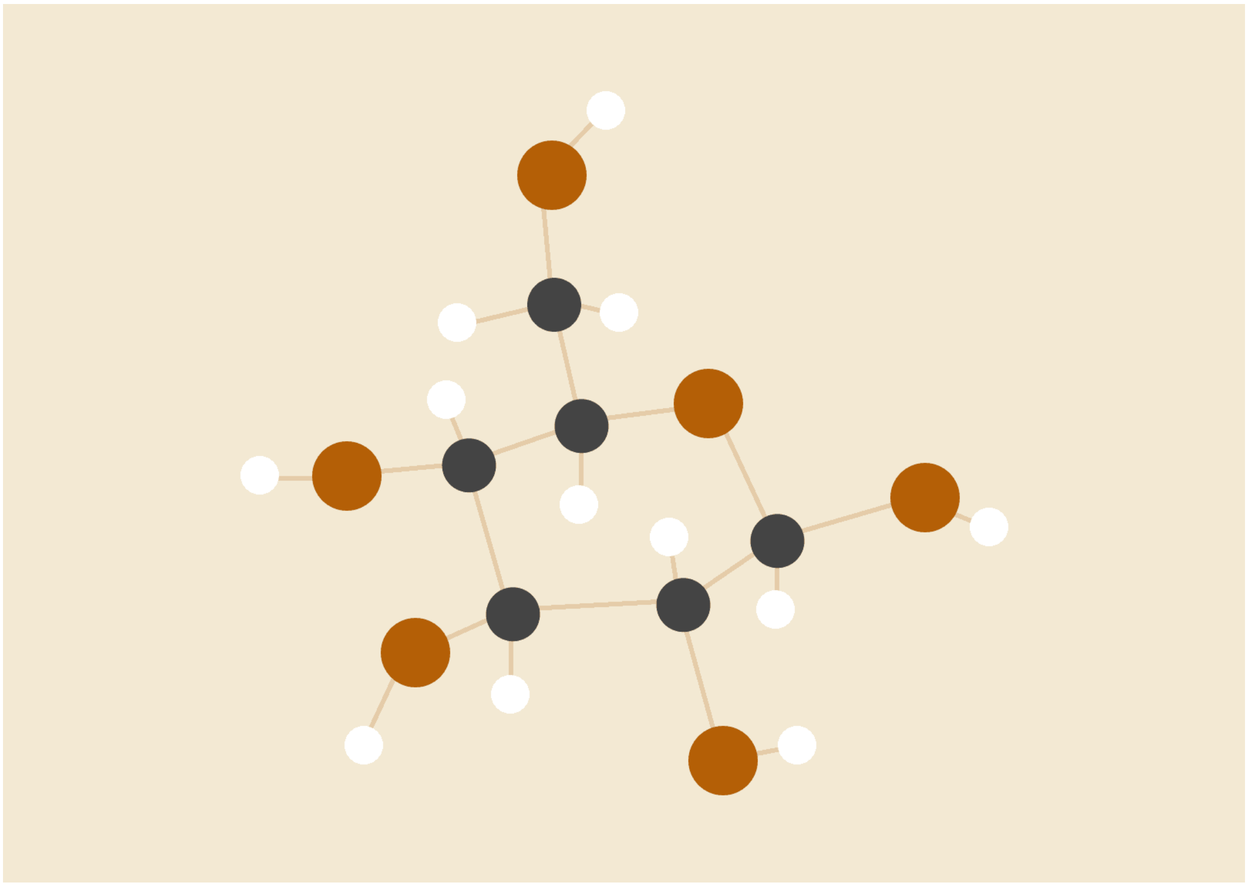
TECHNICAL ASSIGNMENT REPORT

Internship, Simulation of Soft Robotics (TUD)



Prakarsh Kaushik

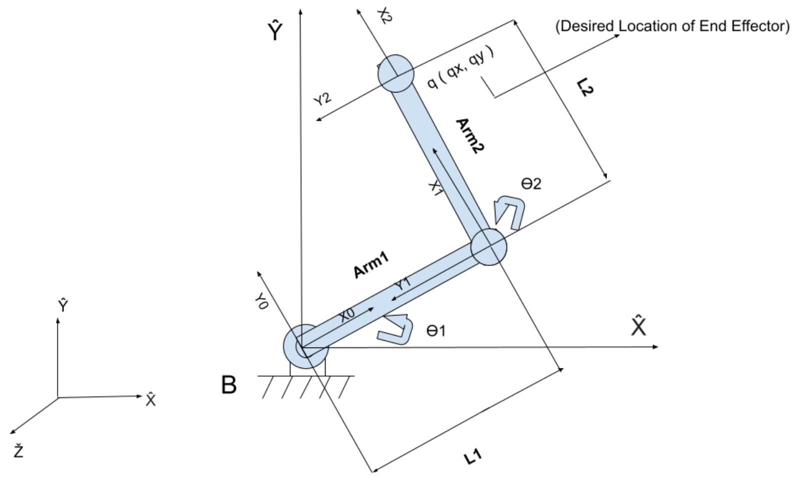
2 July, 2021

INTRODUCTION

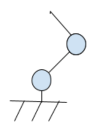
We are considering a robotic arm which consists of two rigid segments or links in 3D task space. Every link has a total mass of 1kg with uniform mass distribution and is of length 1m. The two segments are connected to each other with a rotational or revolute joint which can also be actuated (e.g., we can apply a torque to the rotational joint). We connect the two segments with an additional rotational and actuated joint to the ground plane. Gravity is acting in vertical direction - so rectangular to the ground plane. The exercises surround around the simulation and classical control of this manipulator.

Exercise 1 : Derivation of Kinematics

Derive the kinematics for the above described system mapping the joint angles to the cartesian position of the end-effector. The world coordinate system is centered at the base of the Robot.



Trigonometric Diagram for 2-DOF Manipulator



Kinematic Diagram of 2-DOF Manipulator

Forward Kinematics :- To determine position & orientation of the end effector w.r.t base coordinate system

Frame	Screw Z		Screw X	
	Θi	di	αi	ai
1 (wrt 0)	Θ1	0	0	L1
2 (wrt 1)	Θ2	0	0	L2

DH parameters Table

${}^{Base}_{2}T = {}^{Base}_{1}T \ {}^{1}_{2}T$

$({}^{Base}_{1}T = Rot(Z, \Theta1)\ Trans(X, L1)\ ,\ {}^{1}_{2}T = Rot(Z, \Theta2)\ Trans(X, L2))$

$= Rot(Z, \Theta1)\ Trans(X, L1)\ Rot(Z, \Theta2)\ Trans(X, L2)$

${}^{Base}_{1}T =$ (1)

$$\begin{bmatrix} c1 & -s1 & 0 & c1L1 \\ s1 & c1 & 0 & s1L1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

${}^{1}_{2}T =$ (2)

$$\begin{bmatrix} c2 & -s2 & 0 & c2L2 \\ s2 & c2 & 0 & s2L2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using the Eq.s (1) and (2), the homogeneous transformation matrix ${}^{Base}_{2}T$ is :

${}^{Base}_{2}T =$ (3)

$$\begin{bmatrix} c12 & -s12 & 0 & L1c1+L2c12 \\ s12 & c12 & 0 & L1s1+L2s12 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where c1 = cos(Θ1), c2 = cos(Θ2), s1 = sin(Θ1), s2 = sin(Θ2), c12 = cos(Θ1+Θ2), s12 = sin(Θ1+Θ2) with Θ1, Θ2 as joint angles and L1, L2 are the length of the rigid segments/links

From the matrix in equation (3) we can get the position coordinates of the manipulator's end-effector :-

$qx = L1\cos(\Theta 1) + L2\cos(\Theta 1+\Theta 2)$, (4)

$qy = L1\sin(\Theta 1) + L2\sin(\Theta 1+\Theta 2)$ (5)

The orientation matrix of the end-effector is defined by the first three rows and three columns of the transformation matrix in equation (3).

<u>Inverse Kinematics :-</u> To determine joint angles, provided end effector's position and orientation w.r.t base

We know from above the mentioned equations that,

$qx = L1c1 + L2c12$,

$qy = L1s1 + L2s12$

Taking the squares of Eq.s (4), (5) and adding them,

$qx^2+qy^2 = L1^2c1^2+L2^2c12^2+2L1c1L2c12+L1^2s1^2+L2^2s12^2+2L1s1L2s12$ (6)

$qx^2+qy^2-L1^2-L2^2 = 2L1L2c2$

$c2 = \frac{qx^2+qy^2-L1^2-L2^2}{2L1L2}$

$\Theta 2 = \arccos(\frac{qx^2+qy^2-L1^2-L2^2}{2L1L2})$ (7)

$qx = c1(L1+L2c2) - s1(L2s2)$

Assuming, $L1+L2c2 = \rho\sin\psi$,

$L2s2 = \rho\cos\psi$

$qx = c1\ \rho\sin\psi - s1\ \rho\cos\psi$

$qx = \rho\sin(\psi-\Theta 1)$ (8)

Solving we get,

$\rho = \sqrt{(L1 + L2c2)^2 + (L2s2)^2}$ (9)

Ψ = arctan($\frac{L1+L2c2}{L2s2}$) (10)

qy = ρcos(Ψ-Θ1) (11)

Dividing Eq. (8) by Eq. (11) we get,

Θ1 = Ψ - arctan($\frac{qx}{qy}$) (12)

where, Ψ is a function of length of links and cosine of joint angle 2, sine of joint angle 2 (which itself are a function of position coordinates of end-effector & length of links)

Eq.s (7) & (12) gives our joint angles of the manipulator as follows:

Θ1 = Ψ - arctan($\frac{qx}{qy}$) , Θ2 = arccos($\frac{qx^2+qy^2-L1^2-L2^2}{2L1L2}$) (where Ψ = arctan($\frac{L1+L2c2}{L2s2}$))

There are two sets of values for the joint angles for a given end effector’s position and orientation w.r.t base. They are called as ‘Left Hand solution’ & Right Hand solution’

Exercise 2 : Implementation of Simulation

Model the robotic arm in a simulator of your choice (for example Gazebo, Unity etc.). Choose all other parameters (for example diameter of segment) appropriately to your convenience. Connect this simulator to a ROS2 node. The node should have as inputs the demanded torques at both joints and as outputs the current joint angles the joint velocities and the current end-effector position in task space (so the 3D position of the end of the robotic arm). We additionally consider in a second step an articulated soft robot in the sense that the joints are soft. We can model this by adding an elastic potential force and damping to the joints.

:- The implementation of the simulation of the modeled serial manipulator has been done and tested in Gazebo11 as well as Gazebo9 in ROS1 Melodic & ROS2 Foxy respectively.

The ROS2 package for the simulation & control has been tested in a ROS2 Foxy docker container & a VNC server has been used to run the GUI of Gazebo.

The <dynamics> tag with damping factor has been scaled up to 70.0 while describing the joints in .urdf file to model the joints as soft for our articulated soft robot.

To run the solution for the exercise 2, one should follow the below mentioned consecutive commands (for ROS2 package) :

1. Launch the required nodes with the provided launch file by executing the following command in the terminal: **ros2 launch robotic_arm_1 rrbot_gazebo.launch.py**

2. Run your ROS2 Node for exercise 2 by executing the following command in the Terminal: **ros2 run robotic_arm_1 ex2**

To run the solution for the exercise 2, one should follow the below mentioned consecutive commands (for ROS1 package) :

1. Launch the required nodes with the provided launch file by executing the following command in the terminal: **roslaunch rr_robotic_arm rrbot_gazebo.launch**

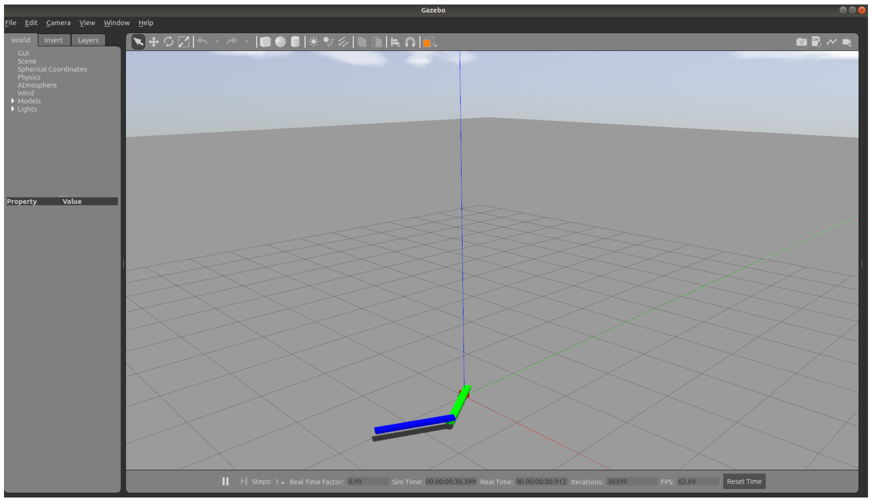
2. Load the joint controller configurations from YAML file to parameter server with the following Command: **roslaunch rr_robotic_arm rrbot_control1.launch**

3. Run your ROS1 Node for exercise 2 by executing the following command in the Terminal: **rosrun rr_robotic_arm ex2.py**

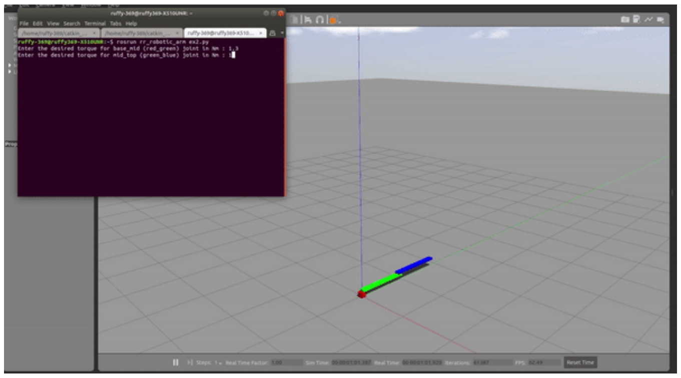
To run the same solution for the exercise 2 but on manipulator with soft joints, do as follows:

For ROS2 package: **ros2 launch robotic_arm_1 rrbot_gazebo_soft.launch.py**
ros2 run robotic_arm_1 ex2

For ROS1 package: **roslaunch rr_robotic_arm rrbot_gazebo_soft.launch**
roslaunch rr_robotic_arm rrbot_control1.launch
rosrun rr_robotic_arm ex2.py

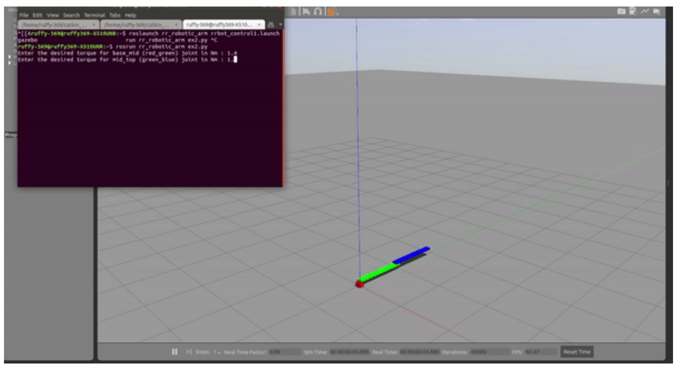


Modeled Robotic Arm in Gazebo Simulator



```
ruffy-369@ruffy369-X510UNR:~$ rosrun rr_robotic_arm ex2.py
Enter the desired torque for base_mid (red_green) joint in Nm : 1.3
Enter the desired torque for mid_top (green_blue) joint in Nm : 1
```

Torque control of the arm with normal joints



```
^[[Aruffy-369@ruffy369-X510UNR:~$ roslaunch rr_robotic_arm rrbot_control1.launch
gazebo                         run rr_robotic_arm ex2.py ^C
ruffy-369@ruffy369-X510UNR:~$ rosrun rr_robotic_arm ex2.py
Enter the desired torque for base_mid (red_green) joint in Nm : 1.4
Enter the desired torque for mid_top (green_blue) joint in Nm : 1.
```

Torque control of the arm with soft joints

Exercise 3 : Baseline Control

Implement an inverse kinematics algorithm in a ROS2 node. This node should take as input the demanded cartesian position of the end-effector and as output the required configuration of the robotic arm (e.g. joint angles) to reach this desired end-effector position. Implement an PID or PD torque controller in configuration space as a baseline in a separate ROS2 node. This PID controller should take the reference position of the end- effector and the current system state (so joint angles and velocities) as an input and give the necessary joint torques to reach the desired configuration (e.g. joint
angles).

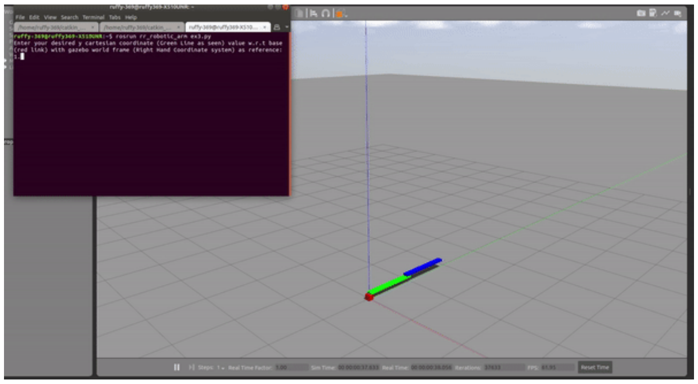
:- For implementation for the baseline PID control ros_control (in ROS1 package) & ros2_control (in ROS2 package) has been used.

For the inverse kinematics, an analytical approach has been used and the required joint angles position has been calculated with the help of the derivation in exercise1.

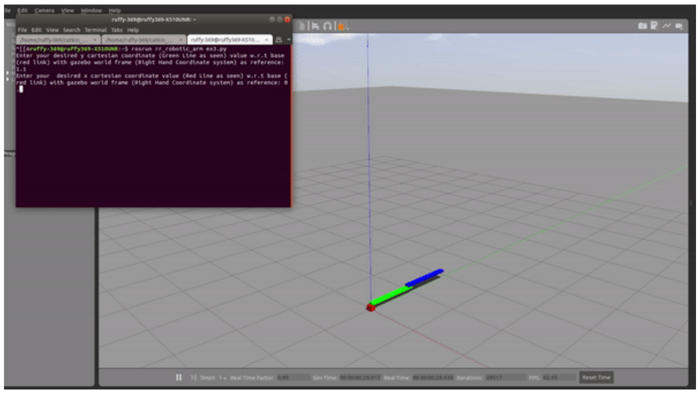
To run the solution for the exercise 3, one should follow the below mentioned consecutive commands (for

ROS2 package) :
1. Launch the required nodes with the provided launch file by executing the following command in the terminal: **ros2 launch robotic_arm_1 rrbot_gazebo1.launch.py**
2. Run your ROS2 Node for exercise 2 by executing the following command in the Terminal: **ros2 run robotic_arm_1 ex3**
To run the solution for the exercise 2, one should follow the below mentioned consecutive commands (for ROS1 package) :
1. Launch the required nodes with the provided launch file by executing the following command in the terminal: **roslaunch rr_robotic_arm rrbot_gazebo.launch**
2. Load the joint controller configurations from YAML file to parameter server with the following Command: **roslaunch rr_robotic_arm rrbot_control.launch**
3. Run your ROS1 Node for exercise 2 by executing the following command in the Terminal: **rosrun rr_robotic_arm ex3.py**

To run the same solution for the exercise 2 but on manipulator with soft joints, do as follows:
For ROS2 package: **ros2 launch robotic_arm_1 rrbot_gazebo1_soft.launch.py**
ros2 run robotic_arm_1 ex3
For ROS1 package: **roslaunch rr_robotic_arm rrbot_gazebo_soft.launch**
roslaunch rr_robotic_arm rrbot_control.launch
rosrun rr_robotic_arm ex3.py


```
ruffy-369@ruffy369-X510UNR:~$ rosrun rr_robotic_arm ex3.py
Enter your desired y cartesian coordinate (Green Line as seen) value w.r.t base
(red link) with gazebo world frame (Right Hand Coordinate system) as reference:
1.
```

Position control of the arm with normal joints



```
^[[Aruffy-369@ruffy369-X510UNR:~$ rosrun rr_robotic_arm ex3.py
Enter your desired y cartesian coordinate (Green Line as seen) value w.r.t base
(red link) with gazebo world frame (Right Hand Coordinate system) as reference:
1.1
Enter your  desired x cartesian coordinate value (Red Line as seen) w.r.t base (
red link) with gazebo world frame (Right Hand Coordinate system) as reference: 0
.
```

Position control of the arm with soft joints

Dependencies for ROS2 Nodes: ros foxy control

ros foxy controllers

ros2 control

ros2 controllers

Important error encountered while working with ROS2 nodes in docker container:

Start at Line 44 in /opt/ros/foxy/lib/python3.8/site-packages/ros2controlcli/verb/load_controller.py

args.state replace it with **args.set_state** in each line

Error may vary from user to user as when I checked the code base on github, the correct argument was accessed.