SAPIENZA
UNIVERSITÀ DI ROMA

Artificial Intelligence and Robotics

Homework 1 - Robot kinematics

Machine Learning

Student:
Giovanni Zara
1929181

Academic Year 2024/2025

Contents

1 Introduction

The study of robot kinematics represents a fundamental aspect of robotics. Geometry enables the study of the movement of a multi-degree of freedom kinematic chain, allowing the calculation of the position and orientation of a robot's end-effector. The term 'forward kinematics' is used to describe the process of calculating the position and orientation of an end-effector, starting from the configuration of its joints. Conversely, the term 'inverse kinematics' is used to describe the problem of determining the required joint configurations to achieve a desired end-effector pose. This project is concerned with the solution of forward kinematics for three distinct robot models, each exhibiting a varying degree of freedom (DOF): a 2-DOF planar robot (R2), a 3-DOF planar robot (R3), and a 5-DOF spatial robot (R5).

The objective of this study is to design and evaluate machine learning models for the purpose of learning forward kinematics functions from simulated datasets. In order to achieve this, techniques such as hyperparameter optimisation and model evaluation will be employed, in addition to numerical differentiation in order to compute Jacobian matrices for each problem. Subsequently, the Jacobians are compared to the analytical ones derived from robot-specific kinematics equations in order to assess the fidelity of the learned models.
The workflow for each robot is as follows:

The initial stage of the data analysis process is exploratory data analysis (EDA). An investigation of the structure and distribution of the datasets, which include joint angles, fingertip positions and orientations. The next stage is the development of the models. Training supervised regression models that are tailored to the specific kinematic complexity of each robot, ranging from simple feedforward neural networks to ensemble learning techniques or deeper neural network models. Performance evaluation is conducted to assess the efficacy of the developed models, and predictive accuracy is iteratively improved by tuning hyperparameters. Each model then passes through a second improvement phase to enhance its performance further, with modifications made to the model architecture in order to optimise its suitability for the specific problem at hand. Finally, the Jacobians of the learned forward kinematics models are numerically estimated and compared with the analytically derived counterparts, in order to provide a more detailed analysis of the models' performance.

1.1 Dataset

The dataset employed for this analysis was generated with "Mujoco", a physics engine that enables the creation of sophisticated simulations, which are useful, for instance, in the field of robotics. A variety of simulations were conducted for each robot, with the number of samples simulated and the random seed utilized for data generation specified in each case. In order to guarantee both better generalisation and a better fit for each model, different seeds and sample numbers were used for each robot configuration. The two distinct seeds were employed to partition the training and testing datasets, generating them with wholly disparate seeds to optimise model generalisation. Each dataset contains the joint angles, directional sin and cosines, the fingertip position and the fingertip orientation.

2 R2 robot

The R2 robot is a planar manipulator comprising two revolute joints and a two-link structure. The simplified model exhibits relatively straightforward forward kinematics equations, which should facilitate a satisfactory analysis even with a relatively simple model. The dataset for the R2 robot comprises two input features, namely joint angles j0 and j1, and four output variables, namely the corresponding end-effector position and orientation.

2.1 EDA

The robot workspace demonstrates the position of the fingertip in Cartesian space and its associated density. This indicates that the dataset is significantly more dense in the centre and in proximity to the border, which suggests that the model will probably predict those positions better.

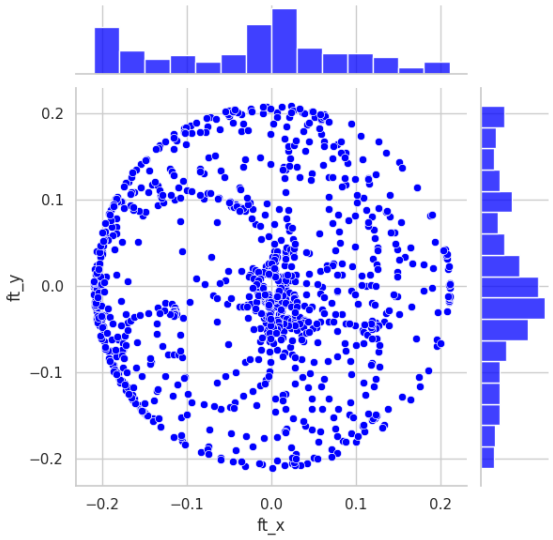

Figure 1: r2 workspace density

In this instance, the model has been trained using a relatively modest subset of the available data, comprising 1,000 samples. Given the relatively straightforward configuration, my objective was to achieve satisfactory results using a modest number of samples in order to present a more challenging problem.
As is the case with all other robot configurations, both data seeds have been utilised. The same seed has been employed to split the features and the labels for the training set, while a different seed has been used to split the features and labels for the test set. This approach is intended to avoid the model overfitting on the training data and to maximise generalisation.

2.2 Model

A feedforward neural network (NN) was deemed an appropriate choice of modelling technique. The initial model will be a linear simple model with no middle layers (which approximates a linear regression) that will most likely perform poorly, and then a more complete one will be created to demonstrate the differences between the two. The objective of the model is to learn a mapping from joint angles to the fingertip position (x, y) and orientation (quaternion values). In both models, six distinct combinations of learning rate and epochs will be searched while the optimiser and loss function are held constant.

The optimiser that will be employed is Adam, a stochastic gradient descent method based on the adaptive estimation of first-order and second-order moments. Computationally efficient and requiring minimal memory, Adam is a commonly used method that has demonstrated efficacy in a range of applications. It therefore seemed an appropriate choice for this type of problem.

With regard to the loss function, the mean squared error (MSE) will be employed in all subsequent analyses. The mean squared error (MSE) is a statistical measure that quantifies the average of the squares of the errors, or the average squared difference between the estimated values and the actual values.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

The squared error function accentuates significant discrepancies, thereby rendering the mean square error (MSE) susceptible to the influence of outliers. This is advantageous in regression tasks where substantial errors must be reduced. The MSE function is continuous and differentiable, rendering it appropriate for optimisation algorithms such as gradient descent (given the use of Adam). By reducing MSE, the model strives to diminish bias and variance, thereby establishing a state of equilibrium between underfitting and overfitting. The MSE is particularly well-suited to regression analysis, as it is designed to accommodate continuous output data.

2.3 First application

The initial Keras model comprises an input layer with a feature size matching that of the training data, followed by a dense layer with neurons equal to the number of output variables. A linear activation function is employed, and the model is compiled with the Adam optimiser and mean squared error loss. Subsequently, the model was wrapped in a KerasRegressor, enabling the utilisation of the GridSearchCV function to identify the optimal combination of hyperparameters (thanks to SciKeras). This function, derived from the scikitlearn library, performs a grid search on a set of parameters while conducting cross-validation. A grid search was conducted on learning rates(0.001, 0.01, 0.1) and epochs(20, 30, 50) with the objective of determining the most effective combination of parameters. The search was cross-validated with five splits.
The model was then tested to obtain a test loss.

As anticipated, the performance of the linear regression model was suboptimal. The lowest mean square error (MSE) value observed in the training data was 0.1, while the test loss was 0.2, indicating that the model's predictions were not accurate. This can be retraced to the fact that this model basically tries to infer the spatial coordinates of the robot just using a line. The linear regression pattern does not match with the data distribution, resulting in very poor prediction capabilities.

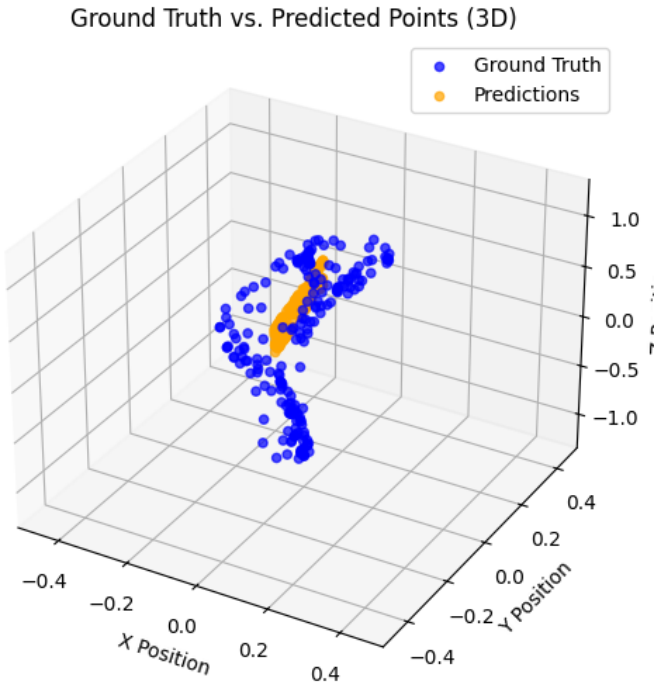

Figure 2: linear regression prediction vs ground truth

2.4 Second application - enhancing performances

In order to optimise the model performance, modifications were implemented to the neural network architecture. In order to more accurately represent the non-linear distribution of the data, a single hidden layer was added, activated with the ReLu function. The Rectified Linear Unit (ReLu) is an activation function that transforms the input by outputting the input itself if it is positive, and zero if the input is negative.

Mathematically, it's defined as:

$$\text{ReLU}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

For any positive input, ReLU simply passes the value through unchanged. For any negative input, it outputs zero. The implementation of this function within the model facilitates the learning of complex relationships by the neural networks. The introduction of non-linearity enables the model to comprehend more intricate data patterns.
All other elements within the model structure were maintained in their original form. A further GridSearchCV was conducted on a total of 45 fits, resulting in 50 epochs and a learning rate of 0.01 being identified as the optimal values.

Figure 3: r2 learning curves

An examination of the model learning curves reveals a convergence of the losses as the epochs progress, ultimately reaching a relatively stable and minimal value. This demonstrates that the model learns in a structured and effective way, with no spikes or oscillations.

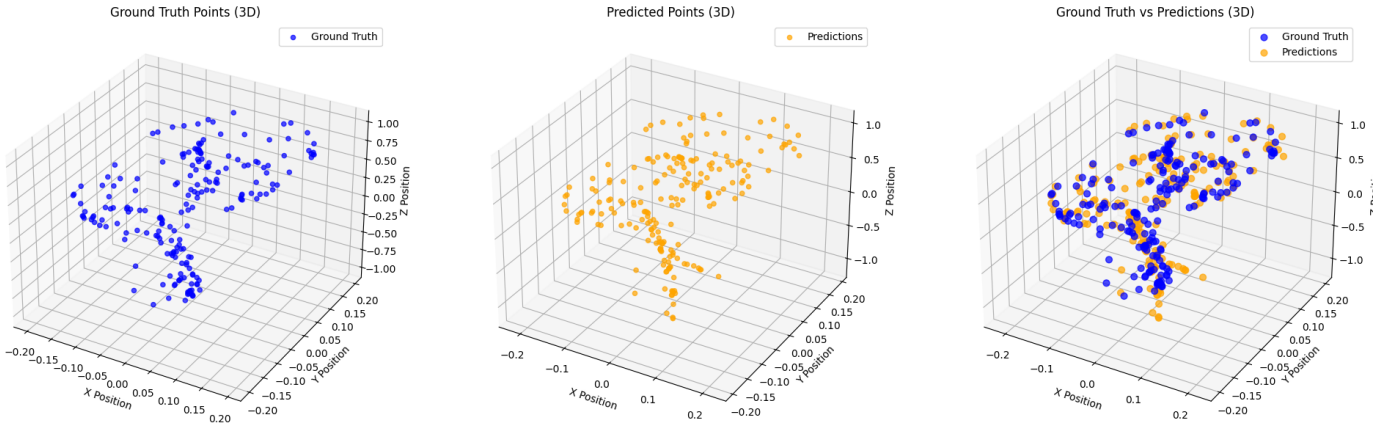

Figure 4: ground truth vs prediction

It is evident that the ground truth vs prediction plot has undergone a notable improvement, exhibiting a striking resemblance between the predicted and actual values.

2.5 Jacobians computation

The Jacobians of the r2 robot are now being calculated. In order to calculate the difference between the learned and analytical Jacobians, it is necessary to compute both. This will provide an additional metric for evaluating the performance of the model predictions. In order to compute the learned Jacobian, it is first necessary to calculate the forward kinematics of the 2DOF problem with the model in question. A forward kinematics (FK) function takes the model and input theta, reshapes theta to a batch of size 1, computes the model on theta, and reshapes the output to a one-dimensional vector of size 4. The FK Jacobian function computes the Jacobian by using TensorFlow's GradientTape to record the operations.

The function monitors the input variable x and calculates the Jacobian of the output variable y (derived from the forward kinematics function) with respect to the input variable x.

This learned Jacobian is confronted with the analytical jacobian corresponding to the 2DOF forward kinematics problem, which is calculated as follows:

$$x = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2)$$
$$y = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2)$$

To compute the Jacobian, we need to take the partial derivatives of x and y with respect to the joint angles θ_1 and θ_2.

$$\frac{\partial x}{\partial \theta_1} = -L_1 \sin(\theta_1) - L_2 \sin(\theta_1 + \theta_2)$$
$$\frac{\partial x}{\partial \theta_2} = -L_2 \sin(\theta_1 + \theta_2)$$
$$\frac{\partial y}{\partial \theta_1} = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2)$$
$$\frac{\partial y}{\partial \theta_2} = L_2 \cos(\theta_1 + \theta_2)$$

The Jacobian matrix J is:

$$J = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} \end{bmatrix}$$

Substituting the computed derivatives:

$$J = \begin{bmatrix} -L_1 \sin(\theta_1) - L_2 \sin(\theta_1 + \theta_2) & -L_2 \sin(\theta_1 + \theta_2) \\ L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2) & L_2 \cos(\theta_1 + \theta_2) \end{bmatrix}$$

In order to compute the difference between the Jacobians, the Frobenius norm was employed, which is a matrix norm that measures the magnitude of a matrix. It is defined as the square root of the sum of the absolute squares of its elements:

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

In other words, it is similar to the Euclidean norm.
The Jacobians were calculated on approximately 200 samples of the test dataset (to avoid using the samples the model trained on) and then the difference between the norms was computed. Subsequently, the mean of these differences was calculated in order to obtain a more reliable value.

The mean of all the differences was found to be **0.078**, which is a very small value. This further supports the assertion that the model demonstrates very good predictive capabilities.

3 R3 robot

The R3 robot is composed of three revolute joints and a three-link structure. Although this model is not particularly complex, it exhibits a greater degree of articulation with respect to the R2 model. The forward kinematics equations remain relatively straightforward, but they are less simple than those of the previous model. This could necessitate the use of a larger number of samples for the model to function correctly. The dataset for the R3 robot comprises three input features, namely joint angles j0, j1 and j2, and four output variables, namely the corresponding end-effector position and orientation ft_x, ft_y, ft_qw and ft_qz. In this analysis, the learning process will be examined on both the smaller (1k) and larger (10k) datasets to identify any differences in the model learning.

3.1 EDA

The robot workspace and Cartesian density are presented below. This configuration appears to be more balanced than the r2 one in terms of data distribution density, although it does exhibit some clustering.

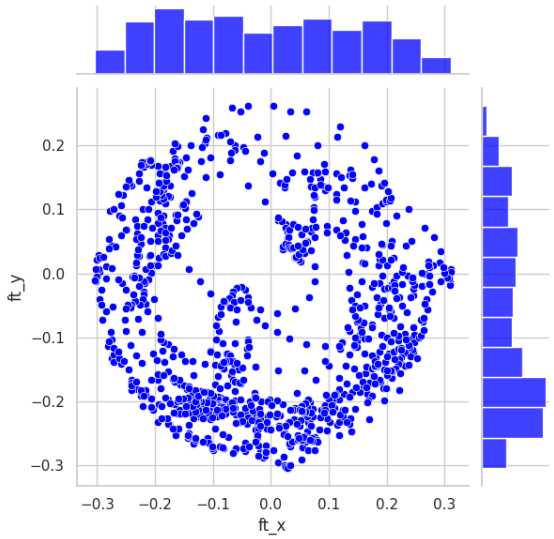

Figure 5: r3 workspace density

Again, both data seeds[1] have been utilized to have different seeds for train and test datas.

3.2 Model

In light of the unsatisfactory outcomes observed in the linear regression trial with R2 configuration, an alternative approach was considered for this model. The objective was to enhance the efficacy of the regression analysis by integrating an ensemble of linear regression models with SciKitLearn AdaBoost. An AdaBoost regressor is a meta-estimator that begins by fitting a regressor on the original dataset. Subsequently, additional copies of the regressor are fitted on the same dataset, with the weights of instances adjusted according to the error of the current prediction. Consequently, subsequent regressors concentrate on cases that are more challenging. Upon analysis of the R2 robot, it became evident that the linear regression model was unable to adequately approximate the problem. Consequently, an attempt will be made to assess the performance of a regression ensemble. In order to identify and capture non-linear features more effectively, I intend to utilise support vector machine regressors (SVR).

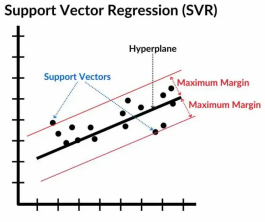

Figure 6: SVR

Support Vector Regressor (SVR) is a machine learning algorithm derived from Support Vector Machines (SVM) and designed for regression tasks. It works by finding a function f(x) that predicts outcomes while allowing for a margin of tolerance ϵ, within which deviations from the actual data points are not penalized (ϵ tube). SVR seeks to maintain a balance between model simplicity (flatness of the function) and predictive accuracy.

In order to achieve this, SVR minimises the deviations that exceed ϵ by introducing slack variables, thereby ensuring that the algorithm is robust to outliers. Furthermore, it employs kernel functions (the kernel trick), including linear, polynomial, and radial basis functions (RBF), to capture non-linear relationships by mapping data into higher dimensional spaces. This flexibility enables SVR to perform well in a variety of regression scenarios while exhibiting particular resilience to noise in the data.
SVR formula:

$$\text{Minimize } \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

Subject to:

$$y_i - w \cdot x_i - b \leq \epsilon + \xi_i,$$
$$w \cdot x_i + b - y_i \leq \epsilon + \xi_i^*,$$
$$\xi_i, \xi_i^* \geq 0.$$

Given that I was required to predict more than one dimension sample, I wrapped the AdaBoost in a MultiOutputRegressor and then performed a GridSearchCV, iterating on the following hyperparameters: 'estimator learning rate' (values: 0.1, 0.01, 0.001), 'number of estimators' (values: 70, 90), 'estimator epsilon' (values: [0.01, 0.1] and 'estimator kernel'(values: 'linear' or 'rbf')

The iteration on the kernels is, in fact, futile, as has been demonstrated through the observation that a linear regression function provides an inadequate approximation of the data set. For illustrative purposes, both options have been presented, and it is anticipated that the model will perform considerably better with the radial basis function kernel(rbf).
The RBF kernel quantifies the degree of similarity between two data points based on their distance in a feature space. This process corresponds to the mapping of data into an infinite-dimensional feature space. This is because a nonlinear dataset represented in a 2D space, may become linearly separable in a higher-dimensional space. This implies that the function $\phi(x)$, which represents the mapping, would generate an infinite number of features. The explicit computation of $\phi(x)$ is a computationally expensive process. In contrast, the kernel trick allows the direct calculation of the dot product between two mapped points directly in the feature space, utilising the kernel function.

3.3 First application

In order to assess the model's capacity to accurately predict outcomes with a limited number of samples, an initial application was made to the smaller 1k dataset, which presented a more challenging problem.
GridSearchCV was computed on 120 fits, resulting in a mean squared error (MSE) of $5 * 10^{-5}$. The optimal parameters were as follows: 'estimator epsilon': 0.01, 'estimator kernel': 'rbf', 'estimator learning rate': 0.1, and 'numbers of estimators': 90.

The failure of this trial is evidenced by the fact that the test loss was 0.002, which is two orders of magnitude higher than the train loss. This discrepancy may be indicative of overfitting (the train loss curve converged too fast and in a too steady way) of the model on the training set. There are a number of potential reasons for this, but the primary one may be the relatively limited dimension of the dataset.

For this reason the training and hyperparameter search will now be repeated with the larger dataset of 10,000 samples.

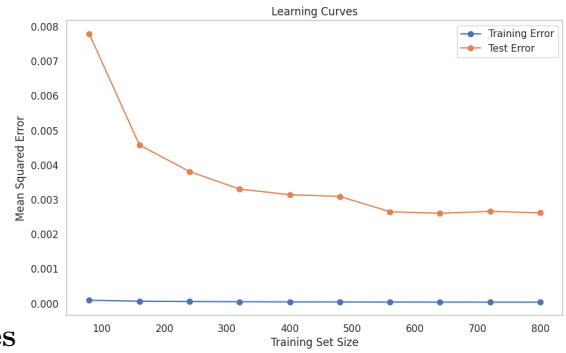

Figure 7: bad learning curves

3.4 Second application - enhancing performances

In order to enhance model performances, I retry the training phase and hyperparameter tuning on the 10k dataset. A few modifications were made to the parameters grid for computational reasons, as the hardware employed was unable to compute all the fits in an acceptable time frame. Consequently, the model was run on the following

parameter grid:'learning rate' : [0.1, 0.01, 0.001],'number of estimators' : [80], estimator epsilon': [0.01, 0.1], 'estimator kernel': [rbf][2].
This new tuning ended with a train loss of $3.15 * 10^{-5}$ and a test loss of $3.08 * 10^{-5}$. The train loss is now aligned with the test loss, exhibiting a comparable order of magnitude. It would appear that the overfitting issue has been resolved.

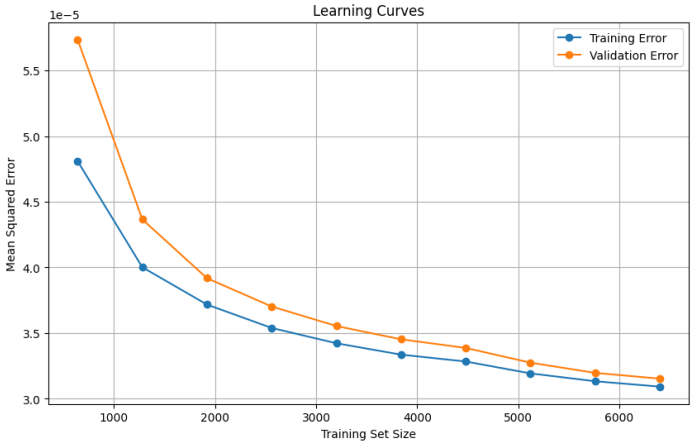

Figure 8: good learning curves

3.5 Jacobians computation

The Jacobian computation and comparison were conducted in a manner analogous to that employed for the R2 robot, with one notable exception. Given that AdaBoost is a non-gradient-based algorithm, it is not capable of supporting the computation of gradients or derivatives in the manner of gradient-based models. In order to estimate the Jacobian, it was necessary to employ a numerical approximation through the central difference method. This method approximates the partial derivatives by observing the effect of small changes in the input on the output.

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$$

Given that AdaBoost is a "black-box" model that does not provide explicit gradients, numerical differentiation is the only viable approach for estimating the Jacobian. By perturbing the input and observing the resulting changes in the output, this method allows us to estimate the matrix.

Once this had been completed, the difference between the learned Jacobian and the analytical one (calculated with 3Dof forward kinematics equations) was computed as previously described. The mean of the differences was found to be **0.020**, which is a relatively satisfactory result.

4 R5 robot

The R5 robot is composed of five revolute joints and a five-link structure, representing the most complex configuration present in the project. To address this issue, a deep neural network will be employed, initially trained on a small dataset and subsequently on a larger one, with the objective of highlighting the discrepancies between the two. The dataset for the R5 robot comprises five input features, namely joint angles j0, j1, j2, j3 and j4 and seven output variables, namely the corresponding end-effector position and orientation ft_x, ft_y, ft_z, ft_qw, ft_qz, ft_qx, ft_qy.

4.1 EDA

The robot workspace is presented below. This configuration appears to be the most spatially complex so far.

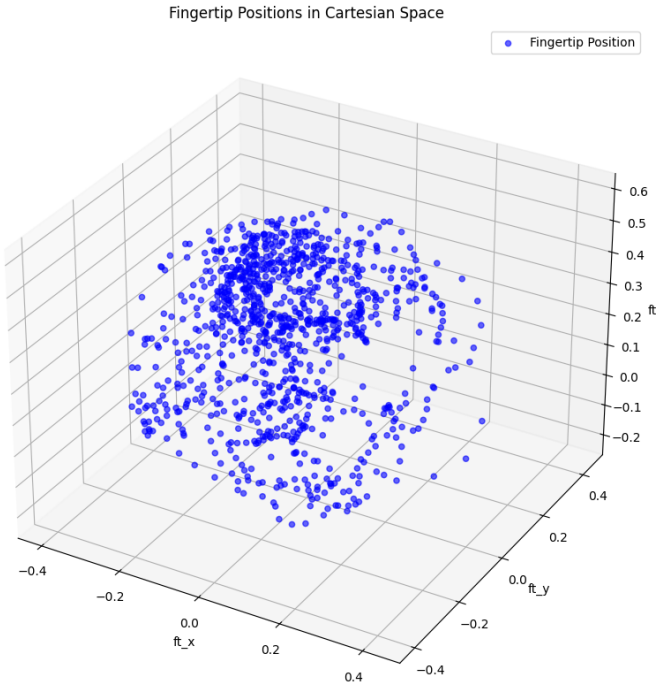

Figure 9: r5 workspace

Also in this analysis, both data seeds[3] have been utilized to have different seeds for train and test datas.

4.2 Model

In this section, a Keras neural network model structured like the one used in the R2 problem (same loss[3] and optimizer[4]), comprising one input and output layer (linear) and a variable number of hidden layers activated by the ReLu[5] function. The depth of the NN and the number of neurons are also subject to hyperparameter tuning, with the objective of developing a dynamic neural network.
So, the GridSearchCV is computed on the following parameters grid: 'learning rate': [0.001, 0.01, 0.1], 'epochs': [20, 30, 50], 'batch size': [16, 32], 'hidden layers': [[16, 32] 2 layers, [16, 32, 64] 3 layers, [16, 32, 64, 126] 4 layers]

4.3 First application

The initial attempt was conducted by fitting the 1K dataset to 270 combinations, resulting in a suboptimal performance in terms of loss and prediction. The train loss was **0.01**, while the test loss was **0.012**. Despite the two losses exhibiting similar magnitudes, their absolute values were relatively low, indicating a limited predictive capacity of the model.
The preceding hypothesis was further validated through an examination of the scatter plot displayed below, which illustrates the disparate structures of the test and predicted values.

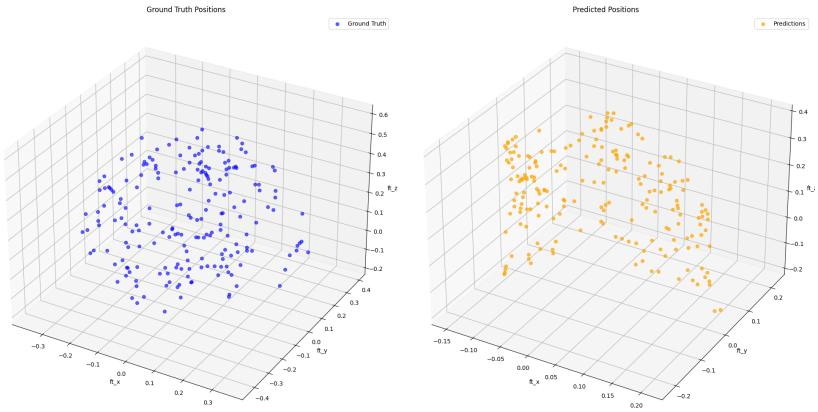

Figure 10: first try-groudn truth vs predictions

4.4 Second application - enhancing performances

In order to enhance the model performance, a refit will be executed on the larger 10K dataset, as the insufficient quantity of samples may be the reason for the model's underperformance in the initial attempt. In order to guarantee the most reliable comparison between the two trials, it was essential to maintain the integrity of the model architecture and parameters grid. Subsequently, the GridSearchCV was recalculated on 270 fits, but on a larger dataset, resulting in a computation that was considerably more time-consuming. This iteration resulted in a notable improvement in performance compared to the previous one, with a training loss of 0.0059 and a test loss of 0.0054.

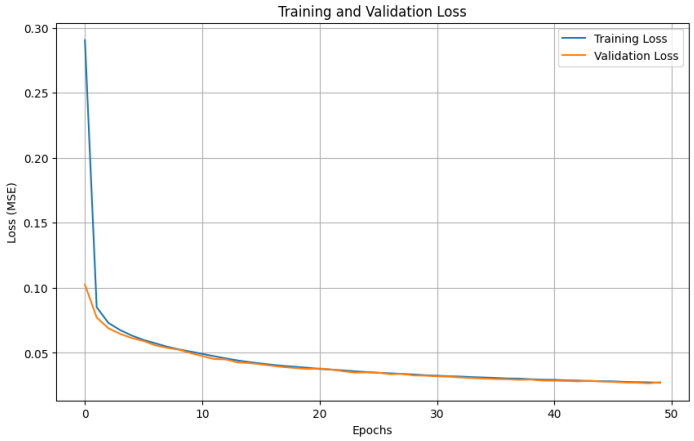

Figure 11: learning curves

It can be observed that the losses converge and reach a low, stable value around the desired epochs.
Not only are the losses comparable in magnitude (in particular, the test loss is slightly lower), but they are also one order of magnitude less than the previous results. Therefore, the second attempt was successful in improving the model's performance, although the achieved loss values may not be considered optimal for a model of this nature.
The scatter plot displayed below illustrates the structure of the test and predicted values. It is notable that the two structures exhibit a considerable degree of similarity despite not being an exact match. Therefore, the performance of the model prediction can be considered satisfactory.

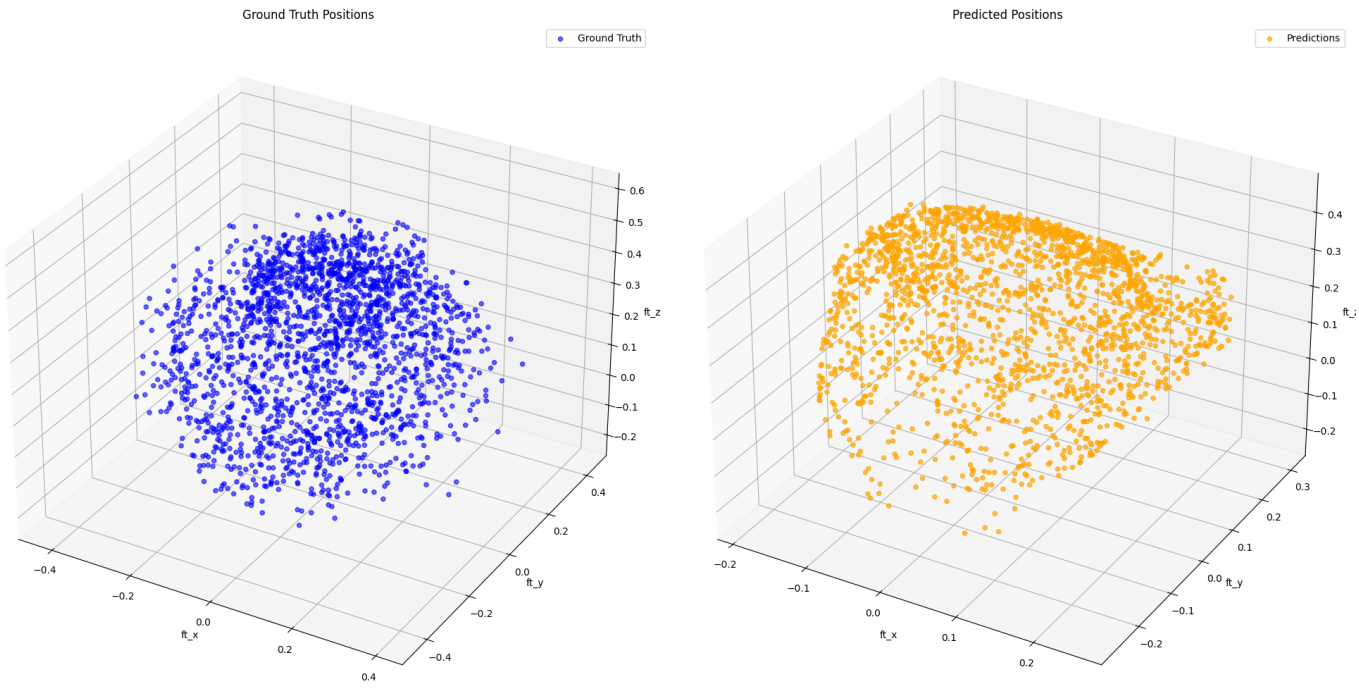

Figure 12: test vs prediction

4.5 Jacobians computation

The Jacobians of the R5 robot are currently being calculated. The learned one is calculated using the same functions of the R2 configurations, with the exception of the number of features in and labels out. In contrast, the analytical Jacobian has been calculated using the forward kinematics equations of a 5-DOF robot, assuming five horizontal revolute joints arranged in a linear configuration (on in top of the other) with the first joint in a vertical position (with the z-axis orthogonal to the remaining four). The forward kinematics is intended as follows.

Forward Kinematics: The position of the end-effector $\mathbf{P} = (x, y, z)$ is a function of the joint angles:

$$\mathbf{P} = f(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5)$$

The Jacobian matrix J is the matrix of partial derivatives of the position vector $\mathbf{P}$ with respect to the joint angles $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5$:

$$J = \frac{\partial \mathbf{P}}{\partial \theta} = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} & \frac{\partial x}{\partial \theta_4} & \frac{\partial x}{\partial \theta_5} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3} & \frac{\partial y}{\partial \theta_4} & \frac{\partial y}{\partial \theta_5} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial \theta_3} & \frac{\partial z}{\partial \theta_4} & \frac{\partial z}{\partial \theta_5} \end{bmatrix}$$

That analytically is:

In order to facilitate comprehension, we will denote the elements of the Jacobian matrix as J_{ij}:

$$J_{11} = -\sin(q_1)\left(L_5\cos(q_2+q_3+q_4+q_5) + L_3\cos(q_2+q_3) + L_2\cos(q_2) + L_4\cos(q_2+q_3+q_4)\right)$$

$$J_{12} = -\cos(q_1)\left(L_5\sin(q_2+q_3+q_4+q_5) + L_3\sin(q_2+q_3) + L_2\sin(q_2) + L_4\sin(q_2+q_3+q_4)\right)$$

$$J_{13} = -\cos(q_1)\left(L_5\sin(q_2+q_3+q_4+q_5) + L_3\sin(q_2+q_3) + L_4\sin(q_2+q_3+q_4)\right)$$

$$J_{14} = -\cos(q_1)\left(L_5\sin(q_2+q_3+q_4+q_5) + L_4\sin(q_2+q_3+q_4)\right)$$

$$J_{15} = -L_5\sin(q_2+q_3+q_4+q_5)\cos(q_1)$$

$$J_{21} = \cos(q_1)\left(L_5\cos(q_2+q_3+q_4+q_5) + L_3\cos(q_2+q_3) + L_2\cos(q_2) + L_4\cos(q_2+q_3+q_4)\right)$$

$$J_{22} = -\sin(q_1)\left(L_5\sin(q_2+q_3+q_4+q_5) + L_3\sin(q_2+q_3) + L_2\sin(q_2) + L_4\sin(q_2+q_3+q_4)\right)$$

$$J_{23} = -\sin(q_1)\left(L_5\sin(q_2+q_3+q_4+q_5) + L_3\sin(q_2+q_3) + L_4\sin(q_2+q_3+q_4)\right)$$

$$J_{24} = -\sin(q_1)\left(L_5\sin(q_2+q_3+q_4+q_5) + L_4\sin(q_2+q_3+q_4)\right)$$

$$J_{25} = -L_5 \sin(q_2 + q_3 + q_4 + q_5) \sin(q_1)$$

$$J_{31} = 0$$

$$J_{32} = L_5 \cos(q_2 + q_3 + q_4 + q_5) + L_3 \cos(q_2 + q_3) + L_2 \cos(q_2) + L_4 \cos(q_2 + q_3 + q_4)$$

$$J_{33} = L_5 \cos(q_2 + q_3 + q_4 + q_5) + L_3 \cos(q_2 + q_3) + L_4 \cos(q_2 + q_3 + q_4)$$

$$J_{34} = L_5 \cos(q_2 + q_3 + q_4 + q_5) + L_4 \cos(q_2 + q_3 + q_4)$$

$$J_{35} = L_5 \cos(q_2 + q_3 + q_4 + q_5)$$

$$J = \begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} & J_{15} \\ J_{21} & J_{22} & J_{23} & J_{24} & J_{25} \\ J_{31} & J_{32} & J_{33} & J_{34} & J_{35} \end{bmatrix}$$

As with the other models, the mean of all the differences between the Frobenius norms was calculated and found to be 0.49, which is not an extremely small value. This could be due to the non-constant definition of the link lengths, which, if slightly different from each other, could result in an erroneous calculation (as, in my code, were considered to be all long the same).