This assignment is concerned with the mathematical description and composition of spatial transformations.

The following exercises are solved with MATLAB and utilize the Robotics System Toolbox from Mathworks. Some tasks involve the use of ROS on a Linux (Ubuntu) system which is also available in the Retina Pool. Note, Matlab Linux uses different keyboard shortcuts per default (copy, paste, etc.). You might change it at *MATLAB - Preferences - Keyboard - Shortcuts - Active Settings: Windows Default Set.*

Prior to programming the assignment, read this document entirely and complete the quiz on Spatial Transformations in the Moodle workspace. In case you are unable to answer the questions in the quiz revisit the lecture slides and study the chapters 2.1 – 2.7 on spatial representations in the book by Siciliano et al [1]. Further recommended reading is the section on position and orientation representation in the chapter on kinematics in the Springer Handbook of Robotics [2].

Questions and Answers

Please first complete the Quiz on Spatial Transformations in the Moodle workspace.

1) Specify the number of parameters and constraints for the following representations of rotations:

   a) Rotation matrix
   b) Euler angles
   c) Roll pitch yaw angles
   d) Unit quaternions
   e) Angle axis representations

   Which of these representations exhibit a singularity?

2) Which statements about rotation matrices are true

   a) $\mathbf{R} = \mathbf{R}^T$
   b) $\mathbf{R}^T\mathbf{R} = \mathbf{I}$
   c) $\det(\mathbf{R}) = 1$
   d) $\mathbf{R}_a\mathbf{R}_b = \mathbf{R}_b\mathbf{R}_a$

3) Denote the elementary rotation matrix $\mathbf{R}_z$ for a rotation along the z-axis by α.

4) Given two rotations $\mathbf{R}_1^0$ and $\mathbf{R}_2^1$. What is the composition $\mathbf{R}_2^0$ of these rotations with respect to the current frame and fixed frame.

5) What is the number of independent parameters of a homogeneous transformation matrix?

6) Denote the homogeneous transformation matrix associated with the rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$.

7) Denote the inverse of the homogeneous transformation matrix in question 6).

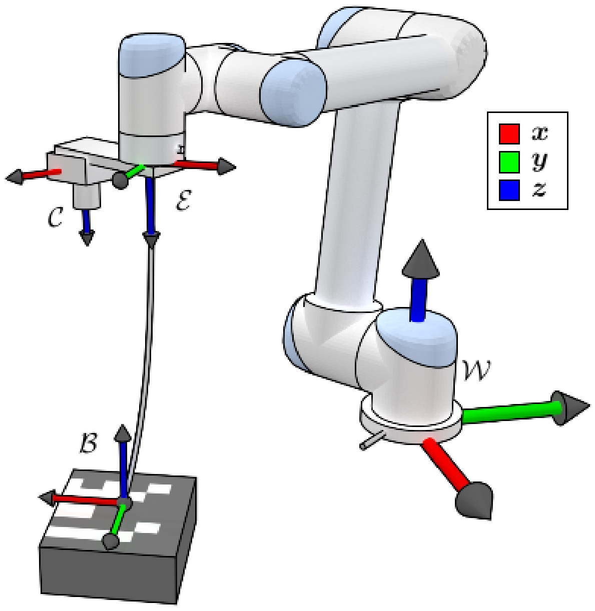

Figure 1: Coordinate frames in robotics. [1]

Spatial Transformations

Robot kinematics establishes the transformations among various coordinate frames that capture the positions and orientations of end-effector, links, rigid bodies and cameras as shown in figure 1. In this scenario $\mathcal{W}$ denotes the robot base frame or world frame, $\mathcal{E}$ the robot end effector frame, $\mathcal{C}$ the camera frame and $\mathcal{B}$ the object frame. The mathematics of spatial transformations that describe rigid motions in Euclidean space is of fundamental importance to robotic manipulation.

The purpose of this assignment is to get familiar with representations of spatial transformations. A recommended reading is the section on position and orientation representation in the chapter on kinematics in the Springer Handbook of Robotics [2] or chapters $2.1-2.7$ in the book by Siciliano et al [1]. Please carefully read these sections prior to the lab.

The mathematics of spatial transformations in Euclidean space is of fundamental importance to robotics. ROS and the Matlab Robotic toolbox support multiple representations of rotations and spatial transformations

- homogeneous transformations (4x4 matrix)
- rotation matrix (3x3 matrix)
- quaternions

[1]Source: Torsteins Webpage Crane Control NTU

- angle and vector orientation
- Euler angles
- roll, pitch, yaw angles

Translations

A general rigid body motion is composed of a translation and a rotation. A translation in Euclidean space is represented by the components of an ordinary 3D vector $p = [p_x\ p_y\ p_z]$ in which the vector $\mathbf{p} = p_x\mathbf{x} + p_y\mathbf{y} + p_z\mathbf{z}$. $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are the unit vectors of the frame axes of an orthogonal frame $O - xyz$. Successive translations are obtained by mere vector addition $\mathbf{p}_{02} = \mathbf{p}_{01} + \mathbf{p}_{12}$.

Matlab employs the abbreviation `trvec` for a translation vector which is represented in 3-D Euclidean space as Cartesian coordinates. Its numeric Representation is a 1-by-3 vector. For example, a translation by 3 units along the x -axis and 2.5 units along the z -axis is expressed as:

```
trvec = [3 0 2.5];
```

Successive translations are composed by mere vector algebra.

```
trvec1 = [2 1 0];
trvec2 = [0 1 1];
trvec12=trvec1+trvec2;
clf;
hold on;
quiver3(0,0,0,trvec1(1),trvec1(2),trvec1(3),0);
text(1,0.5,0,'$trvec_1$','Interpreter','LaTeX');
quiver3(trvec1(1),trvec1(2),trvec1(3), trvec2(1), trvec2(2), trvec2(3),0);
text(2,1.5,0.5,'$trvec_2$','Interpreter','LaTeX');
quiver3(0,0,0,trvec12(1),trvec12(2),trvec12(3),0);
text(1,1,0.5,'$trvec_{12}$','Interpreter','LaTeX');
axis equal;
view(3);
grid on;
xlabel('x');
ylabel('y');
zlabel('z');
```

Rotation Matrices Rotations are represented by 3×3 rotation matrices

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

which entries $\mathbf{R}_{ij}$ correspond to the direction cosines of the axes of frame $O - xyz$ w.r.t. frame $O' - x'y'z'$, namely

$$\begin{aligned} r_{11} &= \mathbf{x}'^T\mathbf{x} = \cos(\angle\mathbf{x}', \mathbf{x}) \\ r_{12} &= \mathbf{y}'^T\mathbf{x} = \cos(\angle\mathbf{y}', \mathbf{x}) \\ r_{13} &= \mathbf{z}'^T\mathbf{x} = \cos(\angle\mathbf{z}', \mathbf{x}) \\ \ldots &= \ldots \\ r_{33} &= \mathbf{z}'^T\mathbf{z} = \cos(\angle\mathbf{z}', \mathbf{z}) \end{aligned}$$

in which $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{x}', \mathbf{y}', \mathbf{z}'$ denote the unit vectors along the corresponding axes.

A rotation matrix $\mathbf{R}$ not only describes the relative orientation between frames but also performs a rotation of a vector in Euclidean space

$$\mathbf{p}' = \mathbf{R}\mathbf{p} \tag{1}$$

The matrix

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

rotates vectors counter-clockwise through an angle θ about the z-axis of the Cartesian coordinate system. To perform the rotation using a rotation matrix $\mathbf{R}$, the position of a point in space is represented by a 1-by-3 column vector $\mathbf{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T$, that contains the coordinates of the point w.r.t. an $X - Y - Z$ frame. A rotated vector is obtained by the matrix multiplication $\mathbf{R}\mathbf{p}$. Coordinate rotations are a natural way to express the orientation of a robot end effector, robot link or camera relative to a reference frame e.g. the robot base frame or a world frame. Once XYZ axes of a local frame are expressed numerically as three direction vectors w.r.t. a global world frame, they together comprise the columns of the rotation matrix $\mathbf{R}$ that transforms vectors in the reference frame into equivalent vectors expressed in the coordinates of the local frame. The inverse of rotation matrix coincides with its transpose, namely $\mathbf{R}^{-1} = \mathbf{R}^T$. Rotation matrices are square matrices that obey the orthogonality constraints: $\mathbf{R}^T = \mathbf{R}^{-1}$ and $\det(\mathbf{R}) = 1$.

A basic or elemental rotation is a rotation about one of the axes of a Cartesian coordinate frame (see figure 2). The three basic rotation matrices rotate vectors by an angle θ about the x, y, or z axis in three dimensions using the right hand rule.

$$\begin{aligned}
\mathbf{R}_x(\theta) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \\
\mathbf{R}_y(\theta) &= \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \\
\mathbf{R}_z(\theta) &= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

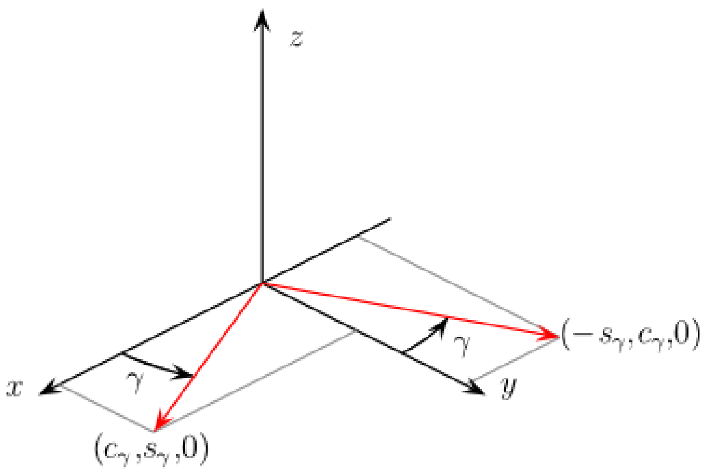

Figure 2: Rotation by an angle γ along the z-axis [2]

Matlab employs the abbreviation `rotm` for a rotation matrix. It is a square, orthonormal matrix with a determinant of 1. Its numeric representation is a 3-by-3 matrix. For example, elemental rotations of $\pi/2$ radians along x, y, z are generated with the commands:

```
theta = pi/2;
% elemental rotation about the x-axis
rotmx = [1 0 0; 0 cos(theta) -sin(theta); 0 sin(theta) cos(theta)];
% elemental rotation about the y-axis
rotmy = [cos(theta) 0 sin(theta) 0; 0 1 0; -sin(theta) 0 cos(theta)];
% elemental rotation about the z-axis
rotmz = [cos(theta) -sin(theta) 0; sin(theta) cos(theta) 0; 0 0 1];
```

[2]source: By Jan Boddez - Own work, GFDL, https://commons.wikimedia.org/w/index.php?curid=3450270

Elemental rotations are more conveniently defined in angle axis notation and then converted to a rotation matrix

```
theta = pi/2;
% elemental rotation about the x-axis
rotmx = axang2rotm([1 0 0 theta]);
% elemental rotation about the y-axis
rotmy = axang2rotm([0 1 0 theta]);
% elemental rotation about the z-axis
rotmz = axang2rotm([0 0 1 theta]);
```

General rotations are obtained from these three elemental rotations by matrix multiplication. For example, the product

$$\mathbf{R} = \mathbf{R}_z(\alpha)\,\mathbf{R}_y(\beta)\,\mathbf{R}_x(\gamma) \tag{3}$$

represents an intrinsic rotation in the order α, β and γ. The corresponding Matlab code is

```
  alpha = pi/2;
  beta=pi/4;
  gamma=pi/6;
  rotmz = axang2rotm([0 0 1 alpha]);
  rotmy = axang2rotm([0 1 0 beta]);
  rotmx = axang2rotm([1 0 0 gamma]);
  rotrpyc= rotmz*rotmy*rotmx;
```

The same extrinsic rotation w.r.t. fixed frame in the order α, β and γ is

$$\mathbf{R} = \mathbf{R}_x(\gamma)\,\mathbf{R}_y(\beta)\,\mathbf{R}_z(\alpha) \tag{4}$$

```
   rotrpyf= rotmx*rotmy*rotmz;
```

The difference between sequences of rotations becomes apparent by plotting the corresponding frames (see figure 3)

```
   plotTransforms(zeros(3,3),[1 0 0 0; rotm2quat(rotrpyc); rotm2quat(rotrpyf)]);
   axis equal;
```

Rotation matrices not only describe the relative orientation between frames but also define the transformation of the coordinates of a vector expressed in one frame to the coordinates of the same vector expressed in another frame. Assume the frame $x_1y_1z_1$ is obtained from frame $x_0y_0z_0$ by applying a rotation $\mathbf{R}_1^0$ followed by a translation with a vector $\mathbf{t}_1^0$. The coordinates of the point $\mathbf{p}^0$ with respect to frame $x_0y_0z_0$ are computed according to

$$\mathbf{p}^0 = \mathbf{R}_1^0\mathbf{p}^1 + \mathbf{t}_1^0 \tag{5}$$

[3]source: RST

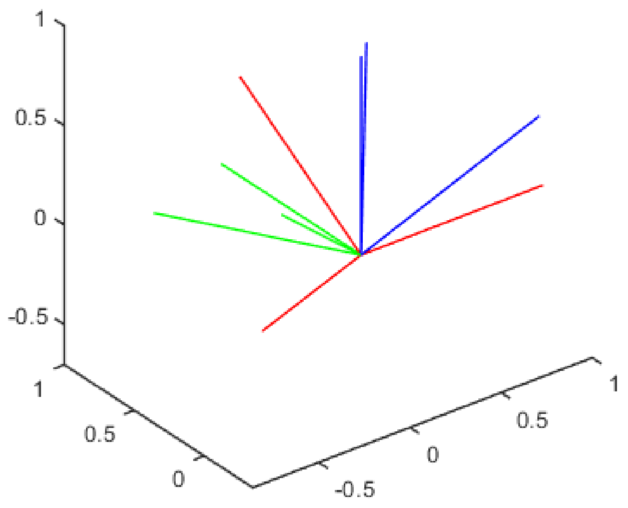

Figure 3: Original frame, frame `rotrpyf` and frame `rotrpyc` [3]

```
p1=[1 1 2];
trvec01=[1 0 -1];
rotmx01=axang2rotm([1 0 0 pi/2]);
p0=rotmx01*p1'+trvec01';
```

The following code first applies the translation $\mathbf{t}_0^1$ followed by a rotation $\mathbf{R}_0^1$ to the vector $\mathbf{p_0}$.

$$\mathbf{p}^0 = \mathbf{R}_1^0(\mathbf{p}^1 + \mathbf{t}_1^0) \tag{6}$$

```
p1=[1 1 2];
trvec01=[1 0 -1];
rotmx01=axang2rotm([1 0 0 pi/2]);
p0=rotmx01*(p1+trvec01)';
```

Given a base frame $x_0y_0z_0$, a rotated current frame $x_1y_1z_1$ obtained from a rotation $\mathbf{R}_1^0$ and third frame $x_2y_2z_2$ obtained from rotation $\mathbf{R}_2^1$. If rotations are carried out w.r.t. the **current frame** $x_1y_1z_1$ **postmultiply** the matrices.

$$\mathbf{R}_2^0 = \mathbf{R}_1^0\mathbf{R}_2^1 \tag{7}$$

The above matrix product corresponds to **intrinsic** rotations which are elemental rotations that occur about the axes of the rotating coordinate system $x_i y_i z_i$, which changes its orientation after each elemental rotation. An intrinsic composition uses rotations about **body-fixed axes** whose directions change in the reference frame after every rotation.

If the second rotation $\mathbf{R}_2^1$ is carried out w.r.t. the fixed frame $x_0 y_0 z_0$ **premultiply** the matrices (**extrinsic** rotation).

$$\bar{\mathbf{R}}_2^0 = \mathbf{R}_2^1 \mathbf{R}_1^0 \tag{8}$$

Extrinsic rotations are elemental rotations that occur about the axes of the fixed coordinate system $x_0 y_0 z_0$. The XYZ system rotates, while $x_0 y_0 z_0$ is fixed. The corresponding extrinsic rotation about xyz is obtained by

$$\mathbf{R}_e = \mathbf{R}_x(\gamma)\,\mathbf{R}_y(\beta)\,\mathbf{R}_z(\alpha) \tag{9}$$

Such a transformation is called an extrinsic composition of rotations. In **extrinsic** rotations the rotations are performed about the **fixed principal axes** directions of the reference frame.

The matrix $\mathbf{R}$ produces the desired effect only if they are used to pre-multiply column vectors. The same point P is either represented by a column vector $\mathbf{v}$ or a row vector $\mathbf{w}$. Rotation matrices either pre-multiply column vectors $\mathbf{Rv}$, or post-multiply row vectors $\mathbf{wR}$. However, $\mathbf{Rv}$ produces a rotation in the opposite direction with respect to $\mathbf{wR}$.

Now, please complete tasks 1 to 5 in the template.

Homogeneous Transformations

Homogeneous transformations are important as they provide the basis for defining robot direct (forward) kinematics. The relationships between frames, e.g. robot base frame and end effector frame, are defined by homogeneous transforms.

In the field of robotics there are many possible ways of representing positions and orientations, but the homogeneous transformation is well matched to MATLABs powerful tools for matrix manipulation. Homogeneous transformations combine the two operations of rotation and translation into a single matrix multiplication.

Homogeneous transformations are 4×4 matrices that describe the relationships between Cartesian coordinate frames in terms of translation and orientation. They are composed of a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$.

$$\mathbf{H} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

Assume the frame $x_1y_1z_1$ is obtained from frame $x_0y_0z_0$ by applying a rotation $\mathbf{R}_1^0$ followed by a translation with a vector $\mathbf{t}_1^0$. The coordinates of the point $\mathbf{p}^0$ with respect to frame $x_0y_0z_0$ are computed according to

$$\mathbf{p}^0 = \mathbf{R}_1^0\mathbf{p}^1 + \mathbf{t}_1^0 \tag{11}$$

Assume another rotation and translation to obtain frame $x_2y_2z_2$ from frame $x_1y_1z_1$ by a rotation $\mathbf{R}_2^1$ followed by a translation $\mathbf{t}_2^1$.

$$\mathbf{p}^1 = \mathbf{R}_2^1\mathbf{p}^2 + \mathbf{t}_2^1 \tag{12}$$

The composition of these consecutive rigid motions is given by:

$$\mathbf{p}^0 = \mathbf{R}_1^0\mathbf{R}_2^1\mathbf{p}^2 + \mathbf{R}_1^0\mathbf{t}_2^1 + \mathbf{t}_1^0 \tag{13}$$

thus

$$\begin{aligned} \mathbf{R}_2^0 &= \mathbf{R}_1^0\mathbf{R}_2^1 \\ \mathbf{t}_2^0 &= \mathbf{R}_1^0\mathbf{t}_2^1 + \mathbf{t}_1^0 \end{aligned}$$

This affine representation of rotation and translation has the drawback that rotational and translational components get mixed e.g. with terms such as $\mathbf{R}_1^0\mathbf{t}_2^1$. A homogeneous representation replaces the matrix multiplication and vector addition in three dimensions by a matrix multiplication in four dimensions. For that purpose the Euclidean vectors $\mathbf{p}^0, \mathbf{p}^1$ are augmented by a fourth component (constant of 1) to obtain the homogeneous vectors:

$$\begin{aligned}\mathbf{P}^0 &= \begin{bmatrix}\mathbf{p}^0\\ 1\end{bmatrix}\\ \mathbf{P}^1 &= \begin{bmatrix}\mathbf{p}^1\\ 1\end{bmatrix}\end{aligned}$$

The transformation in equation (11) is equivalent to the homogeneous matrix transformation

$$\mathbf{P}^0 = \mathbf{H}_1^0\mathbf{P}^1 \tag{14}$$

with the 4×4 homogeneous transformation

$$\mathbf{H}_1^0 = \begin{bmatrix}\mathbf{R}_1^0 & \mathbf{t}_1^0\\ \mathbf{0} & 1\end{bmatrix} \tag{15}$$

The same applies to

$$\mathbf{P}^1 = \mathbf{H}_2^1\mathbf{P}^2 \tag{16}$$

The overall transformation in equation (13) is obtained from

$$\mathbf{P}^0 = \mathbf{H}_1^0\mathbf{H}_2^1\mathbf{P}^2 \tag{17}$$

such that

$$\mathbf{H}_2^0 = \mathbf{H}_1^0\mathbf{H}_2^1 \tag{18}$$

is obtained from 4×4 matrix multiplication of the individual transformations.

Matlab employs the abbreviation `tform` for a homogeneous transformation matrix that combines a translation and rotation into one matrix. Its numeric representation is a 4-by-4 matrix. For example, a rotation of angle `theta` around the x-axis and a translation of `dy` units along the y-axis is expressed as:

```
theta = pi/2;
tform = [1 0 0 0; 0 cos(theta) -sin(theta) dy; 0 sin(theta) cos(theta) 0; 0 0 0 1];
```

You should pre-multiply your transformation matrix with your homogeneous coordinates, which are represented as a matrix of row vectors (n-by-4 matrix of points). Utilize the transpose (') to rotate your points for matrix multiplication to a 4-by-n) matrix of column vectors. For example:

```
points = rand(100,4);
tformPoints = (tform*points')';
```

The Robotics System Toolbox provides conversion functions for transformation representations

```
tform=trvec2tform(trvec);
```

generates the homogeneous transformation matrix `tform` that corresponds to a translation vector `trvec` with components $\mathbf{t} = [t_x, t_y, t_z]$.

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
tform=rotm2tform(rotm);
```

generates the homogeneous transformation matrix `tform` ($\mathbf{H}$) that corresponds to a rotation matrix `rotm` ($\mathbf{R}$) with components r_{ij}.

$$\mathbf{H} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now, please complete tasks 6 to 11 in the template.

Rigid Body Tree Robot Model

Homogeneous transformations are ubiquitous in robotics and therefore in the Robotics System Toolbox. In particular they provide the basis for defining and utilizing robot direct and inverse kinematics as the corresponding frames are defined by homogeneous transforms.

The rigid body tree model is a representation of a robot structure. It is useful to represent robots such as manipulators or other kinematic trees. The `RigidBodyTree` class provides a representation of manipulator kinematic models.

A rigid body tree is composed of rigid bodies (`robotics.RigidBody`) that are connected via joints (`robotics.Joint`). Each rigid body has a joint that defines how that body moves relative to its parent in the tree. The method

```
robotics.Joint.setFixedTransform
```

specifies the transformation from one body to the next by setting the fixed transformation on each joint.

Bodies can be added, replaced, or removed from the rigid body tree model. You can also replace joints for specific bodies. The `RigidBodyTree` object maintains the relationships and updates the `RigidBody` object properties to reflect this relationship. The transformations between different body frames are retrieved with

```
robotics.RigidBodyTree.getTransform
```

Every rigid body tree has a base. The base defines the world coordinate frame and is the first attachment point for a rigid body. The base cannot be modified, except for its `BaseName` property.

The rigid body is the basic building block of rigid body tree model and is created using

```
robotics.RigidBody
```

A rigid body, sometimes called a link, represents a solid body that is non-deformable.

Rigid body objects are added to a rigid body tree with multiple bodies. Rigid bodies possess parent or children bodies associated with them (`Parent` or `Children` properties). The parent is the body that this rigid body is attached to, for example the robot base or some body upstream of the kinematic chain. The children are all the bodies attached to this body downstream from the base of the rigid body tree (see figure 4).

Each rigid body has a coordinate frame associated with them, and contains a `robotics.Joint` object.

Each rigid body has one joint, which defines the motion of that rigid body relative to its parent. It is the attachment point that connects two rigid bodies in a robot model.

[4]source: Robotics System Toolbox, Mathworks

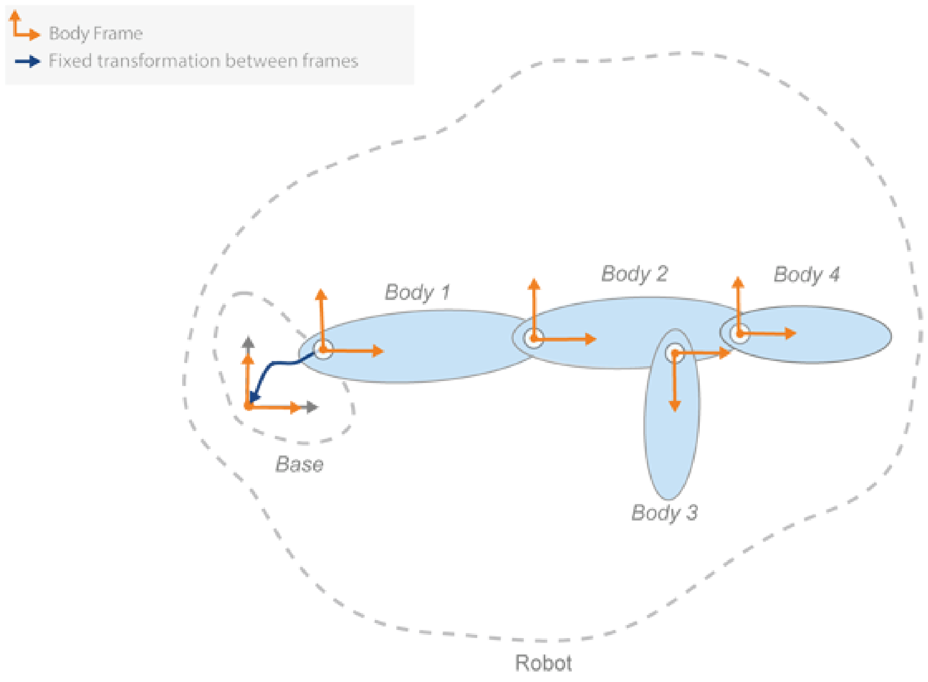

Figure 4: Rigid body tree. [4]

To represent a single physical body with multiple joints or different axes of motion, use multiple `RigidBody` objects.

The `robotics.Joint` class supports fixed, revolute, and prismatic joints as shown in figure 5.

These joints allow the following motion, depending on their type:

- fixed : No motion. Body is rigidly connected to its parent.
- revolute : Rotational motion only. Body rotates around this joint relative to its parent. Position limits define the minimum and maximum angular position in radians around the axis of motion.
- prismatic : Translational motion only. The body moves linearly relative to its parent along the axis of motion.

This assignment only deals with fixed joints or entirely rigid bodies. The upcoming lab on robot kinematics introduces revolute and prismatic joints that form the kinematic chain of a manipulator.

[6]source: Robotics System Toolbox, Mathworks

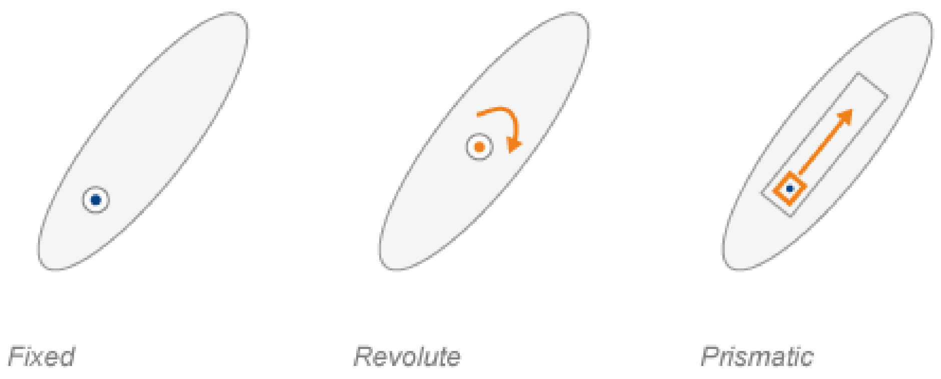

Figure 5: The JointToParentTransform defines where the joint of the child body is in relationship to the parent body frame. The ChildToJointTransform defines where the joint of the child body is in relationship to the child body frame. [6]

Joints also have properties that define the fixed transformation between parent and children body coordinate frames. These properties can only be set using the

```
robotics.Joint.setFixedTransform
```

method. Depending on your method of inputting transformation parameters, either the `JointToParentTransform` or `ChildToJointTransform` property is set. The other property is set to the identity matrix. Figure 6 depicts what each property signifies.

The following code generates a rigid body tree and attaches a rigid body with a fixed joint to the robot base.

```
rbtree = robotics.RigidBodyTree;
body1 = robotics.RigidBody('b1');
jnt1 = robotics.Joint('jnt1','fixed');
body1.Joint = jnt1;
basename = rbtree.BaseName;
rbtree.addBody(body1,basename);
rbtree.showdetails();
rbtree.show(); % plot tree
```

In the following you are supposed to generate a rigid body tree, with four bodies. Note, the underlying body frames coincide with the sequence of Denavit-Hartenberg elemental transformations as you will learn in the next assignment, namely a translation along z-axis by $d = 2$, rotation along z-axis by $\theta = \pi/4$, translation along the new x-axis by $a = 1$ and final rotation along the new x-axis by $\alpha = \pi/2$.

[8]source: Robotics Systems Toolbox, Mathworks

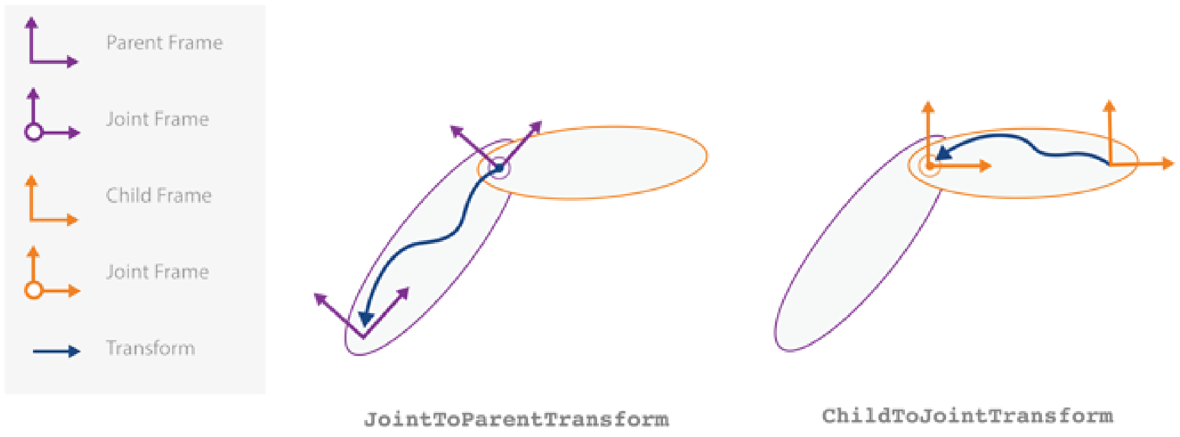

Figure 6: The JointToParentTransform defines where the joint of the child body is in relationship to the parent body frame. The ChildToJointTransform defines where the joint of the child body is in relationship to the child body frame. [8]

Now, please complete tasks 12 to 23 in the template.

Notes:

- The rigid body tree should resemble figure 7.

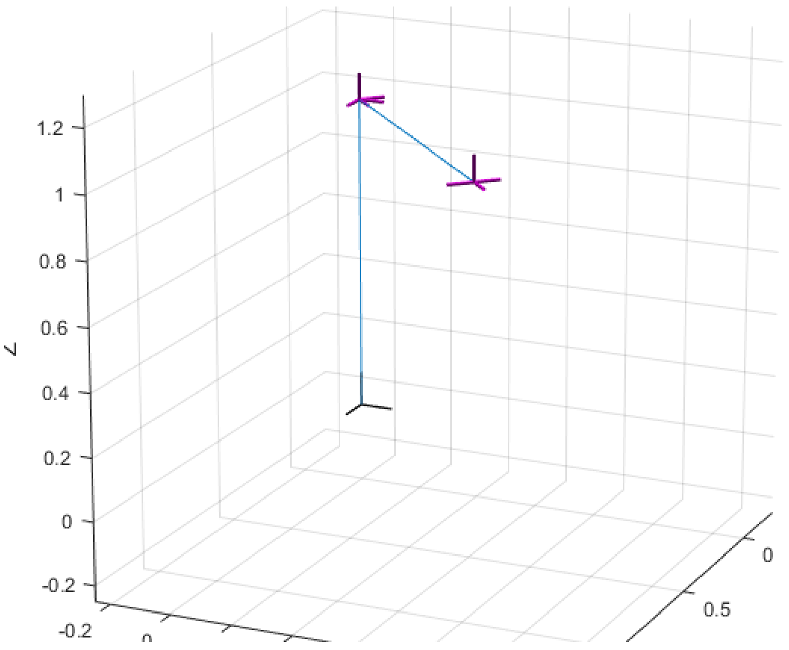

Figure 7: Rigid body tree for elemental Denavit-Hartenberg transforms

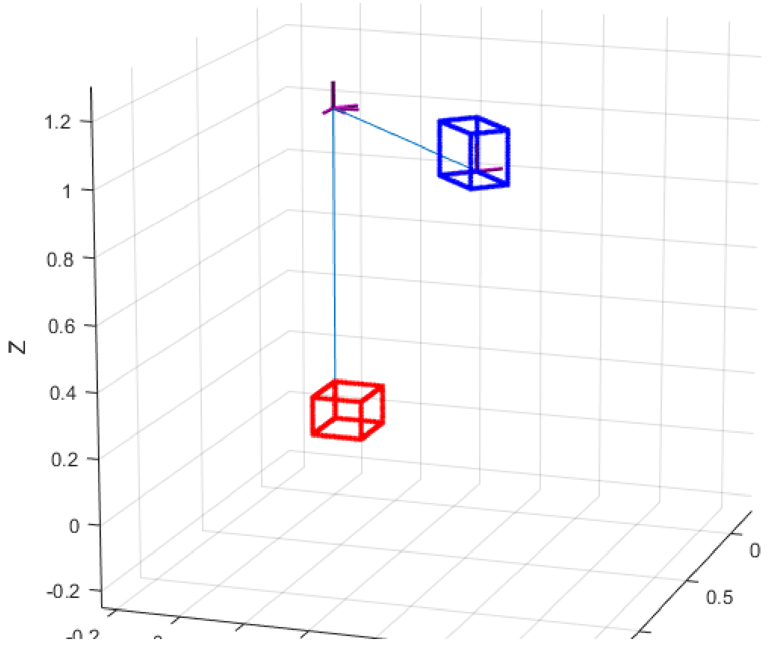

Figure 8: Cuboid w.r.t. base frame (red) and w.r.t. frame 'b4' (blue)

Euler Angles

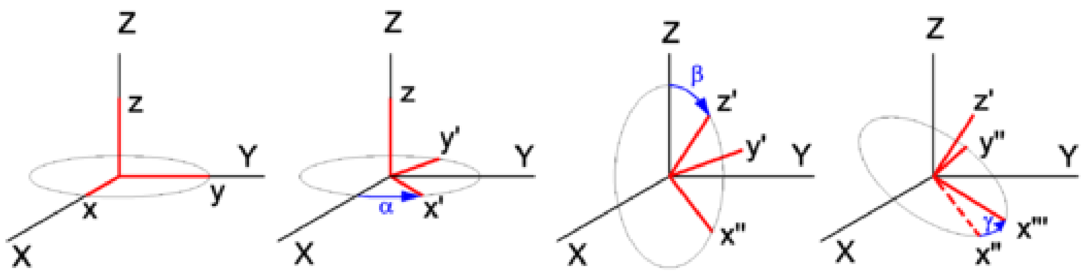

Figure 9: ZYZ - Euler angles [9]

Since a rotation matrix only possess three independent components (orthogonality constraint) one seeks a minimal representation of rotations in term of three independent quantities, in this case angles of three consecutive elemental rotations denoted by Euler angles. The overall rotation is composed of three successive rotations around designated axis: First rotate about the z-axis by the angle ϕ, then rotate about the current y-axis by the angle θ and finally rotate about the current z-axis by the angle ψ. In the figure 9 the angles are denoted by α, β, γ instead of ϕ, θ, ψ. This particular order of rotation axes gives rise to the convention ZYZ-Euler angles. There are other types of combinations, for e.g. ZXZ-Euler angles, but ZYZ-Euler angles are the most common on. The overall rotation is obtained by postmultiplication of rotation matrices

$$\mathbf{R}_{\phi,\theta,\psi} = \mathbf{R}_{z,\phi}\mathbf{R}_{y,\theta}\mathbf{R}_{z,\psi} \tag{19}$$

Matlab supports the 'ZYZ' as well as the 'ZYX' axis order of Euler angles which is denoted as Roll Pitch Yaw (rpy). Knowing which axis order you use is important for apply the rotation to points and in converting to other representations. The numeric representation is an ordinary 1-by-3 vector of scalar angles For example, a rotation around the y -axis of π is expressed as:

```
eul = [0 pi 0]
```

The conversions `tform2eul, eul2tform, rotm2eul, eul2rotm` provide changes of representation between Euler angles and rotation matrices and homogeneous transformations. Unfortunately, the default order for these conversions is 'ZYX' corresponding to roll pitch yaw. Therefore you need to specify the sequence 'ZYZ' as an extra argument.

```
% ZYZ Euler angles
rotmzyz = eul2rotm([phi theta pis],'ZYZ');
```

[9]Citenzendium Euler angles `http://en.citizendium.org/images/thumb/7/7c/Euler_angles.png/550px-Euler_angles.png`

Roll Pitch Yaw Angles

Roll, pitch and yaw angles are an alternative to ZYZ Euler angles, they denote rotations along the axes of a fixed frame rather than the current frame. A rotation matrix $\mathbf{R}$ is described as a product of successive rotations about the principal coordinate axes x, y, z taken in a specific order. These rotations define the roll, pitch and yaw angles usually denoted by ϕ, θ and ψ. The overall rotation with respect to a fixed frame (extrinsic rotation) occurs by an angle ψ along the x-axis (roll), an angle θ along the y-axis (pitch) and an angle ϕ along the z-axis (yaw) resulting from the premultiplication:

$$\mathbf{R}_{\phi,\theta,\psi} = \mathbf{R}_{z,\phi}\mathbf{R}_{y,\theta}\mathbf{R}_{x,\psi} \tag{20}$$

Notice, that instead of roll-pitch-yaw (XYZ) relative to the fixed frames the transformation $\mathbf{R}_{\phi,\theta,\psi}$ can be interpreted as yaw-pitch-roll (ZYX) w.r.t. to the current frame.

Roll Pitch Yaw angles are also denoted as Tait-Bryan angles. They are common in automotive and aerospace applications, so that zero degrees elevation represents the horizontal attitude. They represent the orientation of a vehicle or aircraft with respect to the world frame as shown in figure 10.

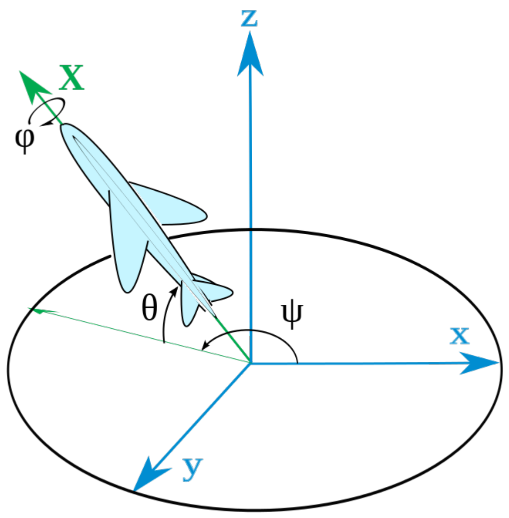

Figure 10: Roll Pitch Yaw angles [10]

Matlab supports the 'ZYX' axis order of Euler angles which is denoted as Roll Pitch Yaw (rpy).

```
% roll pitch yaw
rotmrpy = eul2rotm([phi theta psi],'ZYX');
```

[10]source: By Juansempere - copied from previous workPreviously published: 2011 in wikipedia, CC BY-SA 3.0, `https://en.wikipedia.org/w/index.php?curid=39309443`

Therefore 'ZYX' Euler angles w.r.t. current frame are equivalent to 'XYZ' roll pitch yaw angles w.r.t. fixed frame. For robotics manipulators 'ZYZ' Euler angles are common whereas in mobile robotics 'XYZ' roll pitch yaw angles are common.

Axis - Angle

Euler's rotation theorem postulates that any rotation in 3D can be expressed as a single rotation about a designated axis which itself remains unchanged by the rotation. The magnitude of the angle is also unique, with its sign being determined by the sign of the rotation axis.

Rotations are not necessarily described by consecutive rotations w.r.t principal coordinate axes but instead in terms of a single rotation along an arbitrary axis in space. Let $\mathbf{w} = [w_x, w_y, w_z]^T$ be a unit vector that defines an axis of rotation expressed w.r.t. to the reference frame The pair $\mathbf{w}, \theta$ with unit vector $\mathbf{w} = [w_x, w_y, w_z]^T$ and scalar θ defines a rotation by θ along $\mathbf{w}$. Since the axis is normalized, it has only two degrees of freedom. The angle adds the third degree of freedom to this rotation representation.

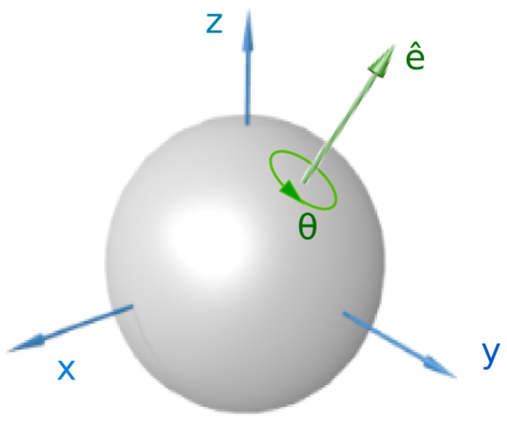

Figure 11: Rotation represented by an Euler axis $\hat{e}$ and angle Θ [11]

If the rotation angle θ is zero, the axis is not uniquely defined. Composition of rotations represented by Euler angles or axis-angle, is not straightforward as the axis vectors do not satisfy the law of vector addition. Typically, one converts from Euler angle / angle-axis to rotation matrix, performs the composition as a matrix product and then transforms back to Euler angle / angle-axis representation.

Matlab employs the abbreviation `axang` for the angle axis representation of a scalar rotation around a fixed axis defined by a vector. The numeric representation is a 1-by-3 unit

[11]source:By DF Malan - Own work, Public Domain, `https://commons.wikimedia.org/w/index.php?curid=1354297`

vector and a scalar angle combined into 1-by-4 vector. For example, a rotation of $\pi/2$ radians around the y-axis: is defined by the vector `axang = [0 1 0 pi/2]`

Now, please complete task 24 in the template.

Representation Singularities

The task starts with the roll, pitch and yaw angles $\alpha = \pi/4, \beta = \pi/2$ and $\gamma = \pi/4$. The minimal representation of Euler and roll pitch yaw angles has the drawback of representation singularities, the so called gimbal lock. In the above example for roll, pitch yaw angles, the first z-axis coincides with the final x-axis, due to the rotation around the y-axis by $\pi/2$. The overall rotation around the common x- and z-axis by the angle $\alpha + \gamma$ becomes zero.

In case of Euler angles if θ is zero, there is no rotation about the y-axis. As a consequence, the final z-axis coincide with the first z-axis, and ϕ and ψ represent rotations about the same axis. The final orientation is obtained with a single rotation about z, by an angle equal to $\phi + \psi$.

Now, please complete tasks 25 to 26 in the template.

Unit Quaternions

Unit quaternions are another representation of orientation. They constitute a compromise between the advantages and disadvantages of rotation matrices and Euler angle sets. Compared to Euler angles they are simpler to compose and avoid the problem of gimbal lock. Compared to rotation matrices they are more numerically stable and may be more efficient. A quaternion $q \in \mathbb{H}$ with

$$q = \eta + \epsilon_x \mathbf{i} + \epsilon_y \mathbf{j} + \epsilon_z \mathbf{k} \tag{21}$$

is composed of a **scalar part** $\mathrm{Re}(q) := \eta$ and an **imaginary part** (sometimes referred to as **vector part**) with three components $\mathrm{Im}(q) := \epsilon_x \mathbf{i} + \epsilon_y \mathbf{j} + \epsilon_z \mathbf{k}$. The imaginary part of a quaternion behaves like a vector in three dimension vector space, and the real part η a behaves like a scalar in $\mathbb{R}$. The abstract symbols $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are similar to the imaginary component of complex numbers and satisfy:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \tag{22}$$

For some operations it is useful to express the quaternion as a 4D vector, composed of the scalar and the vector part (note, we write $\mathbf{q}$ in boldface for this representation):

$$\mathbf{q} = [\eta \ \boldsymbol{\epsilon}]^T = [\eta \ \epsilon_x \ \epsilon_y \ \epsilon_z]^T \tag{23}$$

Unit quaternions are constrained by

$$\begin{aligned} \|\mathbf{q}\| &= \sqrt{\eta^2 + \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}} = 1 && (24) \\ \iff \eta^2 &+ \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} = 1 && (25) \end{aligned}$$

The inverse of a quaternion is defined by

$$\mathbf{q}^{-1} = [\eta \ -\boldsymbol{\epsilon}]^T = [\eta \ -\epsilon_x \ -\epsilon_y \ -\epsilon_z]^T \tag{26}$$

The composition of rotations $q^{(1)}$ and $q^{(2)}$ (not squared!) is achieved by the quaternion product (Hamilton product)

$$\begin{aligned} (\eta^{(1)} + \epsilon_x^{(1)}\mathbf{i} + \epsilon_y^{(1)}\mathbf{j} + \epsilon_z^{(1)}\mathbf{k})(\eta^{(2)} + \epsilon_x^{(2)}\mathbf{i} + \epsilon_y^{(2)}\mathbf{j} + \epsilon_z^{(2)}\mathbf{k}) &= \\ (\eta^{(1)}\eta^{(2)} - \epsilon_x^{(1)}\epsilon_x^{(2)} - \epsilon_y^{(1)}\epsilon_y^{(2)} - \epsilon_z^{(1)}\epsilon_z^{(2)}) &+ \\ (\eta^{(1)}\epsilon_x^{(2)} + \epsilon_x^{(1)}\eta^{(2)} + \epsilon_y^{(1)}\epsilon_z^{(2)} - \epsilon_z^{(1)}\epsilon_y^{(2)})\mathbf{i} &+ \\ (\eta^{(1)}\epsilon_y^{(2)} - \epsilon_x^{(1)}\epsilon_z^{(2)} + \epsilon_y^{(1)}\eta^{(2)} + \epsilon_z^{(1)}\epsilon_x^{(2)})\mathbf{j} &+ \\ (\eta^{(1)}\epsilon_z^{(2)} + \epsilon_x^{(1)}\epsilon_y^{(2)} - \epsilon_y^{(1)}\epsilon_x^{(2)} + \epsilon_z^{(1)}\eta^{(2)})\mathbf{k} & \end{aligned} \tag{27}$$

A unit quaternion resembles the angle axis representation as it can be interpreted as a rotation about the axis denoted by $\frac{\boldsymbol{\epsilon}}{\|\boldsymbol{\epsilon}\|}$ by an angle θ.

$$q = e^{\frac{\theta}{2}(\epsilon_x \mathbf{i} + \epsilon_y \mathbf{j} + \epsilon_z \mathbf{k})} = \cos\frac{\theta}{2} + (\epsilon_x \mathbf{i} + \epsilon_y \mathbf{j} + \epsilon_z \mathbf{k}) \sin\frac{\theta}{2} \tag{28}$$

The rotation of an ordinary 3D vector $[p_x, p_y, p_z]^T \in \mathbb{R}^3$ with a unit quaternion $\mathbf{q}$ is achieved by embedding the vector into $\mathbb{H}$ with $\mathbf{p} = [0, p_x, p_y, p_z]^T$ and applying the conjugation of $\mathbf{p}$ by $\mathbf{q}$:

$$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1} \tag{29}$$

using the quaternion product defined in equation (27). The position vector $\mathbf{p}'$ denotes the coordinates of the point $\mathbf{p}$ after the rotation. The quaternion $\mathbf{p} = [\eta\ \boldsymbol{\epsilon}]^T$ is composed of the vector part $\boldsymbol{\epsilon}$ that is equal to the 3D vector $[p_x, p_y, p_z]^T$ and a real part $\eta = 0$ that equals zero. The vector part $\boldsymbol{\epsilon}'$ of the resulting quaternion $\mathbf{p}'$ is the desired 3D vector obtained by the quaternion product of $\mathbf{q}$, $\mathbf{p}$ and $\mathbf{q}^{-1}$.

The Robotics System toolbox (since Matlab 2018a) supports the quaternion class and quaternion operations. Quaternions are instantiated from other rotation representations for example from rotation matrices

```
quat = quaternion(rotationMatrix,'rotmat','frame')
```

The last argument specifies whether the rotation matrix should be interpreted as a frame or point rotation. Instantiation from Euler angles

```
quat = quaternion(E,'euler','ZYZ','frame')
```

or from a rotation vector

```
quat = quaternion(rotationVector,'rotvec')
```

Rotation vectors $\mathbf{r}_{rotvec}$ are similar to angle-axis representation $(\vartheta, \mathbf{r}_{angleaxis})$, except that the magnitude of rotation ϑ is not a separate parameter but determines the length of the rotation axis vector.

$$\mathbf{r}_{rotvec} = \vartheta \mathbf{r}_{angleaxis} \tag{30}$$

vice versa

$$\begin{aligned} \mathbf{r}_{angleaxis} &= \frac{\mathbf{r}_{angleaxis}}{|\mathbf{r}_{angleaxis}|} \\ \vartheta &= |\mathbf{r}_{angleaxis}| \end{aligned}$$

The transformations from quaternion to Euler angle, rotation matrix and rotation vector are obtained by

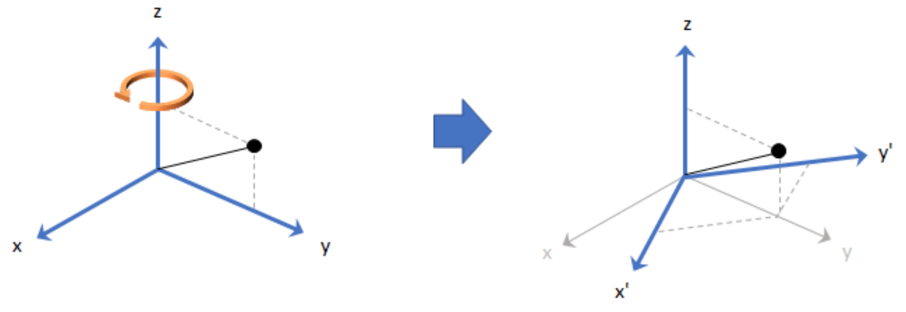

Figure 12: Rotation of frame [12]

```
quat = quaternion([0.7071 0.7071 0 0]);
eulerAngles = euler(quat,'ZYX','frame');
rotationMatrix = rotmat(quat,'frame');
rotationVector = rotvec(quat);
```

The `quaternion` class supports quaternion algebra in particular the Hamilton product

```
Q1timesQ2 = Q1 * Q2;
```

and conjugation

```
Q1c=conj(Q1);
```

and quaternion power

```
Q1square=Q1.^2;
```

The command

```
p=[0.5 , 1.0, 1.0];
quat = quaternion([0,0,pi/4; 0,0,-pi/2],'euler','XYZ','frame');
pHat = rotateframe(quat,p)
```

rotates the frame of reference x-y-z onto the frame x'-y'-z' as shown in figure 12 and maps the Cartesian points `p` in the x-y-z frame onto the same vector in the x'-y'-z' frame using the quaternion `quat`.

The command

```
p=[0.5 , 1.0, 1.0];
quat = quaternion([0,0,pi/4; 0,0,-pi/2],'euler','XYZ','frame');
pHat = rotatepoint(quat,p)
```

rotates the vector `p` in a fixed frame of reference x-y-z onto the new vector `pHat` w.r.t. to the same frame x-y-z as shown in figure 13.

[12]Source: Mathworks

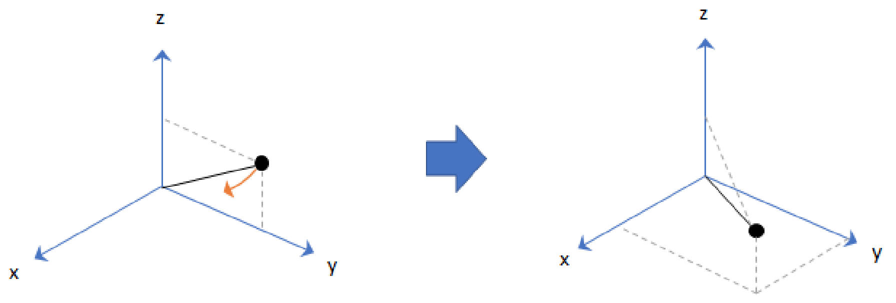

Figure 13: Rotation of points [13]

Euler Angle Interpolation

Interpolation of rotations via Euler angles are non-smooth. The vector moves on an inefficient arc and the axis of rotation is non-constant. The reason for the non-smooth trajectory is the simultaneous motion of two gimbals.

The figure 14 illustrates the trajectory of the vector $\mathbf{p} = [1, 0, 0]$ obtained from linear interpolation from the initial Euler angle $[\psi, \vartheta, \varphi]_i = [\pi/3,\ -\pi/2,\ 0]$ to the final Euler angle $[\psi, \vartheta, \varphi]_f = [-\pi/3,\ \pi/2 + \pi/20,\ 0]$. obtained from the code

```
path = 0:0.05:1;
p = [1,0,0];
interpolatedEuler=eul2rotm([pi/3, -pi/2, 0]...
+[-path*2*pi/3;path*pi*1.05;path*0]','ZYX');
for i=1:21
  rPeul(i,:)=interpolatedEuler(:,:,i)*p';
end
plot3(rPeul(:,1),rPeul(:,2),rPeul(:,3),'r-d');
quiver3(zeros(21,1),zeros(21,1),zeros(21,1),...
rPeul(:,1),rPeul(:,2),rPeul(:,3),'r')
```

Quaternion Slerp

Slerp is shorthand for spherical linear interpolation, with the purpose of animating rotations. A slerp refers to a constant-speed motion along a unit-radius great circle arc, given the ends and an interpolation parameter between 0 and 1. Slerp is applied to unit quaternions such that the quaternion path maps to a path through 3D rotations. The effect is a rotation with uniform angular velocity around a fixed rotation axis. Given an initial quaternion q_0 and a final quaternion q_1 the interpolation over the interval $t = [0, 1]$ is given by the Hamilton product.

$$\mathbf{q}(t) = \mathbf{q_0}(\mathbf{q_0}^{-1}\mathbf{q_1})^t \tag{31}$$

such that $\mathbf{q}(0) = \mathbf{q_0}$ and $\mathbf{q}(1) = \mathbf{q_1}$. Slerp is useful for interpolating rotations for example in path and trajectory planning between a start and a goal pose in workspace.

[13]Source: Mathworks

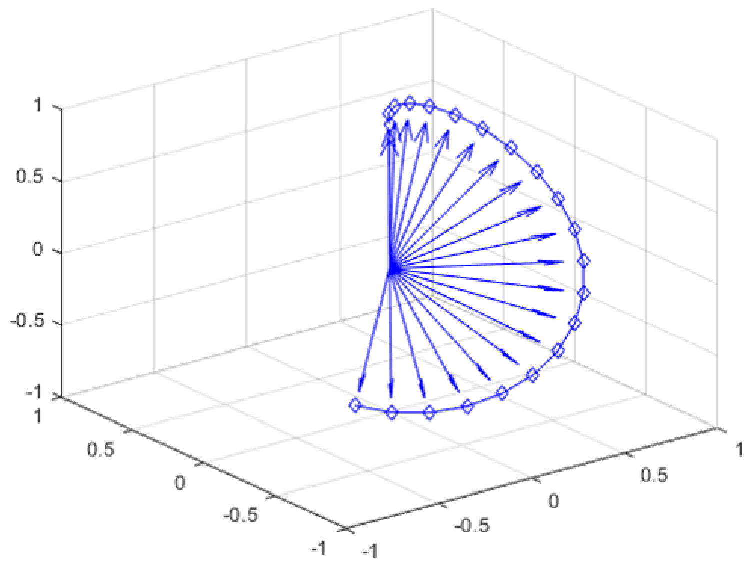

Figure 14: Euler angle interpolation of rotated frames on vector $\mathbf{p} = [1, 0, 0]$.

The function

```
qs = slerp(q0, q1, t)
```

calculates the quaternion spherical linear interpolation according to equation 31 and interpolates between `q0` and `q1` by the interpolation points in `t`.

Now, please complete tasks 27 to 31 in the template.

Notes:

- Compare the quaternion interpolation of task 30 with the Euler angle interpolation in figure 14.

References

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

[2] Kenneth Waldron and James P. Schmiedeler. Kinematics. In Bruno Siciliano and Oussama Khatib, editors, *Springer Handbook of Robotics*, pages 9–33. Springer, 2008.