Efficient Collision Detection and Trajectory Validation in Static Scenes Using Deep Neural Networks[1]*

Solomiia Liaskovska[1,2*,†], Vitalii Mil[1,†], Olga Duran[2†], Yevgen Martyn[3†], Oleksandr Pshenychny[1†]

[1] *Department of Systems of Artificial Intelligence, Lviv Polytechnic National University,Lviv,79905, Ukraine*
[2] *Faculty of Mechanical Engineering, Kingston University, Kingston Upon Thames, KT1 1LQ, London, United Kingdom*
[3] *Department of Project Management, Information Technologies and Telecommunications, Lviv State University of Life Safety, Lviv,79007, Ukraine*

Abstract

In this research, we introduce an approach to collision detection and trajectory validation in static scenes by harnessing the power of deep neural networks (DNNs). Traditional methods have frequently encountered challenges with high computational and memory demands, limiting their real-time applicability, especially in robotics and simulation environments. Our newly developed model addresses these constraints by delivering a swift and memory-efficient solution. Leveraging data from a Bullet physics simulator for training, our approach not only preserves high accuracy but also achieves significant improvements in both speed and memory usage compared to conventional simulation techniques. This innovation marks a substantial step forward, offering promising avenues for advancing real-time applications in robotics and simulation domains, potentially revolutionizing how these systems operate and perform. Furthermore, the adaptability and scalability of our model suggest broader applications beyond the current scope, paving the way for future research in related fields. The integration of deep learning techniques could lead to more intelligent and responsive systems, enhancing overall system reliability and performance.

Keywords

Robotic trajectories, validation, collision detection, deep neural networks

The validation of robotic trajectories in complex static scenes stands as a pivotal yet intricate task in the realm of robotics and automation. As robots become increasingly integrated into various sectors, from manufacturing to healthcare, ensuring the accuracy and safety of their movements becomes paramount. Long paths, in particular, present significant challenges due to the intricate interactions with the environment. Traditional methods employed for trajectory validation, such as dynamic simulation, grid lookup tables, and simplification strategies, have shown limitations. These constraints often manifest as high computational loads, extensive

SMARTINDUSTRY-2024: International Conference on Smart Automation & Robotics for Future Industry, April 18 - 20, 2024, Lviv, Ukraine
* Corresponding author.
† These authors contributed equally.

solomiya.y.lyaskovska@lpnu.ua (S. Liaskovska); vitalii.mil.kn.2021@lpnu.ua (V. Mil); O.Duran@kingston.ac.uk (O. Duran); evmartun@gmail.com (Y. Martyn); oleksandr.y.pshenychnyi@lpnu.ua (O. Pshenychny);
0000-0002-0822-0951(S. Liaskovska); 0009-0002-7779-1640 (V. Mil); 0000-0002-1397-6032 (O. Duran); 0000-0001-9095-7057 (E.Martyn); 0000-0001-8823-7472 (O.Pshenychny);

memory requirements, and compromises in accuracy, making real-time application infeasible in many scenarios[1, 2]. The advent of deep learning technologies offers a promising avenue to address these challenges. Deep neural networks (DNNs) have demonstrated remarkable capabilities in handling complex tasks across diverse domains, including computer vision, natural language processing, and now, robotics. By leveraging the power of DNNs, it becomes possible to devise more efficient and accurate solutions for collision detection and trajectory validation. In this context, this paper introduces a deep learning-based solution tailored to navigate the complexities of validating robotic trajectories in static scenes. Our approach aims to predict collision states and determine nearest geometry distances with heightened efficiency and a reduced memory footprint. By doing so, we aspire to elevate the standards of trajectory validation, paving the way for safer and more reliable robotic operations in complex environments[3-6]. This research contributes to the ongoing efforts in advancing robotics by harnessing the potential of deep learning, ultimately aiming to revolutionize the field's capabilities and expand its horizons[7,8].

1. Analysis of literary sources

The idea of Neural Radience Fields or NeRF is shown in [1]. The main point there is volumetric representation of some scene, also stored in DNN. The model in NeRF is more complex, with 5 degrees of freedom, where X, Y, Z is the position and two angles representing the direction ray. However it also needs much more complex architecture and longer training time.

This research presents a novel approach to collision detection and trajectory validation in static scenes by employing deep neural networks (DNNs).

Traditional methods for collision detection and trajectory validation face challenges like high computational and memory demands[4].

These challenges limit their real-time applicability, especially in robotics and simulation environments. The research introduces a model that harnesses the capabilities of DNNs[9,10].

This model aims to overcome the computational and memory constraints of traditional methods[11, 12]. Data from a Bullet physics simulator is used for training the DNN, ensuring high accuracy while improving speed and memory usage. The proposed approach offers a swift and memory-efficient solution. It achieves significant improvements in speed and memory usage compared to conventional simulation techniques [13,14,15]. The innovation presented in the paper represents a substantial advancement in real-time applications for robotics and simulation.

The adaptability and scalability of the model suggest potential applications beyond the current scope, indicating its versatility and broad applicability [15].

The paper's approach to utilizing DNNs for collision detection and trajectory validation is innovative. By leveraging the power of deep learning, the research aims to address the limitations of traditional methods, potentially leading to more efficient and reliable systems.

The improvements in speed and memory usage can have significant practical implications. Faster and more memory-efficient collision detection and trajectory validation can enable more responsive and reliable robotic systems, which is crucial for real-time applications.

The adaptability and scalability of the proposed model open up possibilities for future research in related fields. The integration of deep learning techniques could lead to the development of more intelligent and responsive systems, enhancing overall system performance and reliability.

This research presents a promising approach to tackling challenges in collision detection and trajectory validation in static scenes. The integration of deep learning techniques offers potential advancements in the field of robotics and simulation, paving the way for future innovations and research.

2. Methods and means

Our research focused on the validation of robotic trajectories, a critical task that requires rigorous optimization for effective implementation.

2.1. Description and justification of selected machine learning models

Motivated by the necessity to represent the room geometry in a more compact and efficient manner, we sought to approach the problem from a novel perspective. Given the importance of ensuring safe robot navigation and the need to choose safer trajectories, our research concentrated on these aspects. For our work, we utilized the Bullet Physics Library, encapsulated within the PyBullet wrapper, and PyTorch for training neural networks due to its popularity and our familiarity with it. We set certain constraints for our experiments: the scene and robot were predefined, and only one scene, one robot, and one model were considered for simplicity. Bullet Physics is a modern physics engine that operates in three-dimensional space. It is provided with open-source code, which allows for easy analysis and study. The physics engine is designed for realistic simulation of object collisions. It is a set of tools that allows for the use of pseudo-realistic behavior of complex objects for gaming, engineering, or scientific purposes. Fig. 1 describes the main actions performed by the engine. The world is updated using the stepSimulation command. This command checks the update frequency of the world through OS tools, then determines the kinematic positions and velocities of all objects, applies gravity, and determines the substeps necessary for Bullet to function correctly.

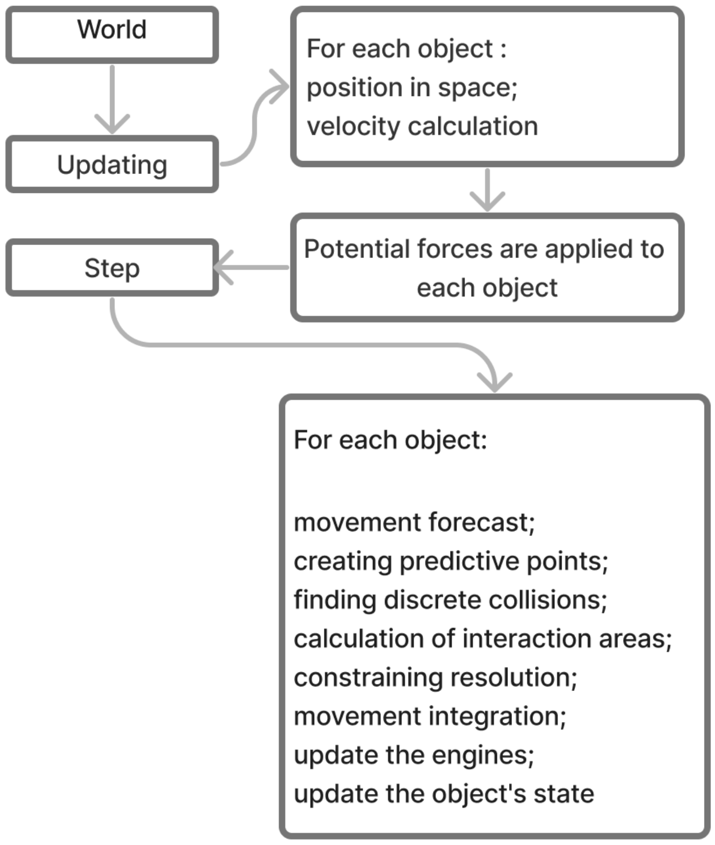

Figure 1: Random forest functional model playback.

Steps are the number of specified time intervals that fit into one Simulation Step. If the world updates quickly, there are fewer Steps, and vice versa. In other words, we have an infinite simulation loop (step) within which its small Steps rotate. Thus, a piecewise simulation is performed based on some system tick, its internal time.

Within each step, a set of actions is performed that determine the position of the object. Damping of objects is necessary partly due to the discreteness of the simulation: issues with infinitely small speeds need to be resolved, for example. Movement prediction of points is carried out by integrating the velocity of the mesh points of the object; subsequently, this data is used for further calculations. will get acquainted with the first classes. The base **class btCollisionWorld** is the foundation of Bullet. On top of this class, the **btDiscreteDynamicsWorld** class is implemented, which describes the specific implementation of collision physics. Inside the Discrete World (DW), auxiliary classes **btClosestNotMeConvexResultCallback** and **btSingleSweepCallback** are implemented, which implement the interaction between the DW and other objects.

The **btRigidBody class** is essential for physical bodies interacting within the engine.

The **btDbvtBroadphase class** implements the main stage of interaction analysis, using two dynamic hierarchies/trees of volume constraints based on the **AABB** (axis-aligned bounding box) algorithm. One tree is used for static objects, and the other is for dynamic ones: objects can move from one hierarchy to another. In principle, the entire interaction analysis is distributed by the author into **BroadPhase and NarrowPhase**.

The **btDbvt class** is closely related to the previous one: it represents a fast-acting tree for computing interaction volume constraints based on an axis-aligned parallelepiped. Otherwise, it looks for interacting object pairs. It can quickly insert, delete, or update nodes. Nodes can dynamically rotate around the hierarchy, changing the structure of the main topology. The **btContinuousConvexCollision class** (belongs to Narrow Phase) performs time estimation for collision during angular and linear movement. Its main task is to maintain consistent motion. Currently, it uses the idea of conservative motion by Brian Mirtich, with the future addition of the Minkowski algorithm (already implemented in the library) planned. The **performDiscreteCollisionDetection algorithm** is executed next after the DPS, and it is presented in the picture below. It involves the well-known **classes btCollisionWorld** and **btDbvtBroadphase**, as well as several new ones.

Next, the perform **DiscreteCollisionDetection** and **calculateSimulationIslands** functions are executed. The **performDiscreteCollisionDetection** function is intended for analyzing the interaction of objects of different types according to predefined algorithms: for sphere-sphere interaction, its own algorithm, etc. All these algorithms are predefined.

The process of loading a mathematical model looks as follows:

1. Creating the model in Blender;
2. Exporting to an fbx file;
3. Loading into Static Mesh through the UE editor;
4. In the target object **AActor**, we add the model to **UStaticMeshComponent**;
5. Next, we unload it into **btCollisionShape**, more precisely into **btConvexHullShape**;
6. Then we associate **btRigidBody** with the obtained shape.

I'd like to draw attention to the following point: two different but coinciding in coordinates points of the object arise, **UStaticMeshComponent** and **btCollisionShape.** They coincide in this example for clarity but remain completely independent entities. The stage is set from the beginning, all the geometry is known in advance. The robot is also defined from the beginning, at the moment no moving parts are provided. One scene - one robot - one model. To start with, a function was chosen that the neural network will be trained on.

$$f(x, y, theta) -> dist,$$

to scene geometry if not in collision and 0 if collision.

This function is continuous and strictly non-negative, which allowed us to employ certain techniques (more on this later). x, y, θ represent the robot's absolute position in the world and its rotation around the Z-axis.

This function was chosen for the following reasons:

It is important to be able to compare the "safety" of two positions, as in the standard implementation through the same simulation; this would take a lot of time. Hence, it usually limits itself to merely checking "collision or no collision."

The first step was dataset generation. PyBullet has built-in functionality that allows checking the distance between objects, so it was utilized. The data generation speed was approximately 2-10 thousand values per second for a relatively simple scene with a basic robot (just a cube).

It's also worth mentioning that Bullet lacks parallel data processing capabilities, necessitating the creation of separate threads/processes to improve efficiency.

For the neural network architecture, a standard DNN with four hidden layers of sizes 256, 256, 256, and 64 was chosen. These values can vary, both decreasing and increasing, depending on the speed/complexity requirements. In practice, it was observed that there is no single "fixed" architecture.

After each layer, there is a ReLU activation layer, even after the last one, which usually doesn't happen. This might be possible because the target function is very similar to ReLU, as the distance linearly increases from the edges of the geometry, and elsewhere, it equals 0. Input parameter transformations were also made:

1) $\theta \rightarrow sin(\theta), cos(\theta)$

2) Encoding input parameters with high-frequency functions allowed the model to train faster and perform better in areas with strong variations in the target function:

np.sin(y),
np.cos(y),
*np.sin(13 * x + 17 * y),*
*np.sin(19 * x - 15 * y),*
*np.sin(23*x+y),*
*np.sin(29*y-x)*

2.2. Indicators of model performance evaluation

Assessing the performance of a machine learning model is a crucial aspect of model development. Therefore, it is essential to understand how the success of a machine-learning model can be gauged.

Evaluation metrics are tailored to specific machine learning tasks, with various metrics available for classification problems. Employing diverse metrics to evaluate performance enables the comprehensive assessment of a model's efficacy before deploying it for real-world data processing.

Since the task at hand is essentially a function regression task, it is possible to visualise accuracy of the model with plots. In this case 2D heatmaps with fixed theta (rotation of the robot) will do great job at showing the accuracy of the trained model.

The confusion matrix allows you to tabulate the number of correct and incorrect predictions made by the model compared to the actual classifications in the test set, showing the types of errors that occur. This matrix evaluates the model's effectiveness in classifying test data with known true values, typically in an *n x n* format, where *n* represents the number of classes. It is constructed after the test data has been predicted. In our case the confusion matrix will be 2 x 2 (Collision/No collision)

There are four possible outcomes of classification prediction:

- True positive outcomes (TP). These are actual positive results that are predicted to be positive.
- False negatives (FN). These are actual negative results that are predicted to be negative.
- False positives (FP). These are actual negative results that were predicted to be positive (type one errors).

- False negatives (FN). These are actual positive results that were predicted to be negative (type two errors).

3. Numerical experiments

3.1. Description of the training data

Two scenes were created for the training (Figure 2).

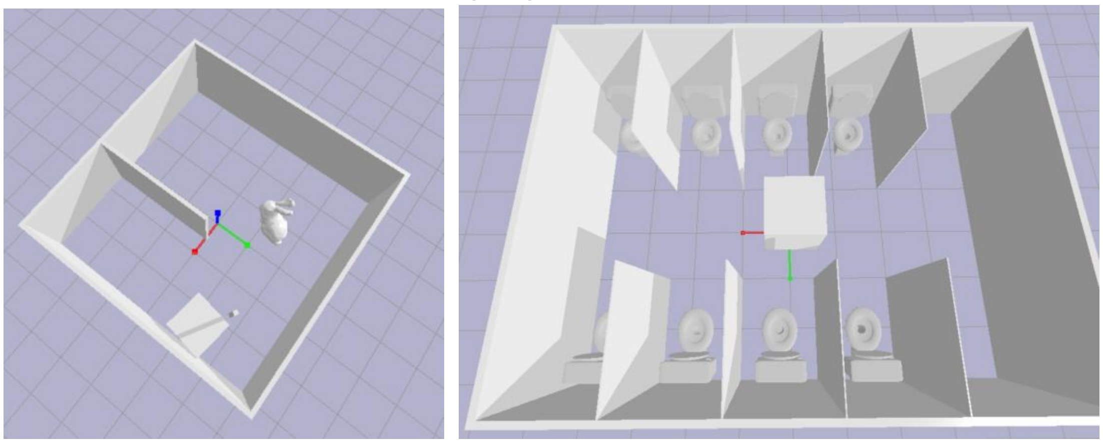

Figure 2: The cube is presents as a Robot

5000000 data entries were generated of the following structure:
X,*Y*, *θ*, Result. *X* and *Y* are the absolute coordinates of the robot center. *θ* is the measure of clockwise rotation of the robot along the Z coordinate.
This means that the robot has three degrees of freedom in our simulation, which are usually the most important for ground based robots which can't travel along *Z* axis and can't rotate their *XY* plane. *θ* is given in radians.

Result is the measure of distance from the robot to the nearest geometry of the scene, excluding the floor. The floor is omitted since it is expected that robot contacts the floor at all times and this will not give any meaningful information. Figure 2 also shows the scenes with the floors omitted. This measurement is in meters. Result field in 0 if the robot is in collision with the world in this respective position. Since the data is generated, the model was trained on all of it. It was not trained on the grid-like visualisation shown later.

3.2. Results of data analysis and preliminary processing

The standard approach to dealing with angle inputs, is to return their sin and cos values. Having made a few tries of training the model on just (*X*,*Y*, *sin(θ)*, *cos(θ)*), it was found that raw position may not give effective accuracy, and the model struggled with correctly capturing some regions where the border between Collision and Not collision lied. To avoid unnecessary model complication, It was decided to add some high frequency features to the model input. The chosen features were

*sin(y), cos(y), sin(13 * x + 17 * y), sin(19 * x - 15 * y), sin(23*x+y),sin(29*y-x)*

This input does not give the model any new data, however including this helped the network train faster and be more precise. These parameters were chosen mostly at random and were not optimized for.

3.3. Evaluation the efficiency of the trained model.

Plots of the target function for the scenes shown in Figure 2 are provided below.

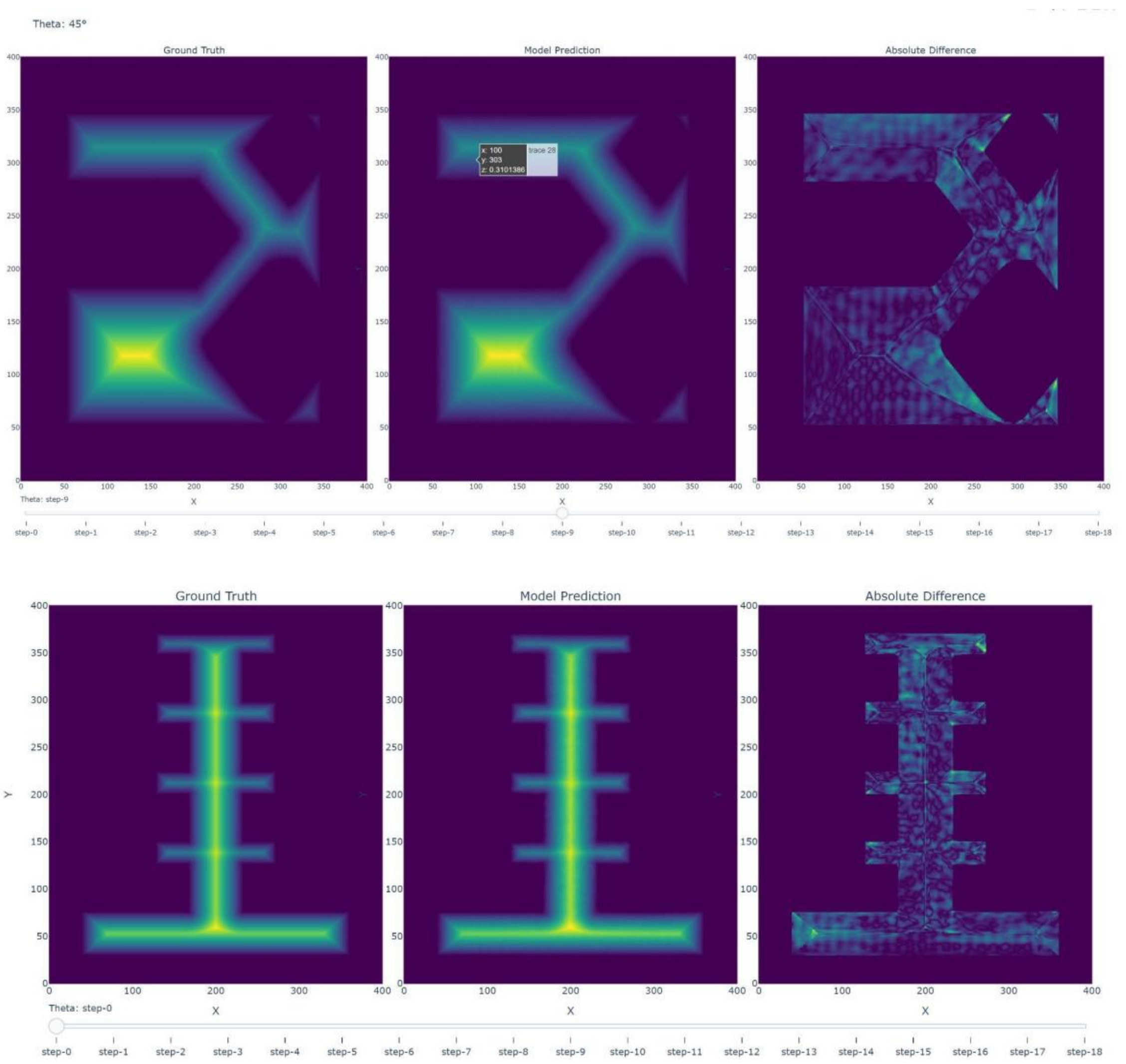

Figure 3: Plots of the target function for the scenes shown in Figure 2.

First one is shown at angle $\theta = 45$, second one is shown at angle $\theta = 0$. Three plots are Ground Truth: taken from a simulation, Model prediction: predicted values from the model, Absolute difference: the absolute difference between first two plots. Purple values are 0. Yellow values are the largest and on first two plots are around 1.5 meters. The maximum absolute difference is around 2 centimeters.

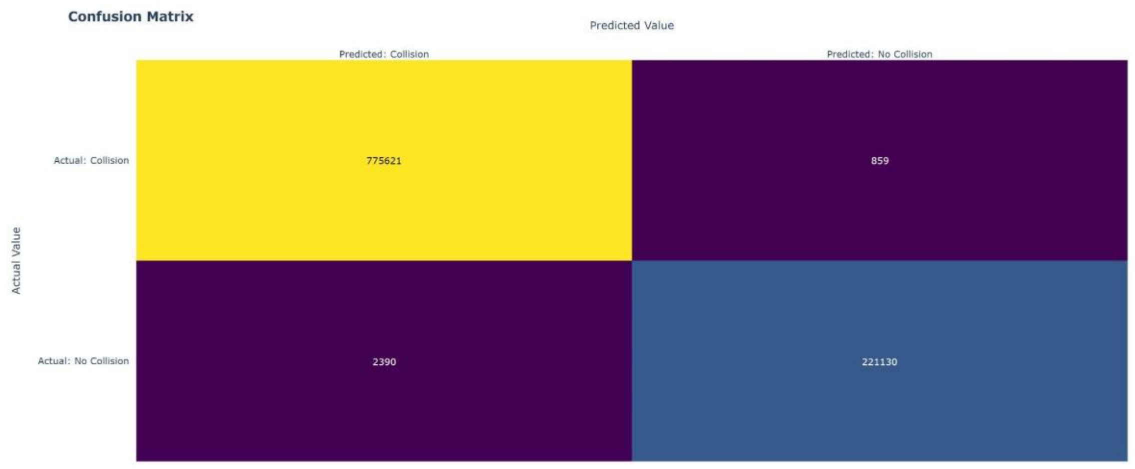

Figure 4: The confusion matrix for the trained model showing Collision vs No collision classification.

The model could have been trained explicitly for this purpose, however it would lose the ability to compare different trajectories based on their distance. This confusion matrix shows 99.8% accuracy. This is the raw result without any finetuning. The amount of FN (Actual Collision predicted as No collision) can be reduced to almost 0, by introducing some Threshold.

4. Discussion of study results

Results of this study show a promising lead into the field of compact scene representation and into the field of autonomous mobile robots. There is a lot of room for improvement regarding the architecture, feature engineering, training processes etc.

The main problem with this particular implementation is rather long training time (around 1 hour) from start to finish, which can disqualify it from some realtime online tasks. However it is most likely possible to reduce the training time significatly to the order of a couple minutes given adequate hardware and some training refining.

The main advantage for this model is its speed and small memory footprint. The ability to quickly construct and validate trajectories is the key to robot relocation tasks. This can get very hard for a complex scene with a complex robot if employing a traditional simulation method.

Also the obvious drawback is that the scene needs to be static in order for this to work, which excludes a lot of possible use cases. A solution to this problem is a combined approach, where only the dynamic elements of the scene are simulated (doors for example).

Table 1.
Comparison of the different methods for this particular task

	Simulation	Lookup table	Our method
Speed	~4000 it/s	Extremelly fast	~100000 it/s
Memory	Only runtime	Extremelly large	~2MB
Dynamic scenes	Yes	No	No
Quantisation	No	Yes	No
Online	Yes	No	No

Table 1 shows a comparison of different methods on a range of properties, and it shows that each method is better than other at some things and each can be leveraged depending on the specifics of the task.

The lookup table is the fastest, but it needs an extraordinary amount of memory if the resolution is high or the scene dimensions are large. It also suffers from quatisation, which leads to smaller precision. Simulation is the most versatile, since it can support dynamic scene updates and does not need to precalculate anything. However the speed can be not enough in some cases. Our method can provide higher speed than a simulation at the cost of training time and exclusion of dynamic items from the scene.

5. Summary and Conclusion

In this research proposed an approach to collision detection and trajectory validation in static scenes using deep neural networks (DNNs). The limitations of traditional methods, such as high computational and memory demands, have constrained their real-time applicability, particularly in robotics and simulation environments. However, our newly developed model offers a swift and memory-efficient solution, addressing these constraints effectively. By leveraging data from a Bullet physics simulator for training, our approach not only maintains high accuracy but also achieves significant improvements in both speed and memory usage compared to conventional simulation techniques. This innovation represents a substantial step forward, presenting promising avenues for advancing real-time applications in robotics and simulation domains. It has the potential to revolutionize how these systems operate and perform, enhancing efficiency. and efficacy.

.

References

[1] Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., & Ng, R. (2020). NeRF. *Communications of the ACM, 65*, 99 - 106..

[2] Suhail, M., Esteves, C., Sigal, L., & Makadia, A. (2022). Generalizable Patch-Based Neural Rendering. *ArXiv, abs/2207.10662*..

[3] Sitzmann, V., Rezchikov, S., Freeman, W.T., Tenenbaum, J.B., & Durand, F. (2021). Light Field Networks: Neural Scene Representations with Single-Evaluation Rendering. *Neural Information Processing Systems*.

[4] Ahmed, Asmau M., Duran, Olga, Zweiri, Yahya and Smith, Mike (2017) Hybrid spectral unmixing : using artificial neural networks for linear/non-linear switching. *Remote Sensing*, 9(8), p. 775. ISSN (online) 2072-4292

[5] Randerath, J. (2023). Syndromes of limb apraxia: Developmental and acquired disorders of skilled movements. In G. G. Brown, T. Z. King, K. Y. Haaland, & B. Crosson (Eds.), APA handbook of neuropsychology, Vol. 1. Neurobehavioral disorders and conditions: Accepted science and open questions (pp. 159–184). American Psychological Association. https://doi.org/10.1037/0000307-008.

[6] Adam Wong Yoon Khang, Shamsul J. Elias, Nadiatulhuda Zulkifli, Win Adiyansyah Indra, Jamil Abedalrahim Jamil Alsayaydeh, Zahariah Manap, Johar Akbar Mohamat Gani, 2020.Qualitative Based QoS Performance Study Using Hybrid ACO and PSO Algorithm Routing in MANET. Journal of Physics, Conference Series 1502 (2020) 012004, doi:10.1088/1742-6596/1502/1/012004.

[7] Zhou, Z., & Tulsiani, S. (2022). SparseFusion: Distilling View-Conditioned Diffusion for 3D Reconstruction. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 12588-12597.

[8] Liaskovska, S.Y., Martyn, Y.V. Development of Information Technologies for the Research of Technical Systems. Lecture Notes in Networks and SystemsThis link is disabled., 2023, 708, pp. 3–15.

[9] B. Zhou, J. Pan, F. Gao, and S. Shen, “Raptor: Robust and perceptionaware trajectory replanning for quadrotor fast flight,” IEEE Transactions on Robotics, vol. 37, no. 6, pp. 1992–2009, 2021

[10] Yixiao Zhang, Zhongguo Zhao, Jianghua Zheng, CatBoost: A new approach for estimating daily reference crop evapotranspiration in arid and semi-arid regions of Northern China, Journal of Hydrology, Volume 588, 2020, https://doi.org/10.1016/j.jhydrol.2020.125087.

[11] Niu, Hanlin & Ji, Ze & Arvin, Farshad & Lennox, Barry & Yin, Hujun & Carrasco, Joaquin. (2021). Accelerated Sim-to-Real Deep Reinforcement Learning: Learning Collision Avoidance from Human Player. pp.144-149.

[12] Baniqued, P. D., Bremner, P., Sandison, M., Harper, S., Agrawal, S., Bolarinwa, J., Blanche, J., Jiang, Z., Johnson, T., Mitchell, D., Lopez, E., West, A., Willis, M., Yao, K., Flynn, D., Giuliani, M., Groves, K., Lennox, B. & Watson, S., 11 Apr 2024, In: Journal of Field Robotics . pp.1-20 20.

[13] Z. Sun and Y. Xia, “Receding horizon tracking control of unicycle-type robots based on virtual structure,” International Journal of Robust and Nonlinear Control, vol. 26, no. 17, pp. 3900–3918, 2016.

[14] Phang, F. A., Pusppanathan, J, Nawi, N. D., Zulkifli, N. A., Zulkapri, I., Harun, F. K. C., Wong, A.Y. K., Alsayaydeh, J. A. J., Sek, T. K., (2021). Integrating Drone Technology in Service Learning for Engineering Students&quot; International Journal of Emerging Technologies in Learning. 16(15), pp. 78-90.

[15] Indra, W.A., Zamzam, N.S., Saptari, A., Alsayaydeh, J.A.J, Hassim, N.B., 2020.” Development of Security System Using Motion Sensor Powered by RF Energy Harvesting”, 2020 IEEE Student Conference on Research and Development, SCOReD 2020 9250984, pp. 254-258.