KINOVA

Inverse Kinematics using Newton's algorithm

2021-04-21

Gen3 lite Cursus

Table of content

Objectives

In our last two labs on the inverse kinematics, we relied heavily on the fact that we were able to get a reasonable approximation of a symbolic model. However, for some robots, this is not always true. In this lab, we will implement Newton's algorithm to solve the inverse kinematics. This method has the advantage of working for any type of manipulator, although like our previous iterative method, its performance depends on the quality of our initial guess.

About Gen3 lite

As usual, all the information we need for this lab is in the User Guide, on pages 122-123. For convenience, a copy of the figure presenting the robot reference frames and dimensions is illustrated in Figure 1.

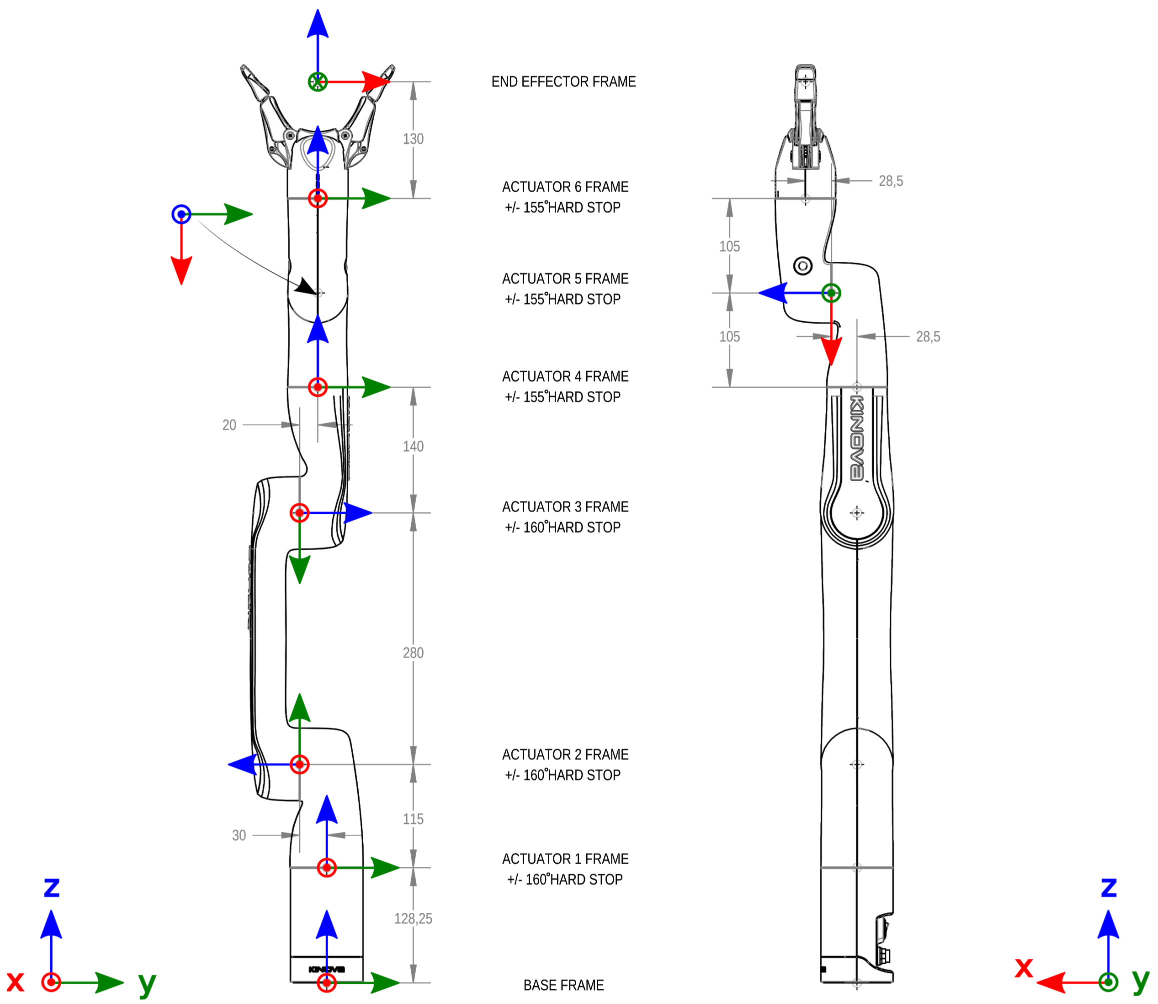

FIGURE 1: Actuator frames and dimensions

In addition to the dimensions of the robot, we will also need the joint position limits, which are indicated in Table 1.

Joint #	1	2	3	4	5	6
Position limit (deg)	± 154.1	± 150.1	± 150.1	± 148.98	-144.97, +145.0	± 148.98

TABLE 1: Gen3 lite Joint position limit

Theory

The idea behind Newton's algorithm is not to figure out the joint positions required to reach a pose directly, but rather to figure out a joint displacement that will get us closer to the desired pose and then repeat the process until we arrive at our destination.

The mathematical structure used to get a sense of cartesian direction when working with angular positions is called the geometric jacobian matrix, which relates joint velocities to the end-effector velocity as follow:

$$\begin{bmatrix} \omega^s_{e/s} \\ v^s_{O\in e/s} \end{bmatrix} = J^s(\mathrm{q})\dot{q}$$

Where $\omega^s_{e/s}$ and $\boldsymbol{v}^s_{O\in e/s}$ are the angular and linear velocities of the end-effector relative to the base respectively. J^s is the jacobian matrix and $\boldsymbol{q}$ is a vector containing the joint positions.

Using this, Newton's algorithm is described by the following steps.

0 - Initialize $\mathbf{q_k}$ as close to the answer as possible.

1 - Compute $q_{k+1} = q_k + \lambda_k J^s(q_k)^* V_k$

Where

- λ_k is an adjustable step size
- $J^s(q_k)^*$ is the pseudo-inverse of the geometric jacobian expressed in the base frame
- $V_k = \theta\xi$ such that $exp(\theta[\xi]) = T^s_{e*}(T^s_e(q_k))^{-1}$, with $T^s_e(q_k)$ being the desired pose at step k.

2 - If $||V_k|| > \epsilon$, where ϵ is a tolerance, return to 1, otherwise output $\mathbf{q_k}$. (In other words, if we are still too far from our target, keep working).

Manipulations

Part 1: Implementation of the inverse kinematics

In this section, we will implement the above algorithm in MATLAB.

1.1 Write a function called **jacob_gen3_lite_s**(q) which takes as an input a joint configuration and outputs the geometric jacobian of Gen3 lite for this configuration.

1.2 Write a function called **ik_gen3_lite_newton**(T, q0), which takes as inputs a desired pose T and an initial guess for the joint configuration which runs Newton's algorithm as described in the previous section. You can use a tolerance of 10^{-2} and a maximum number of iterations of 1000.

Part 2: Validation on the robot

In this section, we will write a script to send cartesian commands to the robot and compare the angular pose calculated by the robot to the one obtained from our own calculation. You should reuse the code that we developed during the *Introduction to Gen3 lite* lab. As a reminder, connection to the robot is established via the **CreateRobotApisWrapper**() function, angular commands are sent using **ReachCartesianPose**() and the angular readings are obtained via **RefreshFeedback**(). For more detail on the MATLAB API, consult the documentation.

2.1 Using either the Web App or your script, use joint commands to identify poses where the robot is in each of the possible configurations. Fill the second and third columns Table 2. You may also reuse the points you had in the previous lab.

Configurati on	Measured Joint positions	Cartesian pose	Calculated Joint positions
'rd+'			
'rd-'			
'ru+'			
'ru-'			
'ld+'			
'ld-'			
'lu+'			
'lu-'			

TABLE 2: Experimental kinematics results

2.2 Input the cartesian pose you noted in Table 2 in your **ik_gen3_lite_newton** function and fill the last column of Table 2. You should reuse the **cart2pose** function we implemented during the lab on *Forward Kinematics*. Since we have it, you can use the simplified model to obtain the initial guess for your iterative algorithm.

2.3 How does Newton's algorithm compare to our two previous methods?

2.4 Try reaching any of your points using [0, 0, 0, 0 ,0 ,0] as your initial guess. Describe what happens and why it happens.

Acknowledgements

This document was produced in collaboration with the following establishments

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Including direct implication from the following people:

- Prof. David Saussié, Eng., M.A.Sc., Ph.D., Department of Electrical Engineering.
- Alexis Tang, M.Eng., Department of Electrical Engineering
- Prof. Jérôme Le Ny, Eng., M.A.Sc., Ph.D., Department of Electrical Engineering.
- Prof. Richard Gourdeau , B.A.Sc., M.A.Sc., Ph.D., Department of Electrical Engineering.
