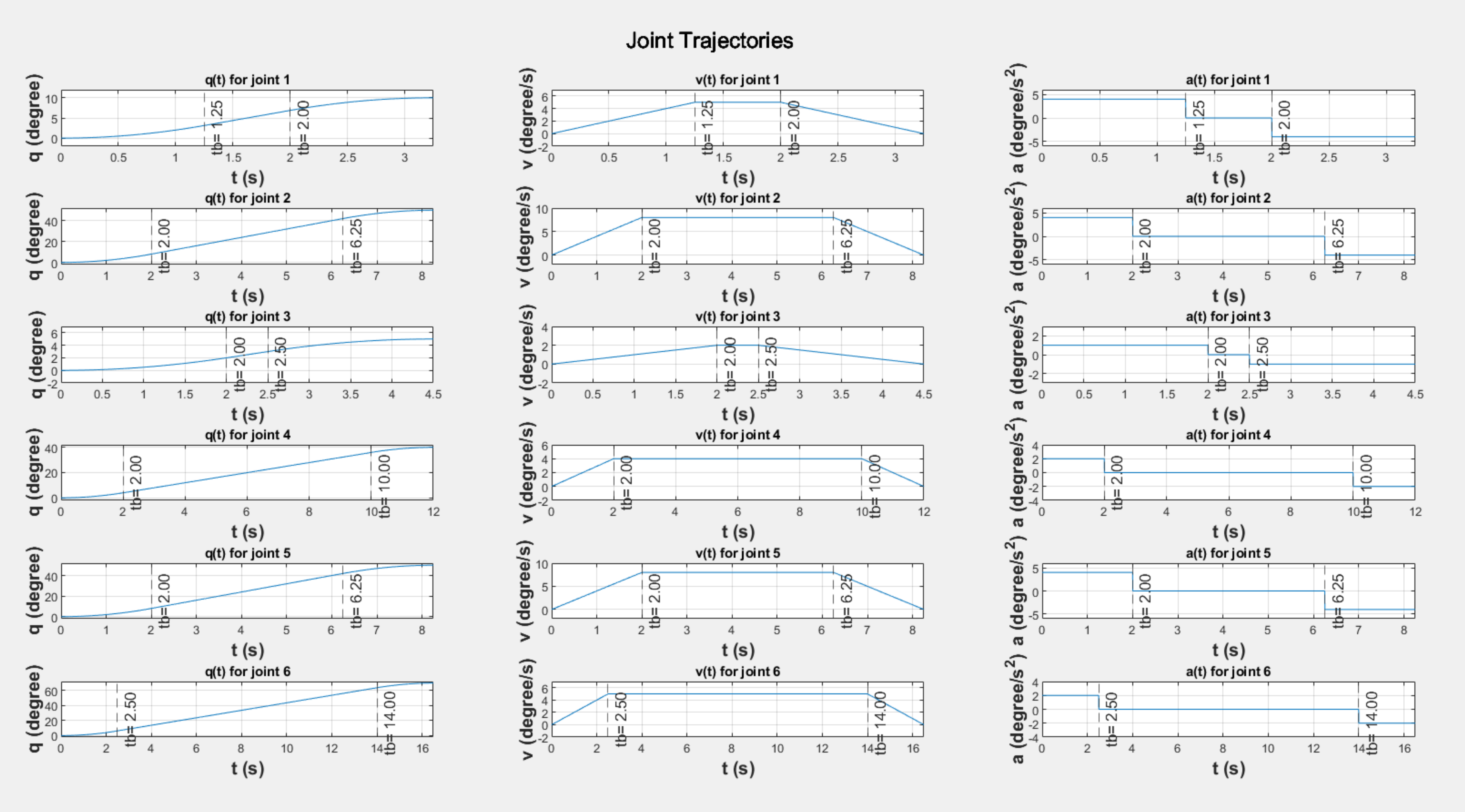
Part 3: Trajectory Planning

Joints Trajectories

- Position, velocity, and acceleration trajectories are plotted for each joint.
- The joint parameters are shown

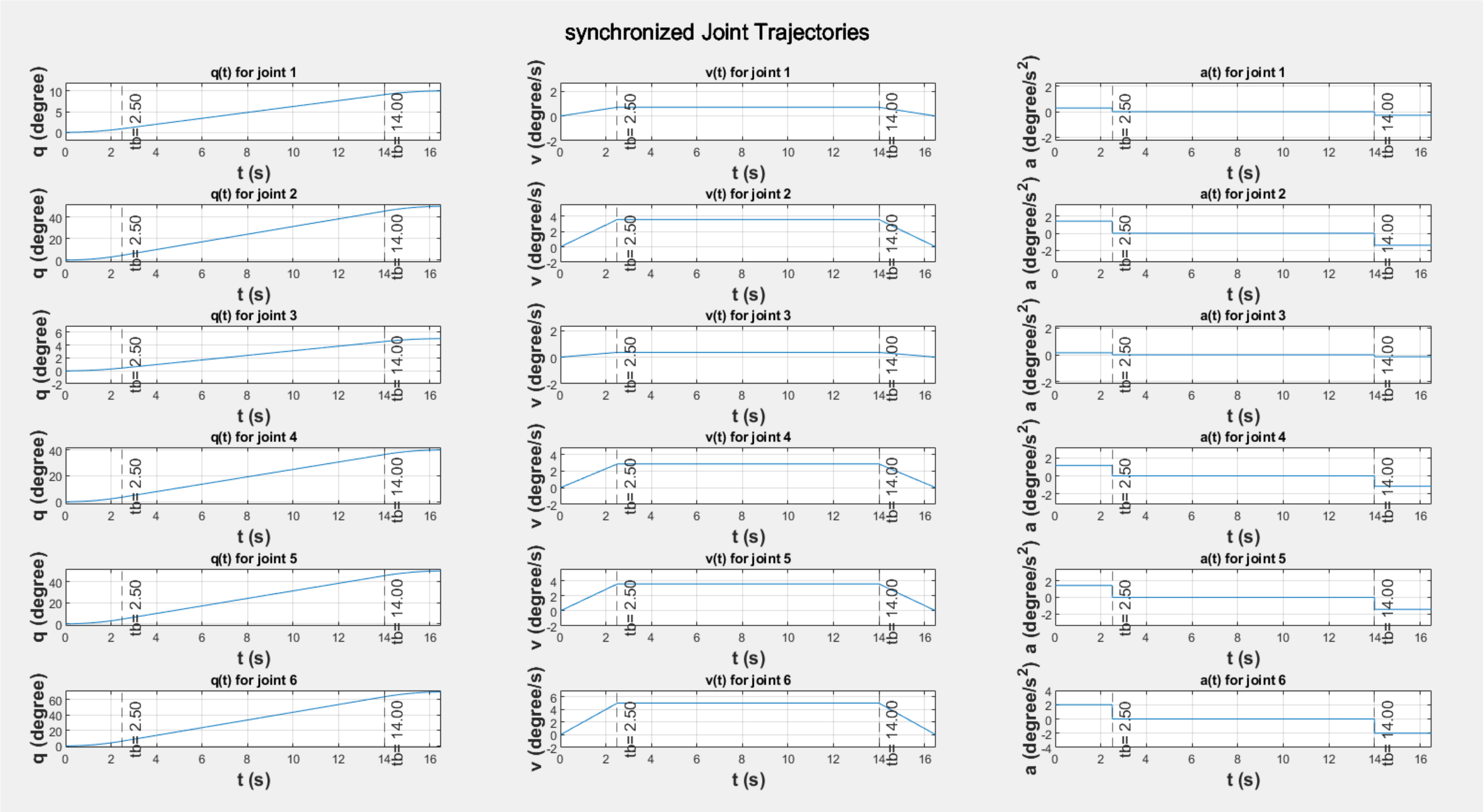
```
%%Enter joint parameters as [q0,qf,dq_m, ddq_m,]
j1 =[0,10,5,4];
j2=[0,50,8,4];
j3 =[0,5,2,1]; %prismatic joint
j4=[0,40,4,2];
j5 =[0,50,8,4];
j6=[0,70,5,2];
```

Joints Trajectories



Synchronized Joints Trajectories

Remark: All joints have now a maximum velocity and acceleration less than before due to the prolongation of execution time.



Numerical Joints Trajectories

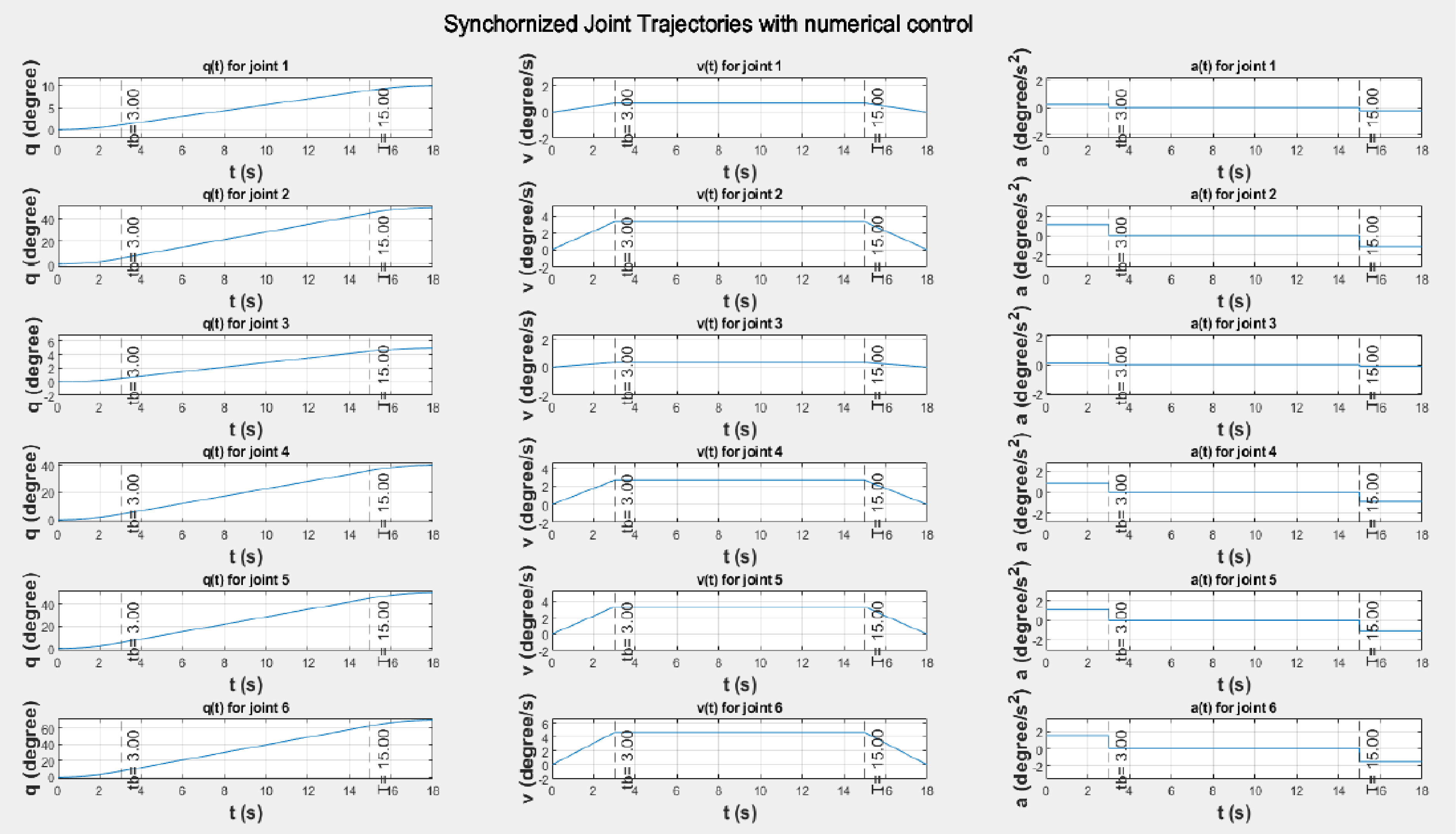
- Numerical control is needed because the robot controller acts at certain frequency

- The new numerical time can be calculated as

$$t_b = \left(\frac{t_b}{\Delta t} + 1\right)\Delta t$$

- The same goes for T

Numerical Joints Trajectories



Propagated Error

- The error is calculated between the end effector before and after numerical control
- Based on the arbitrary values entered, the error was 0

```
%Calculate the difference between x,y,z postion
x_error=Actual_FK(1:4)-Numerical_FK(1:4);
y_error=Actual_FK(2:4)-Numerical_FK(2:4);
z_error=Actual_FK(3:4)-Numerical_FK(3:4);

fprintf("\nError for end effector position is : %.2f in x and %.2f in y and %.2f in z\n",x_error,y_error,z_error)
```

```
Error for end effector position is : 0.00 in x and 0.00 in y and 0.00 in z
```

Part 4: Dynamics

Euler Lagrange

- **1) Center of mass equations**
- **2) Mass matrix M(q)**
- **3)Corriolis matrix C(q,dq)**
- **4) Gravity Matrix g**

- **Dynamics Equation:**

$$M(q)\begin{matrix} ddq1 \\ ddq2 \\ ddq3 \\ ddq4 \\ ddq5 \\ ddq6 \end{matrix} + C(q,dq)\begin{matrix} dq1 \\ dq2 \\ dq3 \\ dq4 \\ dq5 \\ dq6 \end{matrix} + g(q) = \tau$$

Euler Lagrange

- 1) Center of mass equations:
  - It was obtained using DH parameters
  - Equation for x,y,z were obtained

```
%This represents tranformation needed for each COM. first CoM is easy so it
%written directly in the next step
COM_02 = a1*rot_z2*trans_c2;
COM_03 = a1*a2*trans_c3;
COM_04 = a1*a2*a3*rot_z4*trans_c4;
COM_05 = a1*a2*a3*a4*rot_z5*trans_c5;
COM_06 = a1*a2*a3*a4*a5*rot_z6*trans_c6;
```

Euler Lagrange

- **2) Mass matrix M(q):**
  - **Linear Jacobian**
  - **Angular Jacobian**

$$M(q) = \sum_{i=1}^{n} m_i * \left(J_v^i\right)^T * J_v^i + \left(J_w^i\right)^T * R_i * I * (R_i)^T * J_w^i$$

```
syms m1 m2 m3 m4 m5 m6
M_q_lin = m1*transpose(jac_lin_1)*jac_lin_1+m2*transpose(jac_lin_2)*jac_lin_2+m3*transpose
M_q_ang = transpose(jac_ang_1)* rot_z1(1:3,1:3) *I* transpose(rot_z1(1:3,1:3))...
    *jac_ang_1+transpose(jac_ang_2)* rot_z2(1:3,1:3) *I* transpose(rot_z2(1:3,1:3))...
    *jac_ang_2+transpose(jac_ang_3)* rot_z3(1:3,1:3)*I * transpose(rot_z3(1:3,1:3))...
    *jac_ang_3+transpose(jac_ang_4)* rot_z4(1:3,1:3) *I* transpose(rot_z4(1:3,1:3))...
    *jac_ang_4+transpose(jac_ang_5)* rot_z5(1:3,1:3)*I* transpose(rot_z5(1:3,1:3))...
    *jac_ang_5+transpose(jac_ang_6)* rot_z6(1:3,1:3) *I* transpose(rot_z6(1:3,1:3))...
    *jac_ang_6;
M_q= M_q_lin + M_q_ang;
```

Euler Lagrange

- **3) Corilios C:**

$$c_{ij} = \sum_{k=1}^{n} c_{ijk} * \dot{q}_k$$

- **4) gravity g:**

$$g = -\sum_{k=1}^{n} \left(J_{v_i}^{k}\right)^T * m_k * g_0$$

Euler Lagrange Code

For the following parameters:

```
DH=[0 0 1 0 0 0];  %Enter the values for each joint parameter RRPRRR
L=[1 1 0 1 1 1];  %Robot lengthes
C=[.1 .1  .1 .1 .1]; %Location of each COM relative to the local origin
m = [10 10 10 10 10 10]; %mass of each link
dq=[2 ;2; 4 ;3; 1; 5];%joint velocity
ddq=[1 ;.5 ;1 ;.1 ;.5 ;.6];%joint acceleration
izz=1;  %izz valie
g_0=9.81; % gravity
```

The torques were obtained:

```
T =

 -735.4400
  198.2900
   40.0000
   74.4400
  -58.6200
    5.0200
```

Thank You