Dynamic Deep-RRT*

ENPM661 Planning for Autonomous Robots - Final Project

Sachin Jadhav
Master of Engineering in Robotics
University of Maryland
College Park, United States
email: sjd3333@umd.edu

Navdeep Singh
Master of Engineering in Robotics
University of Maryland
College Park, United States
email: nsingh19@umd.edu

Kashif Ansari
Master of Engineering in Robotics
University of Maryland
College Park, United States
email: kansari@umd.edu

***Abstract*—Efficient path planning is crucial in robotics for smooth movement, reduced cycle times, and safe navigation across various domains. This paper presents a novel approach that integrates deep learning techniques with traditional algorithms, focusing on emulating the renowned RRT-star algorithm using a deep neural network (DNN) and a contractive autoencoder. The DNN is trained rigorously to mimic RRT-star's sampling strategies, enhancing path planning efficiency by eliminating the need for state exploration and search strategies. The contractive autoencoder encodes workspaces, reducing input data complexity and improving learning capabilities. Validation and effectiveness demonstration are planned using simulation platforms like Gazebo, targeting real-time applicability and scalability. This research contributes to establishing a robust framework for dynamic path planning in complex environments, with potential applications in robotics, autonomous vehicles, and logistics.**

Keywords— Path planning, robotics, deep learning, RRT-star, contractive autoencoder, simulation, Gazebo, efficiency, scalability.

I. Introduction

Efficient path planning is a fundamental requirement in robotics, essential for achieving smooth movement, minimizing cycle times, and ensuring safe navigation across diverse sectors like manufacturing, logistics, healthcare, and autonomous vehicles. Our project delves into the intricate realm of training a deep neural network (DNN) to autonomously generate optimized paths tailored to specific workspaces. This endeavor involves rigorous training of the DNN to emulate the renowned RRT-star algorithm, a cornerstone in path planning algorithms, using carefully curated training examples. To further enhance training efficiency and reduce computational overhead, we have integrated a contractive autoencoder into our methodology. This autoencoder plays a pivotal role by encoding the workspace, thereby reducing the complexity of the input data and enhancing the learning capabilities of the training network. To demonstrate the effectiveness and validate our approach, we plan to leverage simulation platforms such as Gazebo, which will allow us to simulate and visualize the path planning process for the TurtleBot3 platform. The overarching goal of our project is to establish a robust and efficient framework for path planning in dynamic and complex environments within the robotics domain, aiming for real-time applicability and scalability.

II. Literature Review

Ahmed et al. introduced Motion Planning Networks (MPNet), a groundbreaking approach in motion planning algorithms that addresses the challenge of computational complexity in high-dimensional motion planning problems. Traditional methods face inefficiency as their computational demands grow exponentially with problem dimensionality, making them unsuitable for advanced robotics applications like self-driving cars. MPNet revolutionizes this landscape by presenting a neural network-based planning algorithm that directly encodes workspaces from point cloud measurements, generating end-to-end collision-free paths for specified start and goal configurations. The study evaluates MPNet across diverse 2D and 3D environments, including scenarios involving a 7 DOF Baxter robot manipulator. Results demonstrate the remarkable computational efficiency of MPNet, consistently achieving computation times of less than 1 second across various environments while maintaining generalizability to unseen scenarios. This efficiency surpasses existing state-of-the-art motion planning algorithms, marking MPNet as a significant advancement in the field of motion planning for robotics.

Dang et al. introduced Deep RRT*, merging deep learning with the Rapidly-exploring Random Trees (RRT*) method for efficient path planning in robotics. They utilized a neural network to learn a sample strategy specific to RRT*, evaluated Deep RRT* in 2D scenarios, and compared it with RRT* in terms of computational efficiency and success rate. The study revealed that Deep RRT* achieved faster path planning times but had a slightly reduced success rate near the goal area. Strategies for improvement, such as incorporating weighted entropy loss during training, were suggested. Overall, Deep RRT* presents a promising integration of deep learning with motion planning, with future work focusing on refining its

performance and addressing specific challenges encountered in practical applications.

III. Methodology

3.1 Traditional Path Planning Algorithms

Traditional path planning algorithms such A star algorithm, Dijkstra algorithm or even the sampling based algorithms such as the RRT-star algorithm require either the initial availability or the computation of different states in a workspace so as to come up with either a feasible or an optimized path. No matter if there is initial availability of the states or computation is required the conventional algorithms usually take up search time to generate paths.

```
τa ← {xstart}
τ ← ∅
Reached ← False
for i ← 0 to N do
    xnew ← Pnet(Z, τa(end))
    τa ← τa ∪ {xnew}
    Connect ← steerTo(τa(end))
     return τ
return ∅
```

Fig. 1. Pseudo-code for RRT* planner

3.2 Integration of Deep Learning Techniques

Our project aims at training a deep learning network that can mimic the RRT-star algorithm to generate paths for a robot's controller to follow, in a short amount of time as this network would not require the exploration of different states and nor it would employ any search strategies, therefore the time required to carry out these processes would be cut down. The concept behind how this would work is that the deep neural network through optimization will adjust its weights to produce paths similar to the RRT-star algorithm given a workspace. Through this method the average computational time for generating a path would go down as compared to employing traditional planning algorithms.

3.3 Contractive Autoencoder for Workspace Encoding

Dimensional workspaces can also be encoded using this method and further passed to the planning network for training.
$\Omega(\theta) = \|J_x(h)\|_2^2$

The above is the contractive loss. This is used for regularization which describes the Forbenius norm of the Jacobian matrix. The Jacobian matrix is created through the weight matrix of the last layer.

The above expression is the expression for the Means square error loss. MSE = $\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$

Following is the used equation for the loss equation which combines the MSE loss and the contractive loss term to come up with the overall loss term used for training the network.
$Loss_{\text{total}} = \alpha \cdot L_{\text{MSE}} + \beta \cdot L_{\text{contractive}}$

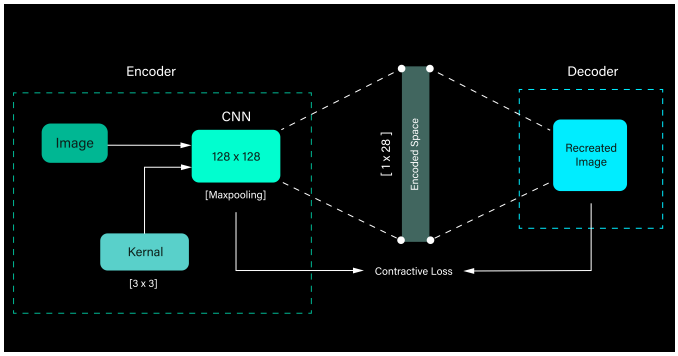

Fig. 2. Contractive Auto Encoder

The bottom set of images are original maps. The bottom set of images are recreated maps that have come up after decoding. These are created during the training process of the contractive autoencoder.

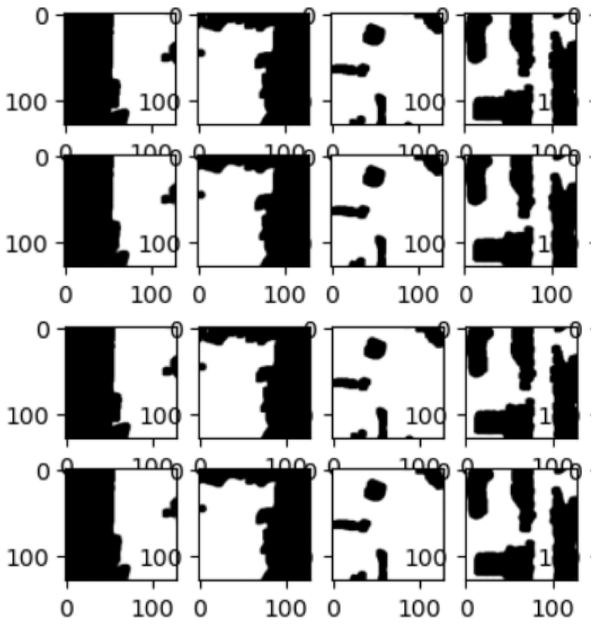

Fig. 3. Comparison of actual map and CAE output map

A. Implementation Details

3.1.1 Training Data Collection and Preprocessing

For our project training data collection has happened through the use of the RRT-star algorithm. We employ this algorithm for maps represented by binary images with the black area representing the obstacle space and the white area representing the free space and the RRT-Star algorithm plans accordingly for different start and goal positions and we essentially collect a certain number of unique paths for different workspaces and then we utilize these paths to train the planner network.

3.1.2 Neural Network Architecture

The Neural Network consists of an Input Layer, a series of Hidden Layers and an Output Layer, Each Layer consists of a PRelu Activation Function which passes only the Rectified Output to the next layer. We also introduce dropout regularization on each layer in order to force the Loss Function to identify important features in patterns of data.

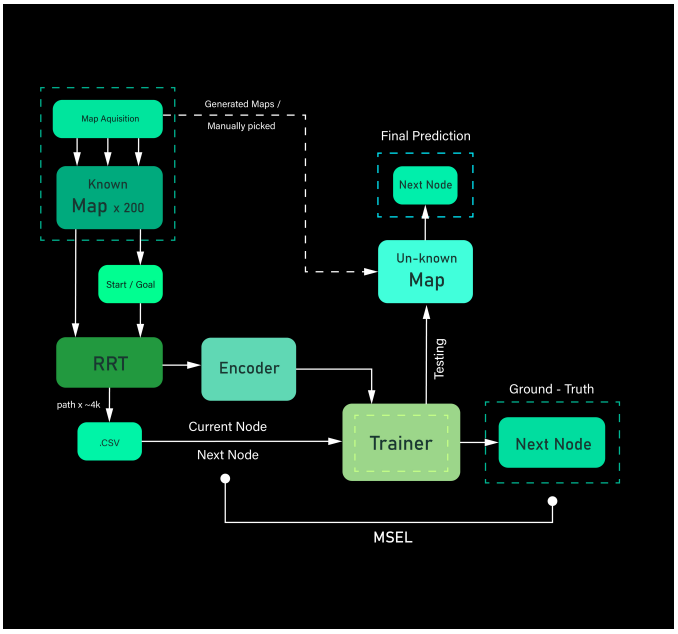

Fig. 4. Neural Network Flow Diagram

3.1.3 Training Process

The Networks receive the encoded image maze as a vector along with the current and goal points to output the next point in the path. This predicted path is compared with the ground truth values collected before and a Mean Squared Error (MSE) is calculated based on which the weights of the network are updated through back propagation. This training process is carried out till the training data lasts and the weights are updated and stored accordingly. These weights can then be used to predict the next step for the test data.

IV. Results and Analysis

4.1 Performance Evaluation Metrics

Algorithm	Succ. Rate (%)	Tree Size	Time (s)
RRT	97.4	149.87 ±150.01	0.60 ±0.72
Deep RRT*	90.1	74.28 ±126.93	0.36 ±1.19

TABLE I
Comparison with RRT.

As per the paper, even though the success rate of Deep RRT* seems a bit less than RRT* but is reasonably well and practically viable. Also the time complexity of the algorithm is reduced multifold by more than 65% which is a remarkable improvement considering that in practical applications quick trajectory generation is necessary.

4.2 Comparison with Traditional Algorithms

DeepRRT-Star incorporates neural networks to learn continuous representations of the state space, allowing for more nuanced decision-making in continuous domains. Deep learning techniques can potentially handle more complex and high-dimensional spaces by leveraging generalization capabilities learned from training data. The main advantage of applying a Neural Network for path planning is that it decreases the time complexity by a huge percentage which has a huge advantage in practical applications.

4.3 Simulations and Implementations

Below are the comparison of the performances between the deep RRT* and the traditional RRT*

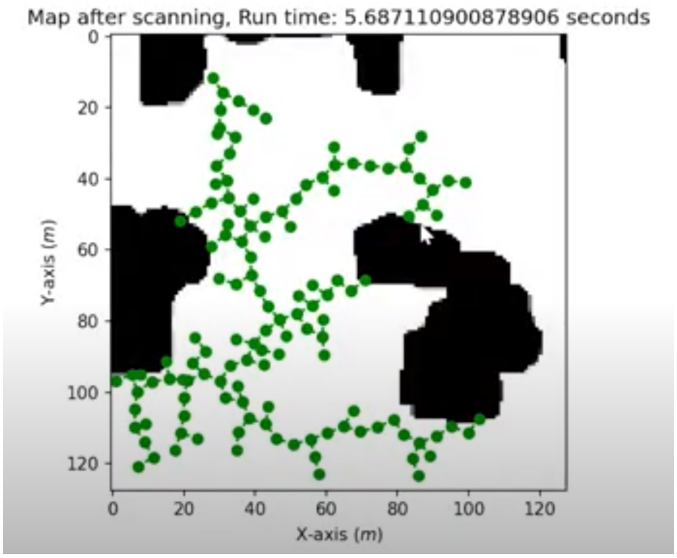

Fig. 5. Static RRT* implementation

Static Simulations shows the comparison for time complexity between traditional RRT and the Deep RRT Star that we

have created. A time duration flashing on the top of the plot shows the reduced computation time for generating the path.

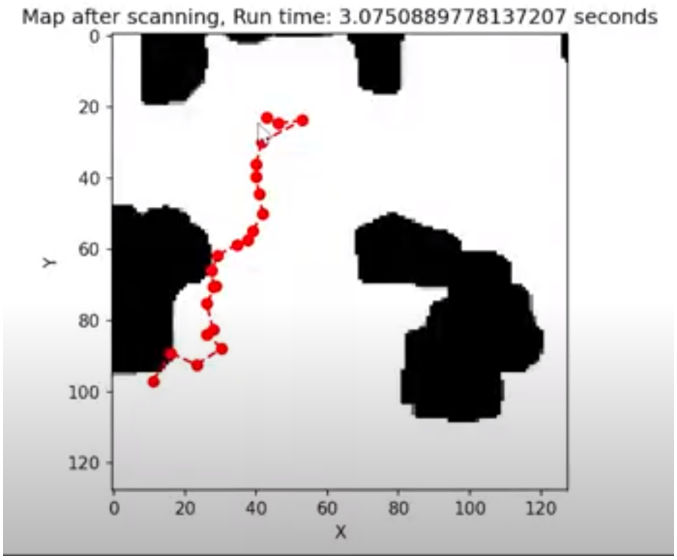

Fig. 6. Static Deep-RRT* implementation

Fig.4 and Fig.5 above shows Static Simulation comparison for time complexity between traditional RRT and the Deep RRT that we have created. A time duration flashing on the top of the plot shows the reduced computation time for generating the path.

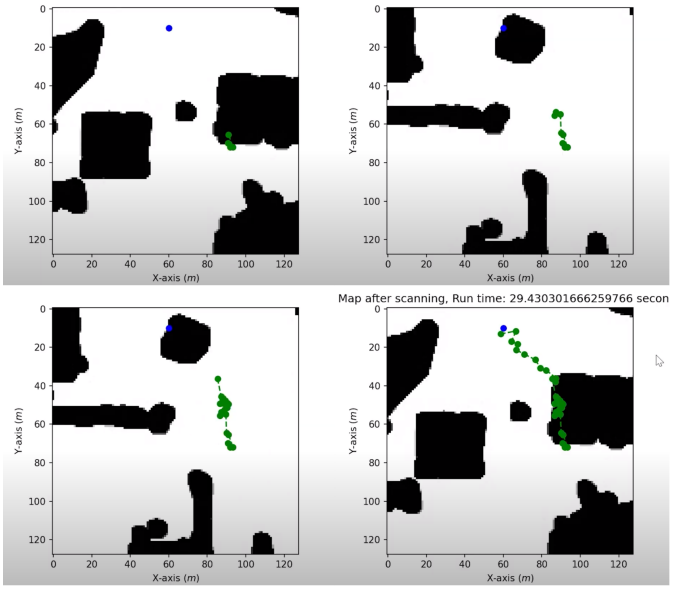

Fig. 7. Dynamic implementation

Above is the figure showing a implementation of the Deep RRT* in a Dynamic environment where obstacles have been found changing their location and their nature with the time as the environment changes. (Paths may be found slightly overlapping with the obstacle this is because the shaded regions are considered without any clearances and MSE is subjected for further improvements

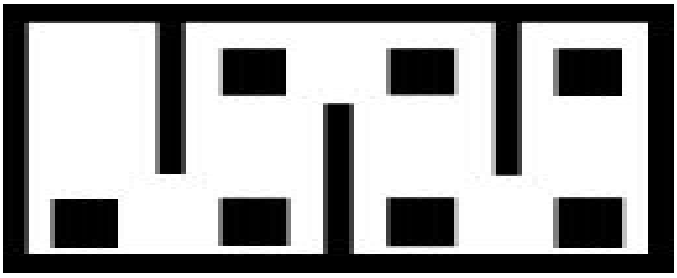

Fig. 8. Reproducing given binary map through CAE

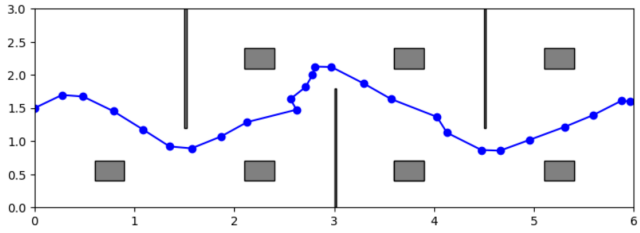

Fig. 9. Producing results on unknown map for simulation

Producing Testing Data and Implementing the trained algorithm on the test data. Based on the accuracy and consistency of the test data results the network can be further trained as and if required.

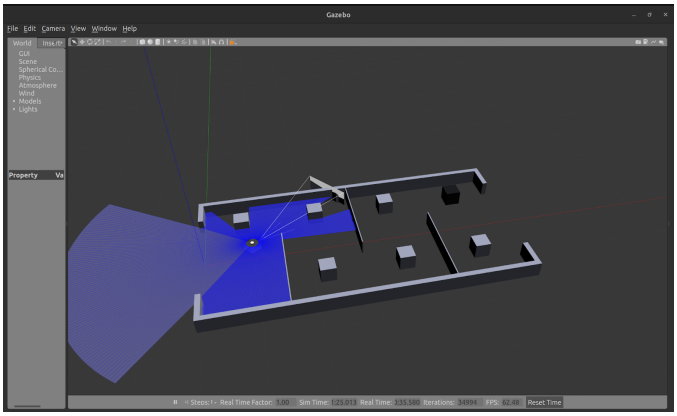

Fig. 10. Simulation in Gazebo Environment

TurtleBot Navigation Enhancement: Implementing our deep learning-based path planning algorithm on the TurtleBot platform can significantly enhance its navigation capabilities. The optimized paths generated by our approach will enable TurtleBot to navigate complex environments efficiently and safely, paving the way for applications in indoor navigation, warehouse automation, and service robotics.

Dynamic Environment Adaptability: One of the strengths of our approach is its ability to adapt to dynamic environments. Future work can focus on real-time adaptation and replanning strategies, allowing TurtleBot to respond effectively

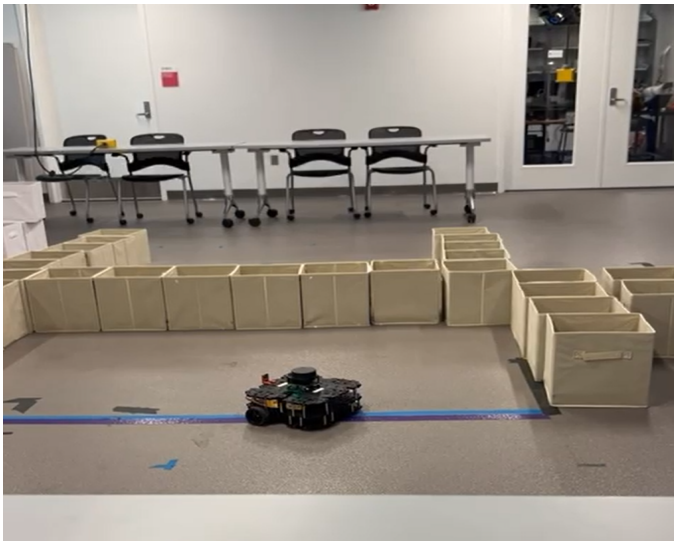

Fig. 11. Implementation on real robot (turtlebot3)

to dynamic obstacles, changes in the environment, and unexpected events. This would be particularly useful in scenarios such as crowded spaces or environments with moving objects.

4.4 Computational Complexity

This section formally highlights the computational complexity of the proposed method. Neural networks are known to have online execution complexity of O(1). Therefore, the execution of Algorithm 1 will have a complexity no greater than O(1).

4.5 Challenges and Limitations

Despite the advantages, AI path planning also faces challenges and limitations. One significant challenge is the computational complexity involved, especially with deep learning-based approaches that require large amounts of training data and computation resources. Ensuring the safety and reliability of AI-driven path planning algorithms is another critical challenge, as errors or biases in the training data can lead to unexpected behaviors or unsafe decisions in real-world settings. Additionally, the interpretability of AI-generated paths can be challenging, making it difficult for humans to understand and trust the decisions made by AI planners. Finally, integrating AI path planning into existing robotics systems and workflows can be complex and require careful consideration of compatibility and scalability issues.

V. Conclusion

In conclusion, our project addresses the critical need for efficient path planning in robotics by integrating deep learning techniques with traditional algorithms like RRT-star. Through rigorous training of a deep neural network (DNN) to emulate RRT-star, along with the use of a contractive autoencoder for workspace encoding, we aim to significantly reduce computational overhead and enhance path generation efficiency. Leveraging simulation platforms like Gazebo, we plan to validate our approach on the TurtleBot3 platform, demonstrating its effectiveness in dynamic and complex environments. Moving forward, future work will focus on producing testing data and implementing the trained algorithm on TurtleBot for navigation enhancement. Additionally, we aim to enhance adaptability to dynamic environments, enabling real-time response and re-planning strategies for improved performance in challenging scenarios.

VI. References

A. H. Qureshi, A. Simeonov, M. J. Bency and M. C. Yip, ”Motion Planning Networks,” 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 2019, pp. 2118-2124, doi: 10.1109/ICRA.2019.8793889.

Dang, X., Chrpa, L., & Edelkamp, S. (Year). Deep RRT*. In *Proceedings of the International Symposium on Combinatorial Search* (Vol. 15, Issue 1). Czech Technical University in Prague. DOI: https://doi.org/10.1609/socs.v15i1.21803