This assignment is concerned with the implementation of the *Inverse Kinematics Algorithm*.

The following exercises are solved with MATLAB and utilize the Robotics System Toolbox from Mathworks.

Before the assignment, read this document entirely and complete the quiz on Inverse Kinematics Control in the Moodle workspace of the course. In case you are unable to answer the questions in the quiz correctly go back to the lecture slides and please carefully study the chapters 3.7 on the inverse kinematic problem in the book by Siciliano et al [2]. Further recommended reading is the chapter on Kinematically redundant manipulators in the Springer Handbook of Robotics [1].

Exercise 1 : Questions and Answers

1) What is the underlying control law of inverse kinematic controls with the Jacobian inverse?

2) Which of the following statements about the inverse kinematic control scheme with Jacobian inverse are true?

   a) The residual tracking error asymptotically converges to zero.

   b) The residual tracking error remains bounded.

   c) The control law results in an asymptotically stable system.

   d) For a constant reference $\dot{\mathbf{x}}_d = \mathbf{0}$ the steady state error becomes zero.

3) How is the orientation error $\mathbf{e}_o$ represented in inverse kinematic control schemes?

4) What are the properties of a Lyapunov function?

5) What is the relationship between joint and task space acceleration (second order inverse kinematic control)?

6) What does the manipulability ellipsoid capture?

Exercise 2 : Inverse Kinematics Algorithm

Recall the assignment on Differential Kinematics in which the joint space velocities that correspond to a reference velocity in task space are obtained by inverting the Jacobian.

$$\dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})^{-1}\dot{\mathbf{x}}, \tag{1}$$

which rests upon the inversion of the Jacobian matrix. The differential equation (1) can be converted into an equivalent difference equation

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \mathbf{J}(\mathbf{q_k})^{-1}\dot{\mathbf{x}}_k\Delta t, \tag{2}$$

for numerical integration with an Euler step.

Notice that the resolved rate motion control scheme[1] is a purely open loop control. The actual and reference trajectory do not coincide due to the assumption of a constant Jacobian $\mathbf{J}(\mathbf{q})$ during the integration step in Eq. (2). In addition the scheme does not respond to disturbances such as friction that effect the joint trajectory. The idea of inverse kinematic control shown in figure 1 is to feedback the error between actual and reference pose and to correct the joint motion accordingly. The feedback control law for inverse kinematic control utilizing the **analytical** Jacobian $\mathbf{J}_A$ (w.r.t. world frame) is given by

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \mathbf{J}_A(\mathbf{q}_k)^{-1}(\boldsymbol{\nu}_k + \mathbf{K}\mathbf{e}_k)\Delta t \tag{3}$$

with the forward kinematics $\mathbf{x}_k = \mathbf{k}(\mathbf{q}_k)$, desired velocity
$\boldsymbol{\nu}_k = \dot{\mathbf{x}}_{d_k} = [\dot{\mathbf{p}}_{d_k}\ \dot{\boldsymbol{\phi}}_{d_k}]^T$,
the pose error $\mathbf{e}_k$ and a proper proportional gain factor $\mathbf{K}$. The pose error $\mathbf{e}_k = [\mathbf{e}_{p_k}\ \mathbf{e}_{o_k}]^T$ is composed of a position error $\mathbf{e}_{p_k} = \mathbf{x}_{d_k} - \mathbf{x}(\mathbf{q}_k)$ w.r.t. the three translational components and an orientation error $\mathbf{e}_{o_k} = \boldsymbol{\phi}_d - \boldsymbol{\phi}_e(\mathbf{q})$ in which $\boldsymbol{\phi}_d$ denotes the minimal representation w.r.t. to the vector of reference Euler angles and $\boldsymbol{\phi}_e(\mathbf{q})$ the actual Euler angles of the end effector frame. Usually $\boldsymbol{\phi}_e(\mathbf{q})$ is not available in closed analytical form but requires computation of the end effector frame rotation matrix $\mathbf{R}_e$ or transform $\mathbf{H}_e$ from forward kinematics. Notice, that the analytical Jacobian $\mathbf{J}_A(\mathbf{q}) = \frac{\partial \mathbf{k}(\mathbf{q})}{\partial \mathbf{q}}$ differs from the geometric Jacobian.

Figure 1 shows the block scheme of the inverse kinematics algorithm.

$$\dot{\mathbf{q}} = \mathbf{J}_A(\mathbf{q})^{-1}(\dot{\mathbf{x}}_d + \mathbf{K}\mathbf{e}) \tag{4}$$

in which $\mathbf{K}$ is a positive definite, usually diagonal gain matrix, and $\mathbf{k}(\cdot)$ denotes the forward kinematics function. This scheme falls into the realm of nonlinear control, $\mathbf{k}(\cdot)$ is required to calculate the task space error $\mathbf{e}$ and $\mathbf{J}_A(\mathbf{q}(t_i))^{-1}$ linearizes the system as it compensates for $\mathbf{J}_A(\mathbf{q}(t_i))$. This leads to the equivalent linear system

$$\dot{\mathbf{e}} + \mathbf{K}\mathbf{e} = 0 \tag{5}$$

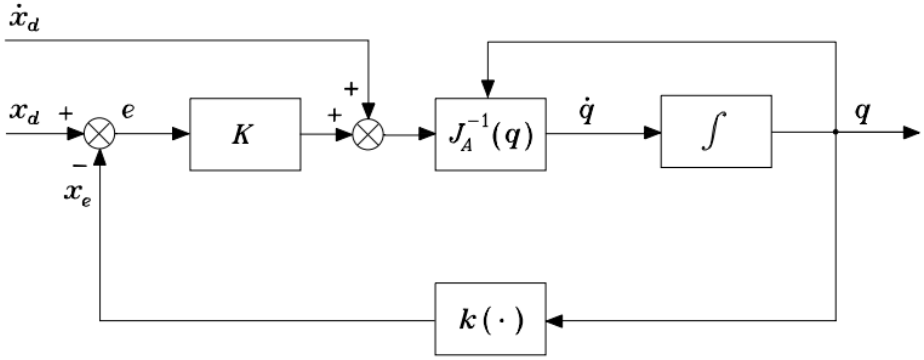

Figure 1: Inverse kinematic control, $\dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})^{-1}(\dot{\mathbf{x}}_d + \mathbf{K})\mathbf{e}$ [2]

which convergence rate to the stable equilibrium $\mathbf{e} = 0$ depends on the eigenvalues of $\mathbf{K}$.

Inverse Kinematic Control with Pseudoinverse for Redundant Manipulators

In case of a redundant manipulator Eq. (4) is generalized to

$$\dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})^{\dagger}(\dot{\mathbf{x}}_d + \mathbf{K}\mathbf{e}) + (\mathbf{I}_n - \mathbf{J}(\mathbf{q})^{\dagger}\mathbf{J}(\mathbf{q}))\dot{\mathbf{q}}_0 \tag{6}$$

in which $\mathbf{K}$ is a positive definite, usually diagonal gain matrix. The n-by-m matrix $\mathbf{J}^{\dagger}$ is the pseudoinverse of $\mathbf{J}$, sometimes called the Moore-Penrose inverse. The pseudoinverse is defined even for none square matrices or matrices that do not have full rank. The pseudoinverse gives the best possible solution

$$\dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})^{\dagger}\dot{\mathbf{x}} \tag{7}$$

to the equation

$$\mathbf{J}(\mathbf{q})\dot{\mathbf{q}} = \dot{\mathbf{x}} \tag{8}$$

in the sense of least squares. Furthermore, $\dot{\mathbf{q}}$ is the unique vector of smallest magnitude which minimizes the equation error norm $||\mathbf{J}(\mathbf{q})\dot{\mathbf{q}} - \dot{\mathbf{x}}||$.

The pseudoinverse has the property that the matrix $(\mathbf{I}_n - \mathbf{J}(\mathbf{q})^{\dagger}\mathbf{J}(\mathbf{q}))$ is a projector onto the nullspace of $\mathbf{J}$. Therefore, $(\mathbf{I}_n - \mathbf{J}(\mathbf{q})^{\dagger}\mathbf{J}(\mathbf{q}))\dot{\mathbf{q}}_0 = \mathbf{0}$ for all joint velocities $\dot{\mathbf{q}}_0$ This allows to add the term $(\mathbf{I}_n - \mathbf{J}(\mathbf{q})^{\dagger}\mathbf{J}(\mathbf{q}))\dot{\mathbf{q}}_0$ to $\mathbf{J}(\mathbf{q})^{\dagger}(\dot{\mathbf{x}}_d + \mathbf{K}\mathbf{e})$ and still retain a joint velocity that minimizes $||\mathbf{J}(\mathbf{q})\dot{\mathbf{q}} - (\dot{\mathbf{x}}_d + \mathbf{K}\mathbf{e})||$

This nullspace method is exploited to achieve secondary goals in addition to the primary task of the end effector tracking the reference poses. Secondary objectives are to achieve joint angles that are close to center or home configuration in order to avoid singularities.

[1] See Assignment on Differential Kinematics, Tasks 11.15.
[2] source: Robotics : Modelling, Planing and Control, Bruno Siciliano et al, Springer 2008

The pseudoinverse tends to instability problems in the vicinity of singularities. The Jacobian looses rank such that the end effector no longer achieves some direction of movement. For joint configurations close to a singularity, the control law from Eq. (6) generates large movements in joint space, even for small movements in the task space.

The proximity of a singularity is measured by the volume of the manipulability ellipsoid

$$w(\mathbf{q}) = \sqrt{\det(\mathbf{J}^{\dagger}(\mathbf{q})\mathbf{J}(\mathbf{q}))}\,. \tag{9}$$

The condition number of the Jacobian $\mathrm{cond}(\mathbf{J})$ measures the ratio λ_1/λ_n of the largest to the smallest generalized eigenvalue λ_i of $\mathbf{J}$.

One option is to choose $\dot{\mathbf{q}}_0$ along the gradient of a cost function for example

$$\dot{\mathbf{q}}_0 = \eta \nabla_{\mathbf{q}} w(\mathbf{q}) \tag{10}$$

to maximize the manipulability ellipsoid or

$$\dot{\mathbf{q}}_0 = -\eta \nabla_{\mathbf{q}} \,\mathrm{cond}(\mathbf{J}(\mathbf{q})) \tag{11}$$

as to minimize the condition number.

Inverse Kinematic Control with Damping

The damped least squares method avoids the problems of the pseudoinverse algorithm with singularities. It determines the joint velocity in a numerically stable manner. The damped least squares method rather than determining the minimum solution to equation (8), determines $\dot{\mathbf{q}}$ such that it minimizes

$$||\mathbf{J}(\mathbf{q})\dot{\mathbf{q}} - \dot{\mathbf{x}}||^2 + \lambda^2 ||\dot{\mathbf{q}}||^2 \tag{12}$$

where λ denotes a damping constant. The least squares normal equations become

$$(\mathbf{J}(\mathbf{q})^T\mathbf{J}(\mathbf{q}) + \lambda^2\mathbf{I})\dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})^T\dot{\mathbf{x}} \tag{13}$$

As $(\mathbf{J}(\mathbf{q})^T\mathbf{J}(\mathbf{q}) + \lambda^2\mathbf{I})$ is non-singular and square, the damped least squares solution becomes

$$\dot{\mathbf{q}} = (\mathbf{J}(\mathbf{q})^T\mathbf{J}(\mathbf{q}) + \lambda^2\mathbf{I})^{-1}\mathbf{J}(\mathbf{q})^T\dot{\mathbf{x}} \tag{14}$$

In the case of inverse kinematic control algorithm the task space velocity $\dot{\mathbf{x}}$ is composed of the feedforward reference velocity and the error scaled with the gain $\mathbf{K}$ according to figure 1.

$$\dot{\mathbf{q}} = (\mathbf{J}(\mathbf{q})^T\mathbf{J}(\mathbf{q}) + \lambda^2\mathbf{I})^{-1}\mathbf{J}(\mathbf{q})^T(\dot{\mathbf{x}}_d + \mathbf{K}\mathbf{e}) \tag{15}$$

The damping constant λ needs to be large enough to ensure low joint velocities in the vicinity of singularities. On the other hand large values λ increase the tracking error.

Inverse Kinematic Control with Geometric Jacobian and Angle and Axis Orientation

The inverse kinematic control can also be defined utilizing the **geometric** Jacobian in order to avoid singularities of Euler angles.

$$\mathbf{q}(t_{i+1}) = \mathbf{q}(t_i) + \mathbf{J}_w(\mathbf{q}(t_i))^{-1} \begin{bmatrix} \dot{\mathbf{p}}_d(t_i)\Delta t + \mathbf{K}_p \mathbf{e}_p(t_i) \\ \mathbf{L}^{-1}\Big(\mathbf{L}^T \boldsymbol{\omega}_d(t_i)\Delta t + \mathbf{K}_o \mathbf{e}_o(t_i)\Big) \end{bmatrix}. \tag{16}$$

The velocity $\boldsymbol{\nu}(t_i) = [\dot{\mathbf{p}}_d(t_i)\, \boldsymbol{\omega}_d(t_i)]^T$ for the orientation part is here given by the angular velocity at each axis $\boldsymbol{\omega}_d$ instead of Euler angles $\dot{\boldsymbol{\phi}}_d(t_i)$. $\mathbf{J}_w$ denotes the **geometric** Jacobian w.r.t. the world frame.

If $\mathbf{R}_d$ denotes the desired rotation matrix of the end effector and $\mathbf{R}$ the rotation matrix computed from the forward kinematics of the actual joint variables, the orientation error between both frames is expressed in angle and axis notation by $\mathbf{e}_o = \mathbf{r} \sin\vartheta$ where ϑ and $\mathbf{r}$ denote the angle and axis of the relative rotation matrix

$$\mathbf{R}(\vartheta, \mathbf{r}) = \mathbf{R}_d \mathbf{R}(\mathbf{q})^T. \tag{17}$$

Matrix $\mathbf{L}$ is computed by

$$\mathbf{L} = -\frac{1}{2}\Big(\mathbf{S}(\mathbf{n}_d)\mathbf{S}(\mathbf{n}) + \mathbf{S}(\mathbf{s}_d)\mathbf{S}(\mathbf{s}) + \mathbf{S}(\mathbf{a}_d)\mathbf{S}(\mathbf{a})\Big) \tag{18}$$

in which $\mathbf{n}, \mathbf{s}$ and $\mathbf{a}$ denote the column vectors of a rotation matrix $\mathbf{R} = [\mathbf{n}\ \mathbf{s}\ \mathbf{a}]$. $\mathbf{S}(\mathbf{w}) \in \mathbb{R}^{3\times 3}$ denotes the skew symmetric matrix of a vector $\mathbf{w} = [w_x, w_y, w_z]^T$:

$$\mathbf{S}(\mathbf{w}) = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix}. \tag{19}$$

Exploiting the definition of the geometric Jacobian

$$\dot{\mathbf{p}}_e = \mathbf{J}_p(\mathbf{q})\dot{\mathbf{q}} \tag{20}$$
$$\boldsymbol{\omega}_e = \mathbf{J}_o(\mathbf{q})\dot{\mathbf{q}} \tag{21}$$

and expressing $\dot{\mathbf{p}}_e$ and $\boldsymbol{\omega}_e$ as a function of $\dot{\mathbf{q}}$ one obtains

$$\dot{\mathbf{e}} = \begin{bmatrix} \dot{\mathbf{e}}_p \\ \dot{\mathbf{e}}_o \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{p}}_d - \mathbf{J}_p(\mathbf{q})\dot{\mathbf{q}} \\ \mathbf{L}^T\boldsymbol{\omega}_d - \mathbf{L}\mathbf{J}_o(\mathbf{q})\dot{\mathbf{q}} \end{bmatrix}. \tag{22}$$

which realizes an inverse kinematics algorithm based on the geometric in lieu of the analytical Jacobian. The inverse kinematic control law becomes

$$\dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})^{-1} \begin{bmatrix} \dot{\mathbf{p}}_d + \mathbf{K}_p \mathbf{e}_p \\ \mathbf{L}^{-1}(\mathbf{L}^T\boldsymbol{\omega}_d + \mathbf{K}_o \mathbf{e}_o) \end{bmatrix}. \tag{23}$$

Inverse Kinematic Control with Geometric Jacobian and Quaternion

The unit quaternion is suitable representation of the orientation error $\mathbf{e}_o$. In fact this representation results in a formulation of inverse kinematics algorithms more elegant than Euler or angle and axis representation. Assume the unit quaternions $\mathcal{Q}_d = \{\eta_d, \boldsymbol{\epsilon}_d\}$ and $\mathcal{Q}_e = \{\eta_e, \boldsymbol{\epsilon}_e\}$ that correspond to the rotation matrices $\mathbf{R}_d$ and $\mathbf{R}_e$ of the reference and end effector frame. The rotation matrix $\mathbf{R}_d\mathbf{R}_e^T$ denotes the orientation error between both frames with a corresponding quaternion error

$$\Delta\mathcal{Q} = \{\Delta\eta, \Delta\boldsymbol{\epsilon}\} = \mathcal{Q}_d * \mathcal{Q}_e^{-1}. \tag{24}$$

This notation allows the definition of the orientation error in terms of the vector part of $\Delta\mathcal{Q}$

$$\mathbf{e}_o = \Delta\boldsymbol{\epsilon} = \eta_e(\mathbf{q})\boldsymbol{\epsilon}_d - \eta_d\boldsymbol{\epsilon}_e(\mathbf{q}) - \mathbf{S}(\boldsymbol{\epsilon}_d)\boldsymbol{\epsilon}_e(\mathbf{q}) \tag{25}$$

with the skew-symmetric cross product operator $\mathbf{S}(\boldsymbol{\epsilon}_d)\boldsymbol{\epsilon}_e(\mathbf{q}) = \boldsymbol{\epsilon}_d \times \boldsymbol{\epsilon}_e(\mathbf{q})$ (see Eq. (19))

The inverse kinematic control law with quaternion orientation error becomes

$$\dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})^{-1} \begin{bmatrix} \dot{\mathbf{p}}_d + \mathbf{K}_p\mathbf{e}_p \\ \boldsymbol{\omega}_d + \mathbf{K}_o\mathbf{e}_o \end{bmatrix}. \tag{26}$$

The evolution of the orientation error is described by the nonlinear equation

$$\boldsymbol{\omega}_d - \boldsymbol{\omega}_e + \mathbf{K}_o\mathbf{e}_o = \mathbf{0} \tag{27}$$

and governed by the end-effector angular velocities $\boldsymbol{\omega}_d, \boldsymbol{\omega}_e$ instead of the time derivative of Euler angles. The inverse kinematics solution in Eq. (26) is based on the same geometric Jacobian as Eq. (23) but is computationally simpler.

Matlab Programming Assignment Inverse Kinematic Control

In the following you are going to implement the inverse kinematic control based on the geometric Jacobian with quaternion error representation for UR10 robot arm.

The end effector is supposed to describe a motion in task space that is composed of four straight line segments that connect the corners $c_1 = [0.2 \ -0.4 \ 0.0]$, $c_2 = [0.2 \ 0.4 \ 0.2]$, $c_3 = [0.8 \ 0.4 \ 0.0]$, $c_4 = [0.8 \ -0.4 \ 0.2]$. Each corner is associated with a reference orientation defined in terms of roll-pitch-yaw angles. The trajectory exhibits a triangular velocity profile along each segment, with zero velocity at the start and end point of a segment and maximum velocity half way through the segment. This guarantees a continuous reference velocity profile. The acceleration profile is piecewise constant but exhibits jerk at the start, end and midpoint of segments. The triangular velocity profile implies a quadratic profile of position and orientation. The script `generateReferenceTrajectory` generates a sequence of transforms `tformxd` of the reference end effector frame and a vector of reference velocities `xdotd` ($\dot{\mathbf{x}}_\mathbf{d} = [\omega_x \ \omega_y \ \omega_z \ \dot{p}_x \ \dot{p}_y \ \dot{p}_z]$). The time interval between frames

is $\Delta t \approx 0.05$ seconds and the overall duration of the motion is $t \in [0,\ 8]$ seconds with 164 time stamps. `tformxd` is a 4 by 4 by 164 array of transforms, `xdotd` is a 6 by 164 array of reference velocities. The script employs the `slerp` algorithm (see lab on Spatial Transformation) to generate the quaternions of intermediate frame orientations. Make sure that the folder for the quaternion object class is included in your Matlab path. The function `OmegaAxis` generates the instantaneous axis and angle of rotation for a sequence of quaternions. The function `plotTaskSpaceTrajectory(tformx)` plots a sequence of frames defined by the array of transforms `tformx`. Figure 2 illustrates the frames `tformxd` of the end effector closed path. Two consecutive frames are separated by a time interval of $\Delta t \approx 0.05$. Notice, that due to the quadratic pose profile the spatial density of frames is higher at the start and end point of the segment at which the velocities tends to zero.

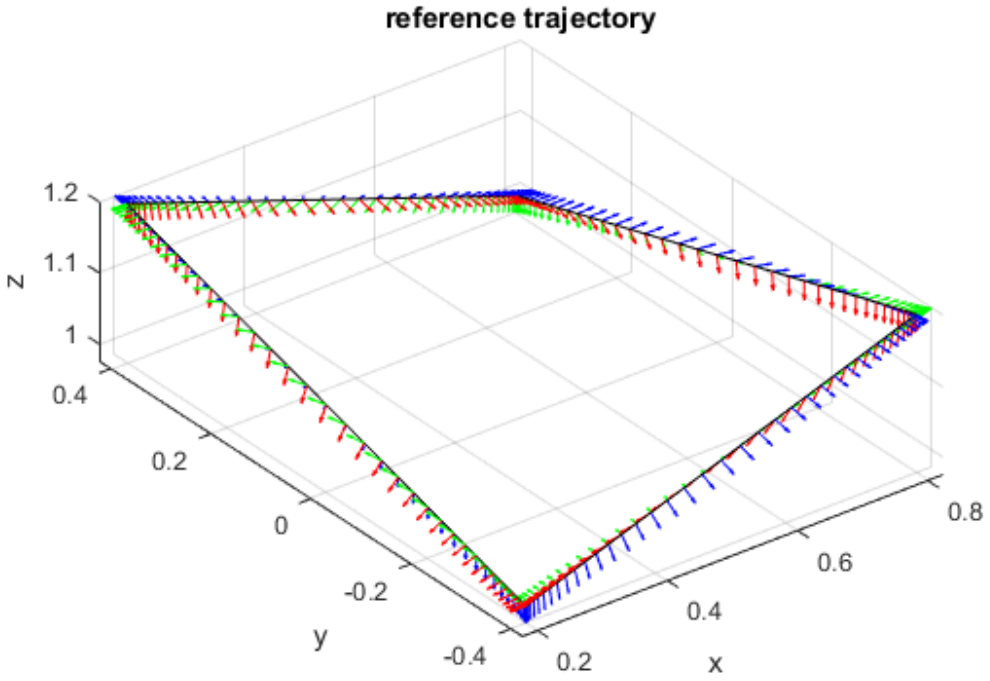

Figure 2: Sequence of reference frames of end effector motion

The script generates and plots the reference trajectory with the following code. The parameter `zoffset` defines the elevation of the base frame above ground and should be set to 1 meter for the UR10 robot and to 0 for the Sawywer robot such that the reference trajectory is located within the robots reachable workspace.

```
% elevation of base frame above ground
if ~exist('zoffset')
  zoffset=1; % UR 10 robot
end
tfinal=8.0;
```

```
ntimestamps=4*41; % number of time stamps, multiple of 4
t=linspace(0,tfinal,ntimestamps);
deltat=tfinal/ntimestamps;
ind=linspace(0,1,ntimestamps/4);

tri=zeros(1,size(ind,2));
qua=zeros(1,size(ind,2));
% triangular velocity profile
tri(1:(size(ind,2)+1)/2)=4*ind(1:(size(ind,2)+1)/2)/(tfinal/4);
tri((size(ind,2)+1)/2:end)=(4-4*ind((size(ind,2)+1)/2:end))/(tfinal/4);
% quadratic position profile
qua(1:(size(ind,2)+1)/2)=2*ind(1:(size(ind,2)+1)/2).^2;
qua((size(ind,2)+1)/2:end)=1-2*(1-ind((size(ind,2)+1)/2:end)).^2;

tformxd=zeros(4,4,size(t,2));
xdotd=zeros(6,size(t,2));

% four straight line segments on opposite corners of a cuboid
% corners of the square in task space coordinates c1, c2, c3, c4
% orientations at the corners RPY [0 pi 0] [pi/2 pi/ 0] [pi/2 pi pi/2] [0 pi/2 pi/2]
c = struct('p',[],'q',[]);
c(1).p=[0.2  -0.4 0.0+zoffset]; c(1).q=quaternion(eul2quat([0 pi 0]));
c(2).p=[0.2  0.4 0.2+zoffset]; c(2).q=quaternion(eul2quat([pi/2 pi/2 0]));
c(3).p=[0.8  0.4 0.0+zoffset]; c(3).q=quaternion(eul2quat([pi/2 pi pi/2]));
c(4).p=[0.8  -0.4 0.2+zoffset]; c(4).q=quaternion(eul2quat([0 pi/2 pi/2]));
c(5)=c(1);

nsegments=size(c,2)-1;

for n=1:nsegments
qslerp= slerp(c(n).q,c(n+1).q,qua);
[omega, axis]=OmegaAxis(qslerp,linspace(0,1,size(qua,2)));
for k=1:size(t,2)/4
tformxd(:,:,k+(n-1)*floor(size(t,2)/4))=trvec2tform((c(n+1).p-c(n).p)*qua(k))*
    trvec2tform(c(n).p)...
*quat2tform(double(qslerp(k))');
end
offset=(n-1)*floor(size(t,2)/4);
xdotd(1:3,(1+offset):(size(t,2)/4+offset))=reshape(axis,3,41).*omega(1,:)*(4/tfinal);
xdotd(4:6,(1+offset):(size(t,2)/4+offset))=[c(n+1).p-c(n).p]'*tri;
end
```

Two helper functions for conversion between joint vector and joint configuration structure are provided. `JointVec2JointConf(robot,q)` converts a the joint vector `q` into a joint configuration structure array with fields `JointName` and `JointPosition` employed by many functions of the Robotics System Toolbox. `JointConf2JointVec(configuration)` converts the joint configuration structure array `configuration` into an ordinary vector.

Now, please complete tasks 1 to 14 in the UR10-template and tasks 1 to 11 in the Sawyer-template.

Notes:

To switch from position controller to velocity controller in task 11 in UR10-template send a switch service command on the following topic:

```
rosservice call /ur10/controller_manager/switch_controller "start_controllers:
- ['joint0_velocity_controller','joint1_velocity_controller','
    joint2_velocity_controller','joint3_velocity_controller','
    joint4_velocity_controller','joint5_velocity_controller']
stop_controllers:
- ['vel_based_pos_traj_controller']
strictness: 1"
```

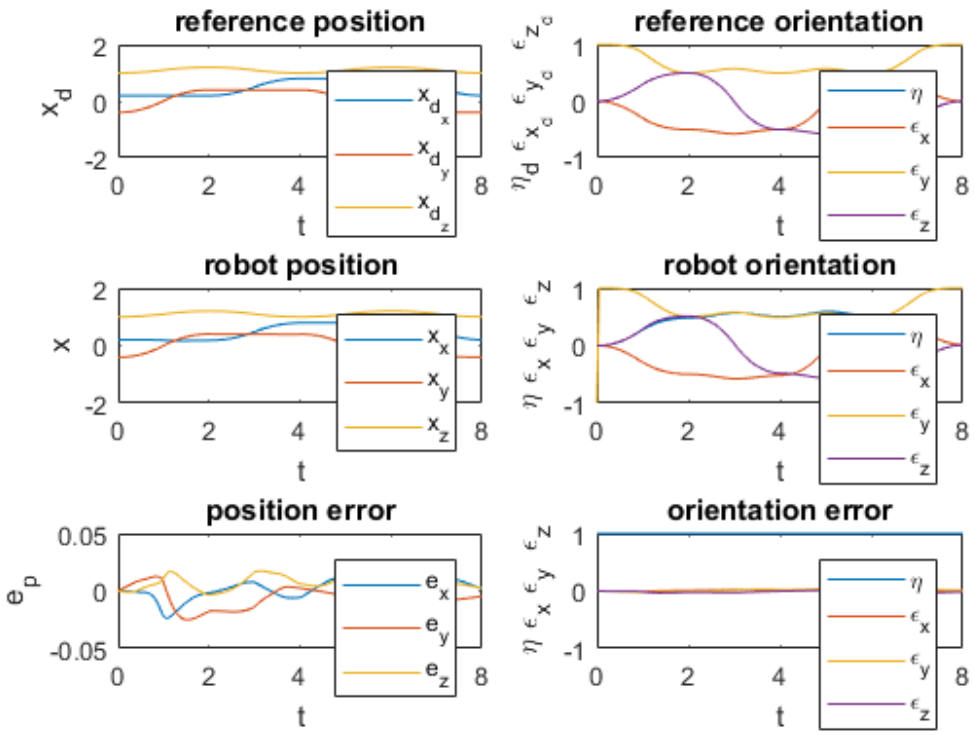

Figure 3: Reference task space position and orientation, end effector position and orientation, task space position and orientation error

.

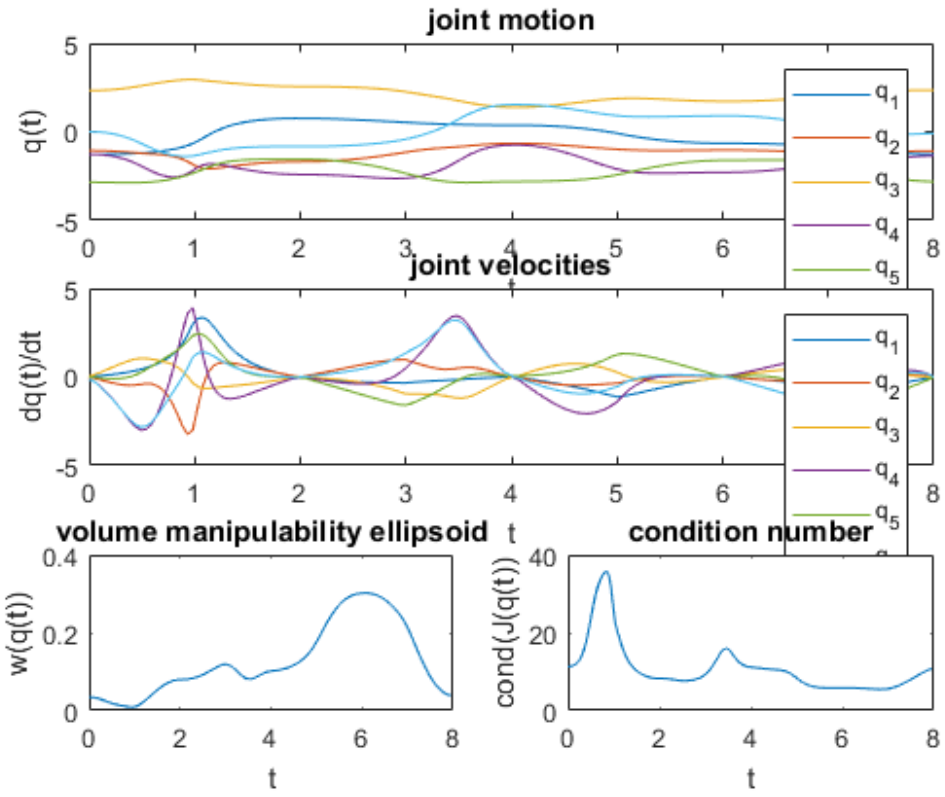

Figure 4: Joint motion for inverse kinematic control, volume manipulability ellipsoid, condition number of Jacobian

References

[1] Stefano Chiaverini, Giuseppe Oriolo, and Ian D. Walker. Kinematically redundant manipulators. In Bruno Siciliano and Oussama Khatib, editors, *Springer Handbook of Robotics*, pages 245–265. Springer, 2008.

[2] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 1st edition, 2008.