UNIVERSITÀ DEGLI STUDI DI GENOVA

DIBRIS

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY,
BIOENGINEERING, ROBOTICS AND SYSTEM ENGINEERING

MODELLING AND CONTROL OF MANIPULATORS

Second Assignment

Manipulator Geometry and Direct Kinematics

Author:

Mamoru Ota

Student ID:

s7784976

Professors:

Enrico Simetti
Giorgio Cannata

Tutors:

Andrea Tiranti
Luca Tarasi
George Kurshakov

December 11, 2024

Contents

Mathematical expression	Definition	MATLAB expression
$< w >$	World Coordinate Frame	w
${}^{a}_{b}R$	Rotation matrix of frame $< b >$ with respect to frame $< a >$	aRb
${}^{a}_{b}T$	Transformation matrix of frame $< b >$ with respect to frame $< a >$	aTb

Table 1: Nomenclature Table

1 Assignment description

The second assignment of Modelling and Control of Manipulators focuses on manipulators' geometry and direct kinematics.

- Download the .zip file called *template_MATLAB-assignment2* from the Aulaweb page of this course.
- Implement the code to solve the exercises on MATLAB by filling the template classes called *geometric-Model* and *kinematicModel*
- Write a report motivating your answers, following the predefined format on this document.

1.1 Exercise 1

Given the following CAD model of an industrial 7 DoF manipulator:

Q1.1 Define all the model matrices, by filling the structures in the *BuildTree()* function.

Q1.2 Implement the method of *geometricModel* called *updateDirectGeometry()* which should compute the model matrices as a function of the joint position q. Explain the method used and comment on the results obtained.

Q1.3 Implement the method of *geometricModel* called *getTransformWrtBase()* which should compute the transformation matrix from the base to a given frame. Calculate the following transformation matrices: ${}^{b}_{e}T$, ${}^{5}_{3}T$. Explain the method used and comment on the results obtained.

Q1.4 Implement the method of *kinematicModel* called *updateJacobian()* which should compute the jacobian of a given geometric model considering the possibility of having *rotational* or *prismatic* joints. Compute the Jacobian matrix of the manipulator for the end-effector. Explain the method used and comment on the results obtained.

Remark: The methods should be implemented for a generic serial manipulator. For instance, joint types, and the number of joints should be parameters.

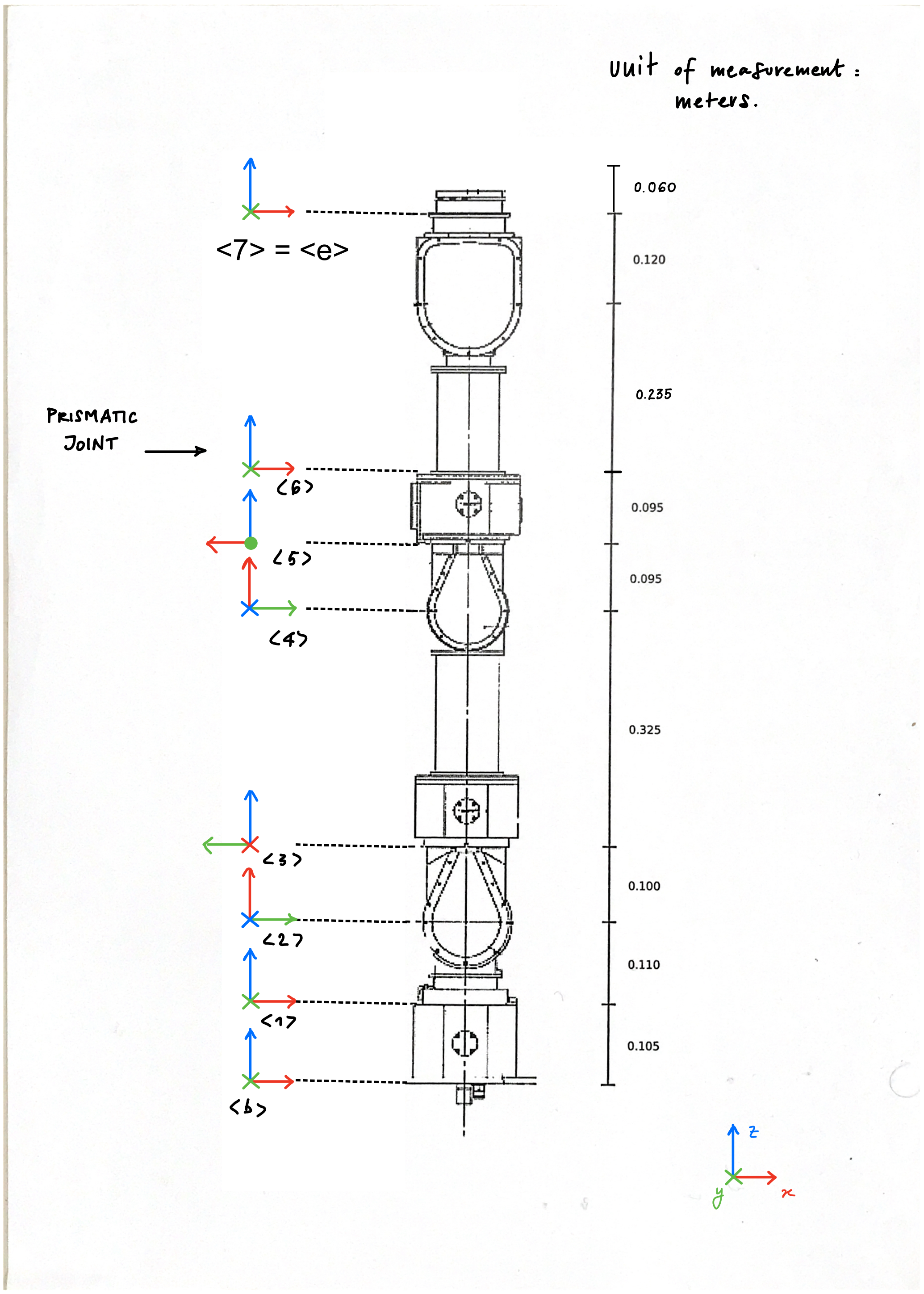

Figure 1: CAD model of the robot

2 Exercise 1

[Comment] For the last exercises include an image of the initial robot image of the final robot configuration.

[Comment] For each exercise report the results obtained and provide an explanation of the result obtained (even though it might seem trivial). The MATLAB code is NOT an explanation of the algorithm.

Abstract

This report documents the computation of kinematic properties for a 7-DoF manipulator. The tasks include defining the initial transformation matrices, computing forward kinematics (direct geometry), deriving the transformation from the base frame to any given frame, and computing the Jacobian matrix for the end-effector. The results have been verified through visualization and analytical checks.

Introduction

In this exercise, we analyze a 7-DoF manipulator. We define the kinematic structure using transformation matrices, compute forward kinematics, and determine the end-effector's pose. We also derive the Jacobian matrix, which relates joint velocities to the linear and angular velocities of the end-effector.

Key objectives:

- Define the model matrices ${}^i_jT$ for the manipulator.
- Compute the transformation from the base to any frame k.
- Compute the Jacobian $\mathbf{J}(\mathbf{q})$, considering both rotational and prismatic joints.

Q1.1: Defining Model Matrices

The initial transformation matrices are defined from the given CAD model. Each matrix describes the pose of link j with respect to link i when all joint variables are zero.

For example, a single link transformation matrix can be expressed as:

$$ {}^i_jT = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} $$

where R is a rotation matrix and $\mathbf{t}$ is a translation vector.

The model matrices were defined using the `BuildTree()` function based on the CAD model. The transformation matrices for ${}^i_jT$ are:

$$ {}^b_1T = \begin{bmatrix} 1.0000 & 0 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 1.0000 & 0.1050 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} $$

$$ {}^1_2T = \begin{bmatrix} 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 1.0000 & 0 \\ 1.0000 & 0 & 0 & 0.1100 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} $$

$$ {}^2_3T = \begin{bmatrix} 0 & 0 & 1.0000 & 0.1000 \\ 0 & -1.0000 & 0 & 0 \\ 1.0000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} $$

$$ {}^3_4T = \begin{bmatrix} 0 & 0 & 1.0000 & 0 \\ 0 & -1.0000 & 0 & 0 \\ 1.0000 & 0 & 0 & 0.3250 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} $$

$$ {}^4_5T = \begin{bmatrix} 0 & 0 & 1.0000 & 0.0950 \\ -1.0000 & 0 & 0 & 0 \\ 0 & -1.0000 & 0 & 0 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} $$

$$
{}^{5}_{6}T = \begin{bmatrix} -1.0000 & 0 & 0 & 0 \\ 0 & -1.0000 & 0 & 0 \\ 0 & 0 & 1.0000 & 0.0950 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}
$$

$$
{}^{6}_{7}T = {}^{6}_{e}T = \begin{bmatrix} 1.0000 & 0 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 1.0000 & 0.3550 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}
$$

Q1.2: Updating Direct Geometry

To compute the direct geometry for a given configuration $\mathbf{q} = [q_1, q_2, \ldots, q_n]^T$, we update each joint transformation. For a rotational joint (type 0), we apply a rotation about the z-axis:

$$
R_z(q_i) = \begin{bmatrix} \cos(q_i) & -\sin(q_i) & 0 \\ \sin(q_i) & \cos(q_i) & 0 \\ 0 & 0 & 1 \end{bmatrix}.
$$

For a prismatic joint (type 1), we apply a translation along the z-axis:

$$
T_z(q_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_i \\ 0 & 0 & 0 & 1 \end{bmatrix}.
$$

Pseudocode for updating direct geometry:

Algorithm 1 updateDirectGeometry($\mathbf{q}$)

```
1: Input: $\mathbf{q}$ (joint positions)
2: Output: ${}^{i}_{j}T$ updated with joint configurations
3: for $i = 1 \to n$ do
4:     if jointType(i) = rotational then
5:         Apply $R_z(q_i)$
6:     else if jointType(i) = prismatic then
7:         Apply $T_z(q_i)$
8:     end if
9:     Update ${}^{i}_{j}T$ by multiplying the initial matrix by the computed rotation/translation.
10: end for
```

The method `updateDirectGeometry()` computes the transformation matrices ${}^{i}_{j}T$ as a function of joint position q. For $q = [\pi/4, -\pi/4, 0, -\pi/4, 0, 0.15, \pi/4]$, the computed transformation matrices ${}^{i}_{j}T$ are:

$$
{}^{b}_{1}T = \begin{bmatrix} 0.7071 & -0.7071 & 0 & 0 \\ 0.7071 & 0.7071 & 0 & 0 \\ 0 & 0 & 1.0000 & 0.1050 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}
$$

$$
{}^{1}_{2}T = \begin{bmatrix} -0.7071 & 0.7071 & 0 & 0 \\ 0 & 0 & 1.0000 & 0 \\ 0.7071 & 0.7071 & 0 & 0.1100 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}
$$

$$
{}^{2}_{3}T = \begin{bmatrix} 0 & 0 & 1.0000 & 0.1000 \\ 0 & -1.0000 & 0 & 0 \\ 1.0000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}
$$

$$
{}^{3}_{4}T = \begin{bmatrix} 0 & 0 & 1.0000 & 0 \\ 0.7071 & -0.7071 & 0 & 0 \\ 0.7071 & 0.7071 & 0 & 0.3250 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}
$$

$$ {}^4_5T = \begin{bmatrix} 0 & 0 & 1.0000 & 0.0950 \\ -1.0000 & 0 & 0 & 0 \\ 0 & -1.0000 & 0 & 0 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} $$

$$ {}^5_6T = \begin{bmatrix} -1.0000 & 0 & 0 & 0 \\ 0 & -1.0000 & 0 & 0 \\ 0 & 0 & 1.0000 & 0.2450 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} $$

$$ {}^6_7T = \begin{bmatrix} 0.7071 & -0.7071 & 0 & 0 \\ 0.7071 & 0.7071 & 0 & 0 \\ 0 & 0 & 1.0000 & 0.3550 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} $$

Q1.3: Any Frame-to-Frame Transformation

To get the transformation ${}^b_kT$ from the base b to a frame k, we accumulate the transformations:

$$ {}^b_kT = \prod_{i=1}^{k} {}^{i-1}_{i}T(\mathbf{q}). $$

This allows us to compute intermediate or end-effector frames with respect to the base.

The method `getTransformWrtBase()` computes the transformation matrix from the base to any given frame k. Below is the pseudocode for the implementation:

Algorithm 2 Compute Transformation from Base to Frame k

```
1: Input: $k$ (frame index)
2: Output: ${}^b_kT$ (transformation matrix from base to frame $k$)
3: Initialize ${}^b_kT \leftarrow I_{4\times4}$                    ▷ Identity matrix
4: for $i \leftarrow 1$ to $k$ do
5:     ${}^b_kT \leftarrow {}^b_k T \cdot \texttt{self.}{}^i_jT$        ▷ Accumulate transformations
6: end for
```

The transformation from the base to the end-effector ${}^b_eT$ for the given configuration is:

$$ {}^b_eT = \begin{bmatrix} -0.5000 & -0.5000 & -0.7071 & -0.7039 \\ 0.5000 & 0.5000 & -0.7071 & -0.7039 \\ 0.7071 & -0.7071 & 0 & 0.5155 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} $$

The transformation matrix ${}^5_3T$ is calculated as follows:

$$ {}^5_3T = {}^b_3T^{-1} \cdot {}^b_5T $$

Resulting in:

$$ {}^5_3T = \begin{bmatrix} 0 & 0.7071 & -0.7071 & 0.2298 \\ -1.0000 & 0 & 0 & 0 \\ 0 & 0.7071 & 0.7071 & -0.3248 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} $$

The matrices ${}^b_eT$ and ${}^5_3T$ accurately describe the transformations based on the manipulator's current configuration.

Q1.4: Computing the Jacobian

The Jacobian $\mathbf{J}$ relates joint velocities $\dot{\mathbf{q}}$ to the end-effector angular velocity ω and linear velocity $\mathbf{v}$:

$$\begin{bmatrix} \omega \\ \mathbf{v} \end{bmatrix} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}.$$

For each joint i: - If the joint is rotational, let $\mathbf{z}_i$ be the unit vector along the joint axis. The linear part of the Jacobian contribution is $\mathbf{z}_i \times (\mathbf{p}_e - \mathbf{p}_i)$ and the angular part is $\mathbf{z}_i$. - If the joint is prismatic, the linear part is simply $\mathbf{z}_i$ and the angular part is $\mathbf{0}$.

Pseudocode:

Algorithm 3 updateJacobian()

```
 1: Input: Current configuration $\mathbf{q}$
 2: Output: Jacobian $\mathbf{J}$
 3: Compute ${}^b_eT$ (base to end-effector)
 4: for $i = 1 \to n$ do
 5:     Compute ${}^b_iT$ (base to joint $i$)
 6:     Extract $\mathbf{p}_i$ and $\mathbf{z}_i$
 7:     $\mathbf{p}_e$ is from ${}^b_eT$
 8:     if rotational then
 9:         $\mathbf{J}_{1:3,i} = \mathbf{z}_i$
10:         $\mathbf{J}_{4:6,i} = \mathbf{z}_i \times (\mathbf{p}_e - \mathbf{p}_i)$
11:     else if prismatic then
12:         $\mathbf{J}_{1:3,i} = \mathbf{0}$
13:         $\mathbf{J}_{4:6,i} = \mathbf{z}_i$
14:     end if
15: end for
```

The Jacobian matrix J is computed using the method `updateJacobian()`. The computed Jacobian matrix J for the given configuration q is:

$$J = \begin{bmatrix} 0 & -0.7071 & -0.5000 & -0.7071 & -0.7071 & 0 & -0.7071 \\ 0 & 0.7071 & -0.5000 & 0.7071 & -0.7071 & 0 & -0.7071 \\ 1.0000 & 0 & 0.7071 & 0 & 0 & 0 & 0 \\ 0.7039 & 0.2125 & 0.3475 & 0 & 0 & -0.7071 & 0 \\ -0.7039 & 0.2125 & -0.3475 & 0 & 0 & -0.7071 & 0 \\ 0 & 0.9955 & -0.0000 & 0.6950 & -0.0000 & 0 & 0 \end{bmatrix}$$

Results and Visualization

- Initial and final configurations of the manipulator are shown in Figure 2.
- The trajectory of motion is visualized in Figure 3.

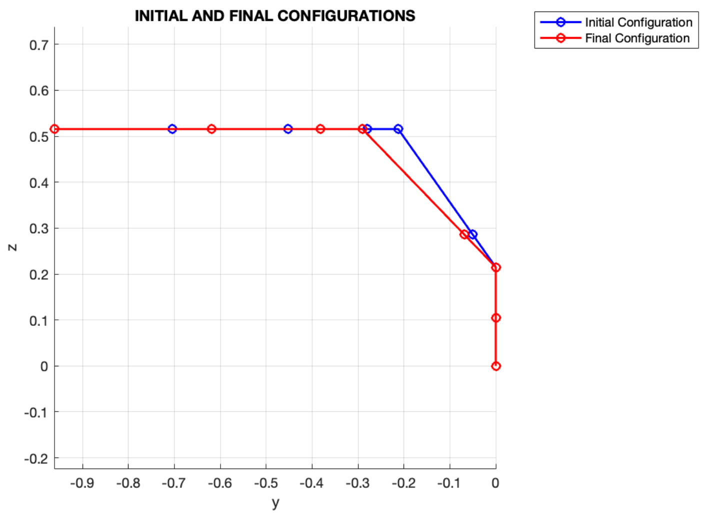

Figure 2: Initial and final configurations of the manipulator.

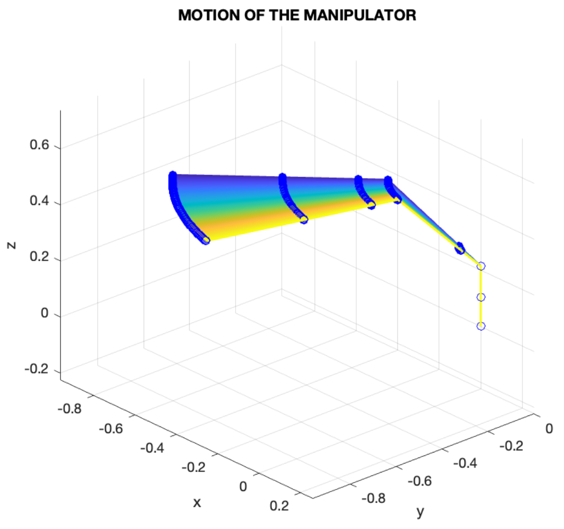

Figure 3: Motion of the manipulator.

3 Appendix

[Comment] Add here additional material (if needed)

3.1 Appendix A

3.2 Appendix B