5ELEN018W Robotic Principles - Tutorial 2

Simulation of Commercial Robots in Matlab and Creating Robot Models in Matlab

Matlab includes predefined models of commercial robots which can be loaded using the loadrobot function:

```
ur10 = loadrobot("universalUR10");  % loads the Universal Robots UR10 robot
```

Universal Robots UR10 Success Story

the `loadrobot()` function returns a `rigidBodyTree` object which describes the dynamics and kinematics of the robot.

```
disp(ur10)  % display the robot model (data only)

show(ur10)  % display with visualisation the actual robot
```

Boston Dynamics Atlas Humanoid Robot

```
atlas = loadrobot("atlas");
show(atlas)
```

Examining Robot Links and Joints

To see the body names of the links (rigid bodies) access the `BodyNames` property of the robot:

```
ur10.BodyNames
```

To access the links (rigid bodies) use the `Bodies` propery

```
ur10.Bodies

ur10.Bodies{1,3}  % access the 3rd link

ur10.Bodies{1,3}.Joint  % access the joint attached to the 3rd link
```

Robot Configurations

Create random configurations for the robot and display it in these configurations:

```
config1 = randomConfiguration(ur10)
show(ur10, config1)
```

```
config2 = randomConfiguration(ur10)
show(ur10, config2)
```

The homeconfiguration of a robot model, i.e. the configuration as predefined by the loaded robot model:

```
homeConfig = ur10.homeConfiguration
show(ur10, homeConfig)
```

Interacting with a Robot Model

To interact with the model of a robot, move it around and see how it behaves, use the `interactiveRigidBodyTree` function (this can be used to set target end-effector positions, manually move joints and select individual elements of the robot):

```
gui = interactiveRigidBodyTree(ur10)
```

Task 1: Use the centre disk to move the end-effector position. The GUI uses an inverse kinematics solver to calculate the joint positions for each body.

Task 2: Use the axes to move linearly and the circles to rotate about an axis.

Task 3: Click on any link to select that body. Its parameters will be displayed.

Task 4: Right click on any link and select "`set body as marker body`". The end-effector will be updated and the inverse kinematics solver will

calculate the joint positions accordingly.

Task 5: To control individual joints manually, right click and select "`Toggle Marker Control Method`". The coloured sphere will disappear. Right click on the link that you would like to manipulate and select "Set body as marker body" and interact with the link. To rotate a revolute joints click and drag the yellow circle.

Exercise

1. Load and display (visualise) the NASA Valkyrie Humanoid robot. Its name is: `valkyrie`
2. Examine some sample links (rigid bodies) of the robot and their names.
3. Display the robot in different random configurations.

Saving Arbitrary Joint Configurations

The `addConfiguration()` function can be used to store the current joints configuration in the `StoredConfigurations` property of the interactive GUI:

```
% use column vectors for joint configurations
ur10 = loadrobot("universalUR10","DataFormat","column");

gui.Configuration = randomConfiguration(ur10);
addConfiguration(gui);
disp(gui.StoredConfigurations)
```

Exercise

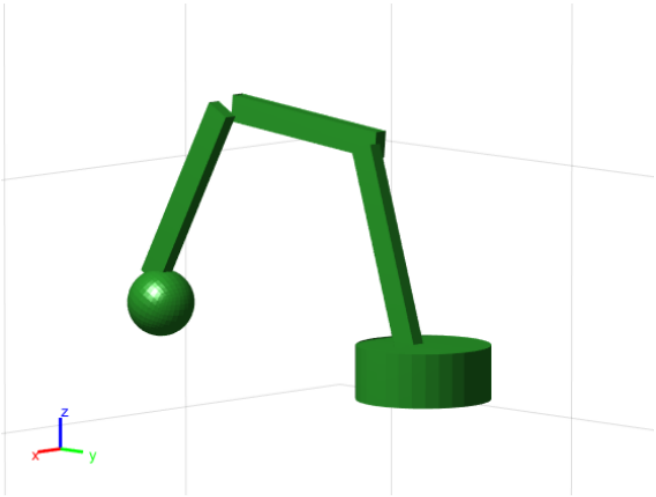
Look at all the commercial robot models available in Matlab at the following page:

Matlab Robot Models

Select your favourite model and repeat the same steps as above.

Building a Rigid Body Tree Robot Model

Here we will be building from scratch a new Matlab robot arm model with 5 degrees of freedom. Do not copy and paste the code but keep typing each section in the **Live Script Matlab editor** and run each section before proceeding to the next one.



Create a rigid body tree. The base of the robot is fixed and defines the world coordinates:

```
my_robot = rigidBodyTree("DataFormat", "column");
base = my_robot.Base;
```

Creating the Links (rigid bodies)

The robot consists of 1 rotating base, 3 rectangular arms and a gripper:

```
rotatingBase = rigidBody("Rotating Base");
arm1 = rigidBody("arm1");
arm2 = rigidBody("arm2");
arm3 = rigidBody("arm3");
gripper = rigidBody("gripper");
```

Create the collision objects for each link (rigid body) with different shapes and dimensions. By default, each collision object has a reference frame which is centered in the object. We set the pose of each object to move the reference frame to the body of each link along the *z*-axis.

The gripper is modelled as a sphere.

```
collBase = collisionCylinder(0.05, 0.04);  % parameters: radius, length
collBase.Pose = trvec2tform([0 0 0.04/2]);

coll1 = collisionBox(0.01, 0.02, 0.15);  % box: length, width, height (x,y,z)
coll1.Pose = trvec2tform([0, 0, 0.15/2]);

coll2 = collisionBox(0.01,0.02,0.15);    % box: length, width, height (x,y,z)
coll2.Pose = trvec2tform([0 0 0.15/2]);

coll3 = collisionBox(0.01,0.02,0.15);    % box: length, width, height (x,y,z)
coll3.Pose = trvec2tform([0 0 0.15/2]);

collGripper = collisionSphere(0.025);    % sphere: radius
collGripper.Pose = trvec2tform([0, -0.015, 0.025/2]);
```

Add the collision objects to the rigid bodies (links):

```
addCollision(rotatingBase, collBase);
addCollision(arm1, coll1);
addCollision(arm2, coll2);
addCollision(arm3, coll3);
addCollision(gripper, collGripper);
```

Create and Attach the Joints

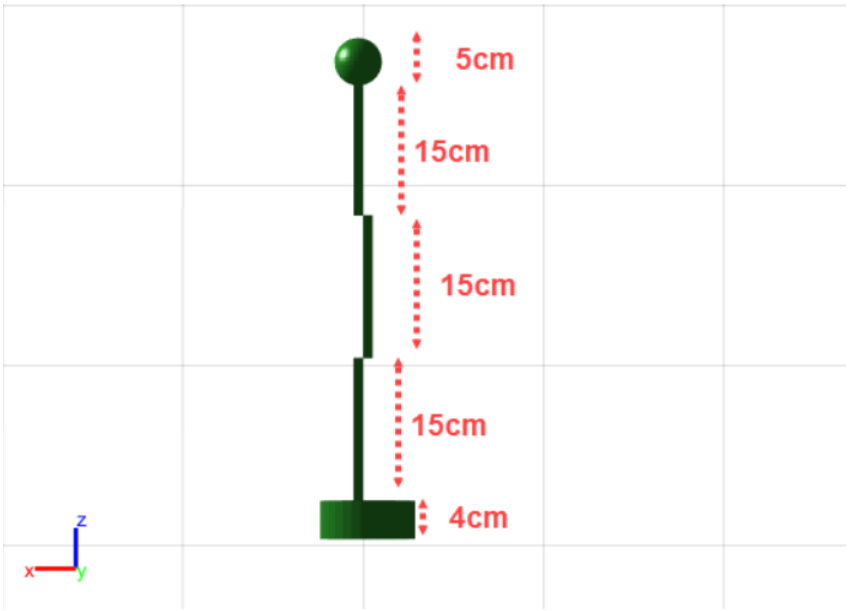
Each rigid body is attached to a revolute joint. The 3 rectangular arms have the *x*-axis as the axis of rotation. The *y*-axis is used for rotation of the gripper.

```
jntBase = rigidBodyJoint("base_joint", "revolute");
jnt1 = rigidBodyJoint("jnt1", "revolute");
jnt2 = rigidBodyJoint("jnt2", "revolute");
jnt3 = rigidBodyJoint("jnt3", "revolute");
jntGripper = rigidBodyJoint("gripper_joint", "revolute")

jnt1.JointAxis = [1 0 0];  % x-axis
```

```
jnt2.JointAxis = [1 0 0];  % x-axis
jnt3.JointAxis = [1 0 0];  % x-axis
jntGripper.JointAxis = [0 1 0];  % y-axis
```

Create transformations which position each joint. The position along the *z-axis* is based on the dimensions of the previous rigid body (link) length. Introduce an offset in the *x*-axis to avoid collisions during rotation.




```
setFixedTransform(jnt1, trvec2tform([0.015 0 0.04]));
setFixedTransform(jnt2, trvec2tform([-0.015, 0, 0.15]));
setFixedTransform(jnt3, trvec2tform([0.015, 0, 0.15]));
setFixedTransform(jntGripper, trvec2tform([0, 0, 0.15]));
```

Assemble the Robot

Create 2 cell arrays one with all links and another with the joints. Add each joint to each link (rigid body) and add the links to the rigid body tree:

```
bodies = {base, rotatingBase, arm1, arm2, arm3, gripper};
joints = {[], jntBase, jnt1, jnt2, jnt3, jntGripper};

figure("Name", "Assemble Robot", "Visible", "on")
for i=2:length(bodies)  % Skip base. Iterate through adding bodies and joints.
    bodies{i}.Joint = joints{i}
    % add the current rigid body with the body with a connection to
    % parent referenced by name
    addBody(my_robot, bodies{i}, bodies{i-1}.Name)
    show(my_robot, "Collisions", "on", "Frames","off")
end
```

Display the details of the robot. Make sure that the parent-child relationship and joint types are correct.

```
showdetails(my_robot)
```

Interacting with the created Robot Model

As previously, you can use the `interactiveRigidBodyTree` function to manipulate the robot you created:

```
figure("Name","Interactive GUI")
gui = interactiveRigidBodyTree(my_robot,"MarkerScaleFactor",0.25);
```