Clearance for Manipulators

Piyush Thapar and Sumukh Porwal
Worcester Polytechnic Institute
Email: {pthapar, sporwal}@wpi.edu

***Abstract*—Robotic manipulation is a complex task, especially when it comes to accurately calculating clearance distances needed for safe and efficient motion planning. Ensuring enough workspace clearance is key to avoiding collisions and boosting the safety and reliability of robots, particularly in dynamic and unpredictable environments. However, traditional motion planning algorithms, like RRT*, often struggle when dealing with high-dimensional robotic systems due to the computational demands of continuous clearance calculations—especially for robots with multiple degrees of freedom. This makes optimizing for workspace clearance a challenging yet crucial goal in building robust autonomous systems. In this project, we explore using learning-based approaches to speed up distance computations in asymptotically optimal planners. By leveraging neural networks to predict minimum clearance, we aim to significantly improve both the convergence rate and efficiency of motion planners for complex robotic manipulators. Focusing on the Fetch robot, we developed a custom dataset of high-dimensional configurations to test how well learning-based methods can optimize motion planning performance. Our initial findings suggest that these techniques can effectively complement traditional algorithms, opening up new possibilities for more scalable and adaptable robotic systems.**

I. Introduction

Traditional algorithms, such as Rapidly-exploring Random Trees (RRT*), are widely utilized for motion planning, but they face substantial difficulties when applied to robots with high degrees of freedom. For instance, solving for the motion of an 8-DOF (Degree of Freedom) Fetch robot under tight computational constraints often proves infeasible within practical timeframes. The bottleneck lies primarily in the computation of clearance metrics, which severely impacts the performance and scalability of these algorithms. Thus, addressing this limitation is pivotal for enabling robotic systems to operate effectively in environments requiring high precision, adaptability, and safety.

A promising avenue for overcoming these challenges is the adoption of learning-based methods for distance computation. These methods aim to replace traditional clearance calculation techniques with approaches that are both faster and more accurate, leveraging advancements in machine learning. The focus of this project is to explore and implement learning-based strategies for optimizing distance calculations within asymptotically optimal motion planners, such as RRT*. By integrating these approaches, we aim to improve the convergence rates and overall efficiency of these planners, particularly for complex robotic systems with high degrees of freedom.

II. Related Work

The problem of efficient and accurate collision detection and clearance estimation in robotic motion planning has been widely studied. Traditional approaches often rely on computationally intensive distance calculations, which can significantly impact the scalability of motion planners, especially for robots with high degrees of freedom. To address these challenges, learning-based methods have been proposed to accelerate collision checking and clearance estimation.

One notable approach is Fastron [1], an online learning-based model that uses active learning strategies for proxy collision detection. Fastron aims to efficiently classify whether configurations are in collision, leveraging a lightweight model to approximate collision checks. This allows for significant speedups in motion planning, especially in dynamic environments where real-time performance is critical.

Similarly, Kew et al. [2] introduced a neural network-based method to estimate collision clearance in batched motion planning scenarios. Their method uses a deep neural network to predict clearance values, reducing the computational load associated with traditional distance calculations. By batching multiple queries, this approach enables planners to perform more efficient trajectory optimization, leading to improved performance in complex robotic environments.

Our work aligns with these learning-based methods by exploring neural network architectures to predict clearance distances for the Fetch robot. By integrating clearance estimators into the RRT* planning framework using the Grapeshot repository, we aim to improve the efficiency of motion planning algorithms for high-dimensional robotic systems. This approach leverages the strengths of prior works while focusing on optimizing performance in environments with dense obstacles and limited clearance.

III. Environment Setup

To facilitate experimentation, a simulation environment was established using the Fetch robot and a spherical obstacle. The initial and final states of the Fetch robot were strategically placed at diametrically opposite sides of the sphere. This configuration was chosen to ensure that the robot's planned path maximizes clearance from the obstacle while navigating between the defined start and goal states. Such a setup effectively mimics real-world scenarios where safe navigation around obstacles is paramount, thereby providing a suitable platform for testing clearance optimization methods.

To streamline the motion planning process, we leveraged the Grapeshot [3] repository, which seamlessly integrates the

Open Motion Planning Library (OMPL) [4] with the PyBullet visualizer. This integration not only enhances the visualization of planned trajectories but also enables rapid experimentation with various motion planning algorithms, facilitating efficient evaluation of clearance optimization strategies in complex robotic environments.

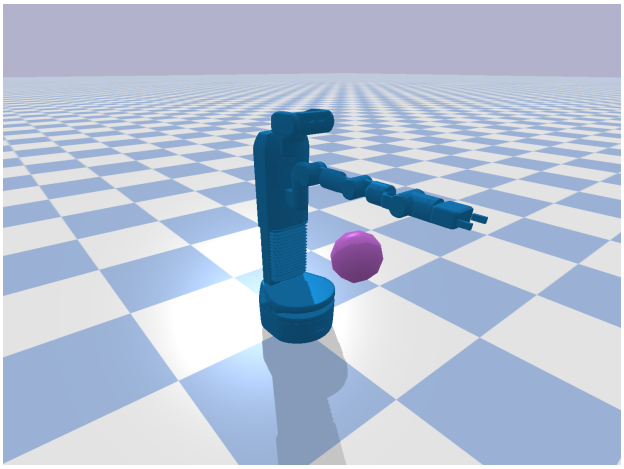


Fig. 1. Fetch robot in PyBullet Environment

IV. Implementing RRT* in Grapeshot using PyBullet

A dedicated class was developed within the existing Grapeshot codebase to implement a custom optimization objective for RRT*. This objective, termed `PathLengthClearanceObjective`, utilizes PyBullet's minimum distance function to calculate both `StateCost` and `MotionCost`. These metrics represent the minimum distance between the robot and obstacles for individual states and transitions, respectively. By integrating this objective into RRT*, the planner was enhanced to prioritize paths that maximize clearance, improving both safety and reliability in complex environments.

V. Methodology

A. Dataset Collection

A comprehensive dataset was generated over 72 hours using multiple WPI Turing Machines, resulting in 17 million unique robotic states. Each state includes annotated joint positions, obstacle attributes, and the minimum distance to the nearest obstacle, computed using PyBullet's distance function. The dataset spans three environments, detailed as follows:

1) Environment with One Obstacle: This dataset contains 10 million samples, each with 15 attributes:

- Eight Fetch robot joint states: `torso lift joint`, `shoulder pan joint`, `shoulder lift joint`, `upper arm roll joint`, `elbow flex joint`, `forearm roll joint`, `wrist flex joint`, `wrist roll joint`.
- Obstacle coordinates (`x`, `y`, `z`) and dimensions (`length`, `breadth`, `height`).
- Minimum distance to the obstacle.

2) Environment with Two Obstacles: This dataset contains 5.3 million samples with 21 attributes:

- Eight joint states as described above.
- Two obstacles' coordinates (`x`, `y`, `z`) and dimensions (`length`, `breadth`, `height`).
- Minimum distance to the nearest obstacle.

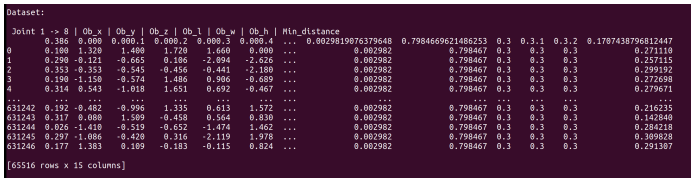
3) Environment with Multiple Obstacles (Table Scenes): For this environment, obstacle positions were fixed relative to the table. Obstacle dimensions were excluded, resulting in a simplified dataset of 1.7 million samples with 12 attributes:

- Eight joint states as described above.
- Obstacle coordinates (`x`, `y`, `z`) relative to the table center.
- Minimum distance to the nearest obstacle.

4) Summary of Datasets: To provide a clear comparison, Table I summarizes the attributes and sample sizes for each dataset:

TABLE I
SUMMARY OF DATASET ATTRIBUTES AND SIZES

Environment	Samples	Attributes	Details
One Obstacle	10M	15	Joint states + obstacle coordinates/dimensions
Two Obstacles	5.3M	21	Joint states + two obstacles' coordinates/dimensions
Multiple Obstacles	1.7M	12	Joint states + obstacle coordinates (table-relative)



Dataset:

	Joint 1 -> 8	Ob_x	Ob_y	Ob_z	Ob_l	Ob_w	Ob_h	Min_distance						
	0.386	0.000	0.000.1	0.000.2	0.000.3	0.000.4	...	0.0029819076379648	0.7984669621486253	0.3	0.3.1	0.3.2	0.1707438796812447	
0	0.100	1.320	1.400	1.720	1.660	0.000	...	0.002982	0.798467	0.3	0.3	0.3	0.271110	
1	0.290	-0.121	-0.665	0.186	-2.894	-2.626	...	0.002982	0.798467	0.3	0.3	0.3	0.257115	
2	0.353	-0.353	-0.545	-0.456	-0.441	-2.180	...	0.002982	0.798467	0.3	0.3	0.3	0.299192	
3	0.190	-1.150	-0.574	1.486	0.906	-0.689	...	0.002982	0.798467	0.3	0.3	0.3	0.272698	
4	0.314	0.543	-1.018	1.651	0.692	-0.467	...	0.002982	0.798467	0.3	0.3	0.3	0.279671	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
631242	0.192	-0.482	-0.996	1.335	0.613	1.572	...	0.002982	0.798467	0.3	0.3	0.3	0.216235	
631243	0.317	0.080	1.509	-0.458	0.564	0.830	...	0.002982	0.798467	0.3	0.3	0.3	0.142840	
631244	0.026	-1.410	-0.519	-0.652	-1.474	1.462	...	0.002982	0.798467	0.3	0.3	0.3	0.284218	
631245	0.297	-1.086	-0.420	0.316	-2.119	1.978	...	0.002982	0.798467	0.3	0.3	0.3	0.309828	
631246	0.177	1.383	0.109	-0.183	-0.115	0.824	...	0.002982	0.798467	0.3	0.3	0.3	0.291307	

[65516 rows x 15 columns]

Fig. 2. Generated dataset of robotic states

B. Neural Network Model Architecture

The core of this project involves designing and implementing a neural network model to predict the minimum distance between the Fetch robot and obstacles in its environment. The chosen architecture, named `DistancePredictor`, is a feedforward neural network tailored to handle the high-dimensional input data and produce accurate distance predictions. Below, we detail the network structure and hyperparameter settings used in the model. While the overall architecture remains the same across all three environments, the **input size varies** based on the number of attributes in the respective datasets.

- **Model Architecture:**
  The `DistancePredictor` neural network comprises a fully connected (dense) architecture with the following key characteristics:

  - **Input Layer:** The network accepts input vectors of varying sizes:
    - **Environment with One Obstacle:** Input size = 14 (8 joint values + 6 obstacle-related parameters).
    - **Environment with Two Obstacles:** Input size = 20 (8 joint values + 12 obstacle-related parameters for two obstacles).
    - **Environment with Multiple Obstacles (Table Scenes):** Input size = 11 (8 joint values + 3 coordinates relative to the table center).
  - **Hidden Layers:** The network includes 2 hidden layers with 1400 neurons each. Each hidden layer is followed by a ReLU (Rectified Linear Unit) activation function, enabling the network to learn complex nonlinear mappings efficiently.
  - **Output Layer:** The final layer consists of a single neuron that outputs the predicted minimum distance between the robot and the nearest obstacle.
  - **Sequential Structure:** The network layers are organized into a sequential block for streamlined forward computation.
- **Loss Function:**
  To handle the regression task of predicting distances, a custom loss function, `WeightedMSELoss`, is employed. This function is based on the Mean Squared Error (MSE) metric but includes an adjustable weighting factor to prioritize predictions for specific distance ranges. The weighting mechanism ensures higher penalties for errors when the true distance is less than 0.5, which is critical for maintaining safety in proximity to obstacles. For this implementation, the weighting factor is set to 30.
- **Optimizer and Learning Rate:**
  The `Adam` optimizer, a widely-used method for stochastic gradient descent, is utilized to update the model parameters. The learning rate is set to 1.7495×10^{-4}, striking a balance between convergence speed and stability during training.
- **Summary of Hyperparameters:**
  - **Hidden Layers:** [1400, 1400]
  - **Activation Function:** ReLU
  - **Output Size:** 1
  - **Loss Function:** Weighted Mean Squared Error (WeightedMSELoss) with weight factor = 30
  - **Optimizer:** Adam
  - **Learning Rate:** 1.7495×10^{-4}
  - **Input Sizes:**
    - Environment with One Obstacle: 14
    - Environment with Two Obstacles: 20
    - Environment with Multiple Obstacles (Table Scenes): 11

This architecture and the associated hyperparameters were chosen to achieve a balance between model complexity and computational efficiency, ensuring robust performance on the task of distance prediction for complex robotic scenarios across all three environments.

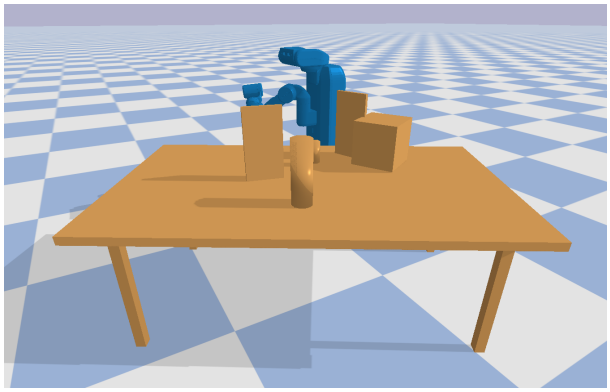


Fig. 3. Fetch Robot Planning in Table Scene (Bonus)

C. Implementation

The proposed implementation focuses on leveraging neural networks for efficient motion planning. Below is a concise breakdown of the methodologies employed:

The distance prediction models are integrated into the motion planning framework to ensure adaptability and efficiency in diverse robotic scenarios. The integration employs the following approaches:

- **Single Obstacle Environments**: The model trained for single obstacle environments is utilized for precise distance predictions.
- **Two Obstacle Environments**: A dedicated model trained for two obstacles is integrated for handling multi-obstacle scenarios effectively.
- **Dual Prediction Method**: The single-obstacle trained model is called twice, once for each obstacle, and the minimum predicted distance is computed to enhance robustness in multi-obstacle scenarios.
- **Table Scene Generalization (Bonus)**: For environments with static obstacles (e.g., a table), the model trained on fixed obstacle positions dynamically adapts to avoid all associated obstacles.
- **Path Length Optimization Objective**: The path length optimization objective is enhanced by integrating the neural network-based distance prediction model. Instead of relying on PyBullet's GJK (Gilbert-Johnson-Keerthi) algorithm for clearance evaluation, the objective uses the predicted distances from the trained neural network model.
- **Dynamic Clearance Evaluation**: Combines neural network predictions with traditional GJK method for robust safety margins.

VI. Results

The results presented in this section demonstrate the performance of the RRT* algorithm in various environments. For each scenario described below, the model was executed for a total duration of 10 seconds, utilizing RRT* to compute a viable path. The evaluation metrics used to compare the results include the path cost and the number of states in the tree after

the fixed planning time. The outcomes provide insights into the efficiency and effectiveness of the neural network-based clearance function compared to traditional methods.

A. Environment with One Obstacle

In an environment containing a single obstacle, the RRT* algorithm was executed using two different clearance functions. Figure 4 illustrates the results when using a neural network as the clearance function. The average cost of the path computed over 20 episodes was found to be **110.36**, while the tree contained **202** states after 10 seconds of planning.

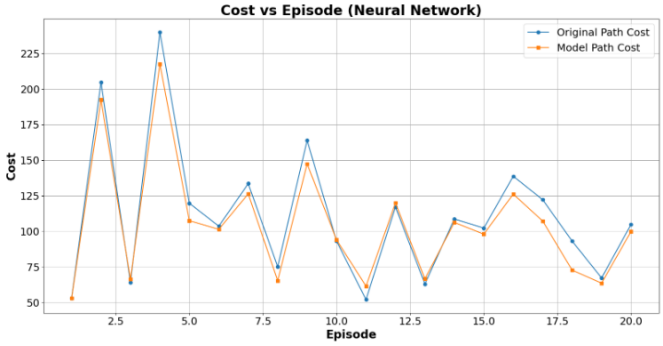


Fig. 4. Path Cost for a Single Obstacle using Neural Network Clearance Function

In comparison, Figure 5 shows the results when the GJK algorithm was used as the clearance function. The average path cost over 20 episodes was **114.98**, and the number of states in the tree after 10 seconds was **160**.

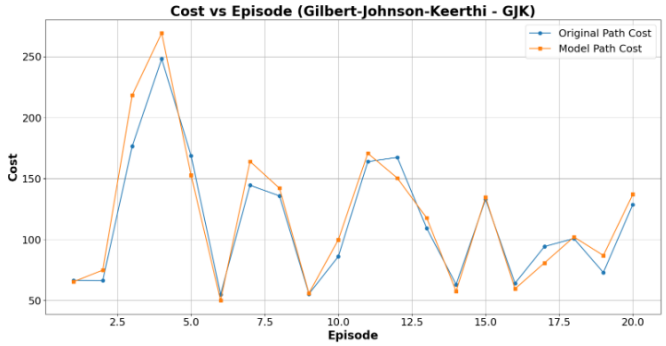


Fig. 5. Path Cost for a Single Obstacle using GJK Clearance Function

B. Environment with Two Obstacles - Double Query

When the environment was configured with two obstacles, a double-query approach was employed. Figure 6 shows the performance of the RRT* algorithm using a neural network for clearance. The average path cost calculated over 20 episodes was **88.95**, and the tree contained **153** states after 10 seconds of computation.

Similarly, Figure 7 presents the results when using GJK for clearance in the same environment. The average path cost for 20 episodes was **98.33**, and the tree contained **129** states after 10 seconds of planning.

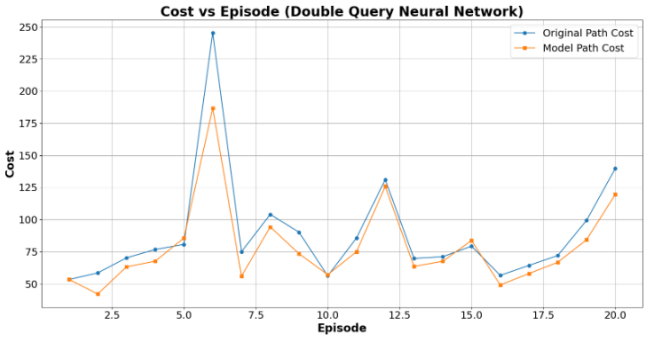


Fig. 6. Path Cost for Two Obstacles using Neural Network with Double Query

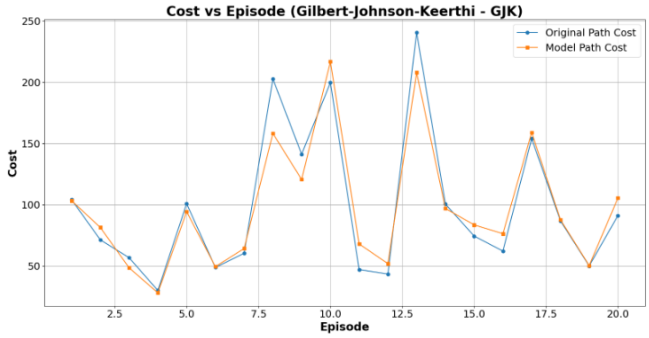


Fig. 7. Path Cost for Two Obstacles using GJK Clearance Function

C. Environment with Two Obstacles - Single Query

In the single-query scenario for an environment with two obstacles, the neural network-based clearance function achieved an average path cost of **88.78** over 20 episodes, with **203** states in the tree after 10 seconds, as shown in Figure 8.

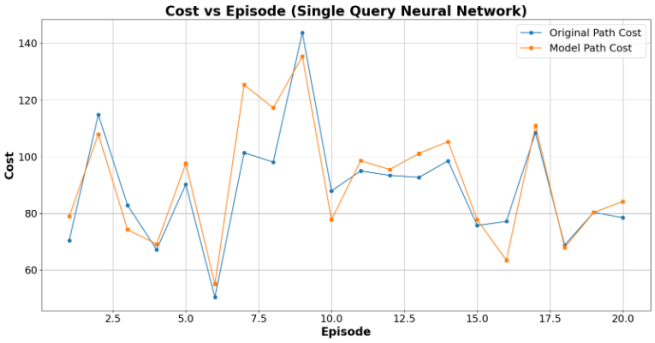


Fig. 8. Path Cost for Two Obstacles using Neural Network with Single Query

In contrast, the GJK clearance function resulted in an average path cost of **114.98** and **129** states in the tree, as illustrated in Figure 9.

D. Environment with Multiple Obstacles (Table Scene) - BONUS

The results for the table scene environment, characterized by multiple obstacles, are reserved for further elaboration. This

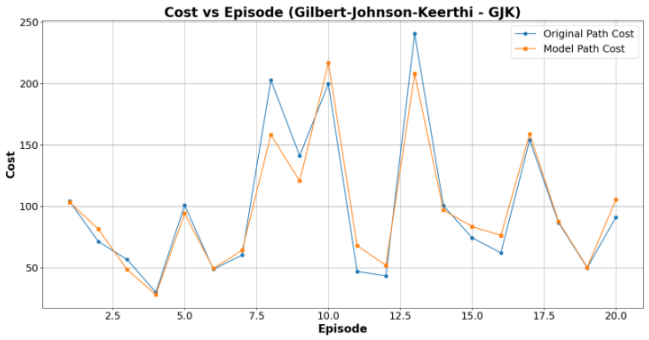


Fig. 9. Path Cost for Two Obstacles using GJK Clearance Function

environment represents a complex scenario, providing a bonus insight into the generalization capabilities of the clearance functions.

E. Model Implications

The findings from these experiments underscore the efficiency of the neural network in accurately predicting minimum distances for a substantial portion of the dataset. To enhance the model's robustness and generalization across diverse environments, several strategies were adopted:

- Increasing the diversity of training data by incorporating more complex scenarios and balancing underrepresented cases.
- Optimizing the network's hyperparameters, such as the learning rate and the size of hidden layers, to improve performance.

Below are the test results for the neural network across various cases:

1) **Environment with One Obstacle and Two Obstacles (Double Query):** Figure 10 compares the predicted and actual values for these configurations.

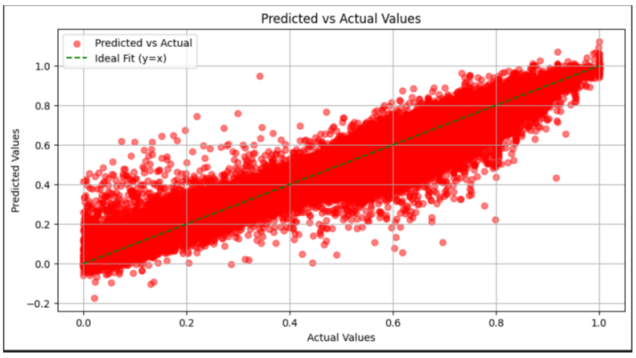


Fig. 10. Test Results for Neural Network with One Obstacle

2) **Environment with Two Obstacles (Single Query):** Figure 11 presents the results for this setup.
3) **Environment with Multiple Obstacles (Table Scene):** Figure 12 showcases the model's predictions in a complex multi-obstacle environment.

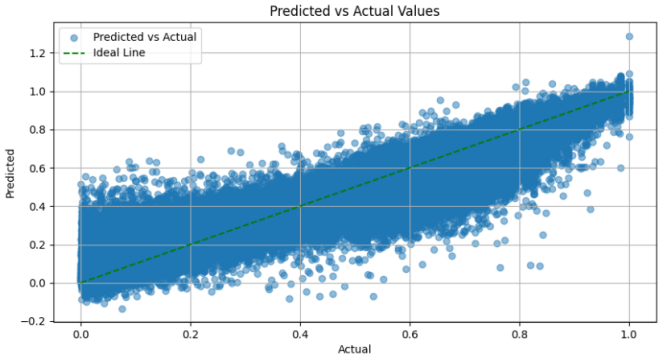


Fig. 11. Test Results for Neural Network with Two Obstacles

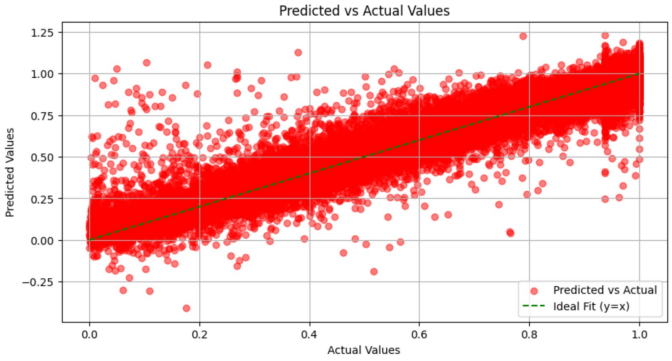


Fig. 12. Test Results for Neural Network with Table Scene

VII. Limitations and Future Work

While the neural network-based clearance function demonstrated promising results in many scenarios, it also exhibited certain limitations that affected its overall performance. One of the key challenges observed during the experiments was the model's inability to accurately predict small clearance values. This limitation became particularly evident in scenarios where obstacles were positioned very close to the potential path of the robot.

Accurate clearance predictions are crucial for generating optimal paths, as they directly influence the cost calculation and tree expansion during the RRT* planning process. When the neural network underestimated or overestimated small clearance values, it led to:

1) **Suboptimal Path Costs:** The paths generated by the neural network-based clearance function often had higher or lower costs compared to those produced by the traditional GJK algorithm. This discrepancy arose because the model failed to provide precise clearance values near obstacles, resulting in either overly cautious or overly aggressive tree expansions.
2) **Deviations from GJK Results:** In several cases, the neural network's predictions diverged significantly from the GJK clearance values, especially in environments with tightly spaced obstacles. This divergence affected the overall reliability of the model in critical planning

scenarios.

3) **Impact on Generalization:** The model's inability to handle small clearances highlighted a need for further improvement in its training process. Specifically, the training dataset lacked sufficient representation of cases with minimal clearance, which limited the model's ability to generalize effectively across diverse environments.

To address these issues, future work could focus on enhancing the model's accuracy for small clearance values. Potential strategies include:

- **Data Augmentation:** Increasing the diversity of the training dataset by incorporating more examples with small clearances to improve the model's learning capabilities in these critical scenarios.
- **Loss Function Optimization:** Modifying the loss function to prioritize accuracy in predicting smaller clearance values, ensuring that the model pays more attention to these critical cases during training.
- **Hybrid Approach:** Combining the neural network with traditional methods like GJK for cases where small clearance values are encountered. This hybrid approach could leverage the strengths of both techniques to achieve better overall performance.

Despite these limitations, the results obtained from this project provide a strong foundation for further research and development. By addressing the identified challenges, future iterations of the model can achieve improved accuracy and robustness, paving the way for more effective path planning in complex environments.

VIII. Break-down of Contributions

This project involved collaborative efforts from both team members, Sumukh Porwal and Piyush Thapar, in various aspects of the work. Below, we provide a detailed breakdown of each member's contributions:

- **Custom Environment Setup and Data Collection:** Both Sumukh and Piyush collaboratively worked on setting up the simulation environment using the Fetch robot in PyBullet. This included configuring the robot's parameters, placing obstacles, and designing the start and goal configurations to maximize clearance. Additionally, both team members contributed to the data collection process, generating the dataset required for training the neural network model.
- **Neural Network Model Development and Training:** Both team members independently developed different neural network models, experimenting with various architectures and hyperparameters. After evaluating the performance of each model, the best-performing components were merged to create an optimized model that accurately predicts clearance distances for efficient motion planning.
- **Report Writing:** Sumukh was responsible for the findings, writing, and structuring the interim report, ensuring a clear presentation of the project's objectives, methodologies, and results.
- **Presentation Preparation:** Piyush was responsible for creating a presentation summarizing the project's progress, key results, and future directions. This included designing slides and organizing content to effectively communicate the project outcomes.

IX. Conclusion

In this project, we explored the application of a neural network-based clearance function within the RRT* path planning framework and evaluated its performance across various scenarios. The results demonstrated the potential of using neural networks to streamline clearance computation, achieving competitive path costs and tree expansion rates compared to traditional methods like GJK. However, the study also highlighted significant limitations, particularly in accurately predicting small clearance values, which affected the model's ability to generate optimal paths in certain environments.

Through systematic experimentation, we identified key areas for improvement, including the need for enhanced data diversity and better loss function design. These findings underscore the importance of balancing innovation with robustness, as neural network-based methods continue to evolve for real-world applications.

Overall, this work provides valuable insights into the capabilities and challenges of neural network-based clearance functions for path planning. Future efforts should focus on addressing the identified limitations, with the ultimate goal of developing models that are both efficient and reliable in navigating complex environments. By doing so, we can unlock new possibilities for robotics and autonomous systems in increasingly dynamic and obstacle-rich settings.

References

[1] N. Das, N. Gupta, and M. Yip, "Fastron: An online learning-based model and active learning strategy for proxy collision detection," 2017. [Online]. Available: https://arxiv.org/abs/1709.02316

[2] J. C. Kew, B. Ichter, M. Bandari, T.-W. E. Lee, and A. Faust, "Neural collision clearance estimator for batched motion planning," 2020. [Online]. Available: https://arxiv.org/abs/1910.05917

[3] Elpis Lab, "Grapeshot - Motion Planning Framework," https://github.com/elpis-lab/grapeshot/tree/master, 2024, accessed: Nov. 04, 2024.

[4] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.