This assignment is concerned with the mathematical description and definition of direct kinematics of serial link robotic arms. In particular you are expected to implement the kinematic model for 6-DOF UR10 robot arm and to visualize the robot arm in ROS via RViz.

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.*

Before the assignment, read this document entirely and complete the quiz on Forward Kinematics 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 2.8 – 2.9 on direct kinematics in the book by Siciliano et al [1]. Further recommended reading is the chapter on kinematics in the Springer Handbook of Robotics [2].

Short Questions

1) What is the objective of forward kinematics?

2) What is the mathematical description of forward kinematics?

3) How do you determine the x, y, z axis of the joint frames in the Denavit-Hardenberg algorithm?

4) Which of the following statements about the Denavit-Hardenberg convention and parameters are true.

   a) $\alpha_i = 0$ if consecutive joint axes intersect.
   b) $a_i = 0$ if consecutive joint axes are parallel.
   c) θ_i is constant for a prismatic joint.
   d) $d_i = 0$ for a revolute joint.

5) What are the elementary transformations associated with the DH-parameters $d_i, \theta_i, a_i, \alpha_i$.

6) Which subset of DH parameters $d_i, \theta_i, a_i, \alpha_i$ is identical to zero for a spherical wrist?

7) Denote the DH parameters of an anthropomorphic arm.

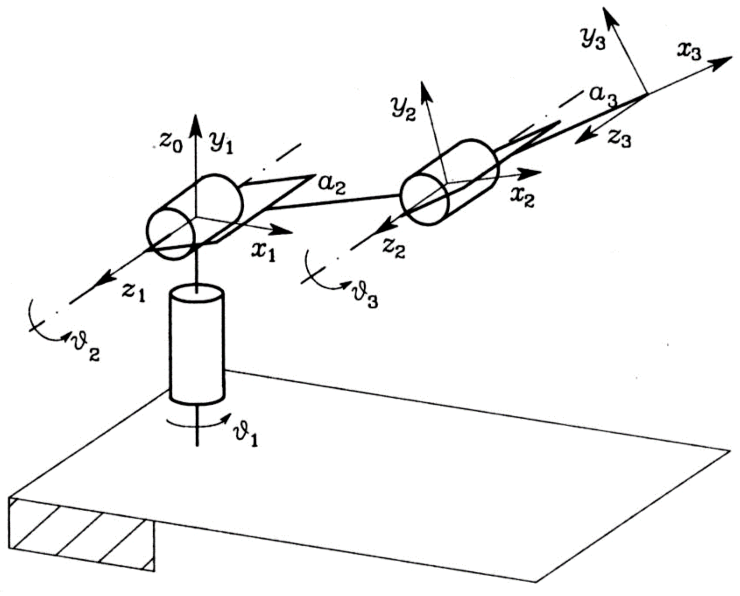

Figure 1: Anthropomorphic arm

Link	d_i	θ_i	a_i	α_i
1				
2				
3				

8) Denote the DH parameters of a spherical wrist.

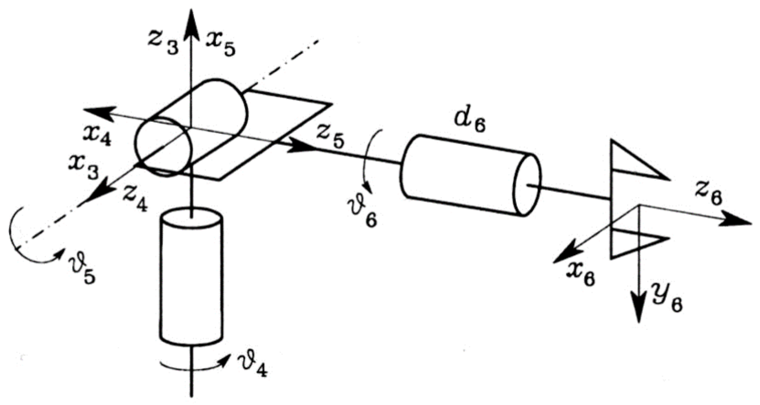

Figure 2: Spherical Wrist

Link	d_i	θ_i	a_i	α_i
4				
5				
6				

Forward Kinematics

Forward kinematics utilizes the kinematic equations of a robot to compute the end-effector pose in task space from the joint configuration. More precisely it establishes a mapping between the joint vector $\mathbf{q} = [q_1, \dots, q_n]^T$ and the homogeneous transform $\mathbf{T}(\mathbf{q})$ that describes the end effector position and orientation w.r.t. the robot base frame. The kinematics equations of a serial link robot arm are obtained from a sequence of transformations that characterize the relative movement at each joint. In case of a revolute joint, the joint variable q_i denotes the angle between the links. In case of a prismatic joint, the joint variable q_i denotes the linear displacement between links. The result is a sequence of rigid transformations T_i between consecutive robot joints from the base frame to the end effector frame parameterized by the joint angles $\mathbf{q} = [q_1, \dots, q_n]^T$.

Denavit-Hartenberg Convention

The Denavit-Hartenberg (DH) convention defines a procedure to uniquely define the joint frames and the corresponding homogeneous transformations in terms of four parameters per joint. The convention standardizes the frames for the geometric linkage among joints.

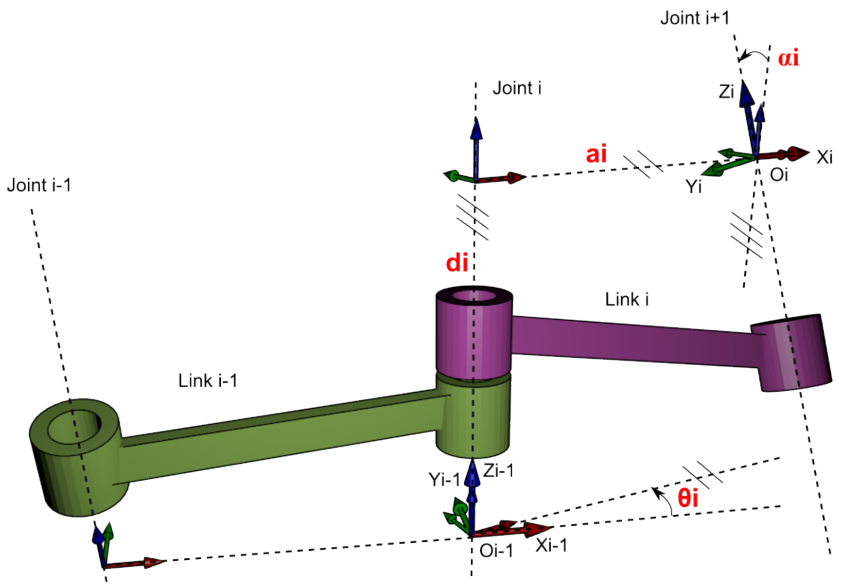

Figure 3: Denavit-Hartenberg parameter (standard convention) [1]

The Denavit-Hartenberg convention in figure 3 represents the homogeneous transformation between the consecutive joint frames i and $i + 1$ as a sequence of four elemental transformations, associated with four parameters $d_i, \theta_i, a_i, \alpha_i$.

- d_i denotes the offset from O_{i-1} to the common normal (line perpendicular to both z axes) along z_{i-1} . In case of a prismatic joint d_i constitutes the joint variable q_{i-1}.

[1]By Ollydbg - I generate this by Blender and Inkscape, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=31730433`

- θ_i is the angle between x_{i-1} and x_i about the z_{i-1}-axis. In case of a revolute joint θ_i constitutes the joint variable q_{i-1}.
- a_i denotes the length of the common normal. Assuming a revolute joint, a is the radius about z_{i-1}.
- α_i is the angle between z_{i-1} and z_i about the x_i-axis (parallel to the common normal).

In order to apply the standard DH procedure, the following properties of the coordinate frames are required:

- The z-axis is the direction of the joint axis.
- The x-axis is parallel to the common normal $x_i = z_{i-1} \times z_i$ and directs from z_{i-1} to z_i. If both z axes are parallel, then d is a free parameter.
- The y-axis follows from the x- and z-axis by choosing it to be a right-handed coordinate system.

Notice, that the literature employs two different Denavit-Hartenberg conventions: the standard convention mentioned above and a modified DH convention in which the locations of coordinate systems and the order of performed transformations differ (modified: $a_i, \alpha_i, d_i, \theta_i$).

A few words on joint offsets: In any DH convention, the joint variables may have arbitrary values. If you wish to define the kinematics drawing with the associated coordinate frames to be a meaningful reference configuration (e.g. all joint variables are zero), you have to define joint offsets. The joint offsets usually do not count to the DH parameters and are thus omitted in DH tables found in literature.

Denavit-Hartenberg Parameters UR10

Figure 4 shows the 3D model and coordinate frames of the UR10 robot arm with 6 degrees of freedom in its reference configuration. The UR10 robot arm is an anthropomorphic arm with a spherical wrist.

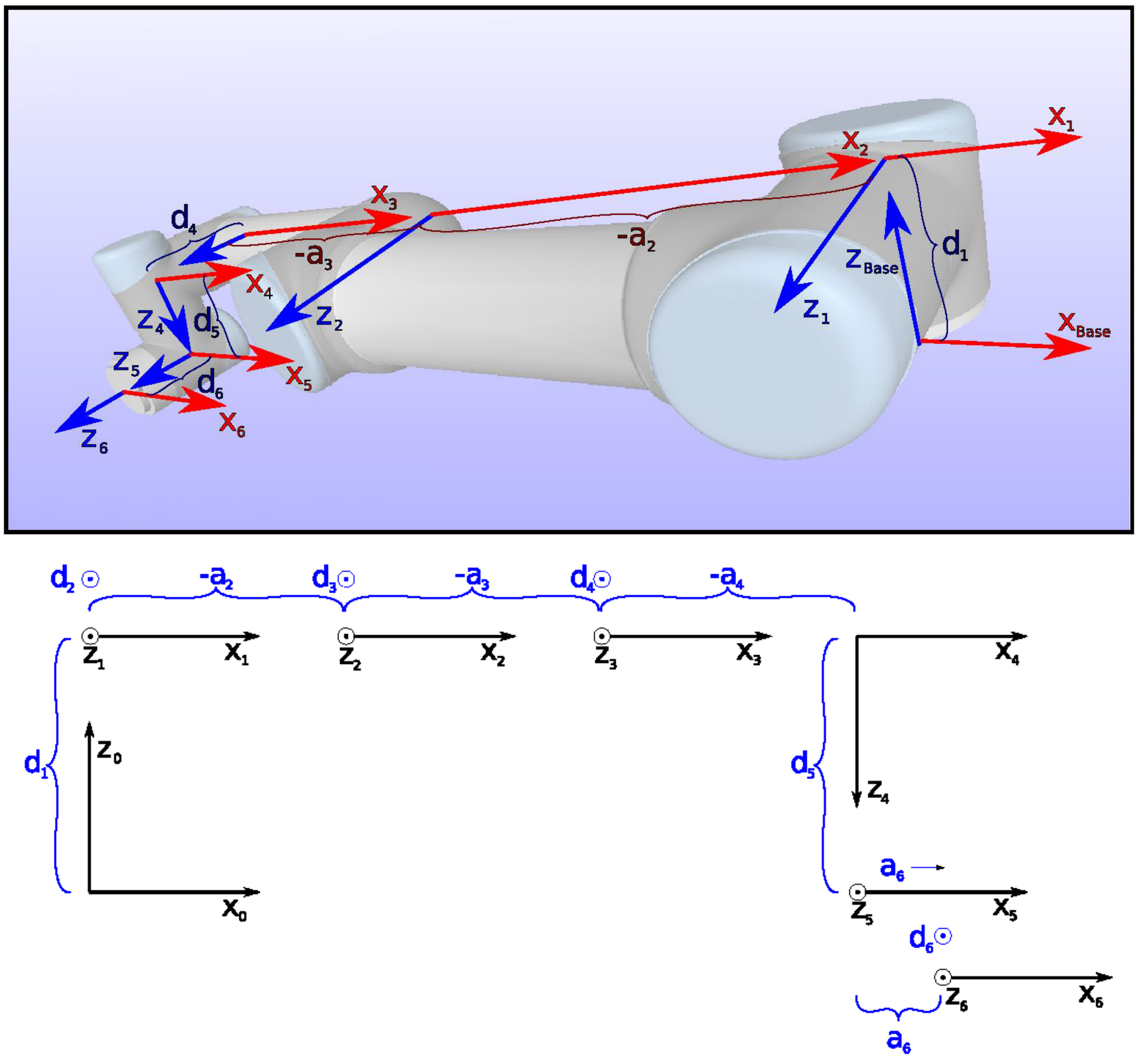

Figure 4: UR10 arm [2]

The frames one to six correspond to base, upper arm, lower arm, wrist 2, wrist 3 and tool center point. All joints are revolute with the joint axis denoted by z_i, therefore the joint variable q_i denotes the rotation θ_i around the z-axis.

[2]source: Universal Robots, `https://www.universal-robots.com/`

a_1	a_2	a_3	a_4	a_5	a_6
0	-0.612	-0.5732	0	0	0
d_1	d_2	d_3	d_4	d_5	d_6
0.1273	0	0	0.163941	0.1157	0.0922
α_1	α_2	α_3	α_4	α_5	α_6

Table 1: Denavit Hartenberg parameters UR10 robot arm [3]

Now, please complete tasks 1 and 2 in the template.

[3]source: Universal Robots, `https://www.universal-robots.com/`

Rigid Body Tree Model

The lab on spatial transformations introduced the `RigidBodyTree` objects for robotic manipulators in Matlab. 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 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. 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. 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.

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.

The method

```
robotics.Joint.setFixedTransform
```

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

The default joint type is 'fixed'. You create a revolute joint with

```
jnt1 = robotics.Joint('jnt1','revolute');
```

Create a rigid body tree object to build the robot.

```
robot = robotics.RigidBodyTree;
```

Create the first rigid body and add it to the robot. To add a rigid body:

- Create a `RigidBody` object and give it a unique name.
- Create a `Joint` object and give it a unique name.
- Use `setFixedTransform` to specify the body-to-body transformation using DH parameters.
- Call `addBody` to attach the first body joint to the base frame of the robot.

```
body1 = robotics.RigidBody('body1');
jnt1 = robotics.Joint('jnt1','revolute');
jnt1.setFixedTransform([a alpha d theta],'dh');
body1.Joint = jnt1;
robot.addBody(body1,'base')
```

In which `a, alpha, d, theta` denote the DH parameters a, α, d, θ of that joint. Proceed in the same manner to add additional bodies and joints to your rigid body tree.

Joint Configurations and Transforms

The joint configuration is described by a `configuration` struct array with fields `JointName` and `JointPosition`. The format is compliant with the ROS message format `sensor_msgs/JointState` of the ROS topic `/joint_states`.

```
configuration = robot.homeConfiguration()
```

returns the home configuration of the robot model. The home configuration is the ordered list of `HomePosition` properties of each nonfixed joint.

With the `robot` you are able to calculate various transforms between robot link frames for a given joint `configuration`.

```
transform = robot.getTransform(configuration,bodyname)
```

computes the transform that converts points in the `bodyname` frame to the robot base frame, using the specified robot configuration.

```
transform = robot.getTransform(configuration,sourcebody,targetbody)
```

computes the transform that converts points from the source body frame to the target body frame, using the specified robot configuration.

Now, please complete tasks 3 to 8 in the template.

Notes:

- Be careful when running sections twice, as the rigid body tree will complain about links that already exist in its structure
- The rigid body tree structure of task 5 should look like this

```
--------------------
Robot: (2 bodies)

Idx          Body Name              Joint Name              Joint Type
      Parent Name(Idx)   Children Name(s)
---          ---------              ----------              ----------
      ----------------   ----------------
1            baselink             worldjoint                   fixed
             world(0)   shoulderlink(2)
2        shoulderlink       shoulderpanjoint                revolute
          baselink(1)
--------------------
```

- The rigid body tree structure of task 8 should look like this

```
Robot: (8 bodies)

Idx          Body Name               Joint Name               Joint Type
        Parent Name(Idx)   Children Name(s)
---          ---------               ----------               ----------
        ----------------   ----------------
1            baselink              worldjoint                    fixed
               world(0)   shoulderlink(2)
2        shoulderlink        shoulderpanjoint                 revolute
             baselink(1)   upperarmlink(3)
3        upperarmlink       shoulderliftjoint                 revolute
         shoulderlink(2)   forearmlink(4)
4         forearmlink              elbowjoint                 revolute
         upperarmlink(3)   wrist1link(5)
5          wrist1link             wrist1joint                 revolute
          forearmlink(4)   wrist2link(6)
6          wrist2link             wrist2joint                 revolute
           wrist1link(5)   wrist3link(7)
7          wrist3link             wrist3joint                 revolute
           wrist2link(6)   eelink(8)
8             eelink                  eejoint                    fixed
           wrist3link(7)
```

Unified Robot Description Format

Unified Robot Description Format (URDF) is an XML format for representing a robot model. The `urdf_to_graphiz` tool generates a graphviz diagram of the URDF model including all components. Figure 5 denotes the graph structure of links and joints of the UR10 robot model in ROS. The blue ellipses denote joints the black boxes denote robot links. The transform between links is stated in terms of a translation vector *xyz* and a rotation expressed in terms of roll pitch yaw angles *rpy*. Notice, the transformation between the `base` frame and the `base_link` frame with a rotation of $-\pi$ around the z-axis.

A URDF-file contains the geometric (kinematic) and dynamic parameters of the links (bodies) and joints. It also refers to the CAD data for visualization of bodies in RViz or Gazebo. The following code describes the geometry, CAD data, mass, center of mass and inertia tensor of the `shoulder_link` in the file `ur10.urdf`.

```
<link name="shoulder_link">
 <visual>
  <geometry>
   <mesh filename="package://ur_description/meshes/ur10/visual/Shoulder.dae"/>
  </geometry>
 </visual>
 <collision>
  <geometry>
   <mesh filename="package://ur_description/meshes/ur10/collision/Shoulder.dae"/>
  </geometry>
 </collision>
 <inertial>
  <mass value="7.778"/>
  <origin rpy="0 0 0" xyz="0.0 0.0 0.0"/>
  <inertia ixx="0.0314743125769" ixy="0.0" ixz="0.0" iyy="0.0314743125769" iyz="0.0"
    izz="0.021875625"/>
 </inertial>
</link>
```

The following code define the joint `shoulder_lift_joint`, it refers to parent and child link of the joint and its specifies the transform asscociated with the joint in terms of translation *xyz* and roll-pitch-yaw angles *rpy*. It also specfies kinematic and dynamic parameters of the joint maximum torque, upper and lower joint limits, maximum joint velocity, dynamic damping and friction.

```
<joint name="shoulder_lift_joint" type="revolute">
 <parent link="shoulder_link"/>
 <child link="upper_arm_link"/>
 <origin rpy="0.0 1.570796325 0.0" xyz="0.0 0.220941 0.0"/>
 <axis xyz="0 1 0"/>
 <limit effort="330.0" lower="-6.2831853" upper="6.2831853" velocity="2.16"/>
 <dynamics damping="0.0" friction="0.0"/>
</joint>
```

The Robotics System Toolbox (release 2017a) provides a URDF file importer which imports URDF robot descriptions as a RigidBodyTree object The function `importrobot` parses the URDF information and returns the corresponding `robotics.RigidBodyTree` object.

Now, please complete tasks 9 to 16 in the template.

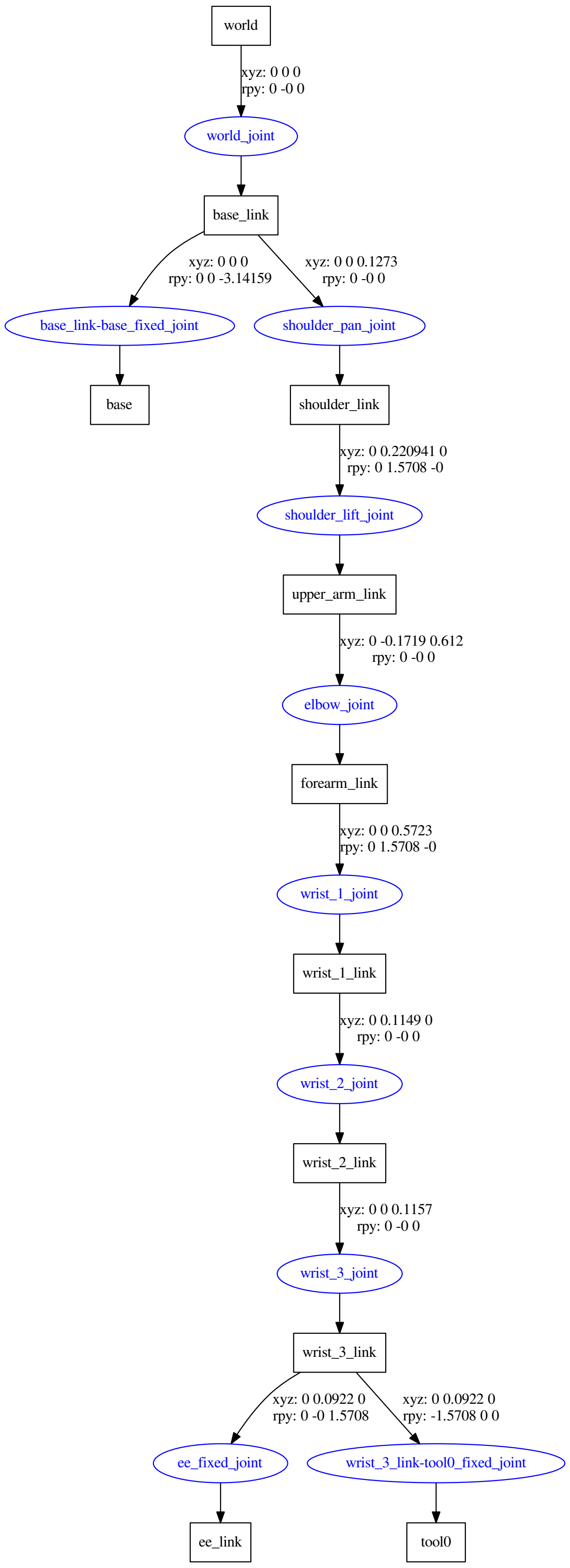

Figure 5: Graph of bodies, joints and transforms in the URDF description of the UR10 robot arm

RViz Visualization Package for UR10 Robot

RViz is the standard ROS tool for visualization of robot related data. RViz provides a number of different camera perspectives so called views to visualize the environment. A display in RViz is something that draws something in the 3D world such as a point cloud or the robot state. The RViz User Guide explains the GUI and the functionalities that RViz provides. The Built-In Display `RobotModel` shows a visual representation of the robot according to current joint and link TF transforms.

This assignment requires that you boot into Ubuntu in order to run ROS. Follow the steps of the following instruction files in the Moodle course, to install a virtual machine with ROS and prepare a workspace with the UR ROS Package:

1) Instructions on VM and ROS
2) Instructions on UR ROS Package
3) Instructions on Matlab and ROS

To avoid installing Matlab twice, you can use Matlab from Windows to communicate with ROS in your virtual machine. See the last instruction for that.

Now, please complete tasks 17 and 18 in the template.

Connect to ROS Master and List Topics

You can skip this description of ROS topics, publishers, subscribes and messages in Matlab in case you are already familiar with ROS commands in the Robotics System Toolbox from the course on Mobile Robots.

Notice: The Robotics System Toolbox mimics the syntax and commands of ROS on the Ubuntu command line. In other words the Robotics System Toolbox duplicates those commands in ROS that you usually enter via a terminal.

When you work with ROS, you will typically follow these steps:

- `rosinit`: tries to connect to a ROS master running on localhost. To connect to a ROS network, you can create the ROS master in MATLAB or connect to an existing ROS master (initiated via the Terminal). In both cases, MATLAB will also create and register its own ROS node (called the MATLAB global node) with the master.
- Exchange Data. Once connected, MATLAB exchanges data with other ROS nodes through publishers (`rospublisher`), subscribers (`rossubsriber`) and services (`rosservices`).
- `rosshutdown`: disconnects from the ROS network.

The command `rosnode list` lists all the nodes that are currently available on the ROS network. Similar `rostopic list` reports the topics that are currently available on the ROS network. With `rostopic info <topic>` you obtain specific information about a particular topic.

Now, please complete tasks 19 to 21 in the template.

ROS Subscriber and Publisher

ROS shares information using messages and services. Messages are a simple data structure for sharing data. Services use a pair of messages for a request-reply interaction.

You subscribe to a topic with `use rossubscriber` and receive messages on this topic with `receive`. You create the subscriber with

```
sub = rossubscriber(topicname,msgtype);
```

Utilize the subscriber to either receive the most recent message with

```
data = sub.LatestMessage;
```

or wait for the next message with

```
data = sub.receive();
```

In the first case the program control flow immediately returns to Matlab, in the second case the Matlab program waits for the next message to be published.

```
pub = rospublisher(topicname);
```

creates a publisher, `pub`, for a topic, `topicname`, that already exists on the ROS master topic list. The publisher gets the topic message type from the topic list on the ROS master. Whenever the Matlab node publishes messages on that topic, ROS nodes that subscribe to that topic receive those messages.

```
pub = rospublisher(topicname,msgname),
```

creates a publisher, `pub`, for a topic, `topicname`, that already exists on the ROS master topic list. If the ROS master topic list already contains a matching topic, the ROS master adds the MATLAB global node to the list of publishers for that topic.

```
msg=rosmessage(pub);
```

creates an empty message determined by the topic published by pub.

```
msg=rosmessage(pub);
```

creates an empty message determined by the topic published by pub.

```
msg = rosmessage(messagetype)
```

creates an empty ROS message object with message type.

The command

```
pub.send(msg);
```

publishes a message to the topic specified by the publisher pub. This message is received by all subscribers in the ROS network that subscribe to the topic.

The `robotics.Rate` object achieves the execution of for or while loops in Matlab at a fixed cycle rate. The command `waitfor`(rateObj) synchronizes the loop execution time with the ROS time. It pauses the execution of the loop until the cycle time for the current iteration expired. With `rateObj.reset` you reset the clock of the rate object `rateObj`. With `rateObj.TotalElapsedTime` denotes the total time that elapsed either since the instantiation of `rateObj` or the last `rateObj.reset`.

```
rateObj=robotics.Rate(rate);
rateObj.reset;  % reset time at the beginning of the loop
while (...)
  ...
  waitfor(rateObj); % delay loop to match control rate
end
```

Now, please complete tasks 22 to 27 in the template.

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.