Scouting Robot with Search-Space Reducing Hybrid Networks for Unknown Environment Path Planning

Charles Zheng and Megan Kuang

Elgin Park Secondary
Advisor

ABSTRACT

Autonomous vehicle navigation is becoming an important problem as fully self-driving cars are becoming a possibility in the next few decades and space exploration is gaining more momentum. One of the most important aspects of autonomous vehicle navigation is a solid path planning algorithm. Most path planning algorithms can be categorized into 2 groups, classical and machine learning algorithms. Classical algorithms can find the shortest path and usually have a 100% accuracy, but it can't function when its environment isn't 100% mapped out. On the other hand, ML algorithms can operate on partial maps or no maps at all, but they have poor success rate and produce long paths. However, a hybrid approach to a path planning algorithm could eliminate the downside of both categories of algorithms. A robot with a hybrid algorithm could have 100% accuracy while maintaining a near optimum route without a map of the environment. This paper proposes a search space reduction hybrid network (SRHN) path planning algorithm that not only combines the advantages of classical methods and machine learning methods, but also reduces the search space and memory usage. SRHN works by dividing up the distance between a start point and an endpoint with landmarks and paths from its current landmark to the next. While calculating an approximate optimal path, it significantly reduces search space. To test the result of SRHN, experimentation was conducted in the real world. Excellent results have been achieved in the real world 2D tests that were conducted.

Introduction

With ample focus on space exploration and NASA starting the first step of returning to the moon through the Artemis mission, scouting robots that are capable of planning routes and moving quickly through unknown environments is becoming more critical[1]. Aside from extraterrestrial exploration, these robots also play a crucial role in many activities on Earth like search-and-rescue, warehouse management, restaurant server robot, and exploring terrains that are too dangerous for humans.

Many popular classical methods of path planning, like A*[2] or Wavefront[2], though generates optimal paths and has 100% accuracy, can only operate under conditions where the environment is known. Because of this, classical algorithms are nearly obsolete in scouting robots in unmapped locations where the goal is to plan accurate and optimum paths and record the surroundings along the way. In recent years, the advancement of Machine Learning (ML) gave rise to ML-based path planning algorithms[3], [4] that can path plan under unmapped environments. However, many ML-based algorithms have difficulty competing with the success rate of classical algorithms, and ML-based algorithm paths are often considerably longer than those of classical methods. By constructing a pros and cons list for classical and ML-based path planning algorithms, we see that the downside of classical algorithms is the upside of ML-based algorithms and vice versa. This suggests that a hybrid approach to a path planning algorithm can theoretically allow a rover to navigate through an unmapped environment while maintaining a near optimum path with 100% accuracy. In this research, a search-space reducing hybrid network (SRHN) is proposed for path planning problems that combines the merits of classic and machine learning approaches. SRHN is a prominent hybrid algorithm that combines a machine-learned algorithm with a traditional algorithm like A*. SRHN has demonstrated significant search space savings and can operate with partial maps while still generating near-optimal routes. The goal of this research is to construct an autonomous scouting robot to test SRHN in real-world scenarios.

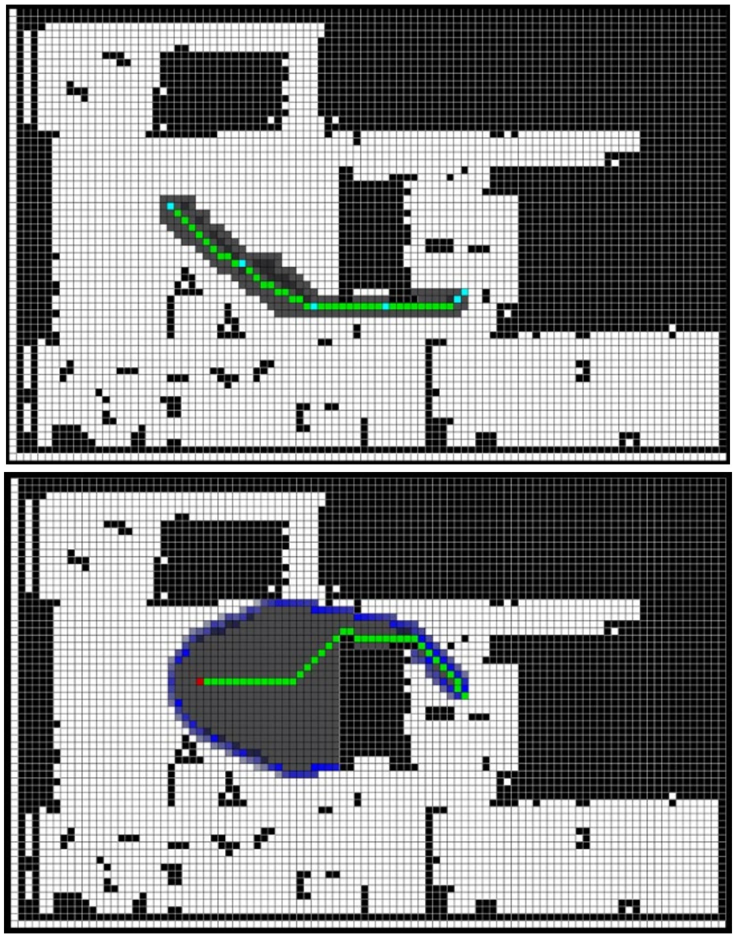

Figure 1. Comparison between SRHN (left) and A* (right) gray area represents the cells searched and green cells represent the final path. For SRHN, the blue cells are the landmarks selected by SRHN. As seen, SRHN has a significantly reduced search space.

Methods

Network Architecture

SRHN consists of a global kernel, to set landmarks towards the end point, and a classical kernel, that plans the path between two landmarks. In our example of the local kernel, we used A*. The global kernel is made up of four modules. The view module, map module, ensemble module, and landmark module.

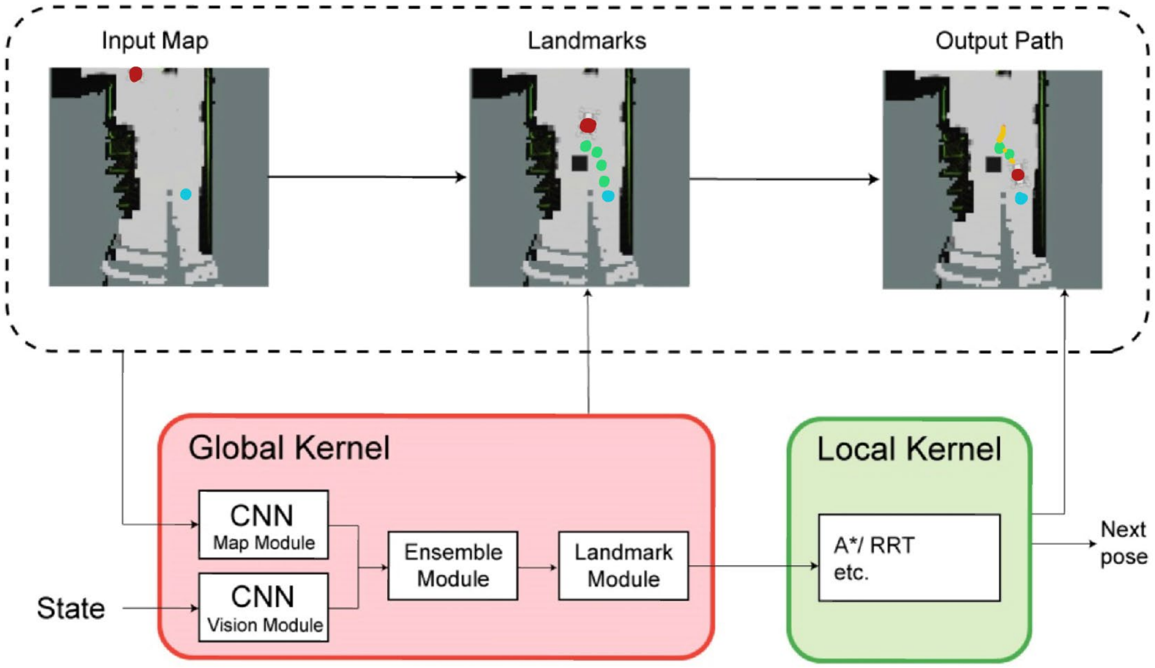

Figure 2. Overview of the search-space reducing hybrid networks (SRHN) architecture. The robot is colored in red and the goal point in blue. Each of the green dots represents a landmark determined by SRHN to guide it around a newly introduced obstacle. The yellow line represents the path found by the local kernel; in our case it represents the path found by A*.

View Module

The view module uses a LSTM network[7] to take in information of the rovers surrounding and suggest the next action it should take based on its environment. The view module bases its suggested next action on four inputs about its surroundings: 1. The normalized distance between the rover and surrounding obstacles in all 8 directions, 2. The normalized direction to the goal, 3. The angle to the direction of the goal, 4. The normalized distance to the goal. The view module exhibits greedy behavior as it performs well when there are direct routes to the goal point but has a low success rate when U-turns are present.

Map Module

The map module's main objective is to fix the greedy behavior of the view modules by trying to predict what the map will look like. This works better in scenarios where there are long corridors or complex formations of barriers are present. The map module works by augmenting the LSTM network input with the compressed global image snapshot. The global snapshot is compressed by using a Convolutional Auto-encoder (CAE)[25]. The CAE encoder contains five layers, four convolutional layers and one linear layer. Each of the convolutional layers contains a series of 4 layers, and the linear contains two layers.

- Convolutional layers: The encoder contains four convolutional layers. Convolutional layers are used to learn spatial hierarchies and extract features from the input data. They perform a convolution operation on the input data, which helps the model learn local patterns.
- Batch normalization layer: After each convolutional layer, there is a batch normalization layer. Batch normalization is used to normalize the input data and improve the training process by reducing internal covariate shift. It helps the model converge faster and achieve better performance.
- Max-pooling layer: Following the batch normalization layer, there is a max-pooling layer used to downsample the input data, reducing its spatial dimensions. This helps the model learn more abstract features and reduces the computational complexity.
- Leaky ReLU activation function: After the max-pooling layer, a Leaky ReLU activation function is applied. Leaky ReLU is a variation of the ReLU activation function that allows a small, non-zero gradient when the input is negative. This helps prevent the "dying ReLU" problem, where some neurons become inactive and stop learning.
- Linear layer: The final layer of the encoder is a linear layer, which is a fully connected layer that maps the output of the previous layers to a lower-dimensional space. This is the encoded representation of the input data.
- Batch normalization layer: After the linear layer, there is another batch normalization layer to normalize the output of the linear layer.

This architecture allows the encoder to progressively extract and compress the input information into a lower-dimensional representation. This compressed representation can be later used by the decoder to reconstruct the original input data.

To decode, we use a CAE decoder that consists of a linear layer followed by four deconvolutional layers. Each deconvolutional layer, which essentially performs the reverse operation of a convolutional layer, is accompanied by a batch normalization layer and a ReLU activation function, except for the last deconvolutional layer, which uses a Tanh activation function. This is because the input data is normalized in the range of [-1, 1], and the output of the Tanh function matches this range. Tanh is a smooth, differentiable function that maps the input to the range of [-1, 1], making it suitable for this purpose. This design allows the decoder to learn complex spatial hierarchies and generate high-quality output. The CAE is trained on 100,000 synthetically generated maps of different environments.

Ensemble Module

The view and map modules behave differently depending on the layout of the map. Additionally, when training the model on different datasets, the same variability in behavior exists. To address this issue, we added a third module called ensemble module. Ensemble machine learning methods use multiple weak learners and outputs a majority voting consensus[9]. The ensemble module focused on parallel ensemble methods that train all weak learners simultaneously on different training datasets sampled from the original dataset. This approach results in uncorrelated weak learners, each learning different features. The voting procedure then increases the accuracy of predictions by incorporating multiple uncorrelated weak learners.

The ensemble module uses the view and map modules as weak learners. By training these models on different datasets, the weak learners learn to adapt to various environments and remain uncorrelated. During runtime, the weak learners are executed in parallel on the map. If one or more kernels find the goal, we select the one with the shortest traversed length. If none of the kernels find the goal, we choose the kernel that has made the most progress. By implementing the ensemble module, we significantly improve the success rate of finding the goal compared to using the view and map modules alone.

Landmark Module

The previous three modules are capable of generating paths from starting point to goal point, but these modules have poor success rate compared to A*, especially in bigger maps or in cases where complex obstacles are present. To improve the success rate, a new module called the landmark module was designed.

The landmark module's primary function is to suggest a series of landmarks between the start and end points to guide the rover through the environment. While any of the previous three modules can be employed to generate landmarks, the ensemble module has proven to be the most effective. Landmarks are generated by constraining the number of iterations of the global kernel, the algorithms used for landmark generation, such as the ensemble module.

Once the landmarks are identified, a local kernel is tasked with planning the actual path for maneuvering between these landmarks. Although any classic solution can serve as the local planner, we used A* because in most cases, it searches less space compared to other classical algorithms. The landmark module plays a crucial role in enhancing the success rates of SRHN.

Rover Design

For our research, we are only concerned with the performance of SRHN in a 2D environment. For this reason, the best design choice for a robot was a rover.

The rover uses a lidar and a camera to get its location and surrounding inputs for the view module. The camera can turn 360 degrees, enabled by a servo, which the camera is attached to. The rover moves with 4 mechanum wheels to allow for omnidirectional movements, which are powered by 4 12V motors. To control everything, we used a Nvidia Jetson Nano, a small but powerful computer for deep learning, to run SRHN. The frame of the rover was 3D printed in carbon fiber nylon, which was then coated with red spray paint for better contrast between the hardware and frame. To power everything, the rover uses a 11.1 V 6000mAh lipo battery that provides the rover with roughly one hour of run time each charge. The Jetson nano runs on Ubuntu 18.04, a Linux distribution. The robot and SRHN are programmed in C++ and Python by using the ROS framework.

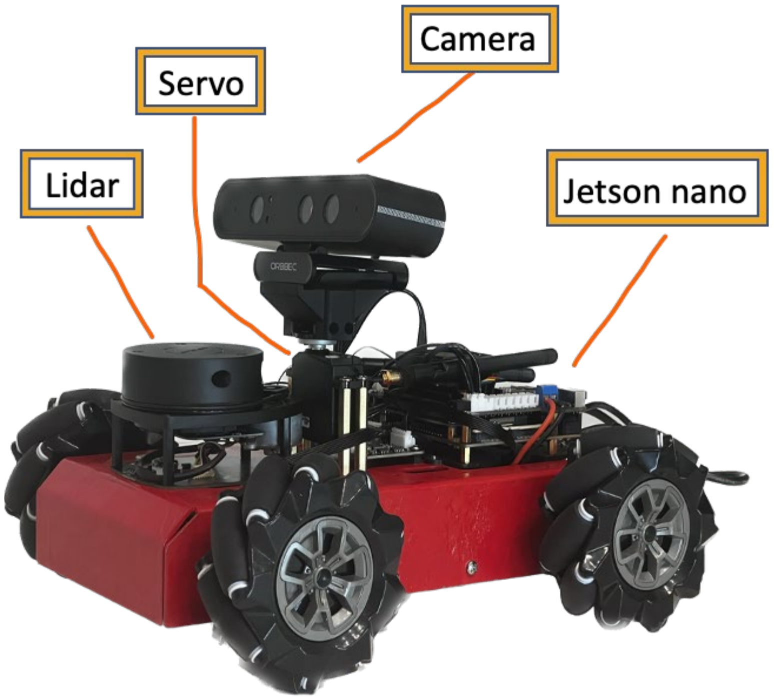

Figure 5. A picture of the rover used to test SRHN. The battery is located under the frame and hidden. The total cost of the rover was under $500USD, making it very economically feasible to reproduce.

Environment Set Up

Every environment was set up to have approximately an area of 150 square meters. This is usually the size of 2 classrooms plus a corridor linking the rooms together. Environments of unusual nature and layout were especially desirable. Once a location is determined, the area is blocked off with plastic dividers and wooden blocks. Random obstacles are then scattered around the environments and special obstacles are arranged to create U-turns and other special natured obstacles. When the environment is set up, a start point and goal point are then randomly chosen, preferably near the edges of the environments to make sure there is a good distance between the start and goal points.

Results

We tested 10 different algorithms under 30 different maps, twice each map. In total, 600 real world tests were conducted. Of the 10 algorithms, 7 were machine learned algorithms. Each machine learning algorithm was trained with the same dataset of 60,000 256x256 sized maps split between random-fill, block, and house style maps generated in PathBench[10]. Six of the algorithms tested were meant to serve as a benchmark to compare SRHN to, while three algorithms were the first three modules of SRHN to see if the modules of SRHN are crucial to its success. From each test, we took 4 measurements:

1. Success rate (Succ R), defined as if the rover was able to reach the goal point from the start point. All maps designed are guaranteed to have at least one navigable path between the start and goal points. Measured as a percentage point.
2. Distance of path (dist. path), defined as the distance traveled from the start point to the goal point in meters.
3. Time taken to reach goal (time), defined as the time it takes for the rover to travel from the start point to the goal point in seconds after the algorithm is initiated.
4. Search space (search), defined as the percentage of the map searched. This measurement was the hardest to take as the environment needed to be mapped before each map was tested on. This required the robot to operate in a closed environment, achieved by placing plastic dividers around a chosen perimeter. The environment was manually mapped by using Gmapping[11]. The final measurement is achieved by dividing the space the algorithm searched by the size of the entire map. Only A*, wavefront, and SRHN had this data.

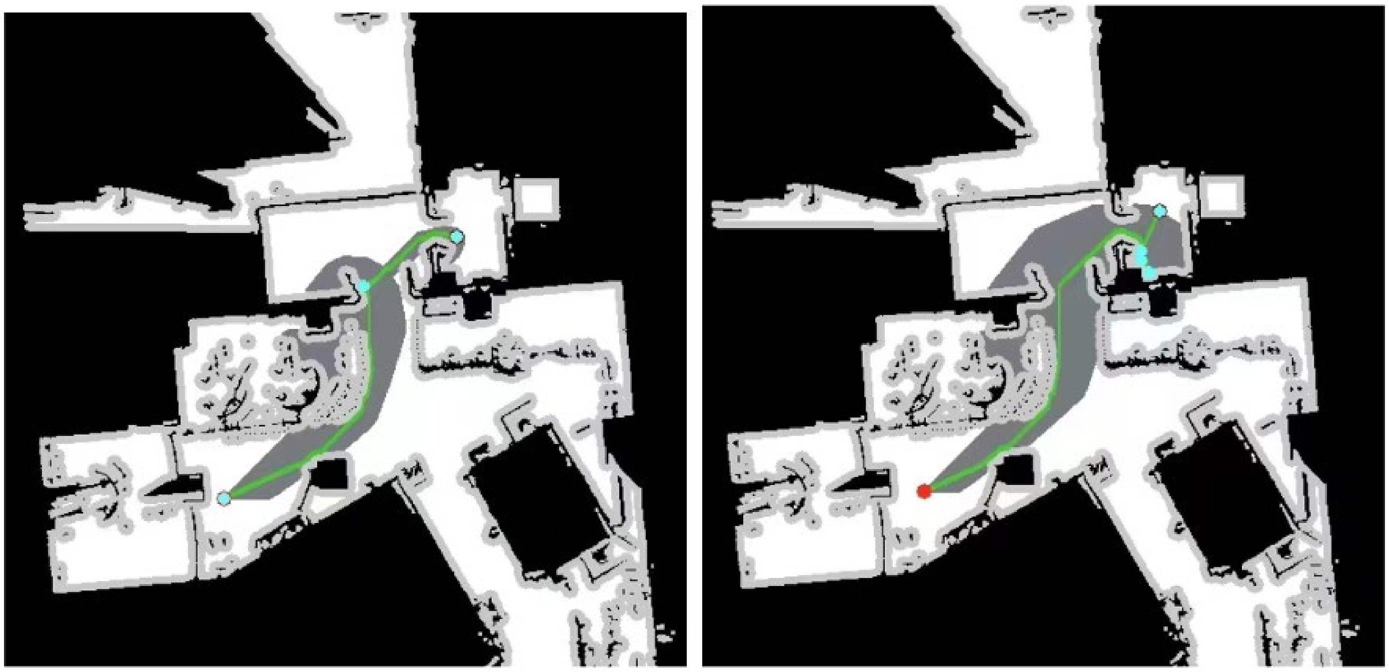

Figure 3. Here is a comparison between SRHN (on the left) and A* (on the right) on a test run in a living room. We can see that SRHN's path is really close to that of A*, but the search space is a lot less. The blue dots on SRHN's map are the landmarks set by SRHN.

Out of the 30 environments tested in, 25 locations were at school and 5 were from a house. Each map was built to be roughly the same size of 150 square meters (the size of roughly 2 high school classrooms). For each map, random obstacles were scattered around random locations. After obstacles were set and the environment was blocked off, a start and goal point were chosen randomly while trying to maximize the distance between the start and goal points.

Table 1. Averaged results of the 4 measurements taken for each of the 10 algorithms tested in real world 2D conditions. In total, the results of 600 tests were computed.

Algorithm	Succ R (%)	Dist. Path (m)	Time (s)	Search (%)
A*[2]	100	15.32	36.91	21.3
Wave-front[2]	100	15.82	37.99	41.29
OMPL RRT[2]	100	33.58	98.58	NA
SRHN[1]	**100**	**17.11**	**53.32**	**10.73**
View Mod[3]	77	24.89	82.56	NA
Map Mod[3]	80	21.04	70.27	NA
Ensemble Mod[3]	90	20.75	66.61	NA
Potential Field[3]	77	25.26	88.77	NA
VIN[5][3]	52	30.88	114.49	NA
MPNet[6][3]	83	25.46	87.34	NA

[1]SRHN data is bolded. It is the only hybrid algorithm.
[2]Classical algorithms.
[3]Machine Learning algorithms.

Each of the data points presented in table 1 are taken as an average. Each algorithm was run under the same condition and same map twice to increase consistency and remove outliers. For classical algorithms, the rover was given full information about the map, while for machine learning algorithms, no information about the map was given other than its goal point. Data presented in a table form can be hard to visualize, but graphs might help us to better understand how SRHN compares to both classical and machine learning algorithms.

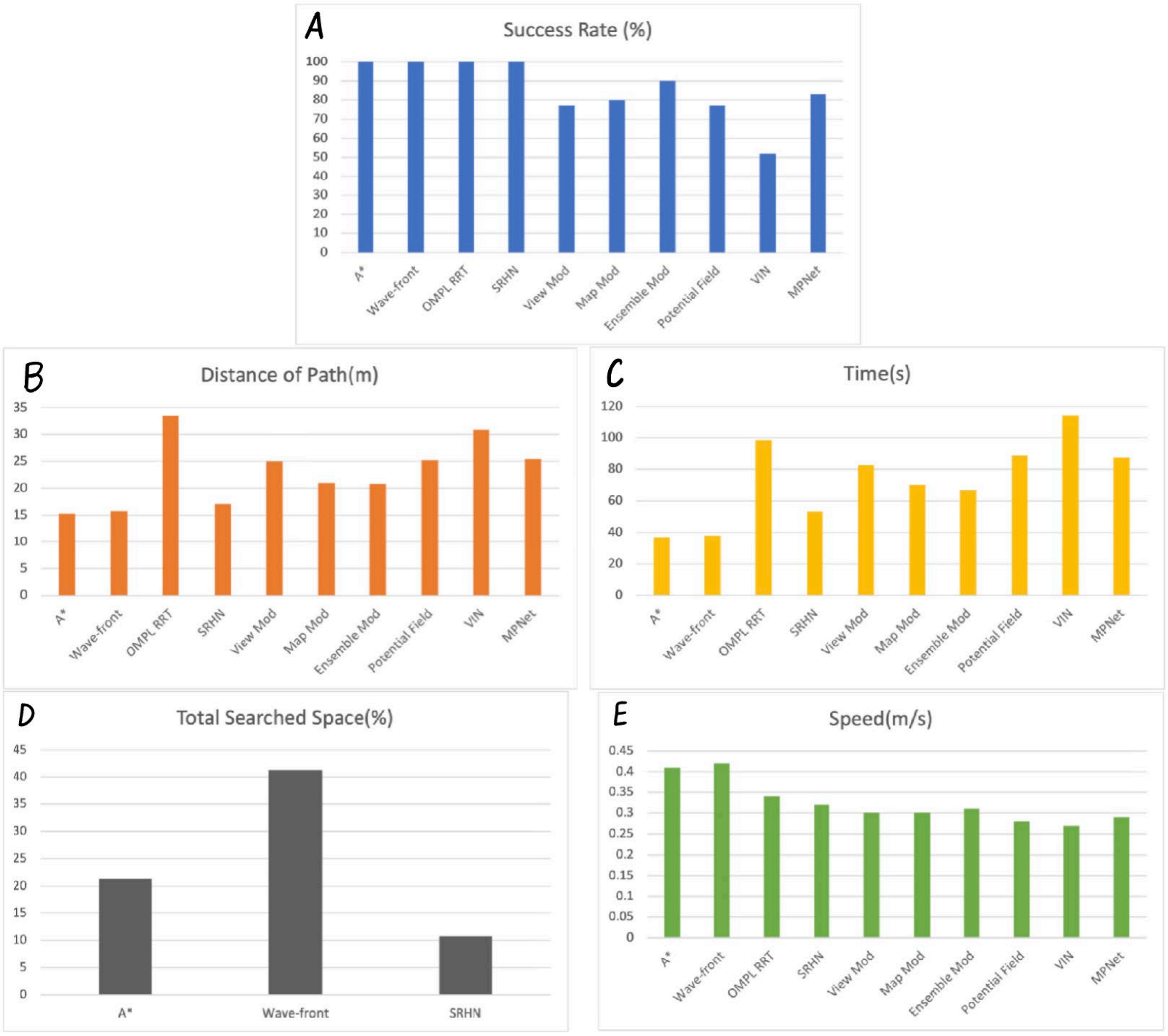

Figure 4. These graphs help to better visualize SRHN's advantage compared to many classical and machine learning algorithms. The graphs compared the success rates, distance of paths, time, search space, and the speed of the 10 algorithms tested.

A speed bar graph was added by dividing the distance the rover took to reach goal point by the time it took to reach the goal point. This graph is crucial in helping us understand the algorithm's behavior during each run.

Discussion

Looking at the results, we begin to see the importance of all four modules in SRHN. On their own, none of the modules can reach a 100% success rate. As predicted the view module's greedy behavior caused it to have longer paths as during a few tests, we observed the view module would walk into a U-turn or other dead ends that seems to be closer to the goal point. As a result, the time taken for the rover to reach the goal point was also longer for the view module. However, the map module was able to avoid these problems as the massive data set it was trained on had many examples of U-turns, so the map module was able to avoid walking into these "traps." These "traps" can sometimes cause the view module to become disorientated or mess up its localization, causing the rover to mess up the location of its goal point. This explains why the success rate of the view module is lower than that of the map module. When we combine the view module and map module together by using parallel ensemble machine learning, we see that there is a noticeable increase in success rate and decrease in time. What this tells us is that the ensemble module walks into U-turns and dead ends less. This can also be seen by looking at the speed data for the ensemble module. Compared to the view and map modules, the ensemble module seems to have a higher speed. However, this isn't necessarily true. The speed data is derived from taking the average distance and dividing by the average time. What this means is that a higher velocity means less distance it traveled compared to the time it took, which means there were less backtracking out of dead ends or U-turns. When the landmark module and a local kernel is added to make SRHN, we see that the success rate skyrocketed to 100% while seeing a huge decline in distance and time. These tests show the importance of each module in SRHN and the necessary role they play in the success of SRHN.

Out of the algorithms tested, the only algorithms that achieved a 100% accuracy were the A*, Wavefront, and OMPL RRT (which are all classical algorithms). However, these algorithms were run while having full knowledge of the map, so having a 100% success rate isn't surprising. SRHN, on the other hand, was run without any knowledge of what the map looks like, yet it still had a 100% success rate. As predicted, none of the machine learning algorithms were close to having a 100% success rate. Not only was SRHN more accurate than the machine learning algorithms tested, but the paths it produced are drastically shorter and the time it takes to reach its goal is also substantially faster. This demonstrates that we have accomplished the first goal of this research: creating a hybrid path planning algorithm that can have a 100% success rate without having prior knowledge of the map.

By examining the distance graph, we see that SRHN's distance of path was close to that of A* (the shortest possible path). Not only was SRHN able to find a path to a goal point with 100% accuracy while having no knowledge of its environment, but it did so while producing nearly the best path possible. Though the distance of the path was close to that of A*, the time it took to reach the goal was much longer compared to A*. The explanation of this is simple: SRHN uses a machine-learned global kernel, which takes more computing than a classical algorithm. On top of that, SRHN must take and store information about the environment in real time. As a result, SRHN moves slower to give itself more time for computation. This is also reflected in the speed data where we see that a rover with SRHN moves slower than a rover running A*. SRHN can become faster by using better hardware (ex. a more powerful GPU).

SRHN shows a more than 50% decrease in search space compared to A* and a more than 75% decrease in search space compared to Wavefront. This occurs because A* exhibits greedy and heuristic behavior. So, by using machine learning to break up the distance between start and goal points, the greedy and heuristic behavior of A*, there will be less trial and error because each segment of the path will become simpler, and in turn lowering the search space.

Conclusion

The combining of classical and machine learning algorithms into a hybrid algorithm is proven to be possible and exhibits many benefits. SRHN not only had 100% accuracy, but also had more than 50% less search space and a near perfect path compared to A*, all while performing with no knowledge of the environment. SRHN also has a significant advantage over machine learning algorithms in terms of success rate, distance of path, and time taken to reach a goal. SRHN is truly able to take advantage of the benefits of both classical and learning-based algorithms. With that, SRHN was able to achieve everything that it was hypothesised to be able to do.

In the future, as more data is collected from test runs, we can begin to train SRHN on real world data aside from just simulated data. There are many real-world events that computational simulations are not able to replicate. Hence by training with real world data, we can expect SRHN's distance to goal to become closer to the optimum path and further reduce the search space.

SRHN has only been experimentally proven to work in a 2-dimensional plane. A future goal would be to extend SRHN into 3D, but this would come with significantly more challenges as it would be hard for sensors to sense in a 3D plane. The rover would also need to be redesigned to accommodate 3D environments. Some possible robot design for that case could be either a hexapod or a drone, but each comes with their own design challenges.

References

[1] D. González, J. Pérez, V. Milanés and F. Nashashibi, "A Review of Motion Planning Techniques for Automated Vehicles," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135-1145, April 2016, doi: 10.1109/TITS.2015.2498841.

[2] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, Principles of Robot Motion: theory, algorithms, and implementation. MIT press, 2005. http://mathdep.ifmo.ru/wp-content/uploads/2018/10/Intelligent-Robotics-and-Autonomous-Agents-series-Choset-H.-et-al.-Principles-of-Robot-Motion_-Theory-Algorithms-and-Implementations-MIT-2005.pdf

[3] J. Wang, W. Chi, C. Li, C. Wang and M. Q. . -H. Meng, "Neural RRT*: Learning-Based Optimal Path Planning," in IEEE Transactions on Automation Science and Engineering, vol. 17, no. 4, pp. 1748-1758, Oct. 2020, doi: 10.1109/TASE.2020.2976560.

[4] M. Inoue, T. Yamashita, and T. Nishida, "Robot Path Planning by LSTM Network Under Changing Environment," in Advances in Computer Communication and Computational Sciences. Springer, 2019, pp. 317329. https://arxiv.org/pdf/2203.10823

[5] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value Iteration Networks," in Advances in Neural Information Processing Systems, 2016, pp. 2154–2162. https://arxiv.org/abs/1602.02867

[6] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019, pp.2118–2124. https://arxiv.org/abs/1806.05767

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997. https://doi.org/10.1162/neco.1997.9.8.1735

[8] M. Inoue, T. Yamashita, and T. Nishida, “Robot Path Planning by LSTM Network Under Changing Environment,” in Advances in Computer Communication and Computational Sciences. Springer, 2019, pp. 317329. https://arxiv.org/pdf/2203.10823

[9] T.G. Dietterich, “Ensemble Methods in Machine Learning,” in international workshop on multiple classifier systems. Springer, 2000, pp. 1–15. https://doi.org/10.1007/3-540-45014-9_1

[10] Alexandru-Iosif Toma, Hao-Ya Hsueh, Hussein Ali Jaafar, Stephen James, Daniel Lenton, Ronald Clark, Sajad Saeedi, “PathBench: A Benchmarking Platform for Classical and Learned Path Planning Algorithms,” in IEEE Conference on Robotics and Vision, Burnaby BC Canada, 2021. https://arxiv.org/pdf/2105.01777

[11] G. Grisetti, C. Stachniss and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," in *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34-46, Feb. 2007, doi: 10.1109/TRO.2006.889486.