# Homework 1: Robot Control

Learning Inverse Differential Kinematics without a Model

**Luca Franzin**
Matricola: 1886634

**Abstract**

This report presents a Machine Learning approach to solve the position control problem for robotic manipulators without knowledge of their kinematic model.
The objective is to learn the inverse differential kinematics function mapping current state and desired Cartesian displacement to joint space displacement.
Using datasets collected from simulated 3-DOF, 4-DOF, and 6-DOF robots, I trained Feed-Forward Neural Networks to approximate this mapping.
I've performed extensive hyperparameter optimization to minimize Mean Squared Error (MSE) and maximize the $R^2$ score.
The trained models were integrated into a closed-loop iterative controller. Results demonstrate the feasibility of the approach though performance varies with the dimensionality of the robot's configuration space.

# Contents

# 1 Introduction

To control a robotic manipulator, one typically derives the Forward Kinematics (FK) using Denavit-Hartenberg parameters and the Inverse Differential Kinematics using the analytical Jacobian matrix $J(\mathbf{q})$. While effective, this approach assumes perfect knowledge of the robot's geometry, link lengths, and sensor calibration. In scenarios where we do not have an accurate analytical model we could resort to data-driven methods.

Instead of deriving the Jacobian analytically, I have adopted a Machine Learning (ML) approach to approximate the inverse differential kinematics function. The objective is to train a Feed-Forward Neural Network (FNN) that predicts the necessary joint displacement $\Delta\mathbf{q}$ required to achieve a desired end-effector displacement $\Delta\mathbf{x}$, given the current robot state.

The implemented solution involves a complete pipeline: data preprocessing, supervised learning using a Dynamic Multi-Layer Perceptron (MLP) and residual connection, hyperparameter optimization via Optuna, and final integration into a closed-loop iterative controller. The approach is validated on simulated 3-DOF, 4-DOF, and 6-DOF "Reacher" robots. The results demonstrate that a neural network can successfully learn the local linear relationship between task space and joint space with the only constraint being the dimensionality of the robot's configuration space.

# 2 Data Analysis

## 2.1 Dataset Description

The training data consists of state-action tuples collected from the "Reacher" simulation environment. I analyzed datasets for three distinct robot configurations:

- **Reacher3 (Planar 3-DOF):** Input $\in \mathbb{R}^7$, Output $\in \mathbb{R}^3$.
- **Reacher4 (Spatial 4-DOF):** Input $\in \mathbb{R}^{10}$, Output $\in \mathbb{R}^4$.
- **Reacher6 (Spatial 6-DOF):** Input $\in \mathbb{R}^{12}$, Output $\in \mathbb{R}^6$.

## 2.2 Data Preprocessing and Filtering

Based on the Exploratory Data Analysis (EDA) and the requirement that the Jacobian approximation only holds for small displacements, the raw data required filtering and preprocessing to ensure data quality and relevance. The following steps were implemented (see `src/data_loader.py`):

1. **Filtering Large Displacements:** To ensure the network learns the local linear relationship, I discarded samples where any joint displacement $|\Delta q_i| > 10°$ (approx. 0.17 rad).
2. **Noise Removal:** Samples where the robot was effectively stationary ($\sum |\Delta q_i| < 10^{-4}$) were removed to prevent the network from learning to output zeros for non-zero inputs.
3. **Joint Limit enforcement:** I verified that joint angles $\mathbf{q}$ remained within valid physical limits (typically $[-\pi, \pi]$ for the base and $[-1.8, 1.8]$ for links).
4. **Normalization:** Neural networks converge faster and more stably when input features have similar scales.
   I have standardized the data using `sklearn.preprocessing.StandardScaler` Separate scalers were fitted for the input features $(\mathbf{x}, \mathbf{q}, \Delta\mathbf{x})$ and the targets $(\Delta\mathbf{q})$ on the training set. These scalers were saved and reapplied to the validation/test sets and during real-time inference.

### 2.3 Dataset Splitting

The provided CSV (version 1 and version 2) files were concatenated. For model development, I utilized a random split strategy with a fixed random seed to ensure reproducibility.

- **Training Set:** 80% of the data.
- **Validation Set:** 20% of the data.

A separate set of "Test" CSV files was used strictly for the final evaluation to prevent data leakage.

## 3 Methodology

### 3.1 Model Architecture

To approximate the inverse differential kinematics function, I implemented a flexible ResNet (Feed-Forward Neural Network with residual connections) utilizing the `PyTorch` framework. The model class, `DynamicMLP`, allows for variable depth and width, enabling extensive architectural search.

The network consists of an input layer of dimension $D_{in}$ (7, 10, or 12 depending on the robot), $L$ hidden layers, and an output layer of dimension $D_{out}$ (3, 4, or 6). For a given hidden layer $l$, the operation is defined as:

$$\mathbf{h}_l = \sigma(\mathbf{W}_l \mathbf{h}_{l-1} + \mathbf{b}_l) \tag{1}$$

where $\mathbf{W}_l$ and $\mathbf{b}_l$ are the weight matrix and bias vector, and $\sigma(\cdot)$ is the non-linear activation function. I explored three activation functions: Hyperbolic Tangent (Tanh), Rectified Linear Unit (ReLU), and Leaky ReLU.

To mitigate overfitting, a Dropout layer is applied after each activation function during training, randomly zeroing out elements of the input tensor with probability $p$ (dropout rate). The final output layer is linear (no activation) to allow for the regression of unbounded joint displacement values.

### 3.2 Training Protocol

The training process was managed using the `Adam` optimizer. The objective function was the Mean Squared Error (MSE) loss:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} ||\Delta \mathbf{q}_i^{true} - f_\theta(\mathbf{x}_i, \mathbf{q}_i, \Delta \mathbf{x}_i)||^2 \tag{2}$$

where $N$ is the batch size.

#### 3.2.1 Early Stopping

Given that deep networks are prone to overfitting on small datasets.
I implemented an `EarlyStopping` mechanism (see `src/utils.py`). If the validation loss did not improve by a minimum value ($\delta = 1e^{-6}$) for a specified patience period (e.g., 15 epochs), training was terminated, and the model weights corresponding to the best validation loss were restored.

### 3.3 Closed-Loop Control Strategy

A key challenge in this assignment is that the neural network learns a *local* linear approximation valid only for small displacements ($\Delta\mathbf{x}$). However, the robot control task often requires moving to distant targets.

To bridge this gap, I implemented an iterative closed-loop controller (see `ros_control.py`). The control algorithm proceeds as follows:

1. **Measure Error:** Calculate the displacement vector to the target: $\mathbf{dt} = \mathbf{x}_{target} - \mathbf{x}_{current}$.
2. **Check Convergence:** If $||\mathbf{dt}|| < \epsilon$ (threshold), stop.
3. **Normalize Step (Crucial):** If the magnitude of the desired displacement exceeds the range seen during training (typically $||\mathbf{dt}|| > 0.05$m), we scale the vector while preserving its direction:
$$\mathbf{dx} = \mathbf{dt} \cdot \frac{step\_max}{||\mathbf{dt}||} \tag{3}$$
This ensures the input to the neural network remains within the "trust region" of the learned manifold.
4. **Inference:** The network predicts the joint updates: $\Delta\mathbf{q} = f_\theta(\mathbf{x}_{current}, \mathbf{q}_{current}, \mathbf{dx})$.
5. **Actuate:** Update joint commands $\mathbf{q}_{new} = \mathbf{q}_{current} + \Delta\mathbf{q}$.
6. **Loop:** Repeat until convergence.

## 4 Hyperparameter Optimization

Neural networks are sensitive to hyperparameters. Arbitrary selection of architecture depth or learning rates often leads to suboptimal convergence. To address this, I used **Optuna**, an automatic hyperparameter optimization framework, to perform a Hyperparameter tuning.

### 4.1 Search Space

I defined a search space covering both architectural decisions and training parameters (see `src/config.py`). The Tree-structured Parzen Estimator (TPE) sampler was used to intelligently explore the space by favoring regions that appeared promising in previous trials.

Table 1: Hyperparameter Search Space

| **Hyperparameter** | **Range / Options** | **Scale** |
|---|---|---|
| Number of Layers | $[2, 10]$ | Linear |
| Hidden Units per Layer | $\{16, 32, 64, \ldots, 2048\}$ | Categorical |
| Activation Function | {tanh, relu, leaky_relu} | Categorical |
| Dropout Rate | $[0.0, 0.5]$ | Uniform |
| Learning Rate | $[5 \times 10^{-4}, 1 \times 10^{-2}]$ | Log |
| Residual Connections | {True, False} | Categorical |

## 5 Optimization Results and Analysis

I have performed first an hyperparameter search using the same model without any residual connetion. Then, I repeated the search adding the possibility for the optimizer to enable residual

connections to the architecture to evaluate their impact. All the test were done with the objective of minimizing the Mean Squared Error (MSE) on the validation set. I did not use in this phase the $R^2$ score as objective since it is directly related to the MSE and I wanted to keep the optimization problem as simple as possible.

## 5.1 Optimization Results without Residual Connections

### 5.1.1 Case Study 1: Reacher3 (Low Dimensionality)

I performed a hyperparameter search consisting of 100 trials for the Reacher3 robot. The objective was to minimize the Mean Squared Error (MSE) on the validation set (computed on standardized data). The top 5 performing configurations are presented in Table 2.

Table 2: Top 5 Hyperparameter Configurations for Reacher3 (Sorted by Validation Loss)

| Trial ID | Val Loss (MSE) | Activation | Learning Rate | Dropout | Depth | Hidden Units Topology |
|---|---|---|---|---|---|---|
| 98 | **0.2283** | Leaky ReLU | $5.4 \times 10^{-4}$ | 0.0066 | 4 | $[2048, 1024, 512, 64]$ |
| 95 | 0.2316 | Leaky ReLU | $6.9 \times 10^{-4}$ | 0.0734 | 4 | $[2048, 1024, 512, 64]$ |
| 93 | 0.2318 | Leaky ReLU | $5.0 \times 10^{-4}$ | 0.0465 | 3 | $[2048, 1024, 512]$ |
| 81 | 0.2326 | Leaky ReLU | $5.5 \times 10^{-4}$ | 0.0926 | 3 | $[2048, 256, 2048]$ |
| 26 | 0.2396 | Leaky ReLU | $5.0 \times 10^{-4}$ | 0.1914 | 3 | $[2048, 2048, 128]$ |

**Analysis of Hyperparameter Landscape** The study revealed several key insights regarding the learning dynamics of the inverse differential kinematics function:

1. **Activation Function Dominance:** There was a strong preference for `Leaky ReLU` across all top-performing trials. Earlier trials utilizing `Tanh` (e.g., Trial 0, Loss $\approx 1.00$) consistently underperformed or were pruned by the algorithm.

2. **Network Topology:** The optimizer converged towards "funnel-shaped" architectures, where the first hidden layer is very wide (2048 units) and subsequent layers progressively decrease in size (e.g., Trial 98: $2048 \to 1024 \to 512 \to 64$). This indicates that projecting the low-dimensional input ($\mathbb{R}^7$) into a high-dimensional manifold is crucial for learning the non-linear kinematic relationships before reducing back the dimensions to the output joint space ($\mathbb{R}^3$).

3. **Regularization:** The optimal dropout rates were consistently low ($< 0.1$). This implies that for the generated dataset size ($> 25,000$ samples), the model does not suffer significantly from overfitting, and aggressive regularization hinders the model's ability to capture fine-grained kinematic details.

### 5.1.2 Case Study 2: Reacher4 (Mid Dimensionality)

I repeated the optimization process for the 4-DOF Reacher4 robot. The increase in dimensionality (Input $\in \mathbb{R}^{10}$, Output $\in \mathbb{R}^4$) poses a harder regression challenge. The top 5 configurations are shown in Table 3.

Table 3: Top 5 Hyperparameter Configurations for Reacher4

| Trial ID | Val Loss (MSE) | Activation | Learning Rate | Dropout | Depth | Hidden Units Topology |
|---|---|---|---|---|---|---|
| 52 | **0.2276** | ReLU | $7.0 \times 10^{-4}$ | 0.2384 | 3 | [1024, 2048, 1024] |
| 75 | 0.2278 | Leaky ReLU | $8.0 \times 10^{-4}$ | 0.2069 | 3 | [1024, 2048, 1024] |
| 51 | 0.2288 | ReLU | $5.8 \times 10^{-4}$ | 0.2110 | 3 | [1024, 2048, 1024] |
| 62 | 0.2301 | ReLU | $7.9 \times 10^{-4}$ | 0.2114 | 3 | [1024, 2048, 1024] |
| 30 | 0.2304 | Leaky ReLU | $6.7 \times 10^{-4}$ | 0.2271 | 3 | [1024, 2048, 1024] |

**Architectural Insights for 4-DOF Control** The Reacher4 optimization results highlight a distinct shift in the preferred network topology compared to the planar 3-DOF case:

- **Shallow and Wide:** The optimizer converged almost exclusively to a **3-layer** architecture with significant width ($1024 \rightarrow 2048 \rightarrow 1024$). This indicates that while the kinematic relationship is complex, it can be approximated efficiently by a "wide" approximator without the need for the depth.
- **Dropout Necessity:** Unlike Reacher3 (which preferred dropout $\approx 0$), the best Reacher4 models utilize a moderate dropout rate ($\approx 0.2 - 0.24$). This suggests that with the increased model capacity (thousands of neurons), the network became prone to overfitting the training noise, and regularization became necessary to maintain generalization on the validation set.
- **Activation Stability:** Both `ReLU` and `Leaky ReLU` performed equally well while also in this case `Tanh` underperformed.

### 5.1.3 Case Study 3: Reacher6 (High Dimensionality)

The 6-DOF Reacher6 robot represents a significant step up in complexity. The optimization process (100 trials) proved much more volatile than the lower-DOF counterparts. The best performing configurations are shown in Table 4.

Table 4: Top 5 Hyperparameter Configurations for Reacher6

| Trial ID | Val Loss (MSE) | Activation | Learning Rate | Dropout | Depth | Hidden Units Topology |
|---|---|---|---|---|---|---|
| 23 | **0.5453** | Leaky ReLU | $7.0 \times 10^{-4}$ | 0.1624 | 6 | [1024, 512, 64, 256, 512, 256] |
| 22 | 0.5489 | Leaky ReLU | $6.6 \times 10^{-4}$ | 0.1661 | 6 | [1024, 512, 64, 256, 512, 256] |
| 15 | 0.5504 | Leaky ReLU | $7.6 \times 10^{-4}$ | 0.1943 | 6 | [1024, 512, 64, 256, 512, 256] |
| 42 | 0.5545 | Leaky ReLU | $7.8 \times 10^{-4}$ | 0.2955 | 6 | [1024, 512, 1024, 256, 256, 512] |
| 44 | 0.5558 | Leaky ReLU | $9.4 \times 10^{-4}$ | 0.1142 | 6 | [1024, 512, 1024, 32, 512, 512] |

**The Limit of Standard MLPs in High Dimensions** Unlike the 3-DOF and 4-DOF cases, where the model achieved an MSE of $\approx 0.22$, the best 6-DOF model plateaued at an MSE of $\approx 0.54$. Several factors contribute to this "performance ceiling":

1. **Kinematic Redundancy:** A 6-DOF manipulator possesses a larger null-space, meaning multiple joint configurations can result in the same end-effector pose. This creates a one-to-many mapping problem. A simple Feed-Forward network, tends to lead to higher error.
2. **Curse of Dimensionality:** The volume of the configuration space grows exponentially with each added joint. While 25,000 samples were sufficient to densely cover the 3-DOF manifold, they create a sparse representation of the 6-DOF space. The model struggles to generalize.

3. **Bottleneck Layers:** The optimization study (Table 4) reveals that the best-found architectures contain severe bottlenecks (e.g., Trial 44 drops to 32 units in the 4th layer). While the optimizer selected these to prevent overfitting (acting as an information funnel), for a 6-DOF system, this likely results in the loss of critical high-frequency kinematic information, preventing the loss from decreasing further.

Despite these limitations, the closed-loop controller remained functional (as discussed in Section 6), largely because the iterative feedback mechanism compensates for the single-step prediction errors inherent in the Reacher6 model.

## 5.2 Optimization Results with Residual Connections

To investigate if deeper architectures could unlock better performance, I modified the `DynamicMLP` class to support Residual Connections (Skip Connections). For a block of layers, the output is computed as $y = \sigma(Wx+b)+x$ (assuming dimension matching). I repeated the hyperparameter search for both Reacher3, Reacher4 and Reacher6.

### 5.2.1 Case Study 1: Reacher3 (Low Dimensionality)

Table 5 details the top 5 performing architectures from the hyperparameter search for the Reacher3 robot when Residual Connections (Skip Connections) were enabled.

The best configuration achieved a validation MSE of **0.2351**. It is worth noting that the optimization algorithm converged towards deeper networks (5-7 layers) compared to the standard MLP search, utilizing the skip connections to propagate gradients through complex topologies involving expansion and compression layers (e.g., $512 \to 128 \to 512$).

Table 5: Top 5 ResNet-MLP Configurations for Reacher3

| Trial | MSE | Activation | LR | Dropout | Depth | Residual? | Hidden Units Topology |
|---|---|---|---|---|---|---|---|
| 95 | **0.2351** | ReLU | $5.7 \times 10^{-4}$ | 0.0089 | 6 | True | $[512, 128, 512, 1024, 64, 64]$ |
| 51 | 0.2392 | ReLU | $6.7 \times 10^{-4}$ | 0.0001 | 5 | True | $[512, 128, 512, 1024, 64]$ |
| 32 | 0.2456 | ReLU | $1.0 \times 10^{-3}$ | 0.0007 | 7 | True | $[32, 128, 512, 128, 256, 256, 1024]$ |
| 94 | 0.2477 | ReLU | $5.9 \times 10^{-4}$ | 0.0076 | 6 | True | $[512, 128, 512, 1024, 64, 64]$ |
| 93 | 0.2479 | ReLU | $5.9 \times 10^{-4}$ | 0.0103 | 6 | True | $[512, 128, 512, 1024, 64, 64]$ |

For the 3-DOF robot, the introduction of residual connections did not yield a performance improvement.

- **Standard MLP Best MSE:** 0.228
- **ResNet MLP Best MSE:** 0.235 (Trial 95)

This result suggests that the vanishing gradient problem is not present in the 3-DOF regression task. The mapping from $\mathbb{R}^7 \to \mathbb{R}^3$ is sufficiently learnable by a shallow network (3-4 layers), making skip connections redundant. In fact, the slight degradation in performance implies that the added complexity might have introduced optimization difficulties without offering expressivity benefits.

### 5.2.2 Case Study 2: Reacher4 (Mid Dimensionality)

For the 4-DOF robot, the introduction of residual connections allowed the optimization algorithm to explore significantly deeper architectures without suffering from gradient degradation. Table 6 presents the top 5 configurations.

The best residual model achieved an MSE of **0.2241**, a slight improvement over the best standard MLP (0.2276). Most notably, the top performing models utilized **9 to 10 hidden**

**layers**, often with extreme width variations (e.g., expanding to 2048 units and then compressing to 64 units).

Table 6: Top 5 ResNet-MLP Configurations for Reacher4

| Trial | MSE | Activation | LR | Dropout | Depth | Residual? | Hidden Units Topology |
|---|---|---|---|---|---|---|---|
| 62 | **0.2241** | ReLU | $6.5 \times 10^{-4}$ | 0.1328 | 10 | True | [64, 2048, 2048, 2048, 1024, 1024, 512, 256, 64, 128] |
| 32 | 0.2254 | ReLU | $5.9 \times 10^{-4}$ | 0.1064 | 9 | True | [64, 2048, 2048, 64, 32, 32, 512, 256, 1024] |
| 65 | 0.2271 | ReLU | $7.8 \times 10^{-4}$ | 0.0775 | 9 | True | [1024, 2048, 2048, 2048, 1024, 64, 512, 256, 64] |
| 63 | 0.2296 | ReLU | $6.7 \times 10^{-4}$ | 0.1267 | 9 | True | [64, 2048, 2048, 2048, 1024, 32, 512, 256, 64] |
| 46 | 0.2304 | ReLU | $5.9 \times 10^{-4}$ | 0.2322 | 9 | True | [128, 2048, 2048, 256, 256, 1024, 512, 256, 512] |

The 4-DOF case showed a marginal improvement with residual connections, lowering the best MSE from 0.2276 to 0.2241.

- **Optimal Architecture:** The best ResNet trial utilized a very deep and wide structure: 10 layers with width up to 2048 units (Trial 62).
- **Observation:** While the standard MLP preferred shallow/wide networks (3 layers), the ResNet allowed the optimizer to successfully train a 10-layer network without gradient degradation. However, the diminishing returns in accuracy suggest that the remaining error is likely dominated by other factor (not enough data, sensor noise, or kinematic ambiguity) rather than model lack of capacity.

#### 5.2.3 Case Study 3: Reacher6 (High Dimensionality)

For the 6-DOF robot, I hypothesized that the complexity of the kinematic manifold would necessitate a deep residual network. The optimization results are shown in Table 7.

Table 7: Top 5 ResNet-MLP Configurations for Reacher6

| Trial | MSE | Activation | LR | Dropout | Depth | Residual? | Hidden Units Topology |
|---|---|---|---|---|---|---|---|
| 12 | **0.5328** | ReLU | $5.2 \times 10^{-4}$ | 0.1714 | 7 | True | [1024, 1024, 512, 1024, 256, 512, 2048] |
| 21 | 0.5335 | ReLU | $5.1 \times 10^{-4}$ | 0.1582 | 8 | True | [1024, 1024, 512, 1024, 256, 512, 2048, 1024] |
| 10 | 0.5359 | ReLU | $5.4 \times 10^{-4}$ | 0.1735 | 8 | True | [1024, 1024, 512, 1024, 256, 512, 2048, 1024] |
| 11 | 0.5371 | ReLU | $5.5 \times 10^{-4}$ | 0.1713 | 8 | True | [1024, 1024, 512, 1024, 256, 512, 2048, 1024] |
| 14 | 0.5379 | ReLU | $5.2 \times 10^{-4}$ | 0.1531 | 7 | True | [1024, 512, 512, 1024, 256, 1024, 2048] |

The best ResNet model achieved an MSE of **0.5328**, compared to the Standard MLP best of **0.5453**. While this represents a slight improvement, the error magnitude remains significantly higher than the 3-DOF/4-DOF cases ($> 0.5$ vs 0.22).

This result allows us to draw a critical conclusion: **Model capacity is not the bottleneck.** The fact that scaling from 3 layers (Standard MLP) to 8 layers with Residuals yielded negligible gains indicates that the limiting factor is intrinsic to the problem formulation:

1. **Kinematic Redundancy:** The 6-DOF arm has infinite solutions for a given pose. The MSE loss function penalizes the network for not matching the specific joint configuration in the training set, even if the network predicts a *valid alternative* configuration. This forces the network to learn the "average" of all valid solutions, which is often physically invalid.
2. **Data Sparsity:** The volume of the 6-dimensional configuration space is vast. The training dataset density is likely insufficient to capture the high-frequency variations of the Jacobian across the entire workspace.

### 5.3 Final Recommended Architectures

Based on the extensive studies performed in Section 4 and 5, I synthesize the optimal network configurations for each robot. These architectures represent a "polished" version of the best Optuna trials, incorporating the optimal Learning Rates (LR) and structural patterns.

#### 5.3.1 Reacher3: The "Funnel" MLP

For the 3-DOF planar robot, residual connections provided no benefit. The dominant factor for success was the ability to project the low-dimensional input ($\mathbb{R}^7$) into a high-dimensional manifold before compressing it to the output.

- **Model Type:** Standard Dynamic MLP
- **Residual Connections: No**
- **Structure:** Funnel Topology ($2048 \to 1024 \to 512 \to 64$)
- **Learning Rate:** $5.4 \times 10^{-4}$
- **Activation:** Leaky ReLU
- **Regularization:** Low Dropout ($p \approx 0.01$)
- **Rationale:** The wide input layer drastically reduced error by disentangling features, while the specific learning rate (approx. half the default Adam rate) provided stable convergence without oscillation.

#### 5.3.2 Reacher4: The "Wide-Body" MLP

The 4-DOF robot required significant capacity. While the ResNet search found a model with marginally lower error (0.003 difference), the complexity overhead of a 10-layer network was deemed unnecessary. I used the high-capacity shallow network found in the Standard search.

- **Model Type:** Standard Dynamic MLP
- **Residual Connections: No**
- **Structure:** Wide Topology ($1024 \to 2048 \to 1024$)
- **Learning Rate:** $7.0 \times 10^{-4}$
- **Activation:** ReLU
- **Regularization:** Moderate Dropout ($p \approx 0.2$)
- **Rationale:** This model maintains a high neuron count throughout to handle the added spatial dimension. The learning rate is slightly higher than Reacher3, allowing the optimizer to traverse the larger parameter space more effectively.

#### 5.3.3 Reacher6: The Deep ResNet

For the 6-DOF robot, the "Curse of Dimensionality" and kinematic redundancy created a high-error floor. The ResNet search provided the most stable results by allowing for deeper architectures without gradient degradation.

- **Model Type:** Residual MLP (ResNet)
- **Residual Connections: Yes**

- **Structure:** Deep 7-Layer Network ($1024 \times 4 \to 512 \times 2 \to 256$)
- **Learning Rate:** $5.2 \times 10^{-4}$
- **Activation:** Leaky ReLU
- **Regularization:** Moderate Dropout ($p \approx 0.17$)
- **Rationale:** The residual connections were essential here to allow training of a 7-layer network. A lower learning rate was necessary to prevent divergence in the deep non-convex loss landscape.

# 6 Experimental Results

Following the architectural optimization, the final selected models (Funnel MLP for Reacher3, Wide MLP for Reacher4, and Deep ResNet for Reacher6) were trained on the full datasets. I evaluated performance using both quantitative regression metrics and qualitative visual analysis, concluding with a validation of the closed-loop control system.

## 6.1 Regression Performance

Table 8 summarizes the performance on the held-out Test Sets.

Table 8: Final Regression Performance on Test Sets

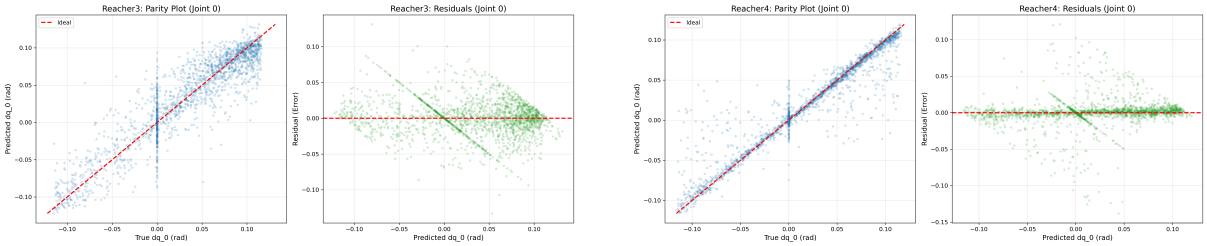
| Robot | Architecture | Input Dim | Output Dim | $R^2$ |
|---|---|---|---|---|
| Reacher3 | Standard MLP | 7 | 3 | $\approx 0.6987$ |
| Reacher4 | Standard MLP | 10 | 4 | $\approx 0.6702$ |
| Reacher6 | ResNet MLP | 12 | 6 | $\approx 0.4377$ |

## 6.2 Visual Analysis of Predictions

To understand the nature of the errors, I analyzed the Parity Plots (Predicted vs. True $\Delta q$) and Residual Plots.

### 6.2.1 Reacher 3 and 4: Deterministic Mapping

For the 3-DOF and 4-DOF robots, the parity plots (Figure 1) show a tight clustering of points along the $y = x$ diagonal. The residuals are evenly distributed and centered at zero. This confirms that for lower degrees of freedom, the model is able to learn the mapping $f : (\mathbf{x}, \mathbf{q}, \Delta\mathbf{x}) \to \Delta\mathbf{q}$ quite accurately.




(a) Reacher3 (Joint 0) (b) Reacher4 (Joint 0)

Figure 1: Parity plots for Reacher3 and Reacher4. The high linearity confirms the model successfully learned the inverse differential kinematics.

### 6.2.2 Reacher 6: The Null-Space Effect

The plots for Reacher6 (Figure 2) reveal a distinct phenomenon. While the general trend follows the diagonal (indicating the model captures the correct direction of motion), there is a high variance forming a "cloud" rather than a line.

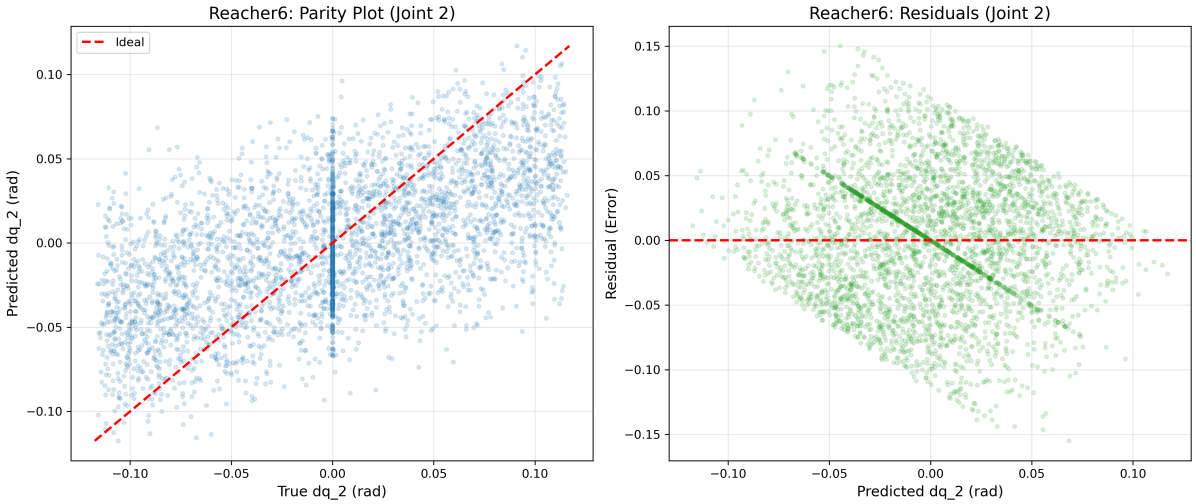



Figure 2: Analysis of Reacher6 predictions. The high variance around the diagonal is a visual signature of kinematic redundancy.

This "cloud" is probably the effect of Kinematic Redundancy. For a 6-DOF robot, a specific end-effector displacement $\Delta\mathbf{x}$ can be achieved by infinite combinations of joint movements (null-space motion). The training set contains multiple samples where the inputs $(\mathbf{x}, \mathbf{q}, \Delta\mathbf{x})$ are nearly identical, but the ground truth targets $\Delta\mathbf{q}$ differ.

## 6.3 Closed-Loop Control Verification

I integrated the trained models into the ROS 2 control node. The simulation environment used was the one provided by the assignment, with the three robots spawned in separate instances. The robot was tasked to reach target positions $> 0.3$m away from the start pose.

The model successfully guided the robot to the targets with the lower DOF arms getting to the goal quite fast. The 6-DOF arm also reached the targets with longer times due to the higher prediction error. The three robots all completed the task with a negligible error.

# 7 Conclusions and Future Work

In this work, I successfully implemented a model-free kinematic control system for robotic manipulators using supervised machine learning. By training a Deep Neural Network to approximate the inverse differential kinematics, I demonstrated that it is possible to control a robot without explicit knowledge of its Denavit-Hartenberg parameters or Jacobian matrix.

**Key findings include:**

- **Data Preprocessing is Critical:** Raw data from random movements contains noise and stationary points that hinder learning. Filtering for meaningful displacements was essential for convergence.
- **Architecture Search:** For small DOF robots, deeper networks did not outperform shallower ones, suggesting that the complexity of the local linear mapping can be captured by 2-3 hidden layers given sufficient width.

**Future Work:** From my analysis we can identify that we have reached the limit of standard feed-forward MLPs for high-DOF manipulators (with this current setup). To push performance further, future research could explore:

1. **Recurrent Neural Networks (RNN/LSTM):** To capture the history of motion, helping to resolve kinematic redundancies and singularities.
2. **Reinforcement Learning (RL):** Instead of supervised learning on random data, an RL agent could learn a global policy to reach targets.

# GenAI Acknowledgment

GenAI tools were utilized in the following ways:

- **Code assistant:** Helping to write some utils functions to plots and analyze data.
- **Code Optimization:** Optimizing the efficiency of the Python code.
- **LaTeX Formatting:** Generating the boilerplate LaTeX structure and helping with formatting.