Final Project

Inverse Kinematics Neural Network

ECE-UY 4563: Introduction to Machine Learning

Omar Rayyan (olr7742)
Mahmoud Hafez (mah9935)

Simulation: https://orayyan.com/projects/IKNN/simulation

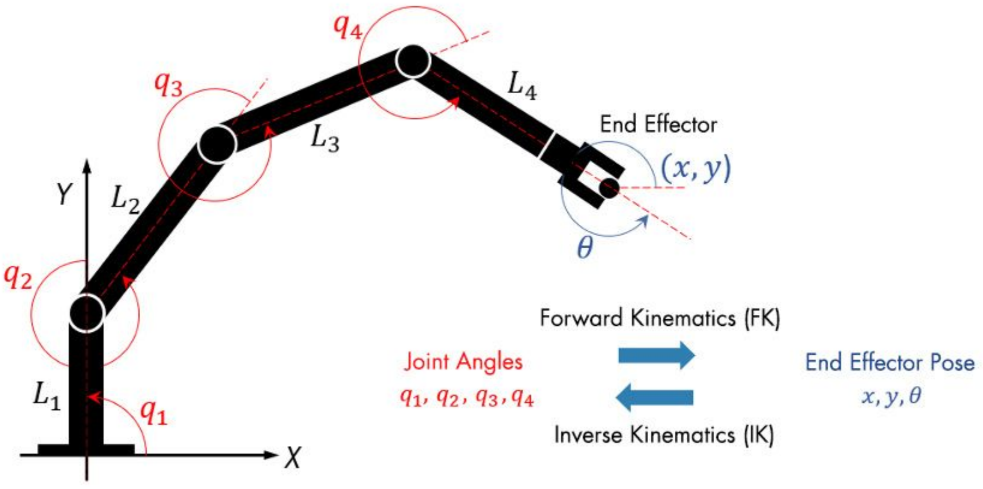
Python Notebook: https://github.com/omarrayyann/IK-NN/blob/master/training$_m$*odel.ipynb*

Presentation Video: https://www.youtube.com/watch?v=yhE8fx0q2Q4feature=youtu.be

Contents

1. Background and Motivation

Forward and Inverse kinematics are concepts used in robotics to describe the relationship between the robot's joints and the position/orientation of its end-effector (such as the gripper or tool it uses). They are crucial for controlling how a robot moves in space. **Inverse kinematics** allows us to compute the joint angles required by a robotic manipulator to have its end-effector at a certain position and orientation. On the other hand, **Forward kinematics** allows us to determine the position and orientation of the end-effector when we know the joint angles. It's like plotting the trajectory of the end-effector based on how the joints move.

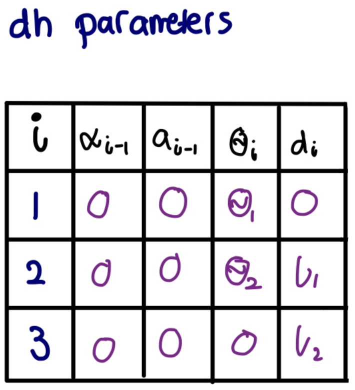


For complex robotic systems with multiple joints and links, inverse kinematics can sometimes have **multiple solutions** or even be impossible to solve in certain configurations. As the number of joints increases, computing closed-form solutions for the inverse kinematics of a manipulator becomes more challenging. The goal of our project is to train a neural network that is capable of computing the inverse kinematics of a 2 joints manipulator allowing it to track a trajectory by the end-effector.

2. Data Generation

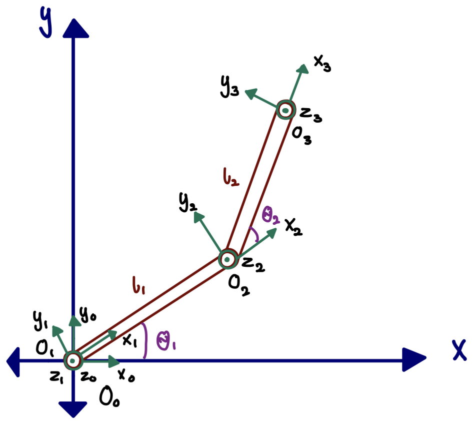
Means of Modeling the Manipulator

To start, classes "Manipulator" and "Link" were created in python which allows the representation of the specifications of manipulators. These classes are later used to generate the data required for the manipulators we will train our models to compute the inverse kinematics for. For an instance of this class to be instantiated, the Denavit-Hartenberg (DH) parameters of the manipulator's links will have to be provided as an input. This allows for the forward kinemtics to be computed in order to find the position of the end effector, given the links' angles. We then created an instant of this class and initiated a manipulator of 2 joints with the following DH parameters, l_1 and l_2 being 5 units long.



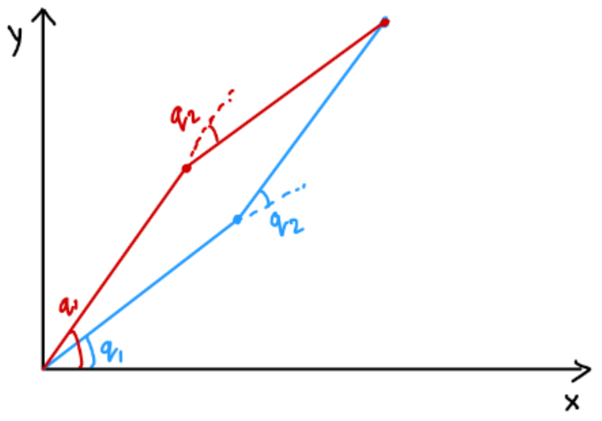
dh parameters

i	α_{i-1}	a_{i-1}	θ_i	d_i
1	0	0	θ_1	0
2	0	0	θ_2	l_1
3	0	0	0	l_2

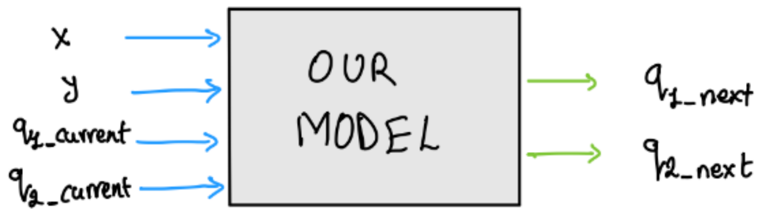


Data Requirement

Given that inverse kinematics yields to an infinite number of solutions, training the model on random points would not enable the development of a neural network capable of achieving seamless transitions between movements. It's essential to use a structured approach to ensure the network learns to navigate these solutions effectively.

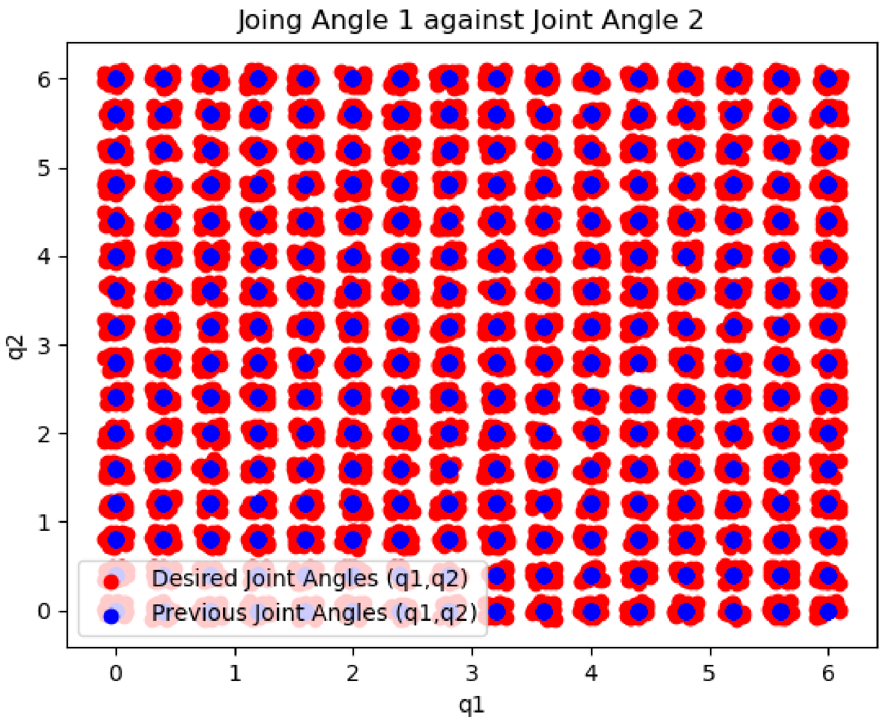


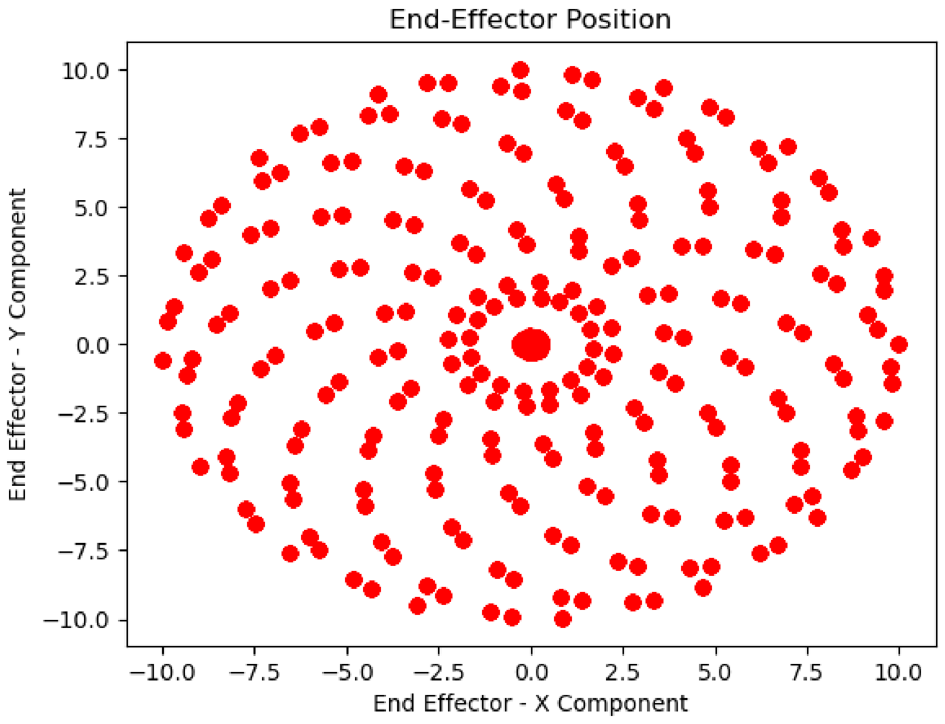
Therefore, having the neural network simply taking the desired (x,y) position of the end effector as an input and output the joint angles required θ_1, θ_2, would obviously result in non-continuous joint angles, making the trajectory tracking task not possible. Therefore, we decided to have our neural network inputs/outputs as follows:



Generating the data

The code generates training data for the manipulator by varying joint angles $q1_{next}$ and $q2_{next}$ with a constant step size of 0.4. For every variation of $q1_{next}$ and $q2_{next}$, we randomly generate 20 random angles +/- a maximum of 0.1 from $q1_{next}$ and $q2_{next}$ respectively and have those as the $q1_{current}$ and $q2_{current}$. We do that in order to have a smooth trajectory and constraint the possible joint angles to a one-to-one relationship. The joint relationship between $q_{current}$ and q_{next} can be seen in the graph below. The random noise generated around every pair of joint angles can be seen. Additionally, the end-effector position distrubution for the genereated data can be also seen in the graph below it. Both the step size of the variation in joint angles as well as the number of random points around each joint angle in this case can be considered a hyper parameter of our data.





Our expectation from the machine learning model is to interpolate the data as if as we had more red points (based on the graph) in which each can be thought of as a sink with respect to the noise around it.

Neural Network Model

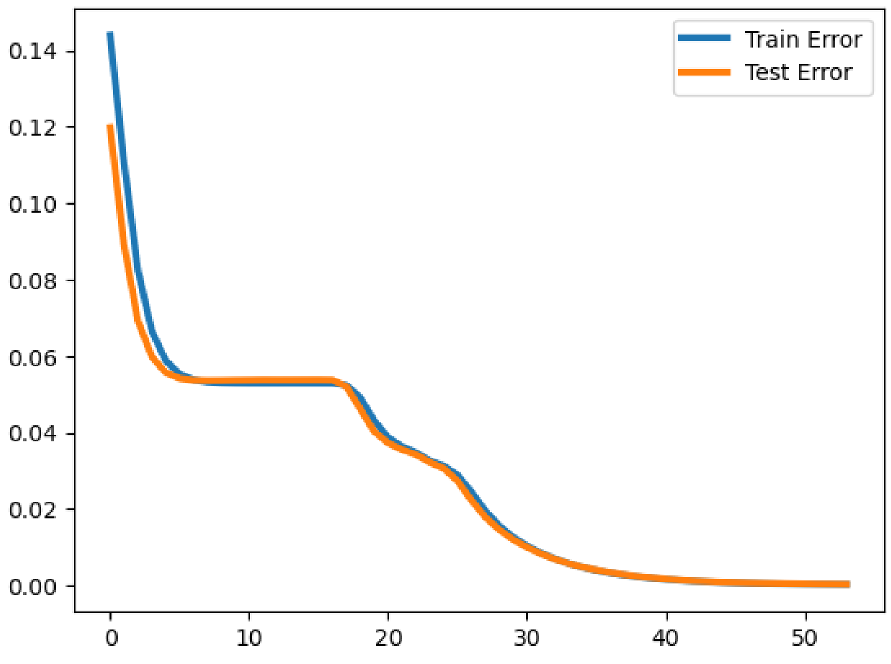
An inverse kinematics (IK) model is defined. It is a simple feedforward neural network with 4 inputs (x position, y position, $q1_{current}$ and $q2_{current}$) several hidden layers, and a 2-unit output layer representing the predicted joint angles

($q1_{next}$ and $q2_{next}$). We have tried and tested different numbers and different dimensions of layers, and eventually what seemed to work best was a Model consisting of five hidden layers and one output layer, with the dimensions: 4x128, 128x64, 64x32, 32x16, 16x8, 8x2.

Training

The neural network is trained using stochastic gradient descent. We experimented with other optimizing algorithms such as Adam Optimizer but SGD seemed to give us better results. The training loop iterates over epochs(200), and batches, and updates the model parameters. The loss is calculated by comparing the model's predictions with the ground truth using Mean Squared Error (MSE). Backward propagation is used and parameters are updated over each epoch. As for the learning rate, we experiemtned with different learning rates and we varied it during th epochs, decreasing it as the epochs increase from 0.01 to 0.001 to 0.0001.

The training and test losses were monitored during the training process. The attached graph illustrates the evolution of these losses over the course of training epochs. The visual representation of the loss graph provides insights into the training dynamics.



As you might have observed from the graph, overfitting is not a concern in the project as overfitting will lead to the right end positions, therefore giving higher accuracy.

3. Simulation (https://orayyan.com/projects/IKNN/simulation)

To test our model in a real example, we exported our model as a (.onnx) which allowed us to run in it in Javascript. We then created a front-end to visualize the trajectory tracking by the 2 joints. The visualization is live and can be tested here https://orayyan.com/projects/IKNN/simulation

The model seemed to have learned to map end-effector positions to joint angles, effectively solving the inverse kinematics problem for the given manipulator. The training loss was very low throughout the training process, indicating successful convergence.

The simulation tool works by creating a unit vector between the current position of the end effector and the desired one (clicked at). It then gives the model the current angles as well as a factor of the unit vector as the end position in order to ensure a smooth trajectory movement.

4. Conclusion

In this project, we focused on developing an inverse kinematics model for a two-joint robotic manipulator. The model was trained using a dataset generated through forward kinematics simulations, capturing the relationships between end-effector positions and joint angles. The training data was perturbed to introduce variability, enhancing the model's ability to find the correct angles to reach the end-effector.

The model learned to map end-effector positions to joint angles, effectively solving the inverse kinematics problem for the given manipulator. The training loss was very low throughout the training process, indicating successful convergence.

We believe we succeeded in achieving our original goal of training a neural network model to carry out inverse kinematics on a 2 joints manipualtor. Had we had more time, we would have tried to analyze whether the neural network was capable of creating more sinks or whether it interpolate the noise regions.