Technion – Israel Institute of Technology

HW3

Numerical Methods

019003

Alon Spinner	305184335	alonspinner@gmail.com
Oren Elmakis	311265516	orenelmakis@gmail.com

November 21, 2021

Question 1

We were asked to solve the Inverse Kinematics (IK) of a serial robot for given target points. To solve the inverse kinematics for points P_i, given the knowledge of the forward kinematics, *FK*, we used *fsolve*, to find a point x such that $FK(x) - P_i \approx 0$. each time with a different starting condition x_0.

$$x = fsolve(FK - P_i, x_0)$$

A solution is considered good when

$$\left|\left|FK(x) - P_i\right|\right|_2 < boundError = 1e - 4$$

As this is a robotics problem, and the solutions are joint angles, the same solutions can be represented with 2π multipliers. To find unique solutions we converted all $x_k, k \in [1, m]$ to the complex plane and then back.

$$x \in \mathbb{R}^{3\times m}$$

$$x_{unique} = angle(unique(e^{ix}, 'rows'))$$

This is good enough, but not perfect due to rounding errors which made it hard for the unique function to distinguish. As such, we introduced a 4-digit round.

$$x_{unique} = angle(unique(round(e^{ix}, 4), 'rows'))$$

1.1

We began by spreading initial conditions as an evenly spaced grid, of 20 points on each axis:

$$q_1, q_2, q_3 = linspace(-\pi, \pi, 20)$$

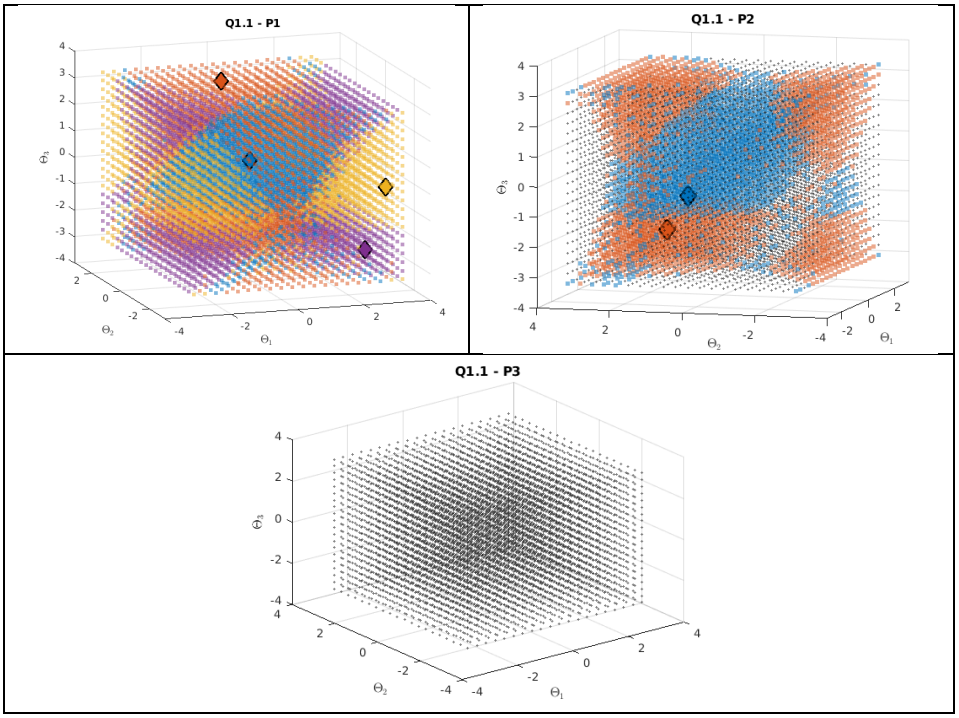
$$x_0(i, j, k) = [q_1(i) \quad q_2(j) \quad q_3(k)]^T$$

The unique solutions that were found are in the table below:

$P_1 = [500 \quad 0 \quad 500]^T$	$P_2 = [700 \quad 500 \quad 400]^T$	$P_3 = [700 \quad 500 \quad 600]^T$
$x_1 = [0 \quad 0.1807 \quad 0.4991]^T$	$x_1 = [0.6202 \quad 1.1973 \quad -0.9110]^T$	
$x_2 = [0 \quad 2.1783 \quad 2.8394]^T$	$x_2 = [0.6202 \quad 1.7636 \quad -2.0036]^T$	
$x_3 = [3.1416 \quad -2.0066 \quad -0.0527]^T$		
$x_4 = [3.1416 \quad -0.5721 \quad -2.8920]^T$		

Some points have all promised 4 solutions, but some are completely out of the manipulator's working space, and hence have none.

We also present point clouds showing the solutions, and the initial guess that led to them. A grid point that provided no solution appears as black.



These graphs can provide a-lot of information for trajectory planning in the configuration space, where one may want (for example) to keep the tool at a specific location but move the joints continuously.

It is worth noting that wrapping the angles between $[-\pi, \pi]$ was our choice, and perhaps a better visualization can be found in the interval $[0,2\pi]$

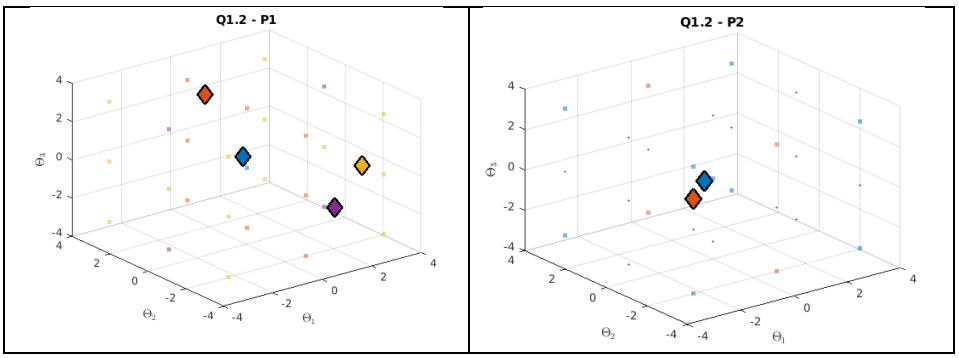
1.2

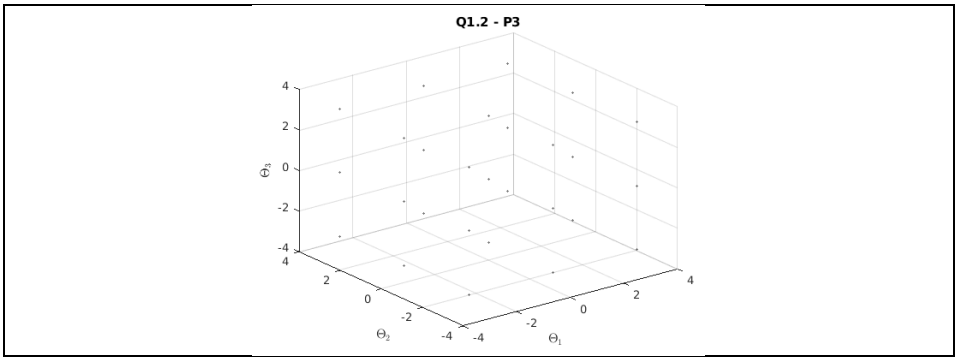
We found (experimentally) the minimum number of initial guesses to find all solutions with an equally spaced grid:

$$q_1, q_2, q_3 = linspace(-\pi, \pi, 3)$$

$$\rightarrow x_0 = x \in \mathbb{R}^{3\times 27}$$

We attach graphs as we did in 1.1.





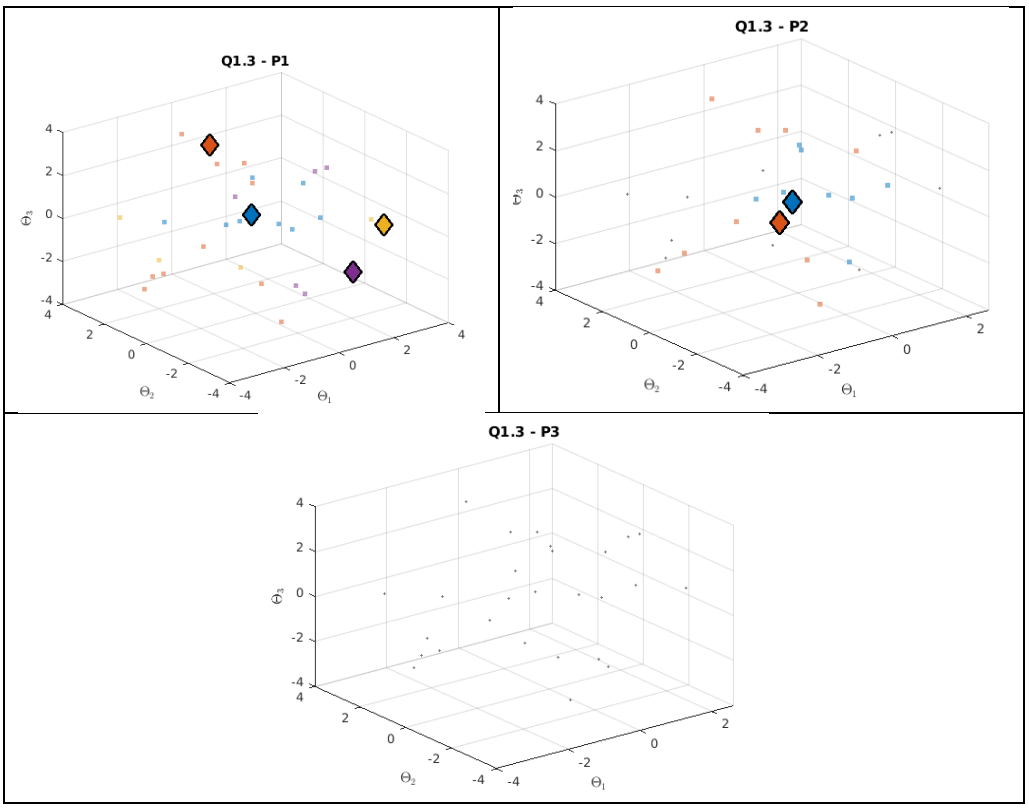
Unsurprisingly, we found the same solutions as before. In a real situation, when solving IK is required online, knowing that we have 4 maximal solutions, and a large configuration working space, allows us to start with a low-density grid and work our way up

1.3

We created a random matrix

$$x_0 = wrapToPi\left(2\pi \cdot rand(3,3^3)\right)$$

On the first try, we found all the solutions:



We ran 10 additional tries and always found all solutions. Having a big configuration working space that is evenly distributed between the solutions is the cause. On other problems this is not the case. The justification for starting with random guesses instead of a grid is usually based on unpredictable convergence volumes: non-continuous, non-convex with small and thin elements.

1. Question 2

We were asked to repeat the experiment in question 1, this time providing the Jacobian of the Forward Kinematics.

2.1

We rewrote the provided *ForwardKinematics* function in a symbolic manner

$$P(\theta_1, \theta_2, \theta_3) = [X, Y, Z]$$

We than used the *diff* function to create the Jacobian

$$J(\theta_1, \theta_2, \theta_3) = \frac{\partial P_i}{\partial \theta_j}$$

Below are Matlab's results:

```
>> pretty(funcbund.symJacobian)
/ -sin(th1) #3,        cos(th1) #1,       cos(th1) #2 \
|                                                     |
|  cos(th1) #3,        sin(th1) #1,       sin(th1) #2 |
|                                                     |
\       0,      - #5 - #4 - 400 sin(th2),  - #5 - #4  /

where
   #1 == 40 cos(th2 + th3) - 405 sin(th2 + th3) + 400 cos(th2)
   #2 == 40 cos(th2 + th3) - 405 sin(th2 + th3)
   #3 == #5 + #4 + 400 sin(th2) + 88
   #4 == 40 sin(th2 + th3)
   #5 == 405 cos(th2 + th3
```

2.2

We ran our *funcbund.findSolutions* for $P_1 = [500 \quad 0 \quad 500]^T$, and the grid setting of $Q1.1$, with and without the use of Jacobian.

With Jacobian	Without Jacobian
$3.422[seconds]$	$6.35[seconds]$

Generally, providing an analytical Jacobian would make things faster, and more accurate as we don't require the gradient based solver to compute one numerically.
Yet, researchers in the field of Machine Learning, where "differentiable programming is all the rage", can point to issues with gradient based optimization of chaotic or volatile systems. As such, sometimes it's better to switch to a different approach rather than computing the Jacobian analytically which is usually no easy task.

https://www.youtube.com/watch?v=EeMhj0sPrhE&ab_channel=YannicKilcher