1. **Create the robot:**

   - Spawn the robot to the ROS-Gazebo environment. Include and image of the robot.
   - Include the robot definition file.

   Include `joint_state_publisher`

   Reference this connecting-gazebo-and-rviz

   To begin the simulation:

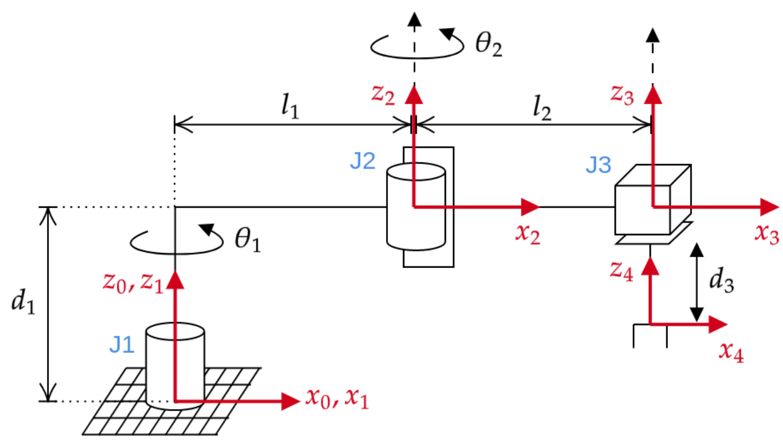
   - `roslaunch rrbot_gazebo rrbot_world.launch`

2. **Forward Kinematics:** Implement a FK node that

   - Subscribes to the joint values topic and reads them from the Gazebo simulator.
   - Calculates the pose of the end-effector.
   - Publishes the pose as a ROS topic (inside the callback function that reads the joint values).

   Print the resulting pose to the terminal using the `rostopic echo` command and include a screenshot of the results.

   We begin by assigning coordinate frames to the manipulator:



   We can then formulate the DH parameters coordinating to the links:

Link	θ_i	d_i	a_i	α_i
1	θ_1^*	d_1	l_1	0
2	θ_2^*	0	l_2	0
3	0	$-d_3^*$	0	0

   Next, we can calculate the transformations for each frame:

$$T_{i+1}^{i} = \text{Rot}_z(\theta_i)\text{Trans}_z(d_i)\text{Trans}_x(a_i)\text{Rot}_x(\alpha_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1\cos 0 & \sin\theta_1\sin 0 & l_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1\cos 0 & -\cos\theta_1\sin 0 & l_1\sin\theta_1 \\ 0 & \sin 0 & \cos 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & l_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & l_1\sin\theta_1 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2\cos 0 & \sin\theta_2\sin 0 & l_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2\cos 0 & -\cos\theta_2\sin 0 & l_2\sin\theta_2 \\ 0 & \sin 0 & \cos 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & l_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & l_2\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^3 = \begin{bmatrix} \cos 0 & -\sin 0\cos 0 & \sin 0\sin 0 & 0\cos 0 \\ \sin 0 & \cos 0\cos 0 & -\cos 0\sin 0 & 0\sin 0 \\ 0 & \sin 0 & \cos 0 & -d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The combined transformation of the end effector is:

$$T_4^0 = T_1^0 T_2^1 T_3^2 T_4^3$$

This will be used in the forward kinematic subscriber's callback function.

To run and test the subscriber/publisher function:

- Run `roslaunch rrbot_gazebo rrbot_world.launch`
- Set all PID gains in `rrbot_control.yaml` to zero
- In a new window `roslaunch rrbot_control rrbot_control.launch` which allows the joint states to be published.
- In a new window `rostopic list` should show /rrbot.joint_states
- `rostopic echo /rrbot.joint_states` should show the joint states printing
- `rosrun scara_robot configuration_to_operational_sub.py` will run the program that subscribes to the joint states, calculates the forward kinematics, and publishes the end-effector pose back to the `Pose` topic. The print out can be seen below:

3. **Inverse Kinematics:**

- Implement an IK node (separate node) that has a service client that take a desired pose of the end effector and returns joint positions as a response.
- Test your node with `rosservice call`. Include a screenshot of the results.

The following definitions can be used to calculate the inverse kinematics: