Dominick Mulder dam3888
Nicolas Brissonneau nb24488

Programming Assignment

Introduction

We aimed to develop a program as reusable as possible which enables the user to visualize his n-degrees of freedom robot in Matlab, with only inputs the zero-angle end-effector transform matrix M, rotation axis ω_s, joints origins q_s, and the Denavit-Hartenberg parameters (compatible with the normal and modified versions).

We mostly used the following **built-in matlab functions**: norm(), cos(), sin(), trace(), cross(), dot(), inv().

We **imported** a 3d cylinder-representation function file **"cylinder2p.m"** from Per Sundqvist to ease the system's visualization.

We cooperated in the understanding and exploration of every question, however we tried to optimize the time spent individually. Dominick was more inclined to work on the theory part and Nicolas was more motivated by the programming assignment. Whenever distributing tasks, we always started by agreeing on an approach to start with for solving our different tasks, let each other deepen it further and whenever an obstacle arised we had the advantage of providing a "fresh look" on it.

We distributed the workload as follows:

- Programming assignment: **70% Nicolas** and **30% Dominick.**
- Homework assignment: **70% Dominick** and **30% Nicolas.**

Solving the assignment

First, we look for the forward kinematics using the space form of the exponential product (PoE).

We have the definition of PoE:

$$T(\theta) = e^{[S_1]\theta_1} \dots e^{[S_n]\theta_n} M$$

Where S_i is the i-th degree of freedom among the robot's n joints, and M is the home position of the robot when $\theta_i = 0$, $\forall\ i \in [1, \dots, n]$.

We can verify that we have succeeded into deriving the correct forward kinematics by plotting the frame resulting from $T(\theta)$ for different and intuitive values θ_i. For example, we can set θ_1 or θ_n *to* $\frac{\pi}{3}$ as it should appear as an easy visual change to confirm our results.

However, we first need to derive the screw axis S_i by deriving the associated rotation ω_i and velocities v_i axis. We identify all ω_i visually and deduce the corresponding v_i as $-\omega_i \wedge q_i$ with q_i identified from the associated screw axis. In this case we consider 7 rotative degrees of freedom so we don't have to consider prismatic joints, however we added the possibility to set a vector t_i for that purpose if need be.

Then, we can get $e^{[S]\theta}$ using:

$$e^{[S]\theta} = \begin{bmatrix} e^{[\omega]\theta} & * \\ 0\ 0\ 0 & 1 \end{bmatrix} \text{; with } * = (I\theta + (1 - cos(\theta))[\omega] + (\theta - \sin sin\ (\theta)\ )[\omega^2])\, v$$

Now if we follow the initial testing plan, we might still be confused if we simply plot the output as it is a frame floating in the air, so we will answer the second question to verify everything altogether.

The second question asks to calculate FK_{space} using the PoE approach and to plot the screw axis along the frames resulting from FK_{space}. We will go one step further and plot all the frames associated with a joint. So we have:

$$FK_{space} = e^{[S_1]\theta_1} \dots e^{[S_7]\theta_7} M_{endeff}$$

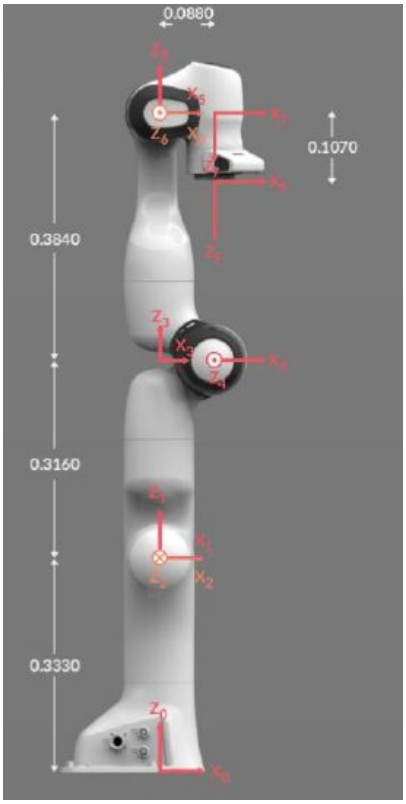
$$FK_{j_7} = e^{[S_1]\theta_1} \dots e^{[S_6]\theta_6} M_7$$

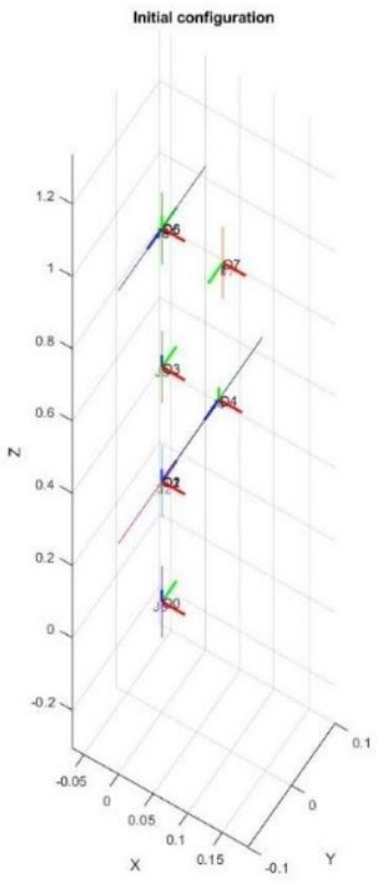
$$\vdots$$

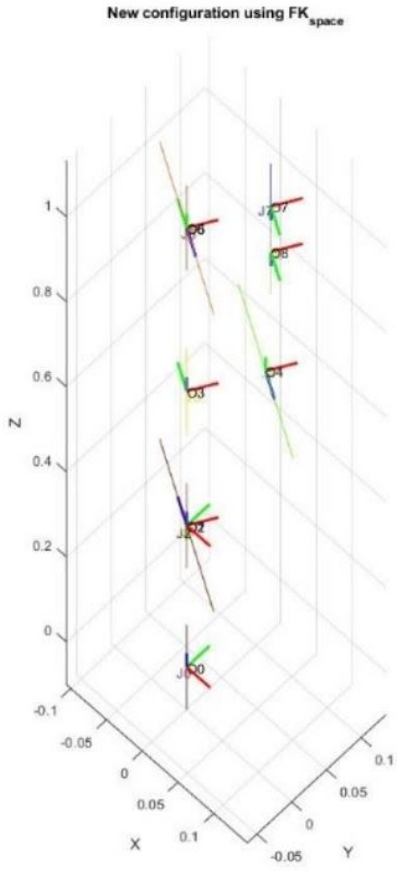
We could "manually" identify each M_i as we did for M_{endeff} but to avoid being redundant we will use the modified Denavit-Hartenberg notation to compute each of them with the table provided in the homework.

In the following figures we present from left to right: the picture of Panda, our initial configuration for $\theta_i = 0$, and our new configuration with $\theta_1 = \frac{\pi}{3}$. In the middle figure we recognize that each frame's x-axis (in red) is pointing toward the same direction as shown in the left-side picture, and in the final

configuration the frame corresponding to S_2 shows a $\frac{\pi}{3}\,rad$ rotation. We conclude that our functions are working properly.







We are then interested in finding the body form of the forward kinematics FK_{body}. We know that:

$$T(\theta) = Me^{[B_1]\theta_1} \ldots e^{[B_n]\theta_n} \text{ with } [B_i] = \left[Ad_{M^{-1}}\right] S_i$$

We can reuse the screw axis derived previously and compute $[B_i]$ with:

$$[Ad_G] = \begin{bmatrix} R_G & 0 \\ [p_G]R_G & R_G \end{bmatrix}, \quad H^1 = \begin{bmatrix} R_H^T & -R_H^T p_H \\ 0 & 1 \end{bmatrix} \text{ with } (G,H) \in SE(3)$$

We can then verify that we have $FK_{space} = FK_{body}$ for a given set of angles with:

$$FK_{body} = M_{endeff} e^{[B_1]\theta_1} \ldots e^{[B_n]\theta_n}$$

The next part of the homework requires to find the space Jacobian J_s and the body Jacobian J_b, we know that:

$$J_{s_i}(\theta) = Ad_{e^{[S_1]\theta_1 \ldots e^{[S_{i-1}]\theta_{i-1}}}} S_i \text{ for } i \in [2, \ldots, n] \text{ and } J_{s_1} = S_1 \text{ with } J_{s_i} \text{ the i-th column of } J_s .$$

$$J_{b_i}(\theta) = Ad_{e^{-[B_n]\theta_n \ldots e^{-[B_{i+1}]\theta_{i+1}}}} B_i \text{ for } i \in [n-1, \ldots, 1] \text{ and } J_{b_n} = B_n \text{ with } J_{b_i} \text{ the i-th column of } J_b .$$

We chose to verify these results by testing them against the written part of the assignment (Exercise 5.12) as it seemed the most straightforward way to do it before using the inverse kinematics algorithm.

We are now interested in the manipulability ellipsoids ant its axes, we know that an ellipsoid is associated with the linear velocities and another one with the angular velocities. The semi-axis directions of the linear velocities ellipsoid is defined as the eigenvectors of A in:

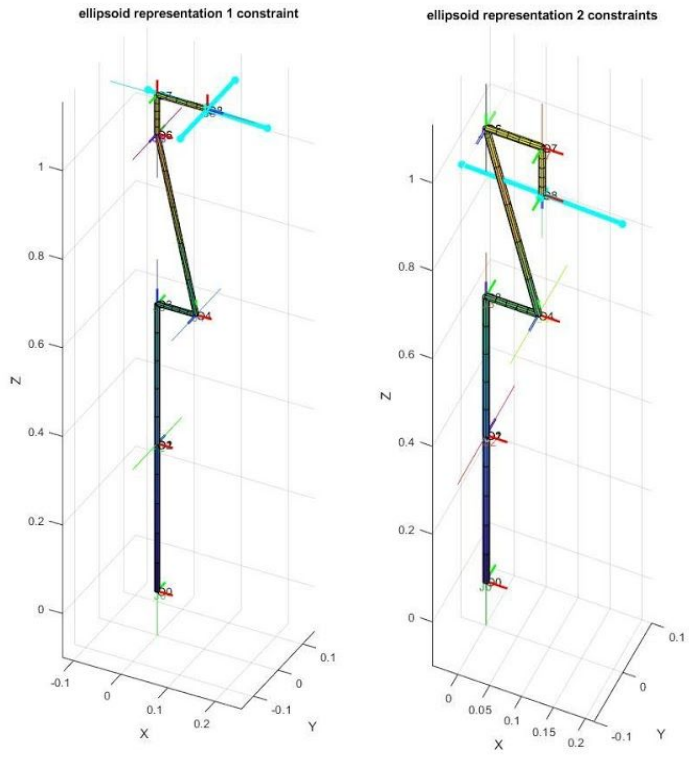
$$A = J_v J_v^T \text{ , with } J = [J_\omega; J_v]$$

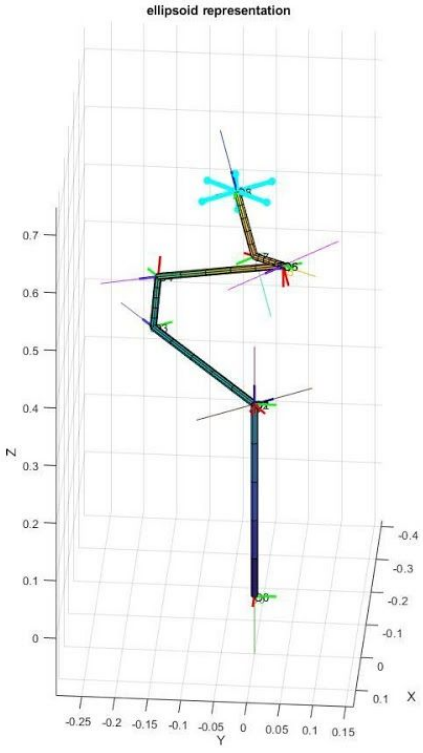
Respectively for the angular velocities the semi-axis directions of the angular velocities ellipsoid is defined as the eigenvectors of B in:

$$B = J_\omega J_\omega^T \text{ , with } J = [J_\omega; J_v]$$

For each of the ellipsoid, the length of each semi-axis is defined as the corresponding eigenvalue's square root $\sqrt{\lambda}$.

To check our results, we chose to compare the results between a common configuration and an extreme case one, close to singularity. Showing the ellipsoids axes we should observe a drastically flattened ellipsoid when being close to singularity, we take advantage of this question to introduce our cylinder-based visualization:





The ellipsoid's axis are visible in cyan, and correspond in these plots to the linear velocities. They have been re-scaled by a factor $\frac{1}{10}$ to improve the visualization, it is here clear from left to right that some configurations a more constraining than others. The extreme case on the right side corresponds to the zero-configuration, when most of the joints are parallel.

We can also learn more about our robot's properties from the ellipsoids' eigenvalues, indeed we know that the volume of each corresponds to V :

$$V = \sqrt{\lambda_1 \dots \lambda_m}$$

The isotropy feature is characterized by $\mu_1 = \sqrt{\frac{\lambda_{max}}{\lambda_{min}}}$, and we have the condition number $\mu_2 = \frac{\lambda_{max}}{\lambda_{min}}$.

Now looking toward moving the robot to a different configuration so that the end effector reaches a desired configuration T_{sd} , we have implemented an inverse kinematics algorithm:

$$\theta^{i+1} = \theta^i + J_b^{\dagger}\left(\theta^i\right) V_b \text{ with } [V_b] = \log log\,(T_{sb}^{-1}\left(\theta^i\right) T_{sd})$$

We can compute $J^{\dagger}$ for $J \in R^{mxn}$ by defining it as:

- $J^{\dagger} = J^T\left(JJ^T\right)^{-1}$ if $n > m$

- $J^{\dagger} = \left(J^T J\right)^{-1} J^T$ if $n < m$

Using this algorithm, we define V_b as the error and have two metrics, $err_{\omega b}$ and $err_{v b}$ corresponding to V_b's two subvectors.

We have also implemented the Jacobian transpose algorithm:

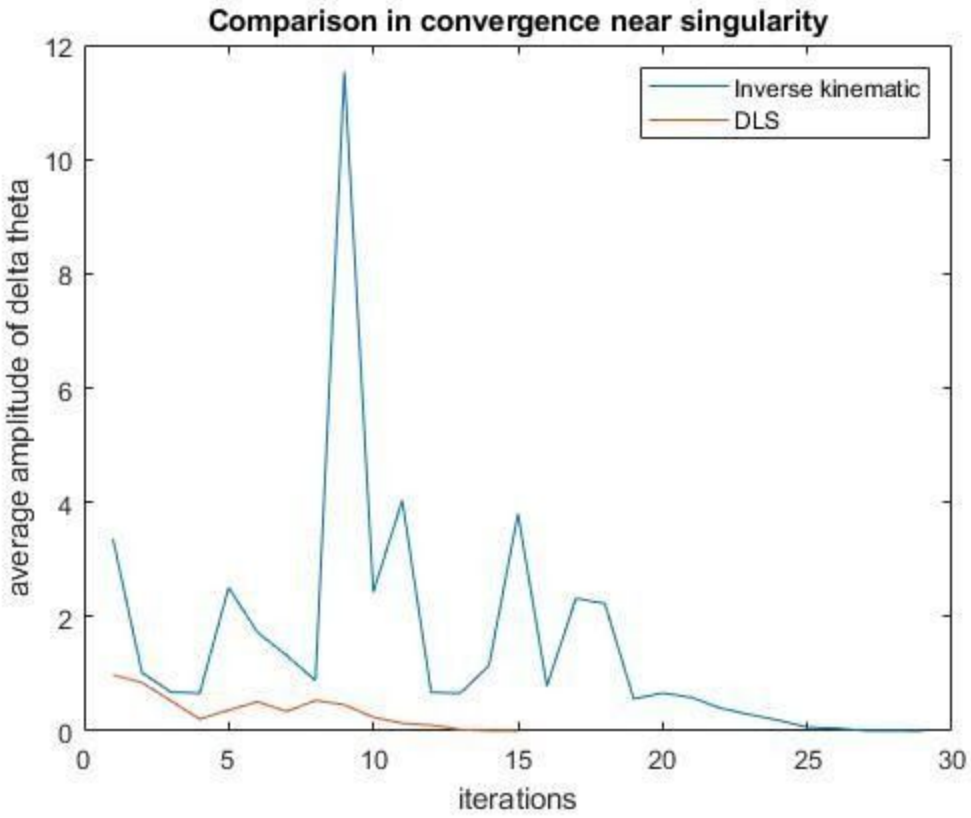
$$\theta^{i+1} = \theta^i + J_{b_v}^T\left(\theta^i\right) v_b$$

To verify our implementation is working properly, we can define a trajectory to follow, or simply verify that our algorithms converge.

We implemented the Damped Least Square (DLS) approach which essentially overcomes the problem of inverting differential kinematics in near singularity configurations by defining the new Jacobian J^*:

$$J^* = J^T\left(JJ^T + k^2 I\right)^{-1}$$

We checked our implementation by keeping track of the amplitude of each "$\Delta\theta$ step" as the iterative algorithm is running. To show off the benefits of this approach, we a configuration very close to the zero-configuration to remain close to a singularity. As we can see the traditional inverse kinematics algorithm spikes when it crosses this singularity, but for each spike of the inverse kinematics algorithm we observe a much attenuated one for DLS.


Comparison in convergence near singularity
Inverse kinematic
DLS
average amplitude of delta theta
iterations
0
2
4
6
8
10
12
0
5
10
15
20
25
30

-