Improving Off-road Planning Techniques with Learned Costs from Physical Interactions

Matthew Sivaprakasam[1], Samuel Triest[2], Wenshan Wang[2], Peng Yin[2], Sebastian Scherer[2]

***Abstract*— Autonomous ground vehicles have improved greatly over the past decades, but they still have their limitations when it comes to off-road environments. There is still a need for planning techniques that effectively handle physical interactions between a vehicle and its surroundings. We present a method of modifying a standard path planning algorithm to address these problems by incorporating a learned model to account for complexities that would be too hard to address manually. The model predicts how well a vehicle will be able to follow a potential plan in a given environment. These predictions are then used to assign costs to their associated paths, where the path predicted to be the most feasible will be output as the final path. This results in a planner that doesn't rely solely on engineered features to evaluate traversability of obstacles, and can also choose a better path based on an understanding of its own capability that it has learned from previous interactions. This modification was integrated into the Hybrid A* algorithm and experimental results demonstrated an improvement of 14.29% over the original version on a physical platform.**

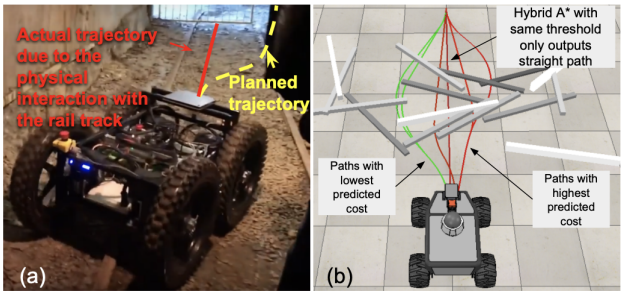

Fig. 1. (a) Environmental interactions often prevent a vehicle from maintaining its planned trajectory. However, this kind of interaction is hard to model by existing methods (b) Potential paths considered by our planner that uses a predictive model. Green paths are predicted to be more successful than red paths. Given the same threshold, an occupancy-grid based planner will output a straight path (which will fail in this case), whereas our planner evaluates multiple paths based on its experiences of the interaction with the environment and chooses the one it predicts to be the most successful.

I. INTRODUCTION

Path planning for ground vehicles in structured environments, such as highways, neighborhoods, and other urban environments, is an increasingly well-understood problem. However, using the same techniques in unstructured areas, such as off-road or cluttered environments, presents unique problems. When an autonomous vehicle is no longer in an area designed for driving (e.g. a paved road), mobility becomes limited and difficult decisions have to be made, such as how to interact with various obstacles and terrains.

Standard path planners often rely on an engineered cost map [1], [2], [3]. When it comes to the design of the cost map, some methods involve a geometry-based approach where the cost map is populated based on the 3D structure of the environment. For example, height of and distance from an obstacle can be used to discourage a vehicle from going too close to it [4], [5], [6]. Other methods involve higher-level costs through concepts such as semantic segmentation, which allows for penalties to be designed based on the terrain itself (e.g. lower cost for driving on a paved road versus gravel) [7], [8]. These approaches have their drawbacks, however. Cost maps become increasingly difficult to tune as they rely on more environment properties, and it is often difficult to tune them in a way that is generalizable to more than one type of environment or scenario. Moreover, the relationship between different properties is not always obvious. For example, as shown in Fig. 1 (a), a vehicle at a given velocity might only be able to cross the rail track by approaching at a certain angle, but a different velocity could require a different angle. Physical interactions based on combinations like this are difficult to model, especially as more properties are considered. Some learning-based methods have tried to address this problem. In [9], Prágr *et al.* propose to learn a traversal cost through online incremental learning. Quann *et al.* predict the energy costs of future paths based on Gaussian process regression [10]. Zhang *et al.* learn a reward function from expert demonstration using inverse reinforcement learning [11]. But the learned costs are often only trained on features from the environment alone, rather than features based on the interactions between a robot and its environment.

There are too many nuances in obstacle interaction to simply hand-design a cost model. In this work, we present a novel approach that models the physical interaction between the robot and the environment using a trajectory prediction network, which is then integrated into a modified Hybrid A* planner [12]. The model is trained on simulation data in order to predict the actual trajectory of a vehicle, based on the planned trajectory in a complex environment. The result is a planner that employs past vehicle experiences to plan flexibly so that the output path is more likely to succeed in unstructured areas compared to that of the same planner without the model (Fig. 1 (b)).

The main contributions of this paper are: 1) a method of planning in off-road environments that takes advantage of a learned model on top of a search-based algorithm in

[1]Matthew Sivaprakasam is with the University of Pittsburgh, Pittsburgh, PA 15213, USA. mjs299@pitt.edu

[2]Samuel Triest, Wenshan Wang, Peng Yin, and Sebastian Scherer are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA. {striest,wenshanw,pyin2,basti}@andrew.cmu.edu

order to output more-feasible plans; 2) an approach to cost calculation formed by training a prediction network on a dataset of previous vehicle interactions; 3) the demonstration of the potential of this approach through the implementation and validation of our idea on a real-world robot.

II. RELATED WORK

There are plenty of both learning and non-learning approaches that relate to the off-road navigation problem already. To address the difficulty in designing the cost map, Lu and Hershberger introduced layered costmaps [13], which provides a way to maintain contextual information in planning. Instead of one costmap, several costmaps are first generated based on specific semantics. For example, different types of obstacles could have their own costmaps, in which the costs are calculated differently depending on the obstacle. These costmaps are then fed into a "master" costmap which is used for planning. Souza and Goncalves introduce an occupancy-elevation grid structure [14]. This has some advantages over a normal occupancy grid, in that information about height and uncertainty is also stored. They provide an opportunity for new planning techniques over their grid, but one limitation is that no other contextual information is stored. Schwarz and Behnke introduced an approach to unknown terrain traversal by taking advantage of omnidirectional height [15]. They used a 2.5D height map to calculate elevation differences in different dimensions in order to create a new map used to calculate traversability. This approach too, however, doesn't incorporate any other concepts in planning besides height.

Some noteworthy learning-based approaches also exist. Prágr *et al*. introduce a way to learn terrain traversal costs online, in the context of exploration [**?**]. The result is a multi-legged robot that can learn new navigation goals and plans as it explores. Quann and Ojeda propose a method of predicting plan costs in terms of vehicle energy usage, by collecting plan and state information and monitoring power consumption to create a probabilistic model [10]. Finn and Levine take advantage of model-based reinforcement learning to achieve object pushing tasks [16].They train a learned model that can predict the outcome of a sequence of actions, given an RGB input. Arruda and Mathew are able to achieve a similar task by instead creating a model that is uncertainty averse [17]. Pfeiffer and Schwesinger were also able to use a model to create a socially-compliant planner trained on human-human interactions [18]. These methods establish the potential for learned models in planning tasks, but none of them yet provide a solution for the specific problem we are addressing in this paper.

III. BACKGROUND AND INITIAL APPROACH

All of our modifications were implemented based off of the PythonRobotics library [19], which includes several path planning algorithms with implementations that are easy to use and modify. Our initial idea was to enhance a simple search-based planner with various additional costs, using Hybrid A* as a starting point (which primarily uses costs

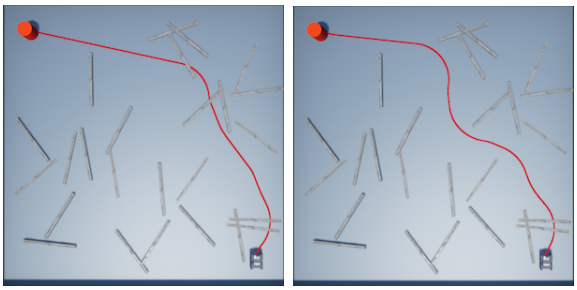

Fig. 2. Paths output by the original algorithm (left), and the modified version with additional height costs (right). The cost inflation around obstacles allowed it to only plan over obstacles when necessary, in an effort to minimize risk of failure without using a learned model.

derived from distance to goal and steering angle). As shown in Fig. 2, a testing environment was set up in AirSim [20] consisting of several obstacles of random height and orientation. We were able to test the limits of the vehicle/planner setup by creating arrangements that can't be easily navigated by relying heavily on a height threshold for obstacles.

A. Integrating an Obstacle-based Cost

In order to add a planning cost derived from obstacles, an approach similar to layered costmaps from [13] was used. In the baseline algorithm, each planning node has an associated cost, calculated based on properties such as length of path from start, distance to goal, forward vs. reverse direction, and steering. We modified the nodes so that each node close enough to an obstacle is given an additional cost that is proportional to the obstacle's height (multiplied by a weight that can be set by the user). This causes the algorithm to guide planning away from a given obstacle, until the cost associated with going around it becomes greater than the cost of going over it.

B. Initial Results With Hybrid A*

Using our modified cost, the planner was able to find more efficient paths than the original version in some cases (see Fig. 2). However, the success of our hand-designed cost did not translate well to various environments unless it was manually tuned each time.

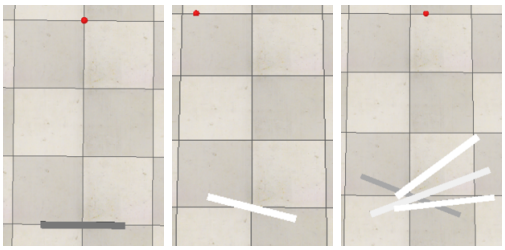

Fig. 3. For data collection, obstacles were systematically placed with varying orientation and height(left,middle), with different goal locations(shown with red dot). Data was also collected by randomly placing multiple obstacles in a similar fashion(right).

IV. LEARNING OBSTACLE TRAVERSAL COSTS

Based on our initial approach, we concluded that accounting for height and angle alone wouldn't be enough to make a significant improvement over the baseline path planner. Moreover, manually choosing obstacle properties, such as height, means potentially excluding a lot of useful information about an environment that can't necessarily be hard-coded into a planner effectively. As obstacle arrangements get more complex, manually modeling planning costs becomes increasingly difficult. We hypothesized that this problem could be addressed, at least in part, by incorporating a predictive model into the planner that predicts an actual path given a desired path. Data was collected in simulation and used to train a predictor, whose task is to predict the actual path of a vehicle given a desired path with given velocity. Our framework involves feeding potential paths at various velocities into the model, and choosing the path/velocity pair that results in the highest similarity between the desired path and the predicted path.

A. Experience Collection

For the rest of our approach, CoppeliaSim (previously known as V-REP) [21] was used instead of AirSim due to its better physics engine. The python toolkit PyRep [22] allowed us to easily interface with the simulator to set up tests and collect data. In order to construct a dataset to train a model on, we aggregated data from two distinct phases of experience collection.

The first phase of experience collection was a systematic process. A single obstacle was placed in between the vehicle and it's goal location and the vehicle was made to go over the obstacle in a straight line towards the goal, with the obstacle having different properties. The following parameters were iterated through:

1) Obstacle angle (-60:60 degrees)
2) Obstacle height (.02:.14 meters)
3) Goal location (-.36:.36 meters laterally with respect to vehicle)
4) Vehicle velocity (.05:.25 m/s)

so that every combination of different obstacle angle/height, goal location, and vehicle velocity were achieved for a total of 5250 samples.

The second phase of experience was collected in a random manner. 1, 2, and 3 obstacles with randomized heights and orientations were placed in the way of the vehicle, which was given a random velocity and a random lateral location for the goal. This gave us an additional 6000 samples (2000 for each quantity of obstacles). Fig. 3 shows examples, from both systematic and random data collection, of how obstacles and the goal locations were placed.

For each sample, a heightmap of the obstacles, planned trajectory and velocity, and resulting simulated trajectory and velocity (where a trajectory consists of an x and y location, as well as the yaw) was recorded. From each trajectory, we generated 50 training samples by randomly sampling a start time, finding the closest path point to the robot's position at that time, and recording the next $0.5m$ of plan and ground-truth trajectory, as well as a cropped local heightmap and robot's current state (position, velocity). All coordinates are transformed to the robot's local coordinates.

TABLE I
DESCRIPTION OF THE VGG BLOCK USED FOR THE TRAJECTORY PREDICTION NETWORK.

Layer	Input Dim	Output Dim
Batch Norm	$C_{in} \times W \times H$	$C_{in} \times W \times H$
Conv1 (3×3)	$C_{in} \times W \times H$	$C_{in} \times W \times H$
Conv2 (3×3)	$C_{in} \times W \times H$	$C_{out} \times W \times H$
Max Pool	$C_{out} \times W \times H$	$C_{out} \times \frac{W}{2} \times \frac{W}{2}$

B. Training the Model

We use a neural network to predict the next $0.5m$ of trajectory (parameterized as the position and yaw of the robot), given the current robot state (as position, yaw and linear velocity), the next $0.5m$ of the planned trajectory, and a local crop of the heightmap.

We obtain the state vector and planned trajectory from the robot's odometry and path planner, respectively. We obtain the local crop of the heightmap by transforming the map to the robot's local coordinate frame and take a $1m \times 1m$ box centered at the robot's location. The heightmap is downscaled to a $24 \times 24 \times 1$ image.

The heightmap is fed through three VGG-style [23] blocks, where each block $VGG_{c_{in},c_{out}}(X)$ transforms a $H \times W \times c_{in}$ image X to a $\frac{H}{2} \times \frac{W}{2} \times c_{out}$ image. This block is described in Table I. Given the flattened heightmap features from the CNN block, we apply four linear layers with *tanh* activations to generate our trajectory prediction. The full network architecture is provided in Table II.

We train our model using Adam [24] to minimize the mean squared error between our prediction and the next $0.5m$ of ground-truth trajectory from our training samples. We observed that the model reached convergence after roughly 200 epochs, achieving a root mean-squared error of 1.33 on held-out validation trajectories.

C. Integrating into the Planner

The baseline Hybrid A* algorithm works by trying to create an unobstructed Reeds-Shepp path [25] from the current node location to the goal. If there is a free path (determined by the lack of any obstacles with a height above the set threshold), it is immediately output as the resulting planned path. Otherwise, more nodes are expanded until an unobstructed path is obtained.

Due to the nature of the planning nodes in the original algorithm, we integrated our model into the planner by associating a new cost with whole paths associated with a node rather than the node itself. Any time a free path is found, it is broken down into segments which are then fed into the learned model, along with a local heightmap and potential velocity. The model outputs a predicted path (rather than a predicted cost) which is then evaluated against the desired path, and a cost is assigned to it by taking the

TABLE II

THE NETWORK ARCHITECTURE

Layer	Input	Input Dim	Output	Output Dim
$VGG_{1,32}$	Heightmap	$1 \times 24 \times 24$	Hmap feat1	$32 \times 12 \times 12$
$VGG_{32,64}$	Hmap feat1	$32 \times 12 \times 12$	Hmap feat2	$64 \times 6 \times 6$
$VGG_{64,128}$	Hmap feat2	$64 \times 6 \times 6$	Hmap feat3	$128 \times 3 \times 3$
Reshape	Hmap feat3	$128 \times 3 \times 3$	Hmap feat flat	1152
$Linear_{1186,512}$	Hmap feat flat $\|$ state $\|$ plan	1186	feat1	512
$Linear_{512,256}$	feat1	512	feat2	256
$Linear_{256,256}$	feat2	256	feat3	256
$Linear_{256,30}$	feat3	256	pred flat	30
Reshape	pred flat	30	pred	10×3

cumulative sum of the squared Euclidean distances between the desired path points and their corresponding predicted path points. This process is done 5 times per evaluated path, each time with a different potential velocity from the list [.05, .1,.15,.20,.25] (determined based on the range of simulated velocities that were seen by the model in training). The path/velocity pair with the lowest cost is kept track of, so any time a new path/velocity has a lower cost, it replaces the current best. This process of expanding nodes and evaluating paths continues until either the best path's cost is below a desired minimum, or the planner has been planning for a maximum amount of time (both of which can be set by the user). The maximum time and minimum cost was implemented to account for the decrease in efficiency caused by iterating over every path. By adjusting these, the planner can be made to prioritize speed over path feasibility, and vice versa. The final trajectory output by this process consists of a set of x,y, and yaw points, as well as a desired velocity which the vehicle then uses to carry out a path.

The result of this modification is a planner that can choose whether or not to go around an obstacle, even if it isn't thresholded into the occupancy grid, as well as adjust it's output based on how the vehicle might interact with its environment (see Fig. 4). Moreover, it can predict which velocity will increase the likelihood of success.

V. EXPERIMENTAL RESULTS WITH COST PREDICTION

In order to evaluate our modifications, we wanted to compare it's performance to that of the original Hybrid A* algorithm both in simulation and in real life. Our primary metric was percentage of successful attempts to reach the goal by using a given planner, as well as the time taken to reach the goal.

A. Testing in Simulation

We set up a series of tests in CoppeliaSim, in which 1, 2, 3, and 4 random obstacles were placed in the way of the vehicle. Both the original Hybrid A* planner and our version were given the same obstacle threshold, and the original planner was given a set velocity of .1. This ended in 400 tests (100 for each quantity of obstacles) with the results shown in Table III.

Fig. 5 (a) and (b) show a compelling example where the impact of the learned model is visible. Even in cases where both planners decide to go over a series of obstacles, there are still different ways to do so. Our version of the planner was able to successfully plan a path with a certain velocity that resulted in a success, whereas the original planner failed to do so.

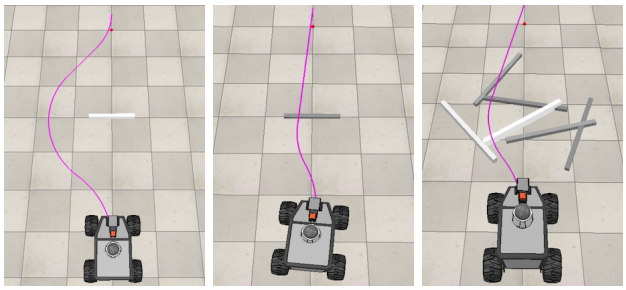

Fig. 4. Some resulting plans output by the modified planner. All obstacle heights are under the set threshold, but the model avoids getting stuck on the higher obstacle (left) and correctly deems it safe enough to go over the lower obstacle (middle) and through the complex example (right).

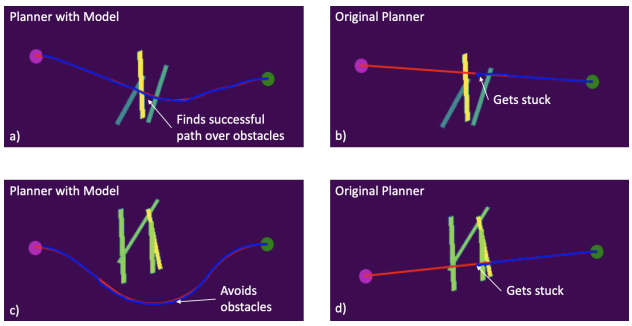

Fig. 5. An example test where both planners try to go over the obstacles, but the baseline planner (right) results in a failure whereas the planner using our learned model (left) results in a success. Red represents the planned path, blue represents the resulting trajectory, and teal and yellow represent obstacles of differing height, where yellow obstacles are taller.

B. Testing on Ground Robot

In order to validate the efficacy of our planner on a physical robot, we constructed a testing environment designed for a Mushr robot [26] to navigate through. The environment was a roughly 5m $\times$ 6m box which we filled with 80/20 of varying lengths, widths and heights. Of particular importance was the heights of our obstacles, which were either 3cm, 4cm, 5cm or 8cm. In our preliminary experiments,

TABLE III

RESULTS OF SIMULATED TESTING.

Planner	Success Rate	Avg. Time to Reach Goal (s)
Without Model	34%	66.97
With Model (ours)	**64%**	**41.63**

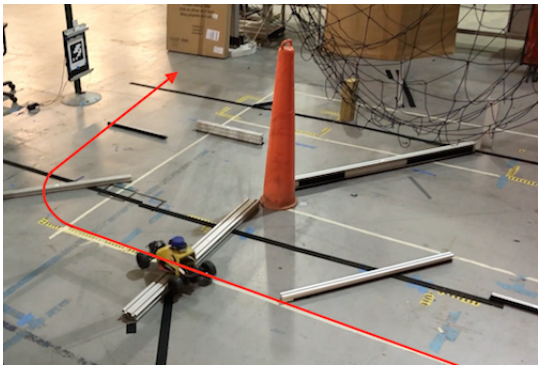

Fig. 6. An image of our testing environment. The robot is traversing a 5cm tall obstacle. The general shape of the the task is indicated with the red arrow. The robot is unable to reach the goal without interacting with obstacles at least 4cm in height.

we observed that the Mushr robot could consistently clear the 3cm obstacle and could never clear the 8cm obstacle. The robot could clear the 4cm and 5cm obstacles if it approached the obstacle a high enough speed at a steep approach angle (we observed that an approach angle within $\pm 15^\circ$ of perpendicular was most likely to succeed).

Since the vehicular dynamics were different for the Mushr robot as compared to the robot in CoppeliaSim, we collected 98 trajectories of the Mushr robot interacting with the obstacles, varying the obstacle height, depth and approach angle. Using these trajectories, we fine-tuned our network to compensate for the change in interactions from the CoppeliaSim scenarios. This was accomplished by repeating the training procedure with the Mushr data using the model trained in CoppeliaSim as an initialization. The trajectories and velocities outputted by our planner were followed using the a feedback controller [27]. The robot localized itself in the environment using a particle filter reading scans from a ydlidar. We compared two planners - hybrid A* without a learned cost, and hybrid A* using our fine-tuned network as a learned cost function.

A visualization of our testing scenario is provided in Fig. 6 and Fig. 7. The robot was tasked with reaching the upper-right of the environment from the lower-left. We ran ten tests for each planner, varying the location and orientation of the start point, and chose one of two orientations for the goal point. To measure the efficacy of our model, we report the success rate of each planner as well as the time taken to reach the goal point.

Shown in Fig. 8 are representative plans and trajectories from our tests. While both planners are tasked with navigating to the same goal point, we can observe that the planner with a learned model plans a much steeper approach towards the 5cm obstacle. This aligns with our observations that the robot is more likely to clear tall obstacles by considering a steeper approach angle. The efficacy of this planning decision is observed in the actual executed trajectories (Fig. 8), as the robot is able to smoothly execute the trajectory given by the planner with a learned model, but gets stuck on the obstacle when executing the baseline planner's trajectory.

TABLE IV

RESULTS OF PHYSICAL TESTING.

Planner	Success Rate	Avg. Time to Reach Goal (s)
Without Model	70 %	10.6
With Model (ours)	**80 %**	**7.8**

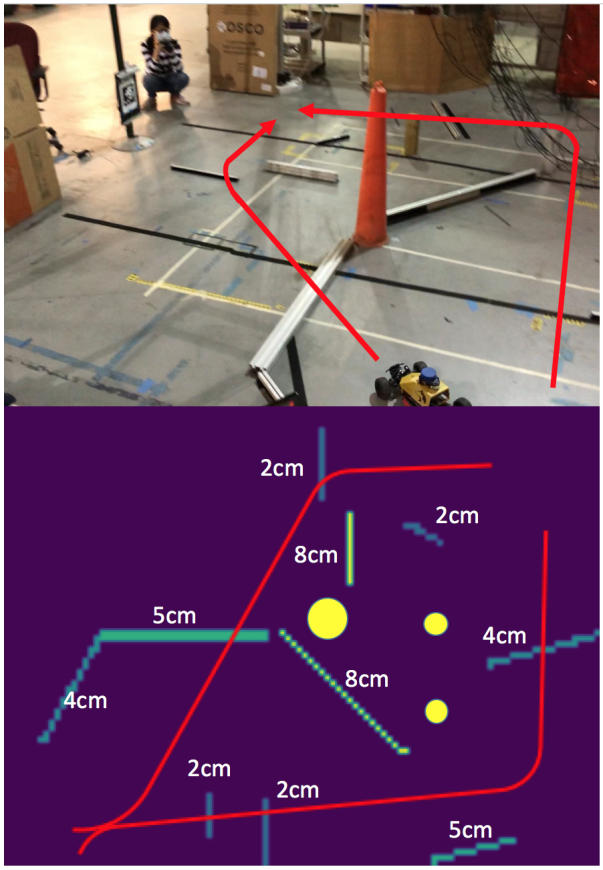

Fig. 7. A visualization of our physical test environment, where the robot's goal is to reach the upper-right from the lower-left. Shown above is a view of the testing environment, with the plans from the heightmap superimposed on it. Below is the heightmap, with two representative plans for how to reach the goal, depending on the desired orientation. The robot is unable to reach the goal without interacting with obstacles at least 4cm in height.

1) Failure Modes: During testing, we observed two primary factors that causes the planner to fail; decoupling of planning and control, and incorrect model predictions. Fig. 9 shows an example where the path-following controller deviated from the planned trajectory, resulting in the vehicle colliding with an adjacent obstacle. Fig. 10 illustrates two similar trajectories where one succeeds in clearing the obstacle, and the other doesn't.

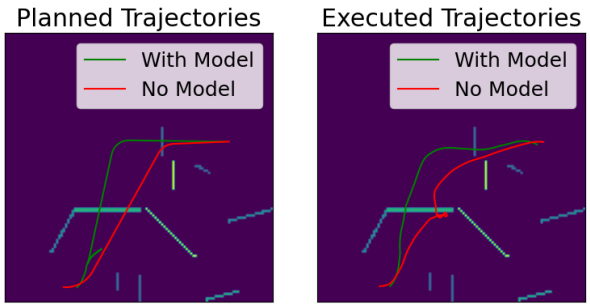

Fig. 8. A visualization of one of our test environments, where the robot is tasked with navigating from the lower-left to the upper-right. Shown in green is the plan produced by Hybrid A* with a learned model, and shown in red is the plan produced by Hybrid A* without a learned model. We can observe that the planner with the model chooses a path with a steeper approach angle, which we observed empirically to result in a higher change of successfully clearing the obstacle. We observe that the plan from the planner with the model results in the vehicle successfully clearing the obstacle, while the plan produced by the baseline planner results in the vehicle getting stuck on the obstacle for a few seconds.

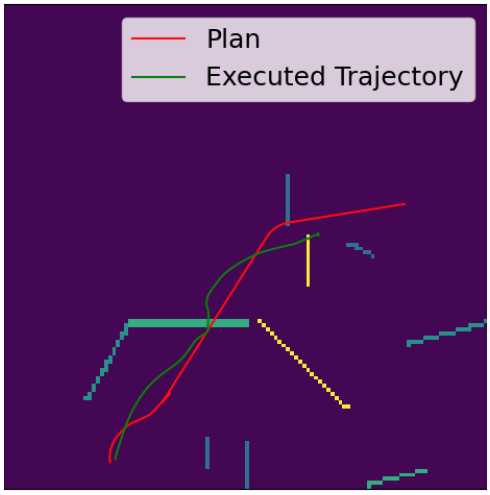

Fig. 9. A failure case in which the controller deviates from the intended path.

VI. CONCLUSION AND FUTURE WORK

This paper presents one method of improving standard path planning in complex environments by introducing effective cost calculations that account for properties that would otherwise be difficult to represent. The key contributions that distinguish this work from prior methods are the introduction of a new planning framework for off-road driving that allows for the combination of search-based planning with learning-based models, and the demonstration of the potential of predictive models in representing physical interactions in a way that can be used to improve planning. Moreover, we were able to validate our results on a real system.

There are plenty of future directions to be taken based off of this work. On of the main advantages of our method is that it can choose the best velocity for a path. However, it is still limited to one velocity for the whole path. A promising next step for improving the planner in complex environments would be to implement a stage where it chooses the best velocity to follow along each point in a path.

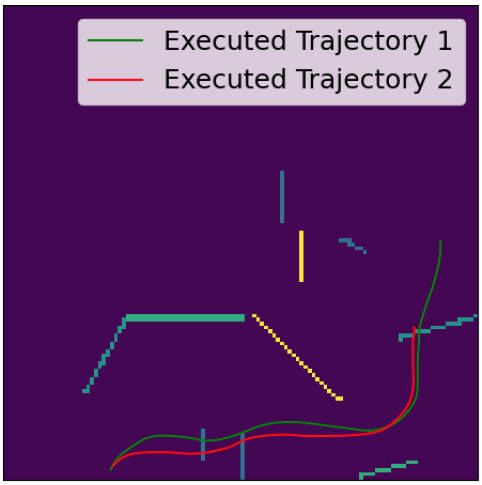

Fig. 10. Two similar trajectories in which one succeeds in clearing the obstacle and the other doesn't.

Another direction to take could be improving the learning component, perhaps by allowing for more generalizability. Throughout testing, only one type of obstacle was used (rectangular). Additionally, all of our data was collected using one vehicle, so it is unlikely that our planner will be as effective when applied to other vehicles with different dynamic constraints. These problems could potentially be addressed by setting up an online learning component in the planner. One could then use the integrate the planner into their own vehicle, or in a different environment, and the model could continuously be adjusted and in turn improve the performance of the planner. This could help mitigate the failure case observed in Fig 9.

Another direction for future work could be using more sophisticated trajectory prediction techniques. As can be observed in Fig. 10, minute changes in state can result in very different resulting trajectories over obstacles. We expect that incorporating techniques that can handle these discontinuities can significantly improve the performance of the planner.

Finally, we are hopeful that our predictive model approach will integrate well with other path planning algorithms, but this has yet to be tested. A thorough evaluation of its effectiveness in other planners would help determine whether this approach is an appealing method to provide another way for planners to include more contextual information that would otherwise be hard to model.

ACKNOWLEDGMENT

This work was supported by ARL award #W911NF1820218.

REFERENCES

[1] James Bruce and Manuela M Veloso. Real-time randomized path planning for robot navigation. In *Robot Soccer World Cup*, pages 288–295. Springer, 2002.

[2] Sebastian Scherer and Sanjiv Singh. Multiple-objective motion planning for unmanned aerial vehicles. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2207–2214, San Francisco, CA, September 2011. IEEE.

[3] Sanjiban Choudhury, Sebastian Scherer, and Sanjiv Singh. Realtime alternate routes planning: the rrt*-ar algorithm. Technical Report December, Carnegie Mellon University, Pittsburgh, PA, 2012.

[4] Pierre Sermanet, Raia Hadsell, Marco Scoffier, Matt Grimes, Jan Ben, Ayse Erkan, Chris Crudele, Urs Miller, and Yann LeCun. A multirange architecture for collision-free off-road robot navigation. *Journal of Field Robotics*, 26(1):52–87, 2009.

[5] Haris Balta, Geert De Cubber, Daniela Doroftei, Yvan Baudoin, and Hichem Sahli. Terrain traversability analysis for off-road robots using time-of-flight 3d sensing. In *7th IARP International Workshop on Robotics for Risky Environment-Extreme Robotics, Saint-Petersburg, Russia*, 2013.

[6] Rogerio Bonatti, Yanfu Zhang, Sanjiban Choudhury, Wenshan Wang, and Sebastian Scherer. Autonomous Drone Cinematographer: Using Artistic Principles to Create Smooth, Safe, Occlusion-Free Trajectories for Aerial Filming. In *International Symposium on Experimental Robotics*, pages 119–129. Springer, November 2020.

[7] Daniel Maturana, Po-Wei Chou, Masashi Uenoyama, and Sebastian Scherer. Real-Time Semantic Mapping for Autonomous Off-Road Navigation. In *Field and Service Robotics*, pages 335–350. Springer, Cham, 2018.

[8] Dong-Ki Kim, Daniel Maturana, Masashi Uenoyama, and Sebastian Scherer. Season-Invariant Semantic Segmentation with a Deep Multimodal Network. In *Field and Service Robotics*, pages 255–270. Springer, Cham, 2018.

[9] Miloš Prágr, Petr Čížek, Jan Bayer, and Jan Faigl. Online incremental learning of the terrain traversal cost in autonomous exploration. In *Robotics: Science and Systems*, 06 2019.

[10] Michael Quann, Lauro Ojeda, William Smith, Denise Rizzo, Matthew Castanier, and Kira Barton. Off-road ground robot path energy cost prediction through probabilistic spatial mapping. *Journal of Field Robotics*, 37(3):421–439, 2019.

[11] Yanfu Zhang, Wenshan Wang, Rogerio Bonatti, Daniel Maturana, and Sebastian Scherer. Integrating kinematics and environment context into deep inverse reinforcement learning for predicting off-road vehicle trajectories. In *Conference on Robot Learning*. Journal of Machine Learning Research, October 2018.

[12] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. In *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08*, 2008.

[13] D. V. Lu, D. Hershberger, and W. D. Smart. Layered costmaps for context-sensitive navigation. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 709–715, 2014.

[14] Anderson Souza and Luiz M. G. Gonçalves. Occupancy-elevation grid: an alternative approach for robotic mapping and navigation. *Robotica*, 34(11):2592–2609, 2016.

[15] M. Schwarz and S. Behnke. Local navigation in rough terrain using omnidirectional height. In *ISR/Robotik 2014; 41st International Symposium on Robotics*, pages 1–6, 2014.

[16] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion, 2017.

[17] Ermano Arruda, Michael J. Mathew, Marek Kopicki, Michael Mistry, Morteza Azad, and Jeremy L. Wyatt. Uncertainty averse pushing with model predictive path integral control. *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, Nov 2017.

[18] M. Pfeiffer, U. Schwesinger, H. Sommer, E. Galceran, and R. Siegwart. Predicting actions to act predictably: Cooperative partial motion planning with maximum entropy models. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2096–2101, 2016.

[19] Atsushi Sakai, Daniel Ingram, Joseph Dinius, Karan Chawla, Antonin Raffin, and Alexis Paques. Pythonrobotics: a python code collection of robotics algorithms, 2018.

[20] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[21] E. Rohmer, S. P. N. Singh, and M. Freese. Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013. www.coppeliarobotics.com.

[22] Stephen James, Marc Freese, and Andrew J. Davison. Pyrep: Bringing v-rep to deep robot learning. *arXiv preprint arXiv:1906.11176*, 2019.

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[25] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific J. Math.*, 145(2):367–393, 1990.

[26] Siddhartha S. Srinivasa, Patrick Lancaster, Johan Michalove, Matt Schmittle, Colin Summers, Matthew Rockett, Joshua R. Smith, Sanjiban Chouhury, Christoforos Mavrogiannis, and Fereshteh Sadeghi. MuSHR: A low-cost, open-source robotic racecar for education and research. *CoRR*, abs/1908.08031, 2019.

[27] R Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.