COMP0245 Final Project

Fuquan Zhang Fangshu Wu Nehith Vemulapalli

October 8, 2025

1 Introduction

This project focuses on improving the control of a robotic arm by employing machine learning techniques specifically designed for trajectory optimization and motor behavior modeling. The primary objective is to refine the movements of a simulated robotic arm in a controlled environment using data-driven approaches. By utilizing datasets generated through robotic motion simulations, we aim to correct erroneous motor models and enhance the accuracy of joint position predictions. This work is contextualized within the broader field of robotics, where precise manipulation tasks are increasingly critical, particularly in automated warehouses and assembly lines. The technical foundation of this project incorporates machine learning frameworks like PyTorch for developing neural networks and Scikit-learn for regression analysis. These tools will enable the robotic arm to learn and adapt to various tasks, thereby improving its operational efficiency and versatility in real-world applications.

2 Materials

- **Python:** A versatile programming language used for implementing control algorithms, state estimation, and data analysis, providing a robust environment for numerical computations and simulations.
- **NumPy:** A fundamental package for scientific computing in Python used for numerical operations and efficient array manipulations, essential for handling data in simulations.
- **Matplotlib:** A plotting library in Python utilized for visualizing the results of simulations, including robot trajectories and estimation errors.
- **SciPy:** A library used for advanced mathematical computations, including solving the Discrete Algebraic Riccati Equation (DARE) as part of the control algorithms.
- **PyTorch:** This open-source machine learning library is used for building and training neural networks, enabling complex function approximations necessary for modeling motor behavior.
- **Pinocchio:** A robotic framework used to facilitate the modeling and kinematic analysis of the robot, providing tools for simulation and interaction with the physical dynamics of robotic systems.

- **PyBullet:** A physics simulation library used to model and simulate the robotic system. It enables real-time control and evaluation of robot behavior under various conditions.

3 Methods

This project focuses on enhancing robotic arm control through the correction of motor behavior models, trajectory generation using machine learning techniques, and ultimately utilizing these models to command a simulated robot arm. The methodology is organized into three main tasks, each with specific sub-tasks that outline the processes, algorithms used, and the mathematical formulations applied.

3.1 Task 1: Correcting an Erroneous Model of Motor Behaviour

In this task, we employ neural network models to correct for errors in the control of robotic motors. To analyze the impact of model configurations, several key parameters are varied, including hidden node counts, learning rates, and batch sizes.

3.1.1 Data Generation

To begin, we generate synthetic training data that simulates motor dynamics:

- **Torque Calculation:** The control torque is calculated using the equation:

$$\tau(t) = K_p(\theta_T(t) - \theta(t)) + K_d(\dot{\theta}_T(t) - \dot{\theta}(t)) \tag{1}$$

  where $\tau(t)$ represents the torque, with K_p and K_d being the proportional and derivative gains, respectively.

- **Acceleration Dynamics:** The ideal and actual acceleration of the motor are given by:

$$\ddot{\theta}^*(t) = \frac{\tau(t)}{m} \tag{2}$$

$$\ddot{\theta}(t) = \ddot{\theta}^*(t) - \frac{b \cdot \dot{\theta}(t)}{m} \tag{3}$$

  where b is the damping coefficient and m is the mass, effectively capturing the impact of damping on motor performance.

The code initializes these computations across a defined time frame, stores joint positions and velocities, and computes the error for training purposes [1].

3.1.2 Model Development

Two neural network architectures[2] are employed for modeling the correction terms:

- **ShallowCorrectorMLP:** This model consists of a single hidden layer, implemented in PyTorch:

```
class ShallowCorrectorMLP(nn.Module):
    def __init__(self, num_hidden_nodes):
        super(ShallowCorrectorMLP, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(4, num_hidden_nodes),
            nn.ReLU(),
            nn.Linear(num_hidden_nodes, 1)
        )
    def forward(self, x):
        return self.layers(x)
```

- **DeepCorrectorMLP:** This model features an additional hidden layer to enhance capacity for learning complex mappings:

```
class DeepCorrectorMLP(nn.Module):
    def __init__(self, num_hidden_nodes):
        super(DeepCorrectorMLP, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(4, num_hidden_nodes),
            nn.ReLU(),
            nn.Linear(num_hidden_nodes, num_hidden_nodes),
            nn.ReLU(),
            nn.Linear(num_hidden_nodes, 1)
        )
    def forward(self, x):
        return self.layers(x)
```

3.1.3 Parameter Variations and Training Procedure

To assess model performance, key parameters are systematically varied [3]:

- **Impact of Hidden Nodes:** The number of hidden nodes in ShallowCorrectorMLP is varied from 32 to 128 in steps of 32. The resulting training losses, prediction accuracy, and trends are analyzed, comparing against models without corrective features.
- **Impact of Learning Rate:** Different learning rates (1.0, 0.1, 0.01, 0.001, and 0.0001) are tested for both ShallowCorrectorMLP and DeepCorrectorMLP. The effect of these rates on training loss and generalization over the entire domain is documented.
- **Effect of Batch Size:** Using 32 hidden units, batch sizes of 64, 128, 256, and 1000 are tested. We examine their impact on training time, stability, efficiency, and both training and test losses.

- **Loss Function:** The Mean Squared Error (MSE) loss function is utilized to quantify the error between predicted and actual acceleration:

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{4}$$

- **Optimization:** The Adam optimizer is employed with a learning rate of 0.00001.
- **Training Loop:** Each model is trained for 1000 epochs, with periodic logging of the training loss to assess performance.

3.2 Task 2: Training to Control the Motion of the Robot

In this task, we utilize the dataset provided to train models for predicting joint positions necessary for accurate robotic arm control. The effectiveness of various machine learning methods is compared for performance.

3.2.1 Data Preparation

The dataset (data.pkl) contains structured data crucial for effective training:

- **Data Attributes:**
  - **time:** Timestamps captured at intervals of 2 ms over a 5-second period.
  - **q_mes_all:** Measured joint positions for seven joints.
  - **goal_positions:** Cartesian coordinates defining target positions.
- **Dataset Handling:** The JointDataset class formats the data for use in training:

```
class JointDataset(Dataset):
    def __init__(self, time_data, goal_data, joint_data):
        x = np.hstack((time_data.reshape(-1, 1), goal_data))
        self.x_data = torch.from_numpy(x).float()
        self.y_data = torch.from_numpy(joint_data).float().unsqueeze(1)

    def __len__(self):
        return len(self.x_data)

    def __getitem__(self, idx):
        return self.x_data[idx], self.y_data[idx]
```

3.2.2 Model Training and Testing

- **MLP Training:** An MLP model is trained for each joint to predict its motion based on time inputs. Training and testing losses are recorded, and generalization capability is evaluated by selecting ten random goal positions and analyzing prediction accuracy [4].

- **Random Forest Regression:** A Random Forest model is employed with initial parameters ($n\_estimators = 100$, $max\_depth = None$). Depths from 2 to 10 are tested to explore effects on trajectory quality and overfitting, as reflected in tree depth, accuracy, and computational cost[5].
- **Performance Comparison:** The accuracy, generalization, and computational requirements of the MLP and Random Forest models are compared, noting advantages and disadvantages of each in modeling joint trajectories.

3.3 Task 3: Control and Evaluation of the Robot Arm

Following model training, this task integrates the learned models into a control loop to command the robot arm accurately.

3.3.1 Control Implementation

The control algorithm involves:

- **Feedback Linearization Control:** The control torque is computed using:

$$\tau(t) = K_p \cdot (q_d(t) - q(t)) + K_d \cdot (\dot{q}_d(t) - \dot{q}(t)) \tag{5}$$

  where q_d and $\dot{q}_d$ represent the desired joint positions and velocities.
- **Simulation Execution:** The simulation is run in real-time, and the measured joint angles are compared against the desired trajectories to analyze tracking performance.

3.3.2 Evaluation of Smoothness and Errors

During the simulation runs, we evaluate:

- **Tracking Error:** The difference between desired and actual joint positions is monitored:

$$\text{Tracking Error} = \|q_d - q\| \tag{6}$$

- **Control Input Analysis:** The control inputs generated throughout the simulation are analyzed for stability and smoothness, focusing on variations and abrupt transitions that could impact robotic arm behavior.

3.3.3 Trajectory Smoothing

To address discontinuities in the trajectories produced by the Random Forest model, we employ an exponential moving average smoothing filter[6]. This process involves:

- **Filtering Method:**

$$S_t = \alpha \cdot X_t + (1 - \alpha) \cdot S_{t-1} \tag{7}$$

  where α determines the degree of smoothing.
- **Impact Measurement:** After applying the smoothing filter, the robotic arm's motion is re-evaluated, and the results are compared against both the original Random Forest and the MLP predictions.

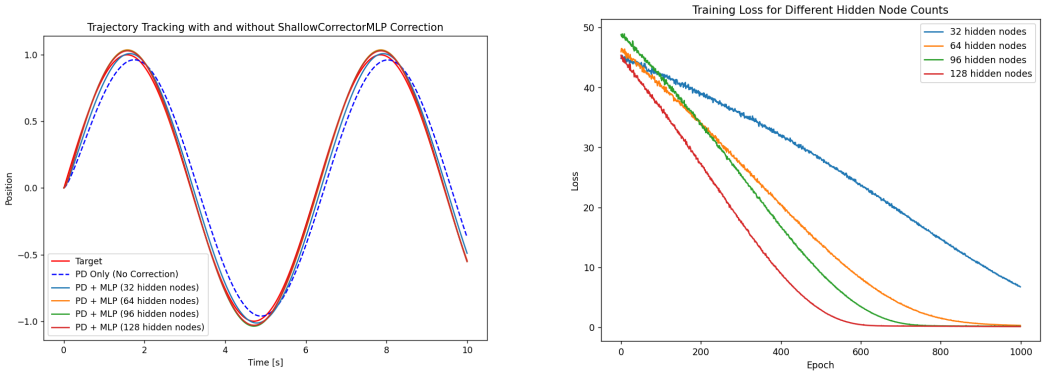
4 Results

4.1 Task 1

In Task 1, the performance of neural network models is tested under different structures and training parameters, with the goal of optimising the ShallowCorrectorMLP and DeepCorrectorMLP models for better adaptation to complex control tasks. The experiments focus on investigating the impact of hyperparameters such as the number of hidden layer nodes, learning rate and batch size on the training effectiveness of the models, and evaluating the tuning effect of these parameters by comparing the results.

4.1.1 Task 1.1: Impact of the number of hidden layer nodes on model training

In the ShallowCorrectorMLP model, the experiments adjusted the number of hidden layer nodes from 32 to 128 incrementally and observed the effect of the number of nodes on training loss and prediction accuracy. The results are as follows:



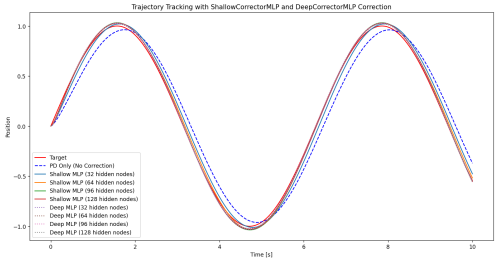
(a) Trajectory tracking.

(b) Training loss.

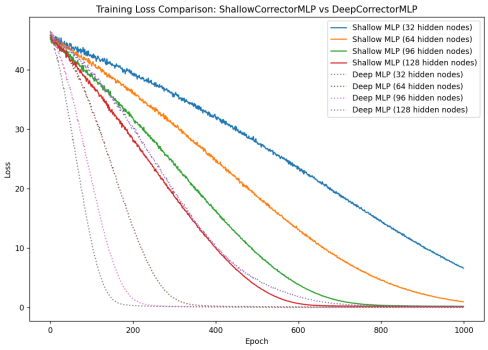
Figure 1: Comparison of models with different number of hidden nodes.

4.1.2 Task 1.2: Comparison of ShallowCorrectorMLP and DeepCorrectorMLP models

In this section, we construct the DeepCorrectorMLP model (two hidden layer structure) and set the number of hidden nodes from 32 to 128 (with a step size of 32) to compare the training performance of the shallow and deep models. The experimental results are as follows:



(a) Trajectory tracking.

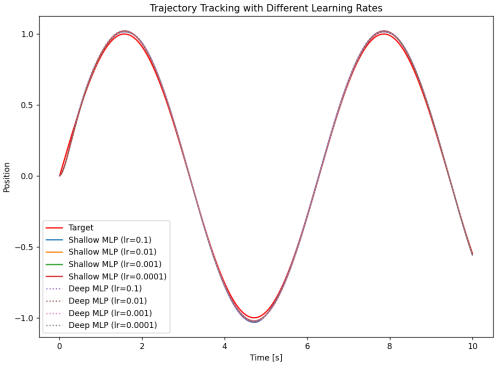


(b) Training loss.

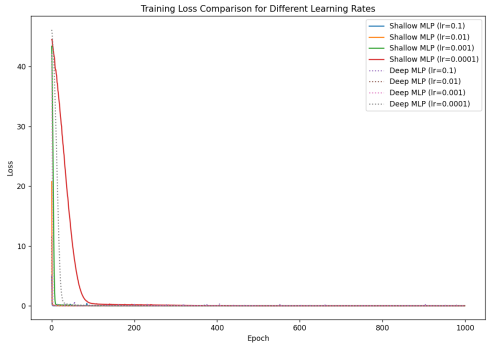
Figure 2: Comparison of shallow and deep models with different number of hidden nodes

4.1.3 Task 1.3: Impact of learning rate on model training

In the ShallowCorrectorMLP and DeepCorrectorMLP models, we tested the effect of different learning rates (1.0, 0.1, 0.01, 0.001, and 0.0001) on model training. It was found that due to the high learning rate of 1.0, the initial loss of training was very high, making the images with other learning rates not easy to observe, so here we divided into two images with and without 1.0 learning rate to facilitate the observation of the results. The experimental results are as follows:

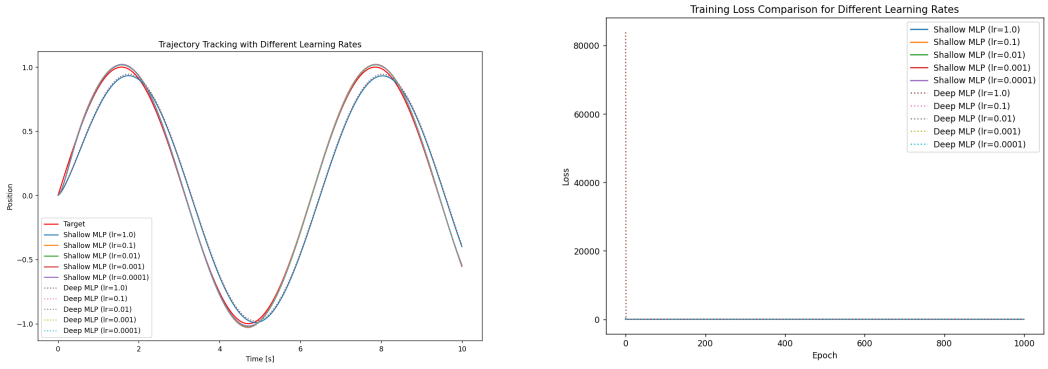


(a) Trajectory tracking.



(b) Training loss.

Figure 3: Comparison of model training and testing losses at different learning rates(without lr = 1.0)

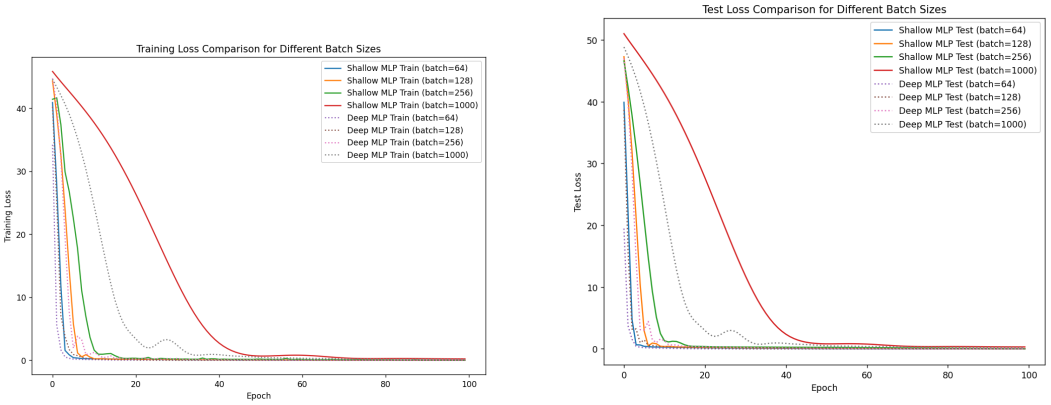


(a) Trajectory tracking. (b) Training loss.

Figure 4: Comparison of model training and testing losses at different learning rates(with lr = 1.0)

4.1.4 Task 1.4: Effect of batch size on model training

In the ShallowCorrectorMLP and DeepCorrectorMLP models, the experiments fixed the number of hidden nodes to 32, and tested batch sizes of 64, 128, 256, and 1000 to observe the effect of batch size on the training results.The experimental results are as follows:



(a) Training Loss. (b) Test Loss.

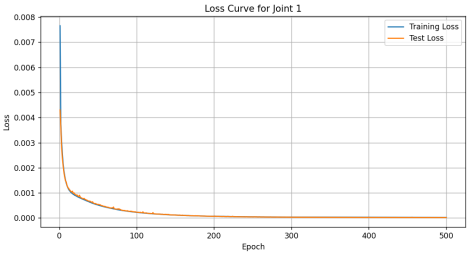
Figure 5: Plot of model training vs. testing losses for different batch sizes

4.2 Task 2

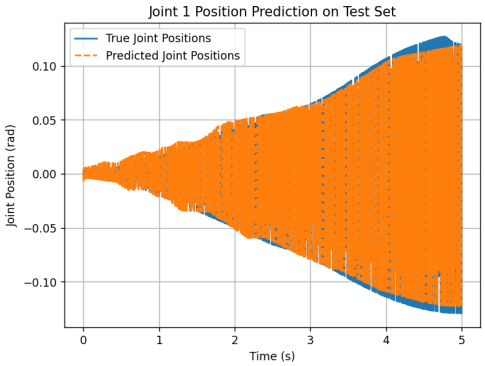
In Task 2, the experiment focuses on training a machine learning model to predict the joint motion trajectories of a robot at a given target position. The performance of the neural network (MLP) and random forest model is mainly investigated, and comparative analyses are conducted in terms of prediction accuracy, model generalisation ability, and computational efficiency, respectively.

4.2.1 Task 2.1: Training Results of Multi-Layer Perceptron (MLP) Model

In Task 2.1, a multilayer perceptron (MLP) model was used to predict the position of each joint. The data was divided into a training set and a test set in order to assess the fitting effect and generalisation performance of the model. In assessing the generalisation ability of the model, 10 target positions were randomly selected for testing. The experimental results are presented below:

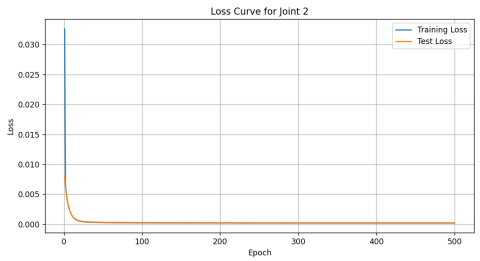


(a) Loss.

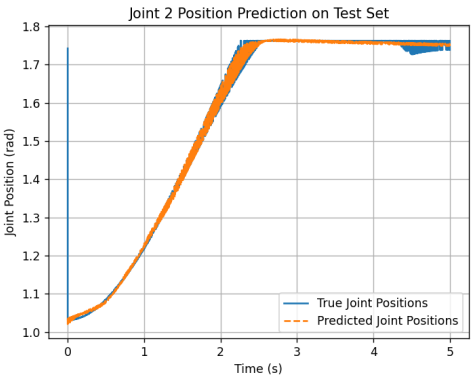


(b) Joint Position.

Figure 6: Joint 1

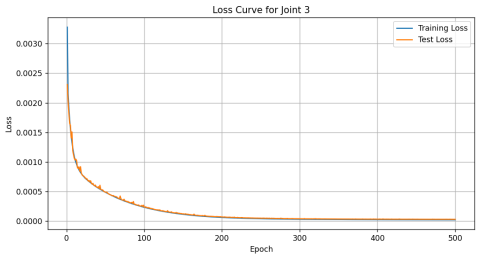


(a) Loss.

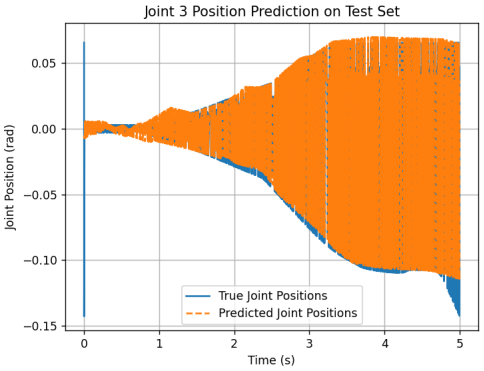


(b) Joint Position.

Figure 7: Joint 2

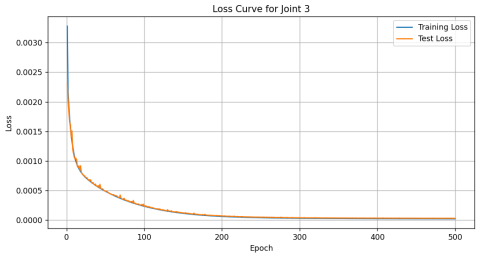


(a) Loss.

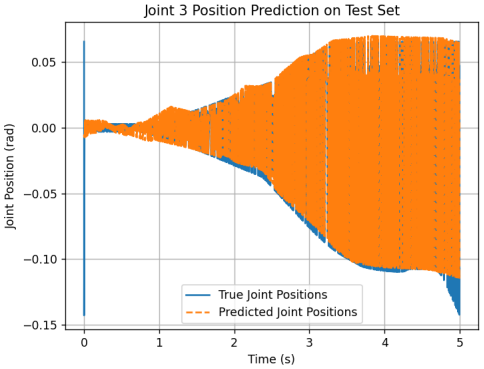


(b) Joint Position.

Figure 8: Joint 3

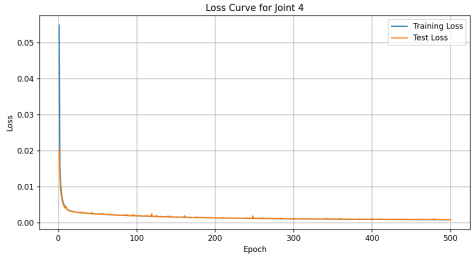


(a) Loss.

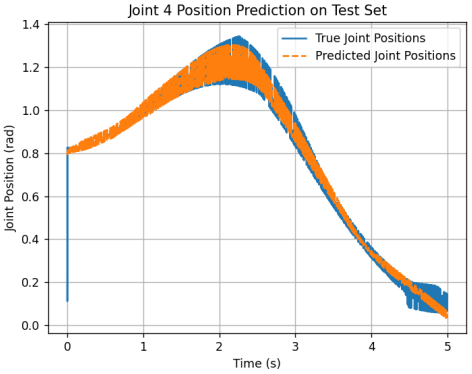


(b) Joint Position.

Figure 9: Joint 3

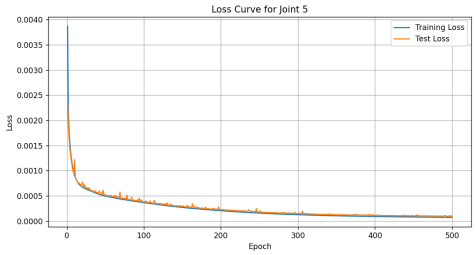


(a) Loss.

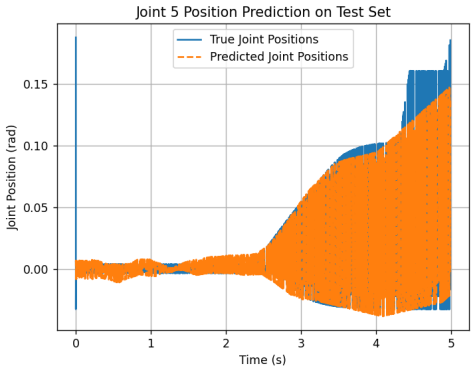


(b) Joint Position.

Figure 10: Joint 4

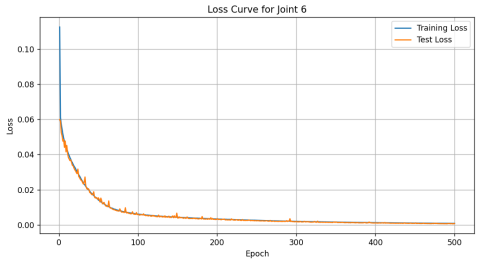


(a) Loss.

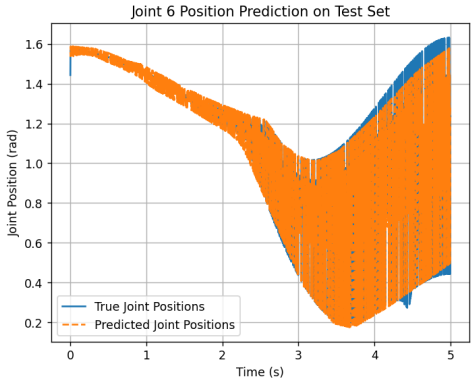


(b) Joint Position.

Figure 11: Joint 5

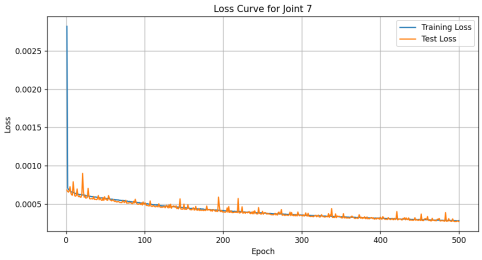


(a) Loss.

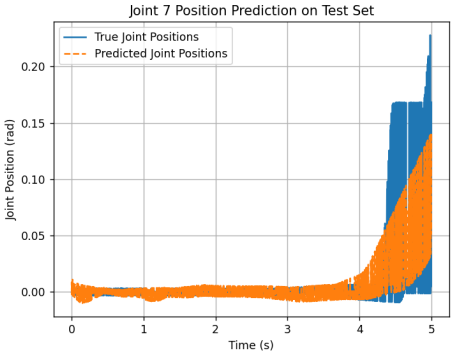


(b) Joint Position.

Figure 12: Joint 6



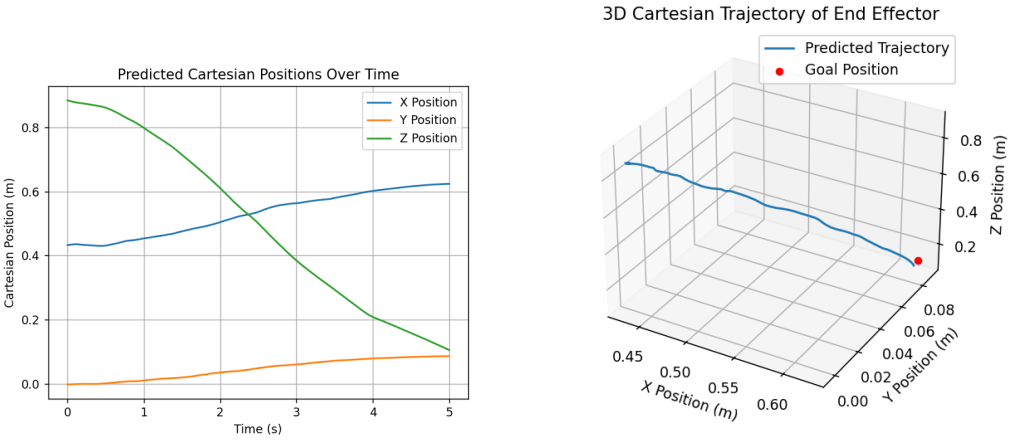
(a) Loss.



(b) Joint Position.

Figure 13: Joint 7

Ten prediction positions were selected for the experiment, only one is shown here due to limited report space, the other results are basically the same and they can be got when running the code "task2.1". The result of the experiment is as follows:



(a) Predicted Position. (b) 3D Trajectory.

Figure 14: Predicted trajectories of end-effector in Cartesian coordinate system vs. target position

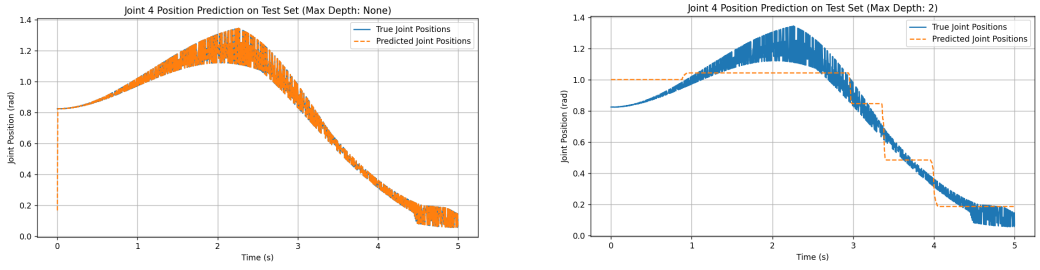
The Position Error and MSE are shown below:

```
Position error between computed position and goal: 0.02750020629212769
Mean Squared Error (MSE) for all tested positions: 0.001134741657916127
```

Figure 15: Position Error and MSE

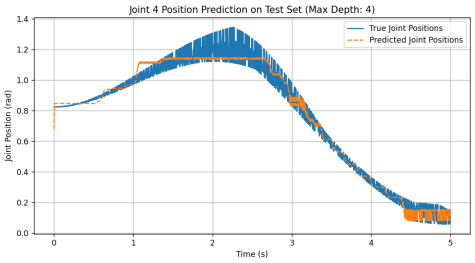
4.2.2 Task 2.2: Random Forest Model Training Results

In Task 2.2, a random forest regression model was experimentally trained for predicting joint positions. The random forest model was experimentally used with different tree depths (ranging from 2 to 10) to observe the effect of tree depth on prediction accuracy. Since each of the seven joints has a position prediction plot at different tree depths, the space for reporting is limited, so only joint 4 is shown here for the discussion of the experimental results, and the other result plots can be got when running the code "task2.2". The experimental results are as follows:

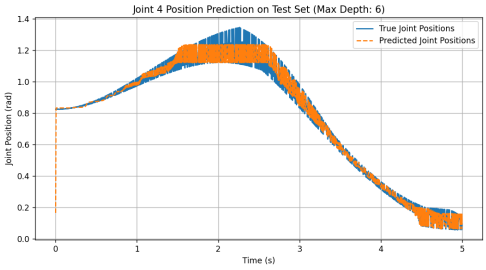


(a) Depth = None (b) Depth = 2

Figure 16: Position prediction curve

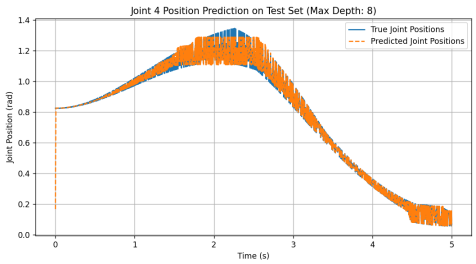


(a) Depth = 4

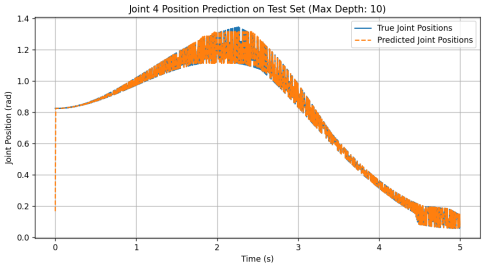


(b) Depth = 6

Figure 17: Position prediction curve



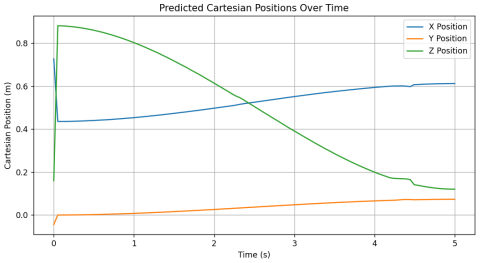
(a) Depth = 8



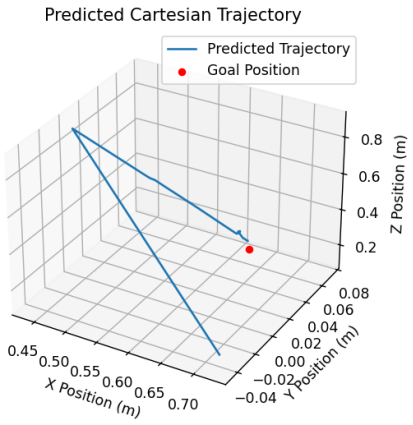
(b) Depth = 10

Figure 18: Position prediction curve

Ten prediction positions were selected for the experiment, only one is shown here due to limited report space, the other results are basically the same and they can be found in the zip file. The result of the experiment is as follows:



(a) Predicted Position.



(b) 3D Trajectory.

Figure 19: Predicted trajectories of end-effector in Cartesian coordinate system vs. target position

The Position Error and MSE are shown below:

```
Summary of All Test Positions:
Total MSE: 0.0009889371435781768
Total Position Error: 0.04682126961021679
```

Figure 20: Position Error and MSE

4.2.3 Task 2.3: Comparative analysis of models

In Task 2.3, experiments were conducted to compare the prediction effectiveness of the multilayer perceptron (MLP) model and the random forest (RF) model at different tree depths. Five random target locations were selected for testing in the experiment, and the performance of the models in terms of accuracy, ability to generalise to new target locations, and computational performance was analysed by calculating the mean squared error (MSE)[7], prediction time, and positional error for each model, and by combining them with the comparative plots of trajectories of the joint locations over time in Tasks 2.1 and 2.2. The computational results are as follows:

Target Position	Model Type	Position Error	Prediction Time
Position 1	MLP	0.037704	0.02s
	RF (Depth = None)	0.078135	1.79s
	RF (Depth = 2)	0.103155	0.51s
	RF (Depth = 4)	0.092724	0.47s
	RF (Depth = 6)	0.087037	0.51s
	RF (Depth = 8)	0.077612	0.53s
	RF (Depth = 10)	0.079770	0.65s
Position 2	MLP	0.030535	0.01s
	RF (Depth = None)	0.062538	1.56s
	RF (Depth = 2)	0.062012	0.48s
	RF (Depth = 4)	0.050397	0.48s
	RF (Depth = 6)	0.063227	0.50s
	RF (Depth = 8)	0.064472	0.54s
	RF (Depth = 10)	0.061890	0.65s
Position 3	MLP	0.036936	0.01s
	RF (Depth = None)	0.048996	1.48s
	RF (Depth = 2)	0.068071	0.49s
	RF (Depth = 4)	0.071029	0.48s
	RF (Depth = 6)	0.060697	0.51s
	RF (Depth = 8)	0.049009	0.54s
	RF (Depth = 10)	0.052414	0.63s
Position 4	MLP	0.031744	0.01s
	RF (Depth = None)	0.038453	1.39s
	RF (Depth = 2)	0.041093	0.51s
	RF (Depth = 4)	0.033685	0.50s
	RF (Depth = 6)	0.035021	0.49s
	RF (Depth = 8)	0.038303	0.55s
	RF (Depth = 10)	0.038881	0.65s
Position 5	MLP	0.033927	0.01s
	RF (Depth = None)	0.039954	1.34s
	RF (Depth = 2)	0.043361	0.50s
	RF (Depth = 4)	0.035422	0.56s
	RF (Depth = 6)	0.038524	0.49s
	RF (Depth = 8)	0.039404	0.52s
	RF (Depth = 10)	0.040163	0.60s

Table 1: Position Error and Prediction Time for Each Target Position Using MLP and RF Models

The results of the MSE calculations are shown below:

```
Overall MLP MSE across all test goals: 0.002331
Overall RF MSE (depth=None) across all test goals: 0.001033
Overall RF MSE (depth=2) across all test goals: 0.001513
Overall RF MSE (depth=4) across all test goals: 0.001238
Overall RF MSE (depth=6) across all test goals: 0.001198
Overall RF MSE (depth=8) across all test goals: 0.001040
Overall RF MSE (depth=10) across all test goals: 0.001071
```

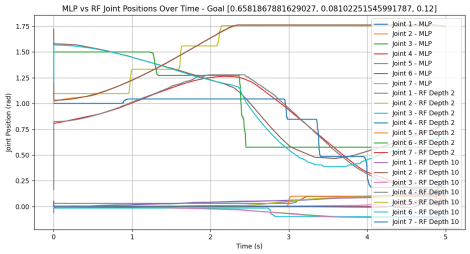
Figure 21: MSE

4.3 Task 3

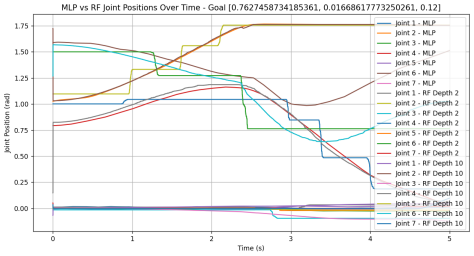
In Task 3 experiments, we compare the trajectory smoothness generated by the MLP model with the random forest model at different depths, and improve the continuity of the random forest trajectory by smoothing filter, and then analyse the effect of the smoothing treatment on the control performance. The results show that the smoothing filter significantly reduces the trajectory abruptness and improves the stability of the control moment, but introduces a slight response delay.

4.3.1 Task 3.1: Comparison of MLP and Random Forest Models on Trajectory Smoothness

In Task 3.1, experiments are conducted to generate the joint trajectories of the robotic arm using the MLP model and the Random Forest model with depths of 2 and 10, respectively, and the performance of each model in terms of trajectory smoothness and control effect is compared. It is also possible to observe the actual motion of the robotic arm after running the code. The results of the experiments are as follows:

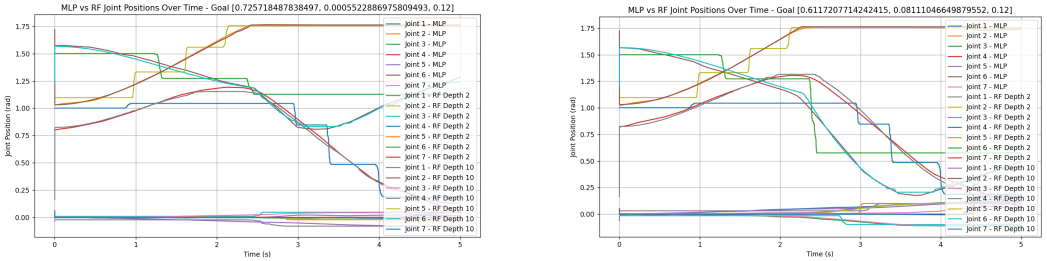


(a) Position 1



(b) Position 2

Figure 22: Joint position trajectory diagram



(a) Position 3 (b) Position 4

Figure 23: Joint position trajectory diagram

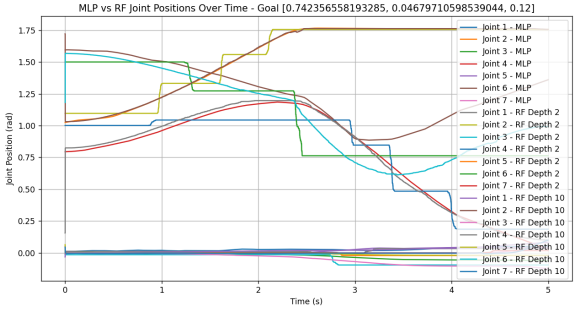


Figure 24: Position 5

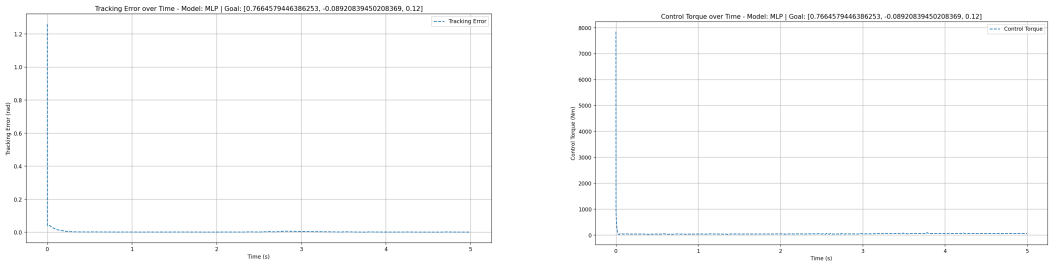
The experiments also recorded the smoothness scores (based on the average of neighbouring position changes) and the final position error (the distance between the target position and the final position) for each model at different target positions. The results of the experiments are presented below:

Target Position	Model Type	Trajectory Smoothness	Final Position Error
Position 1	MLP	0.000125	0.028824
	RF (Depth = 2)	0.000080	0.036672
	RF (Depth = 10)	0.000186	0.018934
Position 2	MLP	0.000107	0.032707
	RF (Depth = 2)	0.000073	0.101741
	RF (Depth = 10)	0.000168	0.076720
Position 3	MLP	0.000111	0.034486
	RF (Depth = 2)	0.000061	0.064100
	RF (Depth = 10)	0.000155	0.062221
Position 4	MLP	0.000136	0.008195
	RF (Depth = 2)	0.000084	0.035241
	RF (Depth = 10)	0.000189	0.007908
Position 5	MLP	0.000111	0.037980
	RF (Depth = 2)	0.000073	0.088645
	RF (Depth = 10)	0.000171	0.065101

Table 2: Trajectory Smoothness and Final Position Error for Each Target Position Using MLP and RF Models (Depths = 2 and 10)

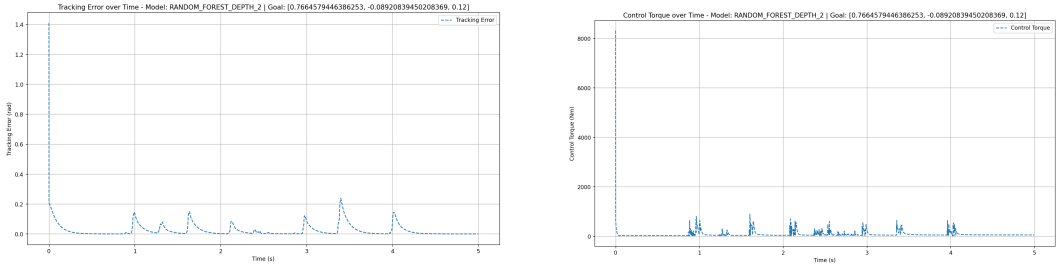
4.3.2 Task 3.2: Tracking error and control input torque

In this section, the experiment was conducted to obtain the tracking error by calculating the difference between the desired joint position and the actual joint position. At the same time, the variation of the torque input generated by the controller is recorded to observe the demand of the model on the control input. Due to the large number of result plots, only one set of them is shown here, the rest of the results can be got when running the code "task3.2". The experimental results are shown below:



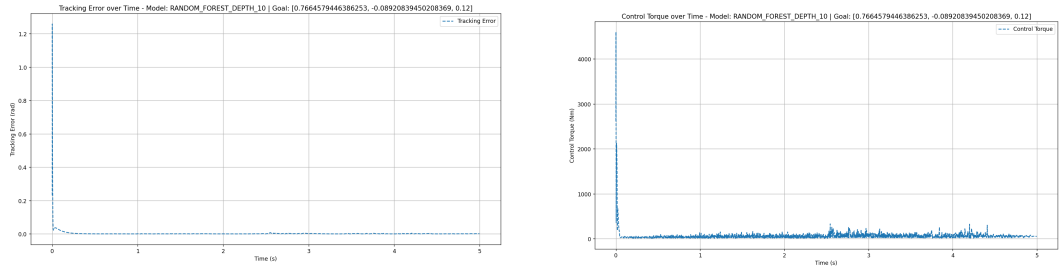
(a) Tracking Error (b) Control Torque

Figure 25: MLP



(a) Tracking Error (b) Control Torque

Figure 26: RF(depth = 2)



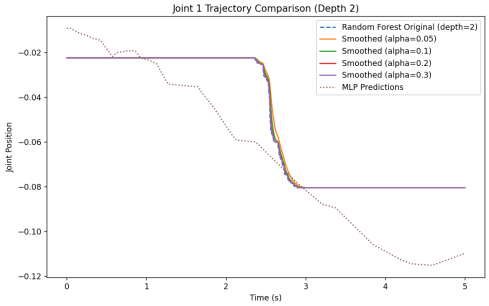
(a) Tracking Error (b) Control Torque

Figure 27: RF(depth = 10)

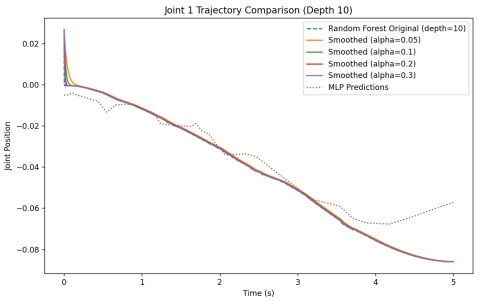
4.3.3 Task 3.3: Trajectory smoothing filter effect

An exponential moving average filter was applied to the trajectory output from the Random Forest model and the simulation was re-run to observe the effect of the smoothing.

Compare the smoothed trajectories with the original Random Forest trajectories and the results from the MLP model to analyse the effect of the smoothing filter on the improvement of trajectory continuity and control stability[8]. The experimental results are as follows:

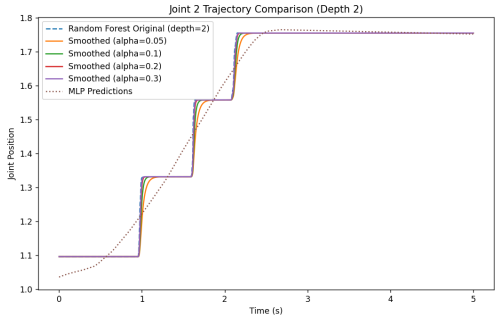


(a) Depth = 2

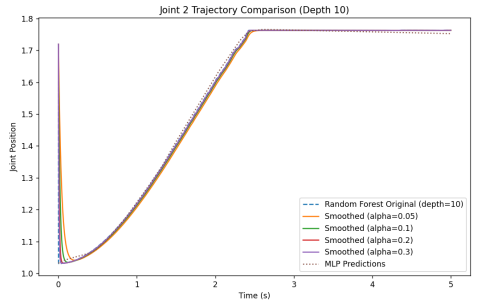


(b) Depth = 10

Figure 28: Joint 1 trajectory

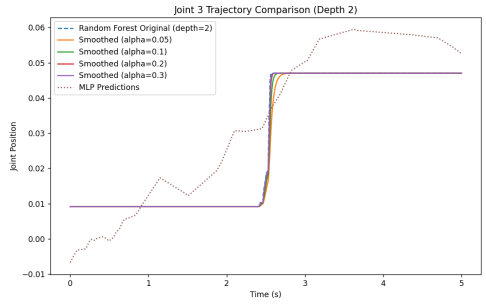


(a) Depth = 2

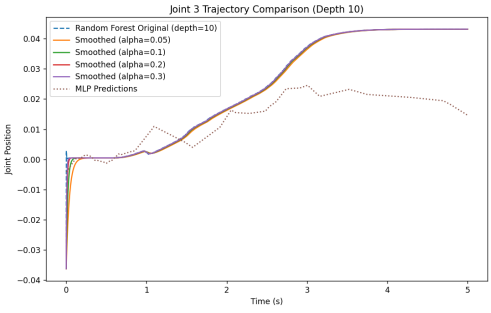


(b) Depth = 10

Figure 29: Joint 2 trajectory

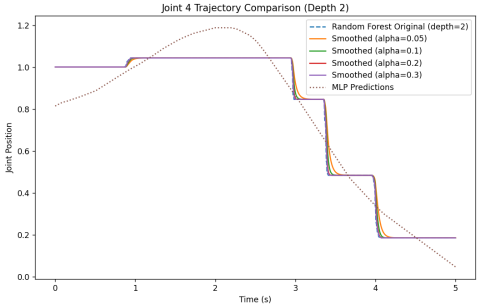


(a) Depth = 2

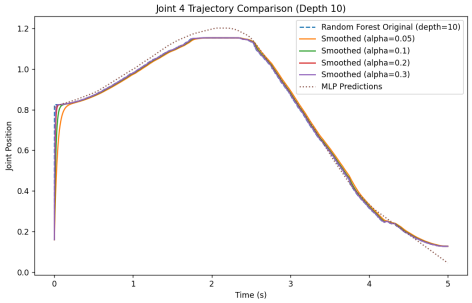


(b) Depth = 10

Figure 30: Joint 3 trajectory

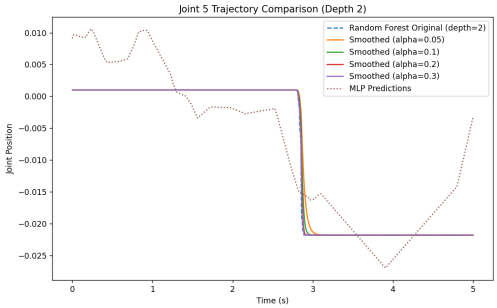


(a) Depth = 2

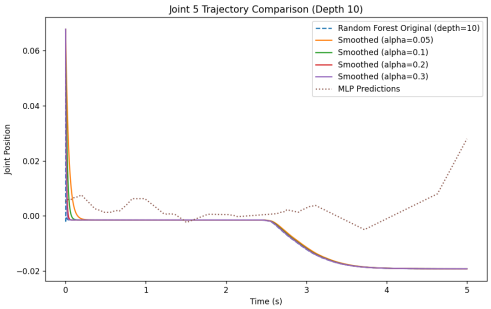


(b) Depth = 10

Figure 31: Joint 4 trajectory

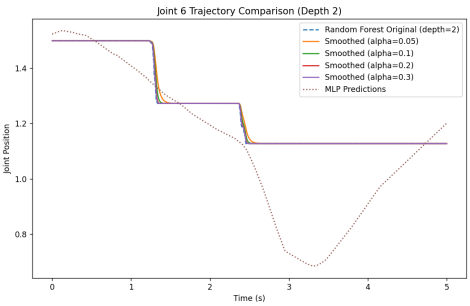


(a) Depth = 2

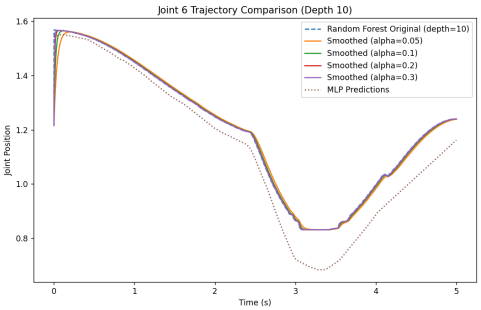


(b) Depth = 10

Figure 32: Joint 5 trajectory

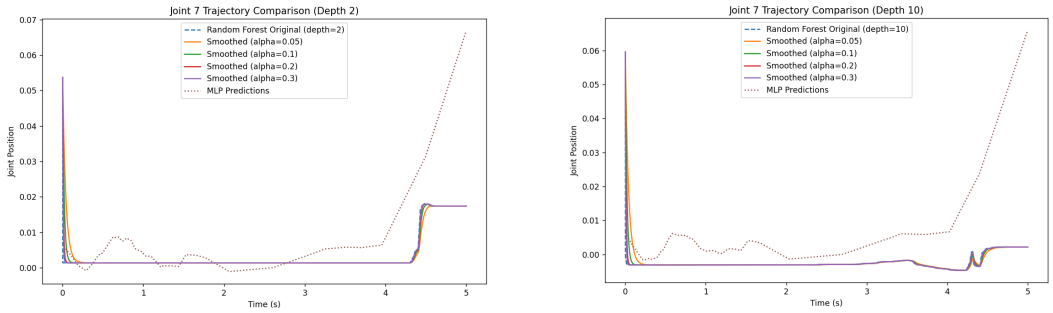


(a) Depth = 2



(b) Depth = 10

Figure 33: Joint 6 trajectory



(a) Depth = 2 (b) Depth = 10

Figure 34: Joint 7 trajectory

5 Discussion

5.1 Task 1

Task 1 experiments explored the scope for improving the performance of neural networks in complex control tasks by adjusting their structure and training parameters. By setting different numbers of hidden layer nodes, network depths, learning rates and batch sizes, the specific effects of these factors on model training loss, convergence speed and prediction accuracy were analysed. Overall, the experimental results show that:

- The network structure (e.g., number and depth of hidden layer nodes) plays a key role in the expressive power of the model, and a more complex structure helps to improve the fitting accuracy of the model.
- The training parameters (e.g., learning rate and batch size) have a direct impact on the convergence speed and stability of the model, and there is a trade-off between different settings in terms of training smoothness and computational resource usage.

These results show that with appropriate structure and parameter settings, the model can significantly improve its performance in trajectory control tasks, providing a reference for further optimisation of the control method. The following is a specific discussion for each sub-task:

5.1.1 Task 1.1: Impact of the number of hidden layer nodes on model training

Experiments were conducted to examine the effect of the ShallowCorrectorMLP model on training loss and prediction accuracy under different settings of the number of hidden layer nodes (ranging from 32 to 128 with a step size of 32). The results show that as the number of hidden layer nodes increases, the training loss gradually decreases and the prediction accuracy improves, especially at 128 nodes where the results are best. Compared with the model without using any correction, ShallowCorrectorMLP shows significant improvement in the error, indicating that increasing the hidden layer nodes effectively improves the expressive ability of the model, which is closer to the real motor data.

However, the increase in the number of nodes also significantly increases the computational complexity and resource requirements. At 128 nodes, although the accuracy is significantly improved, the computational resource overhead also increases, which affects the training efficiency. Therefore, a moderate number of nodes (e.g., 64 or 96) can be chosen for finding a balance between accuracy and resource consumption in practical applications.

5.1.2 Task 1.2: Comparison of ShallowCorrectorMLP and DeepCorrectorMLP models

The DeepCorrectorMLP model with dual hidden layers was constructed based on ShallowCorrectorMLP with different node counts in the range of 32 to 128. The results show that DeepCorrectorMLP has lower training loss and higher prediction accuracy than ShallowCorrectorMLP with the same node settings, indicating that the increase in network depth has a positive effect on expressiveness in complex control tasks.

However, the training complexity and computational requirements of the deeper models also increase significantly, especially in high node count settings, where the training time is significantly longer. Therefore, deep models are more appropriate in scenarios with high control accuracy requirements, while shallow models may be more suitable in cases with limited resources or high real-time requirements. The choice of model depth needs to consider both accuracy and computational resources to ensure that the model is suitable for the application scenario.

5.1.3 Task 1.3: Impact of learning rate on model training

Task 1.3 tested the effect of different learning rates (1.0, 0.1, 0.01, 0.001 and 0.0001) on the ShallowCorrectorMLP and DeepCorrectorMLP models. Experiments show that higher learning rates (e.g., 1.0) lead to higher initial loss fluctuations and difficulty in stable convergence, while lower learning rates (e.g., 0.0001) ensure smoothness but slower convergence. Moderate learning rates (e.g., 0.01 or 0.001) achieved a better balance between convergence speed and training stability, and performed best in terms of training loss and prediction accuracy.

The experimental results also show that the generalisation ability of the model is more desirable with moderate learning rate, i.e. the model performs better on the test set. To improve the stability of the model in practical applications, the learning rate can be dynamically reduced in the later stages of training in the future to further reduce the final loss.

5.1.4 Task 1.4: Effect of batch size on model training

The experiments analysed the effect of batch size on the training of ShallowCorrectorMLP and DeepCorrectorMLP by fixing the number of hidden nodes to 32 and setting different batch sizes (64, 128, 256 and 1000). The results show that small batches (e.g., 64) increase the volatility of the training process and improve the learning speed but are less stable, while large batches (e.g., 1000) have a smoother training process but have a slower convergence rate.

A moderate batch size (e.g., 128 or 256) strikes a balance between training stability and convergence speed. With 128 and 256 batch settings, the model is balanced in terms of training loss and testing loss, which provides faster convergence, ensures the stability of the training process, and prevents the model from falling into local optimal solutions. The choice of batch size can be flexibly adjusted according to the task requirements and resource conditions, especially in high-precision scenarios, small batches can help improve the generalisation ability.

5.2 Task 2

The experimental results of Task 2 show that MLP and Random Forest models have their own advantages and disadvantages in predicting robot joint positions. Overall, the MLP model is able to generate smooth and stable trajectories with low training and testing losses, and has better generalisation ability, which is suitable for application scenarios that require continuous and smooth control. In contrast, the random forest model can achieve higher prediction accuracy at larger depths, but it is prone to overfitting phenomenon and poorer trajectory smoothness, while the computational complexity is higher than that of the MLP model. The experimental results show that the random forest can find a certain balance between accuracy and trajectory smoothness at moderate depths, but computational resources and real-time requirements need to be taken into account in practical applications.

5.2.1 Task 2.1: Training Results of Multi-Layer Perceptron (MLP) Model

In Task 2.1, experiments were conducted to train the MLP model for predicting joint positions. The dataset was divided into a training set and a test set to evaluate the fitting effect and generalisation performance of the model. Ten target positions were randomly selected for testing in the experiments, and the results show that the MLP model has low loss on the training set and maintains good performance on the test set, indicating that the model has strong prediction ability for new data. The prediction curves of each joint position generated by the model show relatively smooth trajectories, which is important in ensuring stable control of joint motion.

In addition, experimental observations of 3D trajectories show that the MLP model is able to generate trajectories close to the target position. Although the accuracy varies from joint to joint, the smoothness of the overall trajectory meets the requirements of robot motion, verifying the advantages of MLP in terms of trajectory smoothness and continuity.

5.2.2 Task 2.2: Random Forest Model Training Results

In Task 2.2, experiments were conducted to predict joint locations using a random forest regression model and different depths of the tree (2 to 10) were tested to analyse the effect of depth on prediction performance. The results show that as the depth increases, the model's fitting accuracy on the training set improves, but there are signs of overfitting on the test set, especially at a depth of 10, where the test error increases significantly, indicating that too much depth can lead to the model overfitting the

training data and a decrease in the generalisation ability.

In terms of trajectory generation, a random forest model with shallow depth (e.g., depth of 2) generates trajectories with better smoothing but slightly lower positional accuracy; a deeper model (e.g., depth of 10), despite higher prediction accuracy, shows large fluctuations in trajectories with poor smoothing, which may lead to sharp changes in the joint positions, thus affecting the stability of the motion. Experiments showed that moderate depths (e.g., 6 or 8) achieved a good balance between accuracy and trajectory smoothness.

5.2.3 Task 2.3: Comparative analysis of models

In Task 2.3, the experiment compares the prediction accuracy, generalisation ability and computational efficiency of MLP and Random Forest models at different target positions. The results show that the MLP model has a significant advantage in trajectory smoothing and computational efficiency, especially in terms of smaller errors at different target locations, better generalisation ability, and is suitable for tasks that require stable and smooth control.

In contrast, although the random forest model with larger depth can achieve higher prediction accuracy, the trajectory smoothness is poorer and is prone to sharp changes, which affects the stability of joint motion. In addition, the random forest model with larger depth has high computational complexity and long training time, which is not conducive to applications with high real-time requirements. Therefore, the MLP model is more suitable in tasks that require the generation of smooth trajectories, while the random forest model with reasonable depth adjustment can also be used as an alternative in tasks with very high accuracy requirements.

5.3 Task 3

The experimental results of Task 3 show that the trajectory smoothness generated by different models has a significant effect on the robot control effect. Overall, the MLP model excels in trajectory smoothness and control input stability, and is able to effectively reduce tracking errors and torque fluctuations, while the random forest model, despite its high prediction accuracy at larger depths, has large trajectory fluctuations, leading to unstable control torque. Smoothing filtering of the trajectories generated by the random forest can significantly improve their trajectory smoothness and control performance, but it will introduce some response delays. In practice, the MLP model is suitable for tasks requiring high smoothness, while in scenarios with acceptable delays, random forests combined with smoothing processing can also achieve better results.

5.3.1 Task 3.1: Comparison of MLP and Random Forest Models on Trajectory Smoothness

Joint trajectories of the MLP model and the random forest model with depths of 2 and 10 were generated in the experiments, and the differences in trajectory smoothness and control effects of the different models were compared. The results showed that the MLP generated trajectories had the best smoothness and relatively small changes

in joint motion. In contrast, the random forest model with a depth of 10 has poor trajectory smoothness and exhibits significant fluctuations and discontinuities, despite the improved prediction accuracy. Comparatively, the random forest model with a depth of 2 improved in smoothness but had slightly lower positional accuracy.

5.3.2 Task 3.2: Analysis of the effect of tracking error and control input torque

In Task 3.2, the tracking errors and controller moment variations of the different models were experimentally recorded. The results show that the smooth trajectory generated by the MLP model results in less variation in the control torque and lower tracking error. In contrast, the random forest model, especially at a depth of 10, resulted in significant moment fluctuations due to the trajectory discontinuity, with frequent spikes and large variations, increasing the burden on the controller and affecting the stability of the robot motion.

5.3.3 Task 3.3: Trajectory smoothing filter effect

In Task 3.3, the experiments are conducted by applying exponential moving average filtering to the trajectories generated by the random forest to reduce the adverse effect of the unsmoothed trajectories on the control effect. The results show that the smoothing process significantly reduces the trajectory discontinuities, makes the robot joints move more smoothly, and the fluctuation of the control torque is suppressed to some extent. Compared with the unsmoothed random forest trajectory, the filtered trajectory is closer to the effect of MLP model in terms of smoothness and control stability. However, the experiment also found that the smoothing treatment introduces a certain response delay, which may have some impact on the tracking ability of rapid dynamic changes.

6 Conclusion

This project provides an insight into the effectiveness of machine learning models in robot control tasks through three tasks, analysing the performance of different models in terms of trajectory prediction and smoothness. The following is a summary of each task:

6.1 Task 1

In Task 1, the accuracy of the motor model is improved by adjusting the structure and training parameters of ShallowCorrectorMLP and DeepCorrectorMLP. The experimental results show that increasing the hidden layer nodes and depth significantly enhances the training of the models; the appropriate learning rate and batch size settings strike a balance between convergence speed and training stability. Overall, DeepCorrectorMLP outperforms ShallowCorrectorMLP in complex control tasks, with higher accuracy in motor behaviour fitting and trajectory generation, providing a reliable model base for subsequent control tasks.

6.2 Task 2

In Task 2, the performance of the multilayer perceptron (MLP) and random forest (RF) models are compared on the joint position prediction task. the MLP model has obvious advantages in terms of trajectory smoothing and generalisation ability, and is suitable for tasks requiring smooth and stable control, while the RF model can achieve higher prediction accuracy at higher depths, but the trajectory is not smooth enough, which may lead to unstable control moments. Overall, the MLP model is more suitable for tasks with higher real-time requirements, while the random forest model adjusted to the appropriate depth performs well in non-real-time scenarios with higher accuracy requirements.

6.3 Task 3

Task 3 experiments explored the effect of trajectory smoothness on the control effectiveness, especially on the feedback linearised controller. The results show that the trajectories generated by the MLP model have a high degree of smoothness, the control input torque is more stable, and the tracking error is smaller; whereas the trajectories generated by the random forest model with a larger depth fluctuate more, resulting in unstable control torque. By applying smoothing filter to the random forest trajectory, the continuity of the trajectory and control stability can be effectively improved, but a certain response delay will be introduced. Overall, the MLP model is more suitable for applications with higher smoothing requirements, while the random forest combined with the smoothing process also possesses better results in tasks with acceptable delays.

References

[1] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. John Wiley & Sons, 2020.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade: Second edition*. Springer, 2012, pp. 437–478.

[4] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.

[5] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[6] R. G. Brown, *Smoothing, forecasting and prediction of discrete time series*. Courier Corporation, 2004.

[7] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[8] R. J. Tibshirani and B. Efron, "An introduction to the bootstrap," *Monographs on statistics and applied probability*, vol. 57, no. 1, pp. 1–436, 1993.