Inverse kinematics via gradient descent

Mathematics Internal Assessment

12 pages

1 Introduction

When I think about automation, the first thing that comes to my mind is the image of a fleet of robotic arms on an assembly line, operating at an incredible precision. But these machines are not usually considered as "intelligent" due to their ubiquity in the industry. This essay is an attempt to uncover the mathematical complexity behind robotic arms, by proposing a solution to a problem which is fundamental within robotics: *inverse kinematics*.

Inverse kinematics is the problem of determining the configuration of every joint within an arm (e.g., by configuring its angle) in order to have the end-effector (the tip of the arm) reach a specific target. We humans solve this problem effortlessly after years of experience: every time you control your shoulder, elbow, and wrist joints to sip a cup of coffee, you are solving inverse kinematics. But it is not trivial to distill that intuition into a robotic arm. In fact, it is impossible to derive a closed-form solution[1] for inverse kinematics[1] because it is possible to have an infinite number of solutions, as shown in Fig. 1.

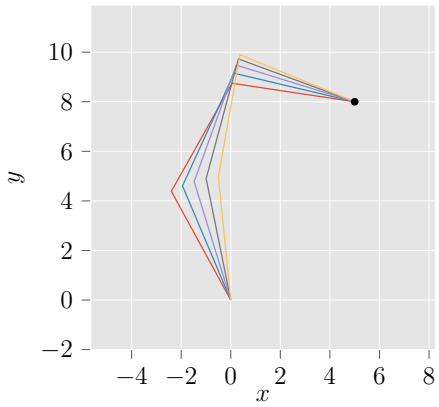

Figure 1: 5 out of an infinite number of inverse kinematics solutions to reach a target coordinate (black) with an arm with 3 segments of equal length 5

As such, inverse kinematics solutions are almost always obtained through iterative methods whose accuracy improves with the number of iterations. Although there are iterative non-gradient based inverse kinematics solutions such as Cyclic Coordinate Descent[2], the scope of this essay is limited to an iterative gradient-based method called gradient descent, which allows me to make the most out of my calculus classes.

The goal of this essay is the answer the question: *can gradient descent be used to solve inverse kinematics?*

[1] except when the arm with only 2 segments

2 Forward kinematics

Throughout this essay, the axes naming convention in Fig. 2 will be used, the base of any arm will be at $s_0 = (0, 0, 0)$ in 3d or $s_0 = (0, 0)$ in 2d, lengths have arbitrary units, and angles are in radians.

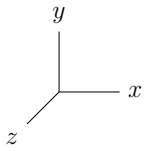

Figure 2: Axes naming convention

2.1 2 dimensions

To describe an arm mathematically, let any arm consisting of n joints/segments be parameterized by $l = [l_1, l_2, ..., l_n]$ and $\theta = [\theta_1, \theta_2, ...\theta_n]$. Where l_i is the length of segment i and θ_i is the angle measured counterclockwise from the extension of the previous segment $(i - 1)$ to segment i. An example configuration is shown in Fig. 3.

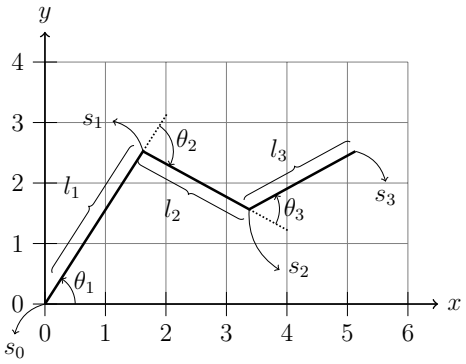

Figure 3: An xy view of an arm parameterized by $l = [3, 2, 2]$ and $\theta = [1, -1.5, 1]$

There are several important observations to make:

1. The arm is only composed of "bending"-joints, which rotate segments with larger values of i in the xy-plane, for example, θ_2 affects the direction of segment 3 but not segment 1,

2. θ_1 is defined as the angle measured counterclockwise from the positive x-axis to segment 1 since segment (-1) does not exist, and

3. θ_i can be negative, as shown through the direction of the angle measure arrows in Fig. 3.

Let $s_i = (x_i, y_i)$ denote the end position of segment i. In order to optimize θ so that the end effector s_n nears a target position, s_n itself must first be calculated — this is the problem of forward kinematics. To calculate s_n, the end positions of previous segments must also be obtained. Using the arm in Fig. 3 as an example, s_1 can be deduced using simple trigonometry:

$$\begin{aligned} x_1 &= x_0 + l_1 \cos\theta_1 = 0 + 3\cos 1 \approx 1.62 \\ y_1 &= y_0 + l_1 \sin\theta_1 = 0 + 3\sin 1 \approx 2.52 \end{aligned} \tag{1}$$

Let θ_i' be defined as the counterclockwise angle between from the positive direction of a horizontal line to segment i. Deriving s_2 is a bit more involved, since cosine and sine functions measure offsets against the x- and y-axes respectively, but θ_2 is not calculated with the x-axis as a reference, but against the extension of segment 1. As such, θ_2' must be obtained.

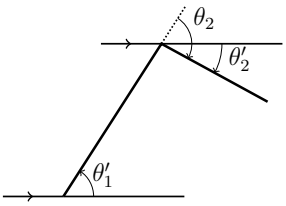

Figure 4: Close-up of Fig. 3 and θ_2'

As shown in Fig. 4, because both horizontal lines are parallel,

$$\begin{aligned} \theta_1' &= \theta_1 = -\theta_2 + \theta_2' \\ \theta_2' &= \theta_1' + \theta_2 \end{aligned} \tag{2}$$

θ_1' is equivalent to θ_1 by definition, as demonstrated in Observation 2. Now s_2 can be obtained with θ_2':

$$\begin{aligned} x_2 &= x_1 + l_2 \cos\theta_2' \\ &= l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \\ &= 3\cos 1 + 2\cos(1 - 1.5) \qquad \approx 3.38 \\ y_2 &= y_1 + l_2 \sin\theta_2' \\ &= l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \\ &= 3\sin 1 + 2\sin(1 - 1.5) \qquad \approx 1.57 \end{aligned} \tag{3}$$

To find s_3, θ_3' can be written as:

$$\theta_3' = \theta_2' + \theta_3 = \theta_1 + \theta_2 + \theta_3$$

Loosely speaking, this is because in Fig. 4, θ_1' can be substituted with θ_2', θ_2 with θ_3, and

θ_2' with θ_3'. Applying this property *ad infinitum*, θ_j' is equivalent to

$$\theta_j' = \theta_1 + \theta_2 + ... + \theta_i = \sum_{k=1}^{j} \theta_k \tag{4}$$

Calculating s_3:

$$\begin{aligned} x_3 &= x_2 + l_3 \cos\theta_3' \\ &= l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ &= 3\cos 1 + 2\cos(1 - 1.5) + 2\cos(1 - 1.5 + 1) \qquad \approx 5.13 \\ y_3 &= y_2 + l_3 \sin\theta_3' \\ &= l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) \\ &= 3\sin 1 + 2\sin(1 - 1.5) + 2\sin(1 - 1.5 + 1) \qquad \approx 2.52 \end{aligned} \tag{5}$$

All example points s_i for $i = [0, 1, 2, 3]$ are plotted in Fig. 3.

Extrapolating from Eq. 1, Eq. 3, and Eq. 5, the general equation for s_i can be written as:

$$\begin{aligned} x_i &= l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) + ... + l_i \cos(\theta_1 + \theta_2 + ... + \theta_i) \\ &= \sum_{j=1}^{i} l_j \cos\left(\sum_{k=1}^{j} \theta_k\right) \\ y_i &= l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) + ... + l_i \sin(\theta_1 + \theta_2 + ... + \theta_i) \\ &= \sum_{j=1}^{i} l_j \sin\left(\sum_{k=1}^{j} \theta_k\right) \end{aligned} \tag{6}$$

2.2 3 dimensions

But robotic arms aren't 2-dimensional, they are usually able to reach objects in 3 coordinates. The easiest way to generalize Eq. 6 to 3 dimensions is to introduce a single rotation joint at the base of the arm which operates in the xz-plane. Let this joint be parameterized by θ_b, defined as the clockwise angle between from the positive x-axis to segment 1 when $\theta_1 = 0$ as measured through a birds-eye view (when $y > 0$).[2] An example configuration is shown in Fig. 5.

[2]the $\theta_1 = 0$ condition is required so that θ_b does not change by π when θ_1 fluctuates between $\theta_1 < \frac{\pi}{2}$ and $\theta_1 > \frac{\pi}{2}$

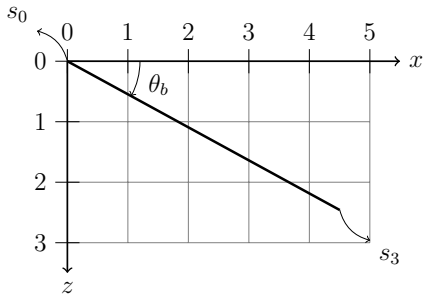

Figure 5: A birds-eye xz-projection of an arm parameterized by $l = [3, 2, 2]$ $\theta = [1, -1.5, 1]$, and $\theta_b = 0.5$

Note that in 3 dimensions, s_i is redefined as (x_i, y_i, z_i). The changes needed to Eq. 6 accommodate this are trivial. x_i is split into x_i and z_i, while y_i remains identical since a rotation in the xz-plane does not affect y-axis coordinates. In summary, s_i can be written as

$$\begin{aligned} x_i &= \cos(\theta_b) \sum_{j=1}^{i} l_j \cos\left(\sum_{k=1}^{j} \theta_k\right) \\ y_i &= \sum_{j=1}^{i} l_j \sin\left(\sum_{k=1}^{j} \theta_k\right) \\ z_i &= \sin(\theta_b) \sum_{j=1}^{i} l_j \cos\left(\sum_{k=1}^{j} \theta_k\right) \end{aligned} \tag{7}$$

Recalculating s_3 for Fig. 5 using values already obtained in Eq. 5:

$$\begin{aligned} x_3 &\approx \cos(0.5) \cdot 5.13 \approx 4.50 \\ y_3 &\approx 2.52 \\ z_3 &\approx \sin(0.5) \cdot 5.13 \approx 2.46 \end{aligned} \tag{8}$$

x_3 and z_3 are also plotted in Fig. 5.

3 Inverse kinematics

Now that s_n can be found given l, θ, and θ_b, how do you find θ and θ_b given s_n and l?

3.1 Gradient descent

Gradient descent[3] is an algorithm to iteratively minimize a loss function, with information of the first derivative. A single iteration of gradient descent is defined as follows:

$$\theta' = \theta - \alpha \frac{dL(\theta)}{d\theta} \tag{9}$$

where θ is the parameter to optimize, θ' is the newly optimized parameter value, L is the loss function, and α is an arbitrary positive scaling factor. Intuitively, this means that θ

is shifted in the direction that minimizes $L(\theta)$ in proportion to how sensitive $L(\theta)$ is to θ (the definition of a derivative).

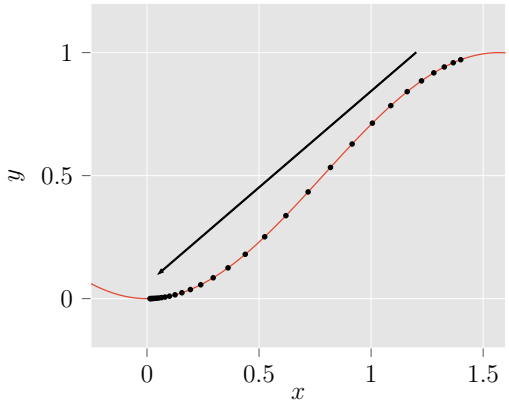

Figure 6: 30 iterations of gradient descent applied on $L(\theta) = \sin^2(\theta)$, initial $\theta = 1.4$, and $\alpha = 0.1$. Each black point represents an updated θ'.

Fig. 6 demonstrates gradient descent's effectiveness in converging to a local minimum on a relatively simple function. The hope is for it to generalize to the inverse kinematics problem.

First of all, Eq. 9 must be modified to use partial derivatives, to be able to optimize multiple θ_i and θ_b in the inverse kinematics problem simultaneously:

$$\theta_i' = \theta_i - \alpha \frac{\partial L(\theta_1, ..., \theta_n, \theta_b)}{\partial \theta_i} \quad \text{and} \quad \theta_b' = \theta_b - \alpha \frac{\partial L(\theta_1, ..., \theta_n, \theta_b)}{\partial \theta_b} \tag{10}$$

3.2 Loss function derivatives

Let $s_t = (x_t, y_t, z_t)$ denote the target position, the desired value of s_n (the end-effector). Inverse kinematics can be framed as a gradient descent optimization problem by choosing a function L that decreases as s_n approaches s_t. Theoretically, almost any distance metric can be used for L, but the squared Euclidean distance[4] d^2_{euclid} will be used here for simplicity.[3] Thus, the value of L can be written as:

$$L(\theta_1, ..., \theta_n, \theta_b) = d^2_{euclid}(s_t, s_n) = (x_t - x_n)^2 + (y_t - y_n)^2 + (z_t - z_n)^2 \tag{11}$$

where s_n is to be calculated with Eq. 7.

Now the partial derivative against θ_i in Eq. 10 can be evaluated:

$$\frac{\partial L(\theta_1, ..., \theta_n, \theta_b)}{\partial \theta_i} = \frac{\partial \left((x_t - x_n)^2 + (y_t - y_n)^2 + (z_t - z_n)^2\right)}{\partial \theta_i}$$

$\rightarrow$ splitting the terms and applying chain rule,

$$\frac{\partial L(\theta_1, ..., \theta_n, \theta_b)}{\partial \theta_i} = 2(x_t - x_n)\frac{\partial (x_t - x_n)}{\partial \theta_i} + 2(y_t - y_n)\frac{\partial (y_t - y_n)}{\partial \theta_i} + 2(z_t - z_n)\frac{\partial (z_t - z_n)}{\partial \theta_i}$$

[3] to avoid a square root term and another chain rule application

$\rightarrow$ omitting x_t, y_t, and z_t in the numerator because they are constant with respect to θ_i,

$$\frac{\partial L(\theta_1, ..., \theta_n, \theta_b)}{\partial \theta_i} = 2(x_t - x_n)\frac{\partial(-x_n)}{\partial \theta_i} + 2(y_t - y_n)\frac{\partial(-y_n)}{\partial \theta_i} + 2(z_t - z_n)\frac{\partial(-z_n)}{\partial \theta_i} \tag{12}$$

$\rightarrow$ computing $\frac{\partial(-x_n)}{\partial \theta_i}$ by substituting in Eq. 7:

$$\frac{\partial(-x_n)}{\partial \theta_i} = -\cos(\theta_b)\frac{\partial}{\partial \theta_i}\left(l_1 \cos(\theta_1) + ... + l_n \cos(\theta_1 + ... + \theta_i + ... + \theta_n)\right)$$

$\rightarrow$ omitting terms which do not contain θ_i because they evaluate to 0,

$$\frac{\partial(-x_n)}{\partial \theta_i} = -\cos(\theta_b)\frac{\partial}{\partial \theta_i}\left(l_i \cos(\theta_1 + ... + \theta_i) + ... + l_n \cos(\theta_1 + ... + \theta_i + ... + \theta_n)\right)$$

$\rightarrow$ taking the derivative of cos,

$$\begin{aligned}\frac{\partial(-x_n)}{\partial \theta_i} &= \cos(\theta_b)\left(l_i \sin(\theta_1 + ... + \theta_i) + ... + l_n \sin(\theta_1 + ... + \theta_i + ... + \theta_n)\right) \\ &= \cos(\theta_b)\sum_{j=i}^{n} l_j \sin\left(\sum_{k=1}^{j}\theta_k\right)\end{aligned}$$

$\rightarrow$ substituting in Eq. 7's y_i,

$$\frac{\partial(-x_n)}{\partial \theta_i} = \cos(\theta_b)(y_n - y_{i-1}) \tag{13}$$

$\rightarrow$ by using the same steps, $\frac{\partial(-y_n)}{\partial \theta_i}$ and $\frac{\partial(-z_n)}{\partial \theta_i}$ were found to be:

$$\begin{aligned}\frac{\partial(-y_n)}{\partial \theta_i} &= -\sum_{j=i}^{n} l_j \cos\left(\sum_{k=1}^{j}\theta_k\right) &&= -\frac{x_n - x_{i-1}}{\cos\theta_b} \\ \frac{\partial(-z_n)}{\partial \theta_i} &= \sin(\theta_b)\sum_{j=i}^{n} l_j \sin\left(\sum_{k=1}^{j}\theta_k\right) &&= \sin(\theta_b)(y_n - y_{i-1})\end{aligned} \tag{14}$$

$\rightarrow$ substituting Eq. 13 and Eq. 14 into Eq. 12 yields the final $\frac{\partial L(\theta_1,...,\theta_n,\theta_b)}{\partial \theta_i}$:

$$\begin{aligned}\frac{\partial L(\theta_1, ..., \theta_n, \theta_b)}{\partial \theta_i} =& 2\cos(\theta_b)(x_t - x_n)(y_n - y_{i-1}) \\ &- 2(y_t - y_n)\frac{x_n - x_{i-1}}{\cos\theta_b} \\ &+ 2\sin(\theta_b)(z_t - z_n)(y_n - y_{i-1})\end{aligned} \tag{15}$$

The partial derivative for θ_b is handled differently:

$$\frac{\partial L(\theta_1, ..., \theta_n, \theta_b)}{\partial \theta_b} = 2(x_t - x_n)\frac{\partial(-x_n)}{\partial \theta_b} + 2(y_t - y_n)\frac{\partial(-y_n)}{\partial \theta_b} + 2(z_t - z_n)\frac{\partial(-z_n)}{\partial \theta_b} \tag{16}$$

$\rightarrow$ substituting in Eq. 7 and taking the derivative,

$$\begin{aligned}\frac{\partial(-x_n)}{\partial \theta_b} &= -\frac{\partial}{\partial \theta_b}\cos(\theta_b)\sum_{j=1}^{n} l_j \cos\left(\sum_{k=1}^{j}\theta_k\right) = z_n \\ \frac{\partial(-y_n)}{\partial \theta_b} &= \frac{\partial}{\partial \theta_b}\sum_{j=1}^{n} l_j \sin\left(\sum_{k=1}^{j}\theta_k\right) = 0\end{aligned}$$

$$\frac{\partial(-z_n)}{\partial\theta_b} = -\frac{\partial}{\partial\theta_b}\sin(\theta_b)\sum_{j=1}^{n} l_j \cos\left(\sum_{k=1}^{j}\theta_k\right) = -x_n \tag{17}$$

$\rightarrow$ substituting Eq. 17 into Eq. 16 yields the final

$$\frac{\partial L(\theta_1, ..., \theta_n, \theta_b)}{\partial\theta_b} = 2(x_t - x_n)z_n - 2(z_t - z_n)x_n \tag{18}$$

The reason θ_b is optimized through gradient descent instead of deducing it from Fig. 5, that is,

$$\theta_b = \arctan\frac{z_t}{x_t} \tag{19}$$

is because Eq. 19 necessitates the use of additional acceleration/deceleration functions in order to prevent an abrupt change in the s_n, whereas gradient descent is inherently smooth in its θ transitions,[4] see Fig. 6 for a visual example.

Although it should have been obvious that the partial derivatives only depend on the values of s_i, I found the simplicity of Eq. 15 and Eq. 18 to be quite surprising because the $\sum$s could be tucked away behind s_{i-1}.

3.3 Gradient descent update

Now that all the partial derivative equations have been obtained, gradient descent optimization can be used to solve the inverse kinematics problem. For example, let an arm have the same the parameters as Fig. 5, where $l = [3, 2, 2]$, $\theta = [1, -1.5, 1]$, and $\theta_b = 0.5$, with target $s_t = [6, 4, -2]$. First, s_i for $i \in [0, 1, 2, 3]$ must be calculated:[5]

i	x_i	y_i	z_i
0	0.00	0.00	0.00
1	1.42	2.52	0.78
2	2.96	1.57	1.62
3	4.50	2.52	2.46

Table 1: Values of s_i for $l = [3, 2, 2]$, $\theta = [1, -1.5, 1]$, and $\theta_b = 0.5$

From Table 1, $d^2_{euclid}(s_t, s_n)$ (Eq. 11) can be calculated:

$$\begin{aligned} d^2_{euclid}(s_t, s_n) &= (x_t - x_n)^2 + (y_t - y_n)^2 + (z_t - z_n)^2 \\ &\approx (6 - 4.50)^2 + (4 - 2.52)^2 + (-2 - 2.46)^2 \\ &\approx 24.31 \end{aligned} \tag{20}$$

Next, the partial derivatives should be computed. Calculating $\frac{\partial L(\theta_1,...,\theta_n,\theta_b)}{\partial\theta_1}$ using Eq. 15:

$$\frac{\partial L(\theta_1, ..., \theta_n, \theta_b)}{\partial\theta_1} = 2\cos(\theta_b)(x_t - x_3)(y_3 - y_0)$$

[4]for relatively small values of α in Eq. 10

[5]since examples have already been given in Eq. 5 and Eq. 8, working will be omitted here for brevity

$$
\begin{aligned}
&- 2(y_t - y_3)\frac{x_3 - x_0}{\cos\theta_b} \\
&+ 2\sin(\theta_b)(z_t - z_3)(y_3 - y_0) \\
\approx &2\cos(0.5)(6 - 4.50)(2.52 - 0.00) \\
&- 2(4 - 2.52)\frac{4.50 - 0.00}{\cos(0.5)} \\
&+ 2\sin(0.5)(-2 - 2.46)(2.52 - 0.00) \\
\approx &6.63 - 15.14 - 10.80 \approx -19.31
\end{aligned} \tag{21}
$$

Using the same method, $\frac{\partial L(\theta_1,...,\theta_n,\theta_b)}{\partial\theta_2} \approx -10.36$ and $\frac{\partial L(\theta_1,...,\theta_n,\theta_b)}{\partial\theta_3} \approx -6.76$.

Calculating $\frac{\partial L(\theta_1,...,\theta_n,\theta_b)}{\partial\theta_b}$ using Eq. 18:

$$
\begin{aligned}
\frac{\partial L(\theta_1,...,\theta_n,\theta_b)}{\partial\theta_b} &= 2(x_t - x_n)z_n - 2(z_t - z_n)x_n \\
&\approx 2(6 - 4.50)2.46 - 2(-2 - 2.46)4.50 \\
&\approx 7.37 + 40.17 \approx 47.53
\end{aligned} \tag{22}
$$

A small $\alpha = 10^{-3}$ will be used so that the maximum $\Delta\theta$ is approximately 0.1. Now the gradient descent update from Eq. 10 can finally be performed:

$$
\begin{aligned}
\begin{bmatrix}\theta_1' \\ \theta_2' \\ \theta_3' \\ \theta_b'\end{bmatrix} &= \begin{bmatrix}\theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_b\end{bmatrix} - \alpha\frac{\partial L(\theta_1,...,\theta_n,\theta_b)}{\partial\theta} \\
&\approx \begin{bmatrix}1 \\ -1.5 \\ 1 \\ 0.5\end{bmatrix} - 10^{-3}\begin{bmatrix}-19.31 \\ -10.36 \\ -6.76 \\ 47.53\end{bmatrix} \approx \begin{bmatrix}1.02 \\ -1.49 \\ 1.01 \\ 0.45\end{bmatrix}
\end{aligned} \tag{23}
$$

Let s_n' denote s_n recalculated with θ_i' and θ_b'. Calculating $d^2_{euclid}(s_t, s_n')$ with the new parameters obtained in Eq. 23 leads to a lower $L(\theta_1,...,\theta_n,\theta_b) \approx 21.62$ because $s_3' = [4.56, 2.67, 2.22]$. The change in Eq. 23 is shown visually in Fig. 7.

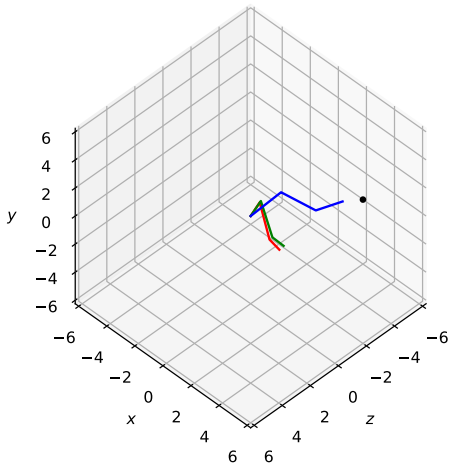

Figure 7: 3d plot of arms in multiple stages of gradient descent. Red: initial parameters $l = [3, 2, 2]$, $\theta = [1, -1.5, 1]$ and $\theta_b = 0.5$, Green: after 1 iteration, Blue: after 100 iterations. $s_t = [6, 4, -2]$ in black.

4 Evaluation

4.1 Advantages

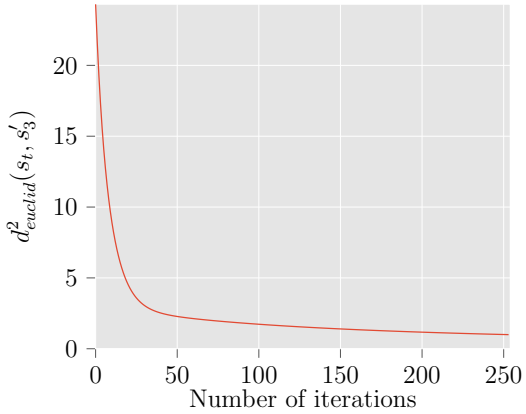

Figure 8: Graph of d^2_{euclid} with an increasing number of gradient descent iterations. Same arm configuration as Fig. 7.

In response to the research question, it can be deduced that gradient descent is a viable approach in tackling inverse kinematics, as made apparent by Fig. 7 and Fig. 8. Fig. 7 shows visually that gradient descent allows the position of the end-effector s'_3 to approach the target s_t. Moreover, Fig. 8 empirically demonstrates that the loss function decays approximately exponentially, confirming that the distance between s'_3 and s_t decreases with the number of gradient descent iterations. Therefore, it can be concluded that the value of $d^2_{euclid}(s_t, s'_3)$ converges asymptotically with gradient descent.

There is an additional observation to make about Fig. 7: *it is impossible to achieve* $d^2_{euclid}(s_t, s'_3) = 0$ *with* $l = [3, 2, 2]$, because the total length of all the segments (7) is less than the magnitude of s_t (≈ 7.48). This highlights an advantage an iterative method like gradient descent has over analytical/closed-form solutions:[6] iterative methods "try their best", leading to solutions which are *optimally close* to s_t where analytical methods would fail.[7] What impressed me the most however was that gradient descent was able to converge onto a single inverse kinematics solution out of the potentially infinite number of solutions Fig. 1. It was outright *magical* for me to be able to finally understand how these machines work.

Note that the forward and inverse kinematics framework provided within this essay are not limited to the case where $n = 3$, specific values of l, or certain initial values θ and θ_b; a single example was used throughout the essay for the sole purpose of maintaining coherence. The only restriction which was found after trial and error was that α had to be adequately small ($\alpha < 10^{-2}$) to ensure convergence and prevent fluctuations in Fig. 8. This is not to say that the framework is foolproof.

4.2 Limitations

First of all, the types of the joints (xy-[8] and xz- rotation joints) used within this essay were purposefully simplistic, the minimum needed to allow s_n to span 3-dimensions. In reality, robotic arms involve more complex joints such as rotation joints which do not operate in a plane perpendicular to x-, y-, or z-axes and shoulder-joints. Although it is technically possible to derive forward kinematics equations with $\sum$-notation like that in Eq. 7, it is more feasible to use the transformation and Jacobian matrices outlined in [5] instead.

Moreover, the framework presented within this essay lacks collision detection, allowing conditions such as that in Fig. 9 to occur.

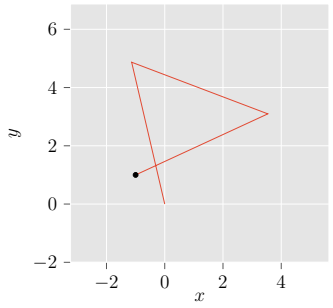

Figure 9: xy-view of an arm with $l = [5, 5, 5]$, $\theta \approx [1.80, -2.16, -2.35]$, and $\theta_b = 0$ with $s_t = [-1, 1]$

[6] if there were any for $n > 2$

[7] e.g. the solution to the quadratic solving inverse kinematics where $n = 2$ would be undefined if s_t is out of range due to a negative discriminant

[8] xy in 2d, any plane perpendicular to xz in 3d

Quite obviously, a physical arm cannot intersect itself and would instead damage the actuators and motors within the arm.

A possible way to counteract this is to perform multi-objective optimization.[6] For each segment, the shortest distance to another segment (which is not directly connected via a joint) should be calculated, and the sum of the negatives of these distances should be added to the loss function in Eq. 11. As a result, gradient descent should minimize the loss by maximizing the distance between segments (which are not already connected via a joint), minimizing collisions. This suggestion is merely hypothetical and has not been tested.

Similarly, there is a way to minimize the energy consumption of the arm by minimizing the total torque applied by all of the joints within the arm. The summation of torque values can also be added to Eq. 11 so that gradient descent will minimize the loss by minimizing the total torque and hence, the necessary power to operate the arm.

Both of these would allow for the gradient descent algorithm to be more robust and applicable to deployments in the industry.

This essay has merely scratched the surface of what is possible by successfully, albeit simplistically, applying gradient descent to the problem of inverse kinematics. There are still an *armful* of possible improvements to extend the reach of these arms.

References

[1] user25658, *What does a "closed-form solution" mean?* Cross Validated. [Online]. Available: `https://stats.stackexchange.com/q/70850`.

[2] R. Juckett, *Cyclic coordinate descent in 2d*, Feb. 2009. [Online]. Available: `http://www.ryanjuckett.com/programming/cyclic-coordinate-descent-in-2d/`.

[3] E. W. Weisstein. (). "Method of steepest descent, From mathworld–a wolfram web resource," [Online]. Available: `https://mathworld.wolfram.com/MethodofSteepestDescent.html` (visited on 06/09/2020).

[4] ——, (). "Euclidean metric, From mathworld–a wolfram web resource," [Online]. Available: `https://mathworld.wolfram.com/EuclideanMetric.html` (visited on 12/05/2020).

[5] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Journal of Robotics and Automation*, vol. 17, no. 1-19, p. 16, 2004.

[6] K. Deb, "Multi-objective optimization," in *Search methodologies*, Springer, 2014, pp. 403–449.