ROBOTICS - ENGDATA302

INVERSE KINEMATICS FOR HUMAN FINGERS

Mehrad Haghshenas (2822865)
Instructor: Prof. Dr. Frank van der Stappen
University College Roosevelt

List of Figures

This page is intentionally left blank.

Introduction

The primary objective of this research project is to study the kinematic analysis of human fingers. In short, we aim to pose the finger accurately such that the tip (end-efector) reaches a specific target destination or moves along a pre-determined trajectory.

Anatomy of a Human Finger

The human hand has three main parts: 1 - palm. 2 - wrist. 3 - five distinct fingers. Each finger consists of multiple joints and bones. The joints of the fingers are primarily revolute joints, allowing for bending and flexion motions.[1] Specifically, the main type of joints in fingers is the *hinge joint*; a type of joint that allows movement in one plane (uniaxial; i.e., they have one degree of freedom) [1]. Each 1-DOF joint facilitates the rotation of the phalanges with respect to each other or to a metacarpal. The index, middle, ring, and little fingers all consist of three individual phalanges, whereas the thumb only possesses two.[2] Furthermore, the three *links* in each of the four fingers (except for the thumb), are called the proximal phalanx (PP), intermediate phalanx (IP), and distal phalanx (DP). Attached to a metacarpal located in the palm is the proximal phalanx (PP), which is also the closest finger bone to the palm. Moreover, the intermediate phalanx (IP) is the middle finger bone; however, this composition does not exist in thumbs. Finally, the distal phalanx is the most distant finger bone from the palm.

Each *hinge joint* has only 1-DOF (degrees of freedom) and each finger excluding the thumb has three of these joints; thus, they will have three degrees of freedom. Note that the thumb having only 2 hinge joints has 2 degrees of freedom, one less than the other fingers. All axes of rotation of the joints within a single finger lay parallel. The three *joints* are named metacarpophalangeal (MCP), proximal interphalangeal (PIP), and distal interphalangeal (DIP). The MCP connects the proximal phalanx and metacarpal, the PIP is the joint between the intermediate phalanx and proximal phalanx; and lastly, the DIP binds the distal phalanx and the intermediate phalanx. However, in contrast to the other four fingers, the thumb only posses one interphalangeal joint.

With regards to measurement information, the following table lists the average lengths of each phalanx in the fingers. The measurements throughout this document are expressed in terms of millimeters.

[1] However, note that finger joints can do a restricted degree of side-to-side (mediolateral) movement, although compared to the rotational movement, this movement is very limited.

[2] *Triphalangeal thumb* is a congenital anomaly where the thumb has three phalanges. In this case, the thumb can be as long as the other fingers.[2]

Finger	Proximal	Intermediate	Distal
Thumb	31.6	-	21.7
Index finger	39.8	22.4	15.8
Middle finger	44.6	26.3	17.4
Ring finger	41.4	25.7	17.3
Little finger	32.7	18.1	16.0

Figure 1: The numbers are taken from the assignment documentation which were originally taken from [3].

The descriptions of a human hand can be better interpreted in the following image:

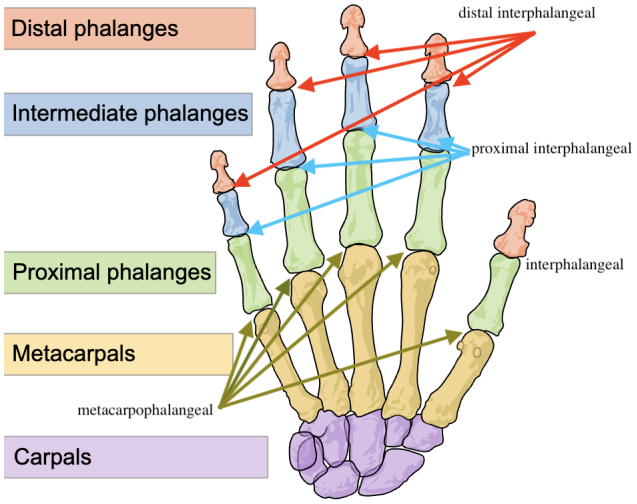

Figure 2: This figure was taken from [4] and the joint locations were added to the figure.

Model

For further analysis of the fingers, we exclude the thumb from our case of study. In each of the other four fingers, the rotation axis of each of the three joints are parallel to each other. It is assumed that the rotation axis is parallel to the $z-axis$. Thus, the motions of the revolute joints occur in the $x-y$ plane. Furthermore, the following assumptions are made:

θ_M: represents the metacarpophalangeal joint angle, defined as the counterclockwise angle between the proximal phalanx and the x-axis. θ_P: represents the proximal interphalangeal joint angle and is the counterclockwise angle between the intermediate phalanx and the extension of the proximal phalanx. θ_D: represents the distal interphalangeal joint angle and is

the counterclockwise angle between the distal phalanx and the extension of the intermediate phalanx.

Note that when all three joint angles are zero the finger is straight in the x-axis. The following shows a model of a single finger where points *M, P, D* show the metacarpophalangeal joint, proximal interphalangeal joint, and distal interphalangeal joint respectively. Furthermore, point *E* shows the location of the tip (end-effector). The three links, proximal phalanx, intermediate phalanx, and distal phalanx are denoted as edges *MP, PD, DE* in turn. From here on after we will, use these representations and the full name of the joints and links interchangeably. The right-handed frames of the joints are also indicated in the figure.

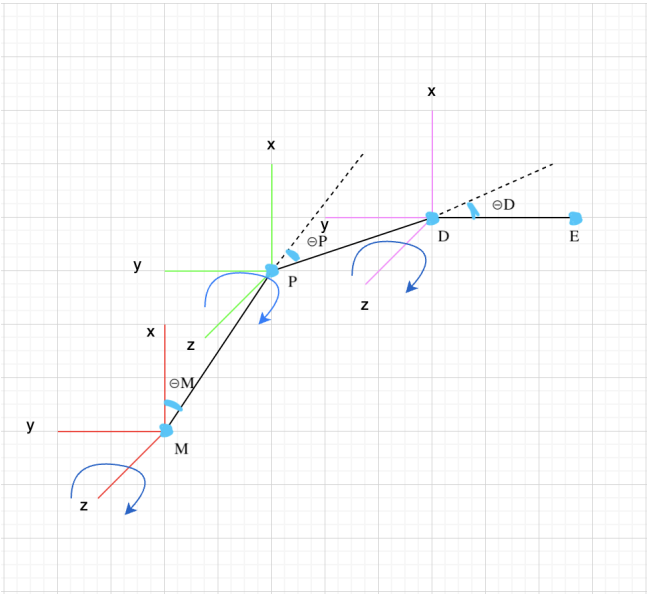

Figure 3: The model of a single finger.

As shown in the figure, the rotations of the joints are around the $z-axis$ and the angles are indicated. Note that in this case, all three angles are negative given that the rotation is done clockwise. To specifically, declare whether the rotation is clockwise or counter clockwise, one can look at the origin of the frame - attached to the joint - from the axis of reference. For example, in the case of the proximal interphalangeal joint, we look at point P from the extension of the proximal phalanx. Thus, it is obvious that PD is rotated in the clockwise direction and θ_P is negative. Similarly for θ_M, we look at point M from the $x-axis$ of the frame attached to the metacarpophalangeal joint; here again MP is rotated in the clockwise direction with respect to x and thus, θ_M is negative. Lastly, to define whether θ_D is negative or positive, we look at point D from the extension of the intermediate phalanx.

It is important to note that the angles of rotation are not from $0-2\pi$; in fact it is quite obvious that our finger cannot bend fully in both directions. Specifically, we explore the following two situations: 1 - Unconstrained Joint Angles. 2 - Constrained Joint Angles, where the situations only differ in the fact that in the latter the proximal interphalangeal joint and distal interphalangeal joint angles are related; however, in the former they are independent angles.

Unconstrained Joint Angles

This is the situation where no dependencies exist between the joint angles. The assumption is $-\frac{pi}{3} \leq \theta_M \leq \frac{pi}{3}$, $-\frac{2pi}{3} \leq \theta_P \leq 0$, and $-\frac{2pi}{3} \leq \theta_D \leq 0$. Also, any potential collisions between the phalanges is neglected. As the first step, using the average lengths in figure 1, we will derive the position of the finger tip in terms of the joint angles. Moreover, there is an object $O = \{(x,y,z) \in R^3 | y + 18 \leq 0\}$ in the space of the finger. The object O actually represents a wall in the space. The second task is to explore the set of finger configurations that place the fingertip on O. Lastly, we explore the set of reachable positions of the fingertip ignoring the obstacle O.

Forward kinematics

To derive the forward kinematics equation, we can use the Denavit-Hartenberg algorithm. However, since all the rotations occur in the $x-y$ plane, we can derive the direct kinematics equation using euclidean geometry as well. We will use the latter approach first as it is much simpler in this case and use the DH algorithm for confirmation. The following figure is used in this process:

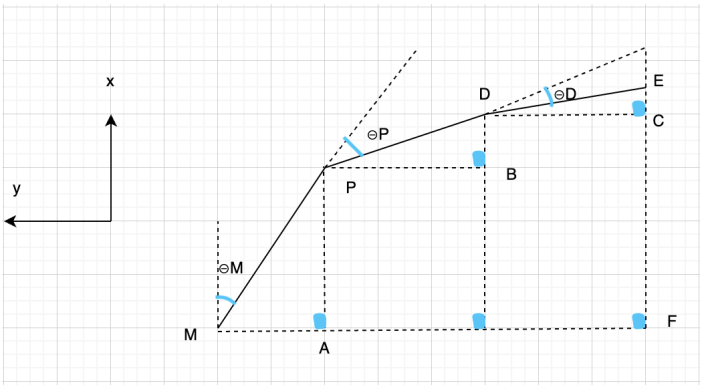

Figure 4: The model of a single finger.

In the shape, we first define the length MA in terms of θ_M. Note that MA represents the distance between the two joints M and P along the $y-axis$. This distance is $MA = cos(\frac{\pi}{2} - \theta_M) * MP$. We know that the cosine of an angle is equal to the sine of the complement of the angle. Thus, given this fact we have:

$$MA = sin(\theta_M) * MP$$

Now we calculate the distance PB, representing the distance between the proximal interphalangeal joint and distal interphalangeal joint in the *y axis*. As the first step we calculate

$$\angle DPB$$

; this angle represents the angle of the intermediate phalanx with respect to the $y-axis$.

$$\angle DPB = \pi - (\angle APM + \angle APB + \theta_P) = \pi - (\theta_M + \frac{\pi}{2} + \theta_P) = \frac{\pi}{2} - (\theta_M + \theta_P)$$

Furthermore we know that $PB = cos(\angle DPB) * PD$, thus:

$$PB = sin(\theta_M + \theta_P) * PD$$

Lastly, we want the distance DC, which is the distance between the distal joint and the end-effector in the y-axis. As the first step, we calculate the angle $\angle PDB = \frac{\pi}{2} - \angle DPB = \frac{\pi}{2} - (\frac{\pi}{2} - (\theta_M + \theta_P) = \theta_M + \theta_P$. Then, we calculate the angle $\angle CDE$:

$$\angle CDE = \pi - ((\theta_M + \theta_P) + \frac{\pi}{2} + \theta_D) = \frac{\pi}{2} - (\theta_M + \theta_P + \theta_D)$$

Now we can calculate DC using the triangle CDE:

$$DC = cos(\frac{\pi}{2} - (\theta_M + \theta_P + \theta_D)) * DE = sin(\theta_M + \theta_P + \theta_D) * DE$$

Having the distances MA, PB, and DC, we can calculate $FM = MA + PB + DC$. FM is actually the distance between the finger tip and the metacarpophalangeal joint in the y-axis. Given that the metacarpophalangeal joint is anchored at the origin of our coordinate frame, i.e. $y_M = 0$, thus FM actually represents the y parameter of the end-effector which we will denote as y_E.

$$y_E = FM = MA + PB + DC = sin(\theta_M) * MP + sin(\theta_M + \theta_P) * PD + sin(\theta_M + \theta_P + \theta_D) * DE$$

Thus, the y parameter of the end effector is calculated in terms of the joint angles and link lengths. The same process will be done to calculate the x parameter of the end-effector. The x parameter of the end effector is the distance between points E and the origin of coordinate frame along the x-axis. Given that the origin is based on point M, thus the parameter x of the end-effector (further denoted as x_E) will be equal to $AP + BD + CE$. Now it suffices to get each of these distances in terms of link lengths and joint parameters.

For AP in triangle APM:

$$AP = cos(\theta_M) * PM$$

For BD in triangle PBD:

$$BD = cos(\theta_M + \theta_P) * PD$$

For CE in triangle CDE:

$$CE = cos(\theta_M + \theta_P + \theta + D) * DE$$

Thus, the x parameter of the end effector will be:

$$x_E = cos(\theta_M) * MP + cos(\theta_M + \theta_P) * PD + cos(\theta_M + \theta_P + \theta_D) * DE$$

Finally, given that the metacarpophalangeal joint is anchored at the origin of our coordinate frame; thus $z_M = 0$. Also all joints lie in the same x-y plane so $z_E = 0$. Therefore, all three parameters of the end effector have been obtained. This is not of importance to us, and only the x_E and y_E will be examined.

To recap the following equation has been obtained in this part:

$$\begin{bmatrix} x_E \\ y_E \end{bmatrix} = \begin{bmatrix} cos(\theta_M) * MP + cos(\theta_M + \theta_P) * PD + cos(\theta_M + \theta_P + \theta_D) * DE \\ sin(\theta_M) * MP + sin(\theta_M + \theta_P) * PD + sin(\theta_M + \theta_P + \theta_D) * DE \end{bmatrix} \tag{1}$$

Where x_E and y_E denote the x and y parameter of the end-effector respectively. Plugging in the numbers in figure 1 for variables MP, DE, and PD will give us the forward kinematics of each finger (excluding the thumb).

As the final note of this subsection, it is also possible to derive the forward kinematics of the finger using Denavit Hartenberg algorithm. Accordingly, we can multiply a series of transformation matrices to computer the composite transformation matrix T_0^n as following:

$$T_0^3 = T_2^3 * T_1^2 * T_0^1$$

Here T_i^j represents the the matrix to configure coordinates with respect to frame i into

coordinates with respect to frame *j*. Moreover, given that each finger has three revolute joints, each finger has three moving frames and one fixed frame and the pre-mentioned equation is derived.

If we translate the moving frame attached to joint M by θ_M degrees and translate it along the MP link until we reach the joint P, we get the transformation matrix T_2^3.

$$T_2^3 = \begin{pmatrix} cos(\theta_M) & -sin(\theta_M) & x_{PM} \\ sin(\theta_M) & cos(\theta_M) & y_{PM} \\ 0 & 0 & 1 \end{pmatrix}$$

x_{PM} and y_{PM} mean the translation along the x-axis and y-axis when translating along the link PM, respectively. We have: $x_{PM}^2 + y_{PM}^2 = PM^2$; where PM is the length of the proximal phalanx. Also $x_{PM} = PMcos(\theta_M)$ and $y_{PM} = PMsin(\theta_M)$.

$$T_2^3 = \begin{pmatrix} cos(\theta_M) & -sin(\theta_M) & PMcos(\theta_M) \\ sin(\theta_M) & cos(\theta_M) & PMsin(\theta_M) \\ 0 & 0 & 1 \end{pmatrix}$$

Similarly for T_1^2 it can be shown:

$$T_1^2 = \begin{pmatrix} cos(\theta_P) & -sin(\theta_P) & PDcos(\theta_P) \\ sin(\theta_P) & cos(\theta_P) & PDsin(\theta_P) \\ 0 & 0 & 1 \end{pmatrix}$$

where PD is the length of the intermediate phalanx. Lastly for T_1^0 it can be seen:

$$T_0^1 = \begin{pmatrix} cos(\theta_D) & -sin(\theta_D) & DEcos(\theta_D) \\ sin(\theta_D) & cos(\theta_D) & DEsin(\theta_D) \\ 0 & 0 & 1 \end{pmatrix}$$

So the transformation matrix can be re-written as:

$$T_0^3 = \begin{pmatrix} cos(\theta_M) & -sin(\theta_M) & PMcos(\theta_M) \\ sin(\theta_M) & cos(\theta_M) & PMsin(\theta_M) \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} cos(\theta_P) & -sin(\theta_P) & PDcos(\theta_P) \\ sin(\theta_P) & cos(\theta_P) & PDsin(\theta_P) \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} cos(\theta_D) & -sin(\theta_D) & DEcos(\theta_D) \\ sin(\theta_D) & cos(\theta_D) & DEsin(\theta_D) \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix}$$

Moreover, the following equations hold:

$$a_{11} = \cos(\theta_M)\cos(\theta_P)\cos(\theta_D) - \sin(\theta_M)\sin(\theta_P)\cos(\theta_D)$$

$$a_{12} = -\cos(\theta_M)\sin(\theta_P)\cos(\theta_D) - \sin(\theta_M)\cos(\theta_P)\cos(\theta_D)$$

$$a_{13} = cos(\theta_M) * MP + cos(\theta_M + \theta_P) * PD + cos(\theta_M + \theta_P + \theta_D) * DE$$

$$a_{21} = \sin(\theta_M)\cos(\theta_P)\cos(\theta_D) + \cos(\theta_M)\sin(\theta_P)\cos(\theta_D)$$

$$a_{22} = -\sin(\theta_M)\sin(\theta_P)\cos(\theta_D) + \cos(\theta_M)\cos(\theta_P)\cos(\theta_D)$$

$$a_{23} = sin(\theta_M) * MP + sin(\theta_M + \theta_P) * PD + sin(\theta_M + \theta_P + \theta_D) * DE$$

Having the transformation matrix, we can write:

$$\begin{pmatrix} x_E \\ y_E \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_M \\ y_M \\ 1 \end{pmatrix}$$

where x_M and y_M are both equal to zero given that the metacarpophalangeal joint lies on the origin. Therefore, we will have $x_E = a_{13}$ and $y_E = a_{23}$ which is exactly the same as equation (1). Note that in the Denavit Hartenberg algorithm, besides calculating the location of the

end-effector with respect to the base of the manipulator (which we had already calculated in another way using geometric calculations); we now know the equations for the orientation of the finger tip using the values of $a_{11}, a_{12}.a_{21}, a_{22}$. The calculated transformation matrix can be used for inverse kinematics problems. Note that in this case, the architecture of the manipulator is relatively simple and thus, calculating the transformation matrix explicitly and solving the inverse kinematics problem directly will be relatively simple. However, in most cases the direct approach requires a lot of computing power and hence, the iterative approach is used as an alternative.

Fingertip on Object

In this subsection, we will investigate the fingertip being on the object. We will assume point P on the object with x-coordinate equal to α and y-coordinate equal to -18. Therefore, we will have:

$$\begin{bmatrix} x_E \\ y_E \end{bmatrix} = \begin{bmatrix} cos(\theta_M) * MP + cos(\theta_M + \theta_P) * PD + cos(\theta_M + \theta_P + \theta_D) * DE \\ sin(\theta_M) * MP + sin(\theta_M + \theta_P) * PD + sin(\theta_M + \theta_P + \theta_D) * DE \end{bmatrix} = \begin{bmatrix} \alpha \\ -18 \end{bmatrix} \tag{2}$$

This equation is generalized for all fingers of the hand (excluding the thumb), we will focus our attention on the index finger. The relevant lengths for the index finger can be found in table (1). Plugging the numbers in equation 2, we will have:

$$\begin{bmatrix} x_E \\ y_E \end{bmatrix} = \begin{bmatrix} cos(\theta_M) * 39.8 + cos(\theta_M + \theta_P) * 22.4 + cos(\theta_M + \theta_P + \theta_D) * 15.8 \\ sin(\theta_M) * 39.8 + sin(\theta_M + \theta_P) * 22.4 + sin(\theta_M + \theta_P + \theta_D) * 15.8 \end{bmatrix} = \begin{bmatrix} \alpha \\ -18 \end{bmatrix}$$

We will further explore the following situation: The fingertip of the index finger is placed on point P such that the distal phalanx has a vertical orientation and the proximal phalanx has a horizontal orientation.

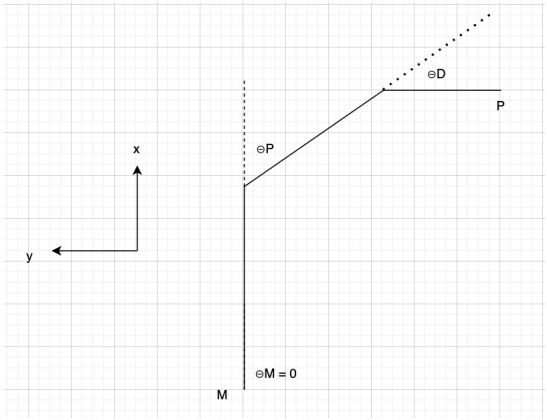

Figure 5: The fingertip of the index finger touching point p such that the distal phalanx is vertical and the proximal phalanx is horizontal. Note the orientation of the x-y plane and the location of point P(α,-18), where in this case $\alpha > 0$.

In this case, we have $\theta_P + \theta_D = -\frac{\pi}{2}$, and $\theta_M = 0$. Note that both θ_P and θ_D are negative. Therefore the following holds:

$$\begin{bmatrix} x_E \\ y_E \end{bmatrix} = \begin{bmatrix} 39.8 + cos(\theta_P) * 22.4 \\ sin(\theta_P) * 22.4 - 15.8 \end{bmatrix} = \begin{bmatrix} \alpha \\ -18 \end{bmatrix}$$

$$cos(\theta_P) = \frac{\alpha - 39.8}{22.4}$$
$$sin(\theta_P) = \frac{-2.2}{22.4} = -0.098$$

Having the $sin(\theta_P) = -0.098$, we will have: $cos(\theta_P) = 0.9952$. Therefore, $\alpha = (0.9952 * 22.4) + 39.8 = 62.094$. Thus, if the index finger is touching the object O where the proximal phalanx is horizontal and the distal phalanx is vertical, it will touch the object in the point $P(62.094, -18)$.

Now what if only the distal phalanx was vertical; meaning that we drop the requirement that the proximal phalanx is also horizontal. Then the following will hold: $\theta_M + \theta_D + \theta_P = -\frac{\pi}{2}$. Note that in the following shape all three joint angles are negative.

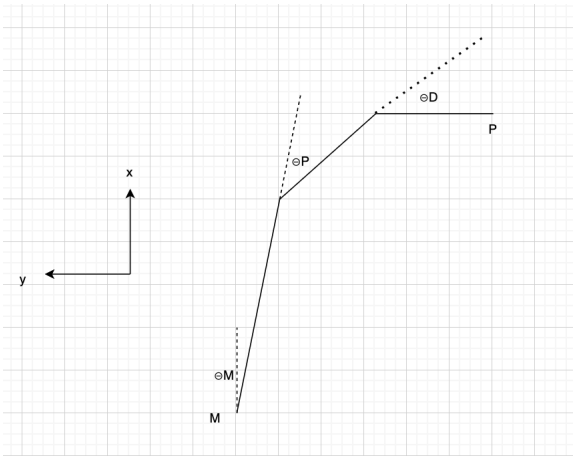

Figure 6: The fingertip of the index finger touching point p such that the distal phalanx is vertical. Note the orientation of the x-y frame and the location of point P(α,-18), where in this case $\alpha > 0$.

$$\begin{bmatrix} x_E \\ y_E \end{bmatrix} = \begin{bmatrix} cos(\theta_M) * 39.8 - sin(\theta_D) * 22.4 \\ sin(\theta_M) * 39.8 - cos(\theta_D) * 22.4 \end{bmatrix} = \begin{bmatrix} \alpha \\ -2.2 \end{bmatrix}$$

Solving the above relation which has two equations with two variables, requires that we square each equation:

$$cos(\theta_M)^2 * 39.8^2 + sin(\theta_D)^2 * 22.4^2 - 2 * 39.8 * 22.4 * sin(\theta_D) * cos(\theta_M) = \alpha^2$$
$$sin(\theta_M)^2 * 39.8^2 + cos(\theta_D)^2 * 22.4^2 - 2 * 39.8 * 22.4 * sin(\theta_M) * cos(\theta_D) = (-2.2)^2$$

Furthermore, we sum up the two equations with each other:

$$\begin{aligned} 39.8^2 + 22.4^2 - 39.8 * 22.4 * 2(sin(\theta_D)cos(\theta_M) + sin(\theta_M)cos(\theta_D)) &= \alpha^2 + 2.2^2 \\ -39.8 * 22.4 * 2sin(\theta_M + \theta_D) &= \alpha^2 + 2.2^2 - (39.8^2 + 22.4^2) \\ 39.8 * 22.4 * 2cos(\theta_P) &= \alpha^2 + 2.2^2 - (39.8^2 + 22.4^2) \\ cos(\theta_P) &= \frac{\alpha^2 + 2.2^2 - (39.8^2 + 22.4^2)}{39.8 * 22.4 * 2} \end{aligned}$$

Given that $-\frac{2\pi}{3} \leq \theta_P \leq 0$ thus $-\frac{1}{2} \leq cos(\theta_P) \leq 1$; thus we will have the following:

$$-\frac{1}{2} \leq \frac{\alpha^2 + 2.2^2 - (39.8^2 + 22.4^2)}{39.8 * 22.4 * 2} \leq 1$$

$$(-39.8 * 22.4) + (39.8^2 + 22.4^2) - 2.2^2 \leq \alpha^2 \leq (39.8 * 22.4 * 2) + (39.8^2 + 22.4^2) - 2.2^2$$

$$1189.44 \leq \alpha^2 \leq 3864$$

$$34.48 \leq \alpha \leq 62.16 or -62.16 \leq \alpha \leq -34.48$$

Therefore we have found out that if the distal phalanx of the index finger is vertical and the fingertip is touching the object, then the range of x will be:

$$34.48 \leq x \leq 62.16$$

$$or$$

$$-62.16 \leq x \leq -34.48$$

Now what if *only* the proximal phalanx was horizontal; meaning that we drop the requirement that the distal phalanx is vertical. Then we will have $\theta_M = 0$ and the following will hold:

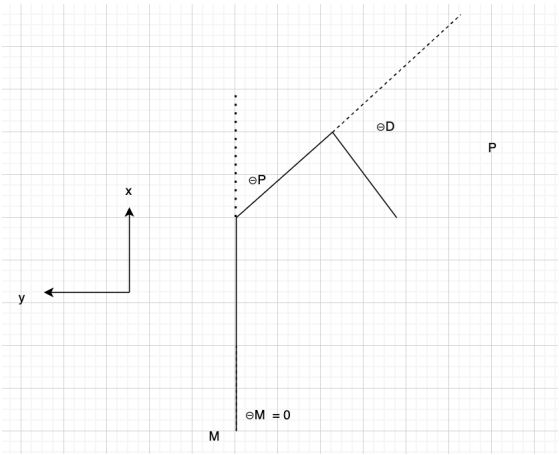

Figure 7: The fingertip of the index finger touching point p such that the proximal phalanx is horizontal. Note the orientation of the x-y frame and the location of point P(α,-18), where in this case $\alpha > 0$.

$$\begin{bmatrix} x_E \\ y_E \end{bmatrix} = \begin{bmatrix} cos(\theta_P) * 22.4 + cos(\theta_P + \theta_D) * 15.8 \\ sin(\theta_P) * 22.4 + sin(\theta_P + \theta_D) * 15.8 \end{bmatrix} = \begin{bmatrix} \alpha - 39.8 \\ -18 \end{bmatrix}$$

We will again, square both equations and sum them up to get the following equation:

$$2 * 22.4 * 15.8(cos(\theta_P)cos(\theta_P + \theta_D) + sin(\theta_P)sin(\theta_P + \theta_D)) = (\alpha - 39.8)^2 + (-18)^2 - 22.4^2 - 15.8^2$$

$$cos(\theta_D) = \frac{(\alpha - 39.8)^2 + (-18)^2 - 22.4^2 - 15.8^2}{2 * 22.4 * 15.8}$$

Similar to θ_P we have: $-\frac{1}{2} \leq cos(\theta_D) \leq 1$, hence:

$$-\frac{1}{2} \leq \frac{(\alpha - 39.8)^2 + (-18)^2 - 22.4^2 - 15.8^2}{2 * 22.4 * 15.8} \leq 1$$

$$-22.4 * 15.8 \leq (\alpha - 39.8)^2 + (-18)^2 - 22.4^2 - 15.8^2 \leq 2 * 22.4 * 15.8$$

$$-22.4 * 15.8 - ((-18)^2 - 22.4^2 - 15.8^2) \leq (\alpha - 39.8)^2 \leq 2 * 22.4 * 15.8 - ((-18)^2 - 22.4^2 - 15.8^2)$$

$$73.48 \leq (\alpha - 39.8)^2 \leq 1135.24$$

$$8.57 \leq |\alpha - 39.8| \leq 33.69$$

$$48.37 \leq \alpha \leq 73.49 or 6.11 \leq \alpha \leq 31.23$$

Therefore we have found out that if the proximal phalanx of the index finger is horizontal and the fingertip is touching the object, then the range of x will be:

$$48.37 \leq x \leq 73.49$$

$$or$$

$$or 6.11 \leq x \leq 31.23$$

Reachability Space

As the final investigation of this part, we will explore the set of reachable positions of the fingertip (and orientations of the distal phalanx) ignoring the obstacle O. If we look at equation 1, after squaring the two relations and summing them up, we will get the following equation:

$$x_E^2 + y_E^2 = MP^2 + PD^2 + DE^2 + 2(cos(\theta_M)cos(\theta_M + \theta_P) * MP * PD +$$
$$cos(\theta_M)cos(\theta_M + \theta_P + \theta_D) * MP * DE + cos(\theta_M + \theta_P)cos(\theta_M + \theta_P + \theta_D) * PD * DE +$$
$$sin(\theta_M)sin(\theta_M + \theta_P) * MP * PD + sin(\theta_M)sin(\theta_M + \theta_P + \theta_D) * MP * DE +$$
$$sin(\theta_M + \theta_P)sin(\theta_M + \theta_P + \theta_D)) * PD * DE$$

$$x_E^2 + y_E^2 = MP^2 + PD^2 + DE^2 + 2(cos(\theta_P) * MP * PD + cos(\theta_D) * PD * DE + cos(\theta_P + \theta_D) * MP * DE)$$

For the index finger this equation will be:

$$x_E^2+y_E^2 = 39.8^2+22.4^2+15.8^2+2(cos(\theta_P)*39.8*22.4+cos(\theta_D)*22.4*15.8+cos(\theta_P+\theta_D)*39.8*15.8)$$

The maximum of $cos(\theta_P)*39.8*22.4+cos(\theta_D)*22.4*15.8+cos(\theta_P+\theta_D)*39.8*15.8$ is 1874.28 and it happens when both joint angles are 0. Given the possible range for the joint angles the minimum of $cos(\theta_P)*39.8*22.4+cos(\theta_D)*22.4*15.8+cos(\theta_P+\theta_D)*39.8*15.8$ occurs when both joint angles are equal to $-\frac{2\pi}{3}$; hence: $min = \frac{-1}{2}(39.8*22.4+22.4*15.8+39.8*15.8) = -937.14$.

$$39.8^2+22.4^2+15.8^2+2*(-937.14) \leq x_E^2+y_E^2 \leq 39.8^2+22.4^2+15.8^2+2*1874.28$$
$$461.16 \leq x_E^2+y_E^2 \leq 6084$$

Therefore, the fingertip of the index finger will definitely be inside the circle with radius $\sqrt{6084} = 78$ however, outside the circle with radius $\sqrt{461.16} = 21.47$. In other words, it will be inside the ring contained between the two pre-mentioned circles. Note that not all points inside this ring are accessible, however, points outside of this ring are definitely not reachable. Lastly, the maximum possible x_E can be obtained when all joint angles are zero and for the index finger we have: $max(x_E) = 39.8+22.4+15.8 = 78$. The maximum possible y_E will be achieved when $\theta_M = \frac{-\pi}{3}$, $\theta_P = \frac{-\pi}{6}$, and $\theta_D = 0$. Note that here by maximum we mean the maximum distance in the y direction not the maximum positive value of y_E.

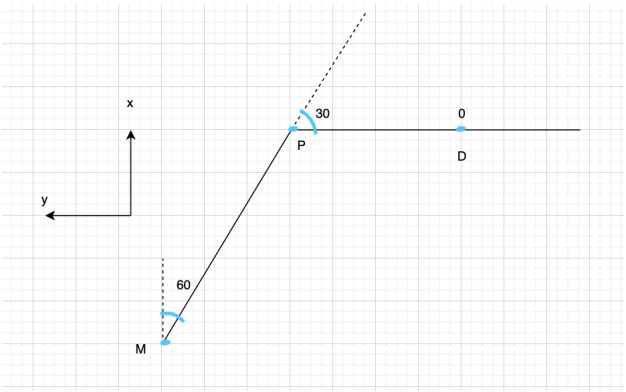

Figure 8: The configuration of the joint angles when the fingertip of the index finger reaches the maximum possible y.

$$max(y_E) = sin(-\frac{\pi}{3}) * 39.8 + sin(-\frac{\pi}{2}) * 22.4 + sin(-\frac{\pi}{2}) * 15.8$$
$$max(y_E) = -(12.13 + 22.4 + 15.8) = -50.33$$

Constrained Joint Angles

In this section we will explore situations where the rotations of the proximal and distal interphalangeal joints are not independent. Specifically, the proximal interphalangeal (PIP) and distal interphalangeal joint variables satisfy the following equation: $\theta_D = 2\frac{\theta_P}{3}$. Therefore, given the constraint $-\frac{2\pi}{3} \leq \theta_P \leq 0$, the constraint for θ_D will be $-\frac{4\pi}{9} \leq \theta_D \leq 0$ (instead of $-\frac{2\pi}{3} \leq \theta_D \leq 0$).

Forward Kinematics

Taking the mentioned fact into account, we adjust the forward kinematics equations accordingly. The forward kinematics of each finger given the added constraint on the joint angles is the following:

$$\begin{bmatrix} x_E \\ y_E \end{bmatrix} = \begin{bmatrix} cos(\theta_M) * MP + cos(\theta_M + \theta_P) * PD + cos(\theta_M + \frac{5}{3}\theta_P) * DE \\ sin(\theta_M) * MP + sin(\theta_M + \theta_P) * PD + sin(\theta_M + \frac{5}{3}\theta_P) * DE \end{bmatrix} \tag{3}$$

This equation is much similar to equation 1, but θ_D has been replaced by $2\frac{\theta_P}{3}$. In this equation, the joint variables are θ_P and θ_M and the configuration space variables are x_E and y_E.

Inverse Kinematics

In this part, given the configuration T, we are going to determine the joint variables that result into this configuration. This is known as the *inverse kinematic problem* and to solve this problem, there are three main approaches: 1) Decoupling 2) Inverse transformation method 3) Iterative solution. In this paper, we focus on the third approach, i.e. we develop an iterative inverse kinematics approach to determine the joint angles, given that the position of the end-effector is defined.

The main process in the *iterative solution* is as follows:
1) We make an initial guess for the joint variables that will lead to the desired configuration. Note that more often than not this initial guess is far from the target destination. Likewise,

the initial guess affects the number of iterations and reaching the local minima (instead of the global minima).
2) We create the Jacobian matrix where each row is the partial derivation of a configuration variable based on each of the joint variables. Furthermore, we calculate the inverse of the Jacobian matrix. Note that if the number of configuration variables and joint variable are not equal, then the Jacobian matrix will not be a squared matrix. Therefore, in this case the *Moore–Penrose pseudo-inverse* should be used.
3) We calculate the difference between the destination position and current position, this will indicate the direction of the movement for optimizing our initial joint variables.
4) We optimize our initial guess for the joint variables by using the following formula:

$$q^{(i+1)} = q^{(i)} + J^{-1}(q^{(i)})\sigma T(q^{(i)}) \tag{4}$$

where $q^{(i)}$ is our current joint variables (current guess), $J^{-1}(q^{(i)})$ is the inverse of the Jacobian matrix, and $\sigma T(q^{(i)})$ is the current position of the fingertip subtracted from the destination position. $q^{(i+1)}$ will be the the next joint variables which in the next iteration will be used as $q^{(i)}$. We repeat this process until the fingertip has reached the predetermined target position. It is worth mentioning that not always can the destination be reached, so in some cases we might end up with the closest position to the target as the result of the iterative solution.

We will apply the described process for the *index finger*. The initial guess for the joint variables will be $\theta_M = -\frac{\pi}{6}$ and $\theta_P = 0 (\theta_D = 0)$, and the destination position is $P(12,-18)$ placed on the object O. Moreover, the Jacobian matrix is:

$$J(\theta_M, \theta_P) = \begin{bmatrix} \frac{\partial x_E}{\partial \theta_M} & \frac{\partial x_E}{\partial \theta_P} \\ \frac{\partial y_E}{\partial \theta_M} & \frac{\partial y_E}{\partial \theta_P} \end{bmatrix}$$

where the partial derivatives are as following:

$$\frac{\partial x_E}{\partial \theta_M} = -MP * sin(\theta_M) - PD * sin(\theta_M + \theta_P) - DE * sin(\theta_M + \frac{5}{3}\theta_P)$$

$$\frac{\partial x_E}{\partial \theta_P} = -PD * sin(\theta_M + \theta_P) - \frac{5}{3}DE * sin(\theta_M + \frac{5}{3}\theta_P)$$

$$\frac{\partial y_E}{\partial \theta_M} = MP * cos(\theta_M) + PD * cos(\theta_M + \theta_P) + DE * cos(\theta_M + \frac{5}{3}\theta_P)$$

$$\frac{\partial y_E}{\partial \theta_P} = PD * cos(\theta_M + \theta_P) + \frac{5}{3}DE * cos(\theta_M + \frac{5}{3}\theta_P)$$

Furthermore, we calculate the inverse of the Jacobian matrix. But before doing so, given that we are discussing the process for the index finger we will substitute the relevant link lengths in the Jacobian matrix:

$$J(\theta_M,\theta_P) =$$

$$\begin{bmatrix} -39.8sin(\theta_M) - 22.4sin(\theta_M+\theta_P) - 15.8sin(\theta_M+\frac{5}{3}\theta_P) & -22.4sin(\theta_M+\theta_P) - \frac{5}{3}15.8sin(\theta_M+\frac{5}{3}\theta_P) \\ 39.8cos(\theta_M) + 22.4cos(\theta_M+\theta_P) + 15.8cos(\theta_M+\frac{5}{3}\theta_P) & 22.4cos(\theta_M+\theta_P) + \frac{5}{3}15.8cos(\theta_M+\frac{5}{3}\theta_P) \end{bmatrix}$$

The inverse of the Jacobian Matrix will be:

$$\begin{bmatrix} -39.8\sin(\theta_M) - 22.4\sin(\theta_M+\theta_P) - 15.8\sin\left(\theta_M+\frac{5}{3}\theta_P\right) & -22.4\sin(\theta_M+\theta_P) - \frac{5}{3}15.8\sin\left(\theta_M+\frac{5}{3}\theta_P\right) \\ 39.8\cos(\theta_M) + 22.4\cos(\theta_M+\theta_P) + 15.8\cos\left(\theta_M+\frac{5}{3}\theta_P\right) & 22.4\cos(\theta_M+\theta_P) + \frac{5}{3}15.8\cos\left(\theta_M+\frac{5}{3}\theta_P\right) \end{bmatrix}^{-1} =$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

where the coefficients are:

$$a_{11} = \frac{1}{\det(J(\theta_M,\theta_P))}(22.4\cos(\theta_M+\theta_P) + \frac{5}{3}15.8\cos\left(\theta_M+\frac{5}{3}\theta_P\right))$$
$$a_{12} = \frac{1}{\det(J(\theta_M,\theta_P))}(22.4\sin(\theta_M+\theta_P) + \frac{5}{3}15.8\sin\left(\theta_M+\frac{5}{3}\theta_P\right))$$
$$a_{21} = \frac{1}{\det(J(\theta_M,\theta_P))}(-39.8\cos(\theta_M) - 22.4\cos(\theta_M+\theta_P) - 15.8\cos\left(\theta_M+\frac{5}{3}\theta_P\right))$$
$$a_{22} = \frac{1}{\det(J(\theta_M,\theta_P))}(-39.8\sin(\theta_M) - 22.4\sin(\theta_M+\theta_P) - 15.8\sin\left(\theta_M+\frac{5}{3}\theta_P\right))$$

The determinant of the matrix $J(\theta_M,\theta_P)$ is equal to:

$$\begin{aligned}\det(J(\theta_M,\theta_P)) = (&-39.8\sin(\theta_M) - 22.4\sin(\theta_M+\theta_P) - 15.8\sin\left(\theta_M+\frac{5}{3}\theta_P\right)) \\ &\times(22.4\cos(\theta_M+\theta_P) + \frac{5}{3}15.8\cos\left(\theta_M+\frac{5}{3}\theta_P\right)) \\ &- \\ &(-22.4\sin(\theta_M+\theta_P) - \frac{5}{3}15.8\sin\left(\theta_M+\frac{5}{3}\theta_P\right)) \\ &\times(39.8\cos(\theta_M) + 22.4\cos(\theta_M+\theta_P) + 15.8\cos\left(\theta_M+\frac{5}{3}\theta_P\right))\end{aligned}$$

This is the generalized Jacobian matrix and in each iteration the joints parameters for that

iteration need to be plugged in. The $q^{(0)}$ matrix is as follows:

$$q^{(0)} = \begin{bmatrix} \theta_M \\ \theta_P \end{bmatrix} = \begin{bmatrix} -\frac{\pi}{6} \\ 0 \end{bmatrix}$$

Finally, to calculate $\sigma T(q^{(0)})$, we first need to calculate the location of the fingertip of the index finger given the initial guess of $\theta_M = -\frac{\pi}{6}$ and $\theta_P = 0$. For this, we plug in these values in the forward kinematics equation (equation (3)):

$$\begin{bmatrix} x_E \\ y_E \end{bmatrix} = \begin{bmatrix} cos(\frac{\pi}{6})(39.8 + 22.4 + 15.8) \\ -sin(\frac{\pi}{6})(39.8 + 22.4 + 15.8) \end{bmatrix} = \begin{bmatrix} 67.54 \\ -39 \end{bmatrix}$$

having the initial position and the target position $\sigma T(q^{(0)})$ is:

$$\sigma T(q^{(0)}) = \begin{bmatrix} 12 \\ -18 \end{bmatrix} - \begin{bmatrix} 67.54 \\ -39 \end{bmatrix} = \begin{bmatrix} -55.54 \\ 21 \end{bmatrix}$$

Now we have all the ingredients to calculate q^1 using equation (4). After doing so, we repeat this process until we reach the desired destination. This is a very complicated task to do manually and we will not go through the details of the remaining steps. The point of describing the first iteration was to become familiar with the details of the process. From here on after we will use an inverse kinematic program written by the author in *Python* to reach the final destination. The following images, represent the process where the index finger goes from the initial point (which was calculated using the initial joint configuration) to the target point $P(12, -18)$.

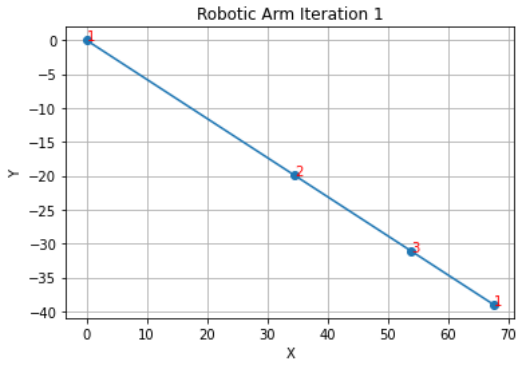

Figure 9: The initial configuration of the index finger. Note that the orientation of the x-y plane is 90 degrees rotated clockwise compared to the pictures drawn previously. In general this is the case for all graphs produced by the python code.

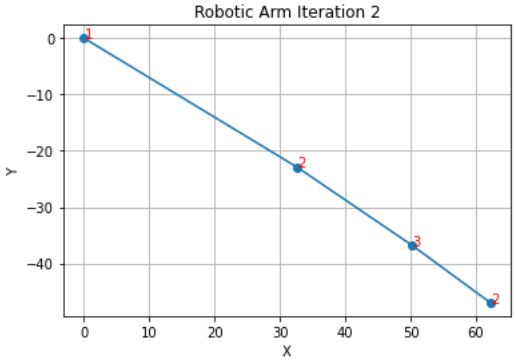

Figure 10: This is the second configuration where the index finger has moved towards the destination.

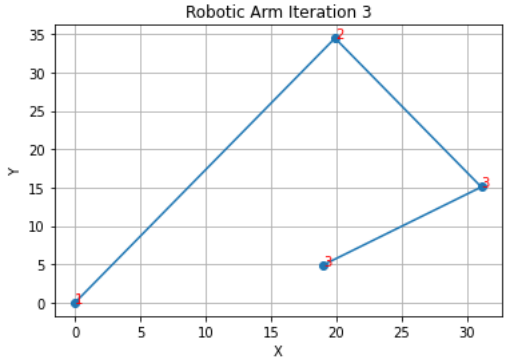

Figure 11: The third pose of the index finger.

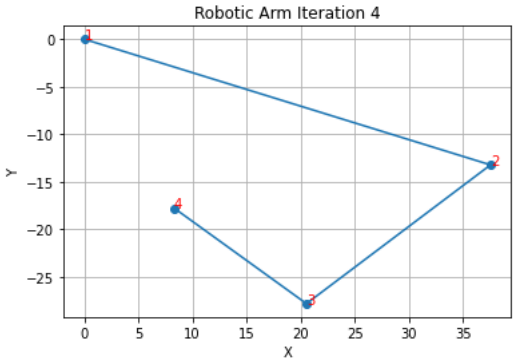

Figure 12: The forth configuration of the index finger.

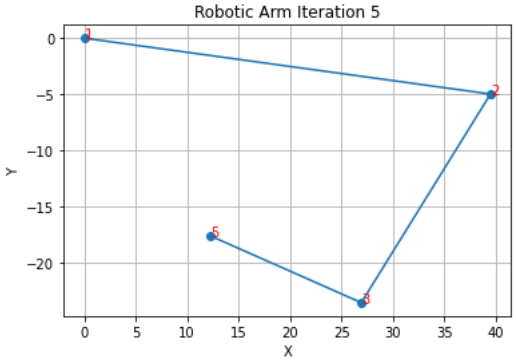

Figure 13: The fifth configuration of the index finger.

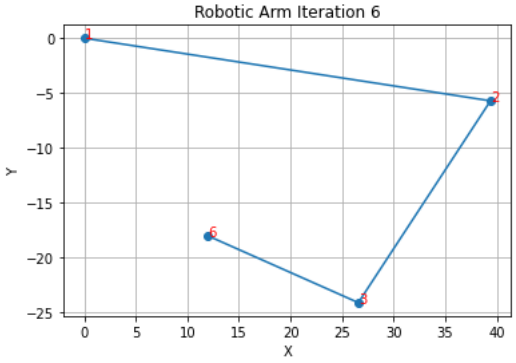

Figure 14: The sixth configuration of the index finger.

The following plot represents an intuitive visualisation of the fingertip of the index finger. It shows how the position of the end-effector has changed from the initial location to the target destination. It is worth mentioning that due to the text not overlapping each other in the graph, we have not labelled every single vertex, particularly the ones that were very near when the process was converging have not been labelled.

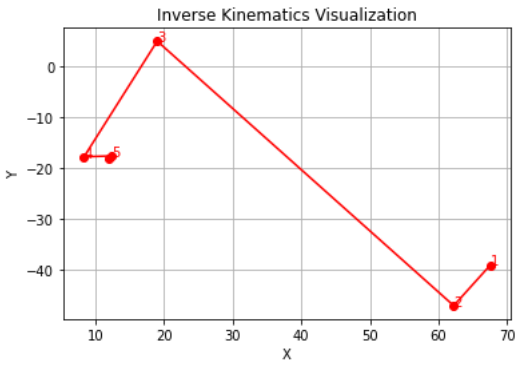

Figure 15: The position of the end effector at each index finger pose. The direction of the edges shows the direction of the changes of the fingertip. It is important to consider that in reality these changes are not linear and these lines only connect the start and end position of the fingertip at each iteration.

The $max_{iterations}$ variable in this examination was set to 10000. This variable defines the maximum number of iterations that the manipulator can undergo. One can increase this number depending on the computing power they have. This variable is mainly used to terminate the process when the process is diverging. Finally, the $tolerance$ variable is set to 0.01 in this case. This variable indicates that if the position of the fingertip of the index finger is within 0.01 distance of the target destination, the process will terminate and return the joint variables. In other words, the $tolerance$ variable defines the desired accuracy that the fingertip gets close to the target position for the process to terminate. We can decrease this variable to get higher precisions depending on the computing power we have. Some useful data about the process are the followings:
1) The solution was found after 5 iterations.
2) The final joint variables are: [-0.1439013 -2.03578803]
3) The final position of the end-effector of the index finger is: [11.9951411 -17.99765616] which is roughly equal to [12,-18] (the destination point).

As the next examination, we investigate whether it is possible for the index finger to reach point $P(7,-6)$ from the initial joint variables $\theta_M = -\frac{\pi}{6}, \theta_P = -\frac{\pi}{4}$. In this examination again $max_{iteration}$ and $tolerance$ are set to 10,000 and 0.01 respectively. As shown below the destination is impossible to achieve with the given constraints.

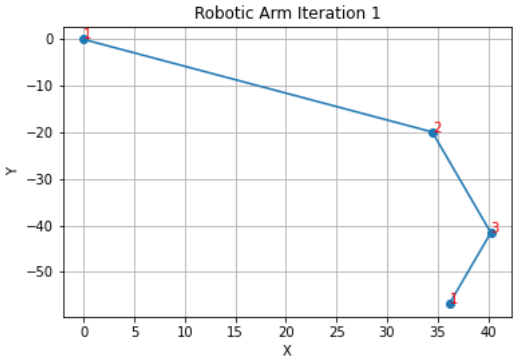

Figure 16: The initial orientation of the index finger.

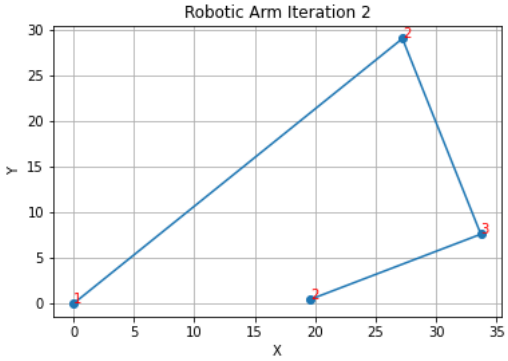

Figure 17: The second orientation of the index finger.

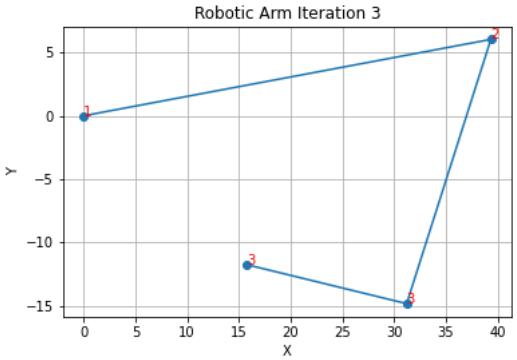

Figure 18: The third orientation of the index finger.

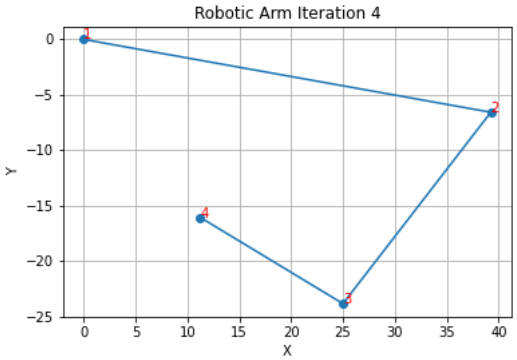

Figure 19: The forth orientation of the index finger.

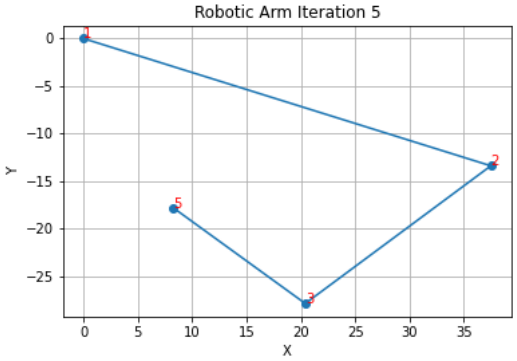

Figure 20: The fifth orientation of the index finger.

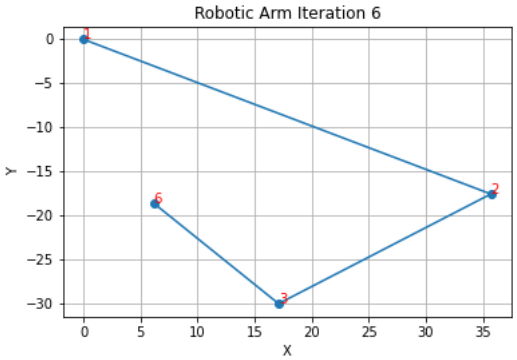

Figure 21: The sixth orientation of the index finger.

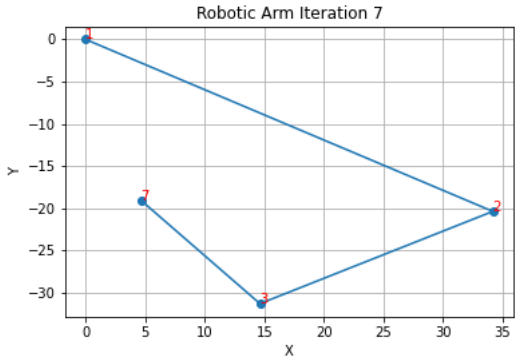

Figure 22: The seventh orientation of the index finger.

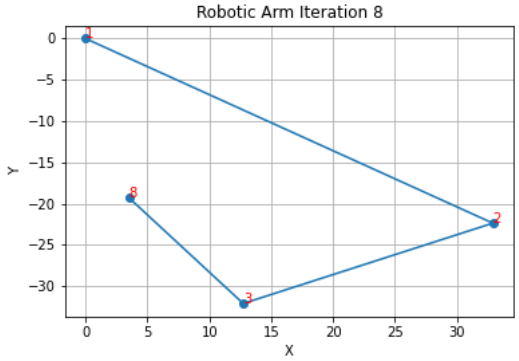

Figure 23: The eighth orientation of the index finger.

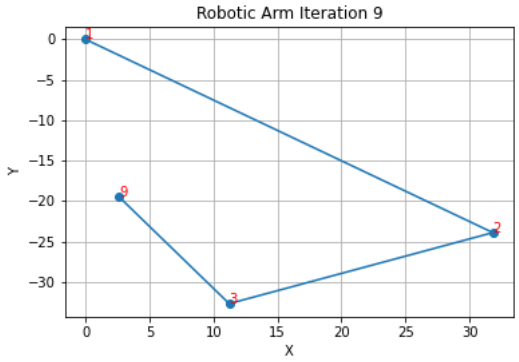

Figure 24: The ninth orientation of the index finger.

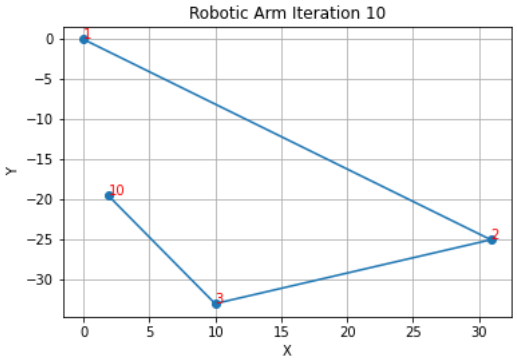

Figure 25: The tenth orientation of the index finger.

These are only the visualisations of the first 10 iterations, but even if we continue the process, we will not get much closer to the target destination. In fact, the target point was not reached even after 10000 iterations. The final joint variables are: $[-1.04719755 - 2.0943951]$ and the final position of the end-effector of the index finger is: $[-5.24364121 - 18.90784857]$.

Illusion of Trajectory

Finally, we sample some nearby points on the surface of object O and solve the inverse kinematics problem to generate the illusion of a finger sliding along the object surface. For this case we assume that the fingertip of the index finger is initially placed in position $[12, -18]$. Therefore the initial joint variables for this position which was derived previously as the result of the first examination will be $[-0.1439013 - 2.03578803]$. Given this position, we want the fingertip of the index finger to move to positions $[13, -18]$, $[14, -18]$, $[15, -18]$, $[16, -18]$, $[17, -18]$, $[18, -18]$ in order. In this process the slight difference is that in each iteration the target of the manipulator is changing. Likewise, the initial position is changing as well; in fact, the target of the previous iteration will be the initial point of the next iteration. Given the modified code for this version, the trajectory of the end-effector, will be:

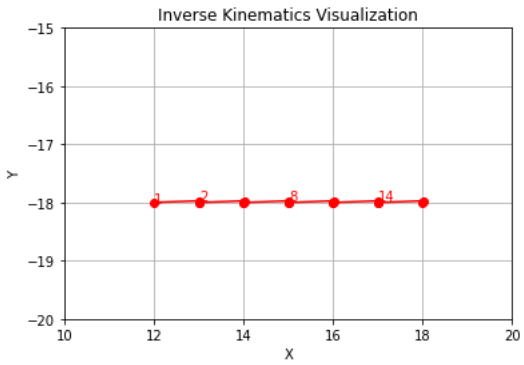

Figure 26: The trajectory of the fingertip of the index finger when moving from point [12,-18] -> [13,-18] -> [14,-18] -> [15,-18] ->[16,-18] -> [17,-18] -> [18,-18]. The image gives an illusion of the fingertip sliding on the surface of the object.

Notice that the fingertip is moving on the surface of the object O where all $y = -18$. However, this is not the real trajectory of the fingertip moving towards the final destination. The following image is a zoomed version of this trajectory.

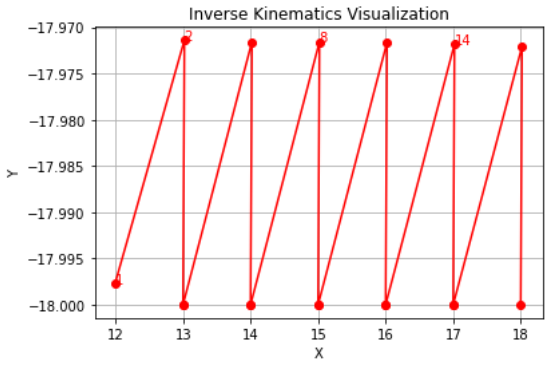

Figure 27: Same image as 26 but the zoomed version which clearly shows that the fingertip of the index finger is not sliding on the surface of the object at all.

This image indicates that in fact the fingertip is not sliding on the surface of the object but instead take steps back and forth on the surface of the object. Therefore, image 26 is showing only an illusion of a finger sliding along the object surface.

Conclusion

This project report represents a comprehensive analysis of the kinematics of a human finger. Particularly, the *index finger* was investigated in this project, however, the process and the

results can be extended for the ring, middle, a little finger. The report starts by providing an thorough explanation and visual representation of the physical structure of a human finger. Subsequently, the finger hand was modelled where each point represented a joint and each edge represented a link. The different configurations of the human index finger are examined using forward kinematics. Accordingly, the joint angles and the corresponding coordinates of the fingertip were computed. Specifically, the report investigates the kinematics of a human finger in the following two situations: 1 - with unconstrained joints 2 - with constraint joints. In the constrained joints case, the desired coordinates of the fingertip are given, and we calculate the required joint angles to achieve the desired position. In other words, we solve the inverse kinematic problem using the iterative approach.

The report suggests that forward kinematics approaches are valuable for studying various configurations of a robotic manipulator. Specifically, to examine the range of reachability forward kinematics is particularly helpful. By exploring different combinations of joint angles, such as maximum, minimum, and other important values, we can observe the resulting configurations of the robot. Furthermore, the report highlights the use of inverse kinematics in facilitating the process of manipulator movement to reach a target destination.

Overall, this study provides insights into the kinematics of a human finger, emphasizing the usefulness of both forward and inverse kinematics approaches in analyzing robotic manipulators.

Appendix

This first block is the code for solving the inverse kinematics problem. The second block is the code used for creating an illusion of sliding on the object surface example.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun May 25 01:02:01 2023

@author: mehradhq
"""
import math
import numpy as np
import matplotlib.pyplot as plt

```

```
# length of index finger links:

# proximal phalanx
MP = 39.8
# intermediate phalanx
PD = 22.4
# distal phalanx
DE = 15.8

# Define an empty list to store the x and y coordinates.
# this is for the plotting the positions of the end-effector from the initial position to the target destination.
x_coordinates = []
y_coordinates = []

# initial guess for joint variables. This can be changed by the user.
theta_M = -0.1439013
theta_P = -2.03578803
# joint variable array
initial_joint_angles = np.array([theta_M, theta_P])

# target destination. This can be changed by the user.
target_pose = np.array([13, -18])
max_iterations = 10000  # Set the maximum number of iterations. depending on the computing power this can be changed.
tolerance = 0.01  # Set the tolerance for convergence. depending on the computing power this can be changed.

# Set the ranges for joint variables
# these ranges have been given in the report.
theta_M_range = np.array([-math.pi / 3, math.pi / 3])
theta_P_range = np.array([-2 * math.pi / 3, 0])
theta_D_range = np.array([-4 * math.pi / 6, 0])

# Jacobian matrix for the index finger
def jacobian_inverse(theta1, theta2):
```

```
    jacobian = np.array([
    [
        -39.8 * math.sin(theta1) - 22.4 * math.sin(theta1 + theta2
           ) - 15.8 * math.sin(theta1 + (5/3) * theta2),
        -22.4 * math.sin(theta1 + theta2) - (5/3) * 15.8 * math.
           sin(theta1 + (5/3) * theta2)
    ],
    [
        39.8 * math.cos(theta1) + 22.4 * math.cos(theta1 + theta2)
            + 15.8 * math.cos(theta1 + (5/3) * theta2),
        22.4 * math.cos(theta1 + theta2) + (5/3) * 15.8 * math.cos
           (theta1 + (5/3) * theta2)
    ]
])

    # Compute the Moore-Penrose pseudoinverse of jacobian_inverse
    jacobian_inverse = np.linalg.pinv(jacobian)
    return jacobian_inverse

def diff_to_target(theta1, theta2):
    forward_kinematics = np.array([
        math.cos(theta1) * MP + math.cos(theta1 + theta2) * PD +
           math.cos(theta1 + (5/3) * theta2) * DE,
        math.sin(theta1) * MP + math.sin(theta1 + theta2) * PD +
           math.sin(theta1 + (5/3) * theta2) * DE
    ])
    diff = target_pose - forward_kinematics
    return diff

def inverse_kinematics(max_iterations, tolerance):
    joint_angles = initial_joint_angles.copy()

    # checking if the joint variables are in the given range.
    # given the written program they should be, but this is just
       for verification.
    assert joint_angles[0] >= theta_M_range[0]
    assert joint_angles[0] <= theta_M_range[1]
```

```
    assert joint_angles[1] >= theta_P_range[0]
    assert joint_angles[1] <= theta_P_range[1]

    for iteration in range(max_iterations):
        # Check if theta_M is within the valid range
        # if the joint M is forced to go out of its motion range
           it will resist.
        if joint_angles[0] < -math.pi / 3:
            joint_angles[0] = -math.pi / 3
        elif joint_angles[0] > math.pi / 3:
            joint_angles[0] = math.pi / 3

        # Check if theta_P is within the valid range
        # if the joint P is forced to go out of its motion range
           it will resist.
        if joint_angles[1] < -2 * math.pi / 3:
            joint_angles[1] = -2 * math.pi / 3
        elif joint_angles[1] > 0:
            joint_angles[1] = 0

        # Forward Kinematics: Compute the end effector pose using
           current joint angles
        # Calculate the error between the target pose and the
           current end effector pose
        target_difference = diff_to_target(joint_angles[0],
           joint_angles[1])
        forward_kinematics_res = np.array([
            math.cos(joint_angles[0]) * MP + math.cos(joint_angles
               [0] + joint_angles[1]) * PD +
            math.cos(joint_angles[0] + (5/3) * joint_angles[1]) *
               DE,
            math.sin(joint_angles[0]) * MP + math.sin(joint_angles
               [0] + joint_angles[1]) * PD +
            math.sin(joint_angles[0] + (5/3) * joint_angles[1]) *
               DE
        ])
        x_coordinates.append(forward_kinematics_res[0])
```

```
103         y_coordinates.append(forward_kinematics_res[1])
104
105         if (iteration<10):
106         # Create a new plot for each iteration
107             plt.figure()
108
109             # Visualize the robotic arm for the current iteration
110             plt.plot([0, math.cos(joint_angles[0]) * MP, math.cos(
                    joint_angles[0]) * MP +
111                       math.cos(joint_angles[0] + joint_angles[1])
                          * PD, forward_kinematics_res[0]],
112                      [0, math.sin(joint_angles[0]) * MP, math.sin(
                         joint_angles[0]) * MP +
113                       math.sin(joint_angles[0] + joint_angles[1])
                          * PD, forward_kinematics_res[1]],
114                      'o-')
115
116             # Number the vertices with a sequence from 1 to n
117             plt.text(0, 0, '1', color='red')
118             plt.text(math.cos(joint_angles[0]) * MP, math.sin(
                    joint_angles[0]) * MP, '2', color='red')
119             plt.text(math.cos(joint_angles[0]) * MP + math.cos(
                    joint_angles[0] + joint_angles[1]) * PD,
120                      math.sin(joint_angles[0]) * MP + math.sin(
                             joint_angles[0] + joint_angles[1]) * PD, '
                             3', color='red')
121             plt.text(forward_kinematics_res[0],
                    forward_kinematics_res[1], str(iteration + 1),
                    color='red')
122
123             plt.xlabel('X')
124             plt.ylabel('Y')
125             plt.title(f'Robotic Arm Iteration {iteration + 1}')
126             plt.grid(True)
127             # saves the image in the given directory. Change it
                    accordingly.
128             #plt.savefig(f'/Users/mehradhq/Downloads/Robotics/img{
```

```
                iteration + 1}.png')
129         if (iteration >= max_iterations - 1):
130             print("Solution not found after", max_iterations, "
                    iterations.")
131             return [joint_angles,  np.array([
132                 math.cos(joint_angles[0]) * MP + math.cos(
                        joint_angles[0] + joint_angles[1]) * PD + math
                        .cos(joint_angles[0] + (5/3) * joint_angles
                        [1]) * DE,
133                 math.sin(joint_angles[0]) * MP + math.sin(
                        joint_angles[0] + joint_angles[1]) * PD + math
                        .sin(joint_angles[0] + (5/3) * joint_angles
                        [1]) * DE
134             ])]
135         if math.sqrt(target_difference[0]**2 + target_difference
                [1]**2) < tolerance:
136             print("Solution found after", iteration, "iterations."
                    )
137             return [joint_angles,  np.array([
138                 math.cos(joint_angles[0]) * MP + math.cos(
                        joint_angles[0] + joint_angles[1]) * PD + math
                        .cos(joint_angles[0] + (5/3) * joint_angles
                        [1]) * DE,
139                 math.sin(joint_angles[0]) * MP + math.sin(
                        joint_angles[0] + joint_angles[1]) * PD + math
                        .sin(joint_angles[0] + (5/3) * joint_angles
                        [1]) * DE
140             ])]
141
142         # Calculate the Jacobian matrix
143         # Compute the pseudo-inverse of the Jacobian matrix
144         jac_inv = jacobian_inverse(joint_angles[0], joint_angles
                [1])
145         joint_angles += jac_inv.dot(target_difference)
146
147     return None
148
```

```
result = inverse_kinematics(max_iterations, tolerance)

if result is not None:
    print("the final joint variables are: ", result[0])
    print("the final position of the end-effector of the index
        finger is: ", result[1])
    # Plot the points and connect them with directed edges
    plt.figure()
    plt.plot(x_coordinates, y_coordinates, '-r', marker='o')  #
        Use '-r' to plot red edges with markers
    # Number the vertices with a sequence from 1 to n
    prev_x, prev_y = x_coordinates[0], y_coordinates[0]
    plt.text(prev_x, prev_y, str(1), color='red')
    for i, (x, y) in enumerate(zip(x_coordinates, y_coordinates),
        start=1):
         # Check the distance between consecutive points
         distance = math.sqrt((x - prev_x)**2 + (y - prev_y)**2)
         if distance > 1:  # Adjust the threshold as needed
             plt.text(x, y, str(i), color='red')
             prev_x, prev_y = x, y
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('Inverse Kinematics Visualization')
    plt.grid(True)
    plt.show()
    # saves the image in the given directory. Change it
        accordingly.
    #plt.savefig('/Users/mehradhq/Downloads/Robotics/img-end.png')
else:
    print("the final joint variables are: ", result[0])
    print("the final position of the end-effector of the index
        finger is: ", result[1])
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun May 25 01:02:01 2023
```

```
5
6  @author: mehradhq
7  """
8  import math
9  import numpy as np
10 import matplotlib.pyplot as plt
11
12 # length of index finger links:
13
14 # proximal phalanx
15 MP = 39.8
16 # intermediate phalanx
17 PD = 22.4
18 # distal phalanx
19 DE = 15.8
20
21 # Define an empty list to store the x and y coordinates.
22 # this is for the plotting the positions of the end-effector from
       the initial position to the target destination.
23 x_coordinates = []
24 y_coordinates = []
25
26 max_iterations = 10000  # Set the maximum number of iterations.
       depending on the computing power this can be changed.
27 tolerance = 0.01  # Set the tolerance for convergence. depending
       on the computing power this can be changed.
28
29 # Set the ranges for joint variables
30 # these ranges have been given in the report.
31 theta_M_range = np.array([-math.pi / 3, math.pi / 3])
32 theta_P_range = np.array([-2 * math.pi / 3, 0])
33 theta_D_range = np.array([-4 * math.pi / 6, 0])
34
35 # Jacobian matrix for the index finger
36 def jacobian_inverse(theta1, theta2):
37     jacobian = np.array([
38     [
```

```
        -39.8 * math.sin(theta1) - 22.4 * math.sin(theta1 + theta2
            ) - 15.8 * math.sin(theta1 + (5/3) * theta2),
        -22.4 * math.sin(theta1 + theta2) - (5/3) * 15.8 * math.
            sin(theta1 + (5/3) * theta2)
    ],
    [
        39.8 * math.cos(theta1) + 22.4 * math.cos(theta1 + theta2)
             + 15.8 * math.cos(theta1 + (5/3) * theta2),
        22.4 * math.cos(theta1 + theta2) + (5/3) * 15.8 * math.cos
            (theta1 + (5/3) * theta2)
    ]
])

    # Compute the Moore-Penrose pseudoinverse of jacobian_inverse
    jacobian_inverse = np.linalg.pinv(jacobian)
    return jacobian_inverse

def diff_to_target(theta1, theta2, target):
    forward_kinematics = np.array([
        math.cos(theta1) * MP + math.cos(theta1 + theta2) * PD +
            math.cos(theta1 + (5/3) * theta2) * DE,
        math.sin(theta1) * MP + math.sin(theta1 + theta2) * PD +
            math.sin(theta1 + (5/3) * theta2) * DE
    ])
    diff = target - forward_kinematics
    return diff

# initial guess for joint variables. This can be changed by the
    user.
theta_M = -0.1439013
theta_P = -2.03578803
# joint variable array
initial_joint_angles = np.array([theta_M, theta_P])

# target destination. This can be changed by the user.
target_pose = np.array
    ([[13,-18],[14,-18],[15,-18],[16,-18],[17,-18],[18,-18]])
```

```
68
69 def inverse_kinematics(max_iterations, tolerance, initial_joint,
       target):
70     joint_angles = initial_joint.copy()
71
72     # checking if the joint variables are in the given range.
73     # given the written program they should be, but this is just
          for verification.
74     assert joint_angles[0] >= theta_M_range[0]
75     assert joint_angles[0] <= theta_M_range[1]
76     assert joint_angles[1] >= theta_P_range[0]
77     assert joint_angles[1] <= theta_P_range[1]
78
79     for iteration in range(max_iterations):
80         # Check if theta_M is within the valid range
81         # if the joint M is forced to go out of its motion range
              it will resist.
82         if joint_angles[0] < -math.pi / 3:
83             joint_angles[0] = -math.pi / 3
84         elif joint_angles[0] > math.pi / 3:
85             joint_angles[0] = math.pi / 3
86
87         # Check if theta_P is within the valid range
88         # if the joint P is forced to go out of its motion range
              it will resist.
89         if joint_angles[1] < -2 * math.pi / 3:
90             joint_angles[1] = -2 * math.pi / 3
91         elif joint_angles[1] > 0:
92             joint_angles[1] = 0
93
94         # Forward Kinematics: Compute the end effector pose using
              current joint angles
95         # Calculate the error between the target pose and the
              current end effector pose
96         target_difference = diff_to_target(joint_angles[0],
              joint_angles[1], target)
97         forward_kinematics_res = np.array([
```

```
98              math.cos(joint_angles[0]) * MP + math.cos(joint_angles
                    [0] + joint_angles[1]) * PD +
99              math.cos(joint_angles[0] + (5/3) * joint_angles[1]) *
                    DE,
100             math.sin(joint_angles[0]) * MP + math.sin(joint_angles
                    [0] + joint_angles[1]) * PD +
101             math.sin(joint_angles[0] + (5/3) * joint_angles[1]) *
                    DE
102         ])
103         x_coordinates.append(forward_kinematics_res[0])
104         y_coordinates.append(forward_kinematics_res[1])
105
106         if (iteration<10):
107         # Create a new plot for each iteration
108             plt.figure()
109
110             # Visualize the robotic arm for the current iteration
111             plt.plot([0, math.cos(joint_angles[0]) * MP, math.cos(
                    joint_angles[0]) * MP +
112                       math.cos(joint_angles[0] + joint_angles[1])
                           * PD, forward_kinematics_res[0]],
113                      [0, math.sin(joint_angles[0]) * MP, math.sin(
                           joint_angles[0]) * MP +
114                       math.sin(joint_angles[0] + joint_angles[1])
                           * PD, forward_kinematics_res[1]],
115                      'o-')
116
117             # Number the vertices with a sequence from 1 to n
118             plt.text(0, 0, '1', color='red')
119             plt.text(math.cos(joint_angles[0]) * MP, math.sin(
                    joint_angles[0]) * MP, '2', color='red')
120             plt.text(math.cos(joint_angles[0]) * MP + math.cos(
                    joint_angles[0] + joint_angles[1]) * PD,
121                      math.sin(joint_angles[0]) * MP + math.sin(
                           joint_angles[0] + joint_angles[1]) * PD, '
                           3', color='red')
122             plt.text(forward_kinematics_res[0],
```

```
                forward_kinematics_res[1], str(iteration + 1),
                color='red')

            plt.xlabel('X')
            plt.ylabel('Y')
            plt.title(f'Robotic Arm Iteration {iteration + 1}')
            plt.grid(True)
            # saves the image in the given directory. Change it
                accordingly.
            #plt.savefig(f'/Users/mehradhq/Downloads/Robotics/img{
                iteration + 1}.png')
        if (iteration >= max_iterations - 1):
            print("Solution not found after", max_iterations, "
                iterations.")
            return [joint_angles,  np.array([
                 math.cos(joint_angles[0]) * MP + math.cos(
                     joint_angles[0] + joint_angles[1]) * PD + math
                     .cos(joint_angles[0] + (5/3) * joint_angles
                     [1]) * DE,
                 math.sin(joint_angles[0]) * MP + math.sin(
                     joint_angles[0] + joint_angles[1]) * PD + math
                     .sin(joint_angles[0] + (5/3) * joint_angles
                     [1]) * DE
             ])]
        if math.sqrt(target_difference[0]**2 + target_difference
            [1]**2) < tolerance:
            print("Solution found after", iteration, "iterations."
                )
            return [joint_angles,  np.array([
                 math.cos(joint_angles[0]) * MP + math.cos(
                     joint_angles[0] + joint_angles[1]) * PD + math
                     .cos(joint_angles[0] + (5/3) * joint_angles
                     [1]) * DE,
                 math.sin(joint_angles[0]) * MP + math.sin(
                     joint_angles[0] + joint_angles[1]) * PD + math
                     .sin(joint_angles[0] + (5/3) * joint_angles
                     [1]) * DE
```

```
            ])]

        # Calculate the Jacobian matrix
        # Compute the pseudo-inverse of the Jacobian matrix
        jac_inv = jacobian_inverse(joint_angles[0], joint_angles
            [1])
        joint_angles += jac_inv.dot(target_difference)

    return None

for i in target_pose:
    result = inverse_kinematics(max_iterations, tolerance,
        initial_joint_angles, i)

    if result is not None:
        print("the final joint variables are: ", result[0])
        print("the final position of the end-effector of the index
            finger is: ", result[1])
        # Plot the points and connect them with directed edges
        plt.figure()
        plt.xlim(10, 20)  # Adjust the x-axis limits as desired
        plt.ylim(-20, -15)  # Adjust the y-axis limits as desired
        plt.plot(x_coordinates, y_coordinates, '-r', marker='o')
            # Use '-r' to plot red edges with markers
        # Number the vertices with a sequence from 1 to n
        prev_x, prev_y = x_coordinates[0], y_coordinates[0]
        plt.text(prev_x, prev_y, str(1), color='red')
        for i, (x, y) in enumerate(zip(x_coordinates,
            y_coordinates), start=1):
            # Check the distance between consecutive points
            distance = math.sqrt((x - prev_x)**2 + (y - prev_y)
                **2)
            if distance > 1:  # Adjust the threshold as needed
                plt.text(x, y, str(i), color='red')
                prev_x, prev_y = x, y
        plt.xlabel('X')
        plt.ylabel('Y')
```

```
172         plt.title('Inverse Kinematics Visualization')
173         plt.grid(True)
174         plt.show()
175         # Adjust the plot limits to show a larger area
176         # saves the image in the given directory. Change it
               accordingly.
177         #plt.savefig('/Users/mehradhq/Downloads/Robotics/img-end.
               png')
178     else:
179         print("the final joint variables are: ", result[0])
180         print("the final position of the end-effector of the index
               finger is: ", result[1])
181
182     initial_joint_angles = result[0]
```

REFERENCES

[1] Hinge joint (2023) Wikipedia. Available at: https://en.wikipedia.org/wiki/Hinge_joint (Accessed: 19 May 2023).

[2] Hovius, S. E. R., Potuijt, J. W. P., & van Nieuwenhoven, C. A. (2019). Triphalangeal thumb: clinical features and treatment. The Journal of hand surgery, European volume, 44(1), 69–79. https://doi.org/10.1177/1753193418797922

[3] [1] A. Buryanov and V. Kotiuk, Proportions of hand segments, Int. J. Morphol. 28(3) (2010), pp. 755-758.

[4] File:scheme human hand bones-en.svg (no date) Wikimedia Commons. Available at: https://commons.wikimedia.org/wiki/File:Scheme_human_hand_bones-en.svg (Accessed: 19 May 2023).

[5] JAZAR, R. N. (2023). Theory of applied robotics: Kinematics, dynamics, and control. SPRINGER NATURE.