MLNav: Learning to Safely Navigate on Martian Terrains

Shreyansh Daftry\textsuperscript{1}, Neil Abcouwer\textsuperscript{1}, Tyler Del Sesto\textsuperscript{1}, Siddarth Venkatraman\textsuperscript{2}
Jialin Song\textsuperscript{3}, Lucas Igel\textsuperscript{4}, Amos Byon\textsuperscript{1}, Ugo Rosolia\textsuperscript{3}, Yisong Yue\textsuperscript{3} and Masahiro Ono\textsuperscript{1}

Abstract—We present MLNav, a learning-enhanced path planning framework for safety-critical and resource-limited systems operating in complex environments, such as rovers navigating on Mars. MLNav makes judicious use of machine learning to enhance the efficiency of path planning while fully respecting safety constraints. In particular, the dominant computational cost in such safety-critical settings is running a model-based safety checker on the proposed paths. Our learned search heuristic can simultaneously predict the feasibility for all path options in a single run, and the model-based safety checker is only invoked on the top-scoring paths. We validate in high-fidelity simulations using both real Martian terrain data collected by the Perseverance rover, as well as a suite of challenging synthetic terrains. Our experiments show that: (i) compared to the baseline ENav path planner on board the Perseverance rover, MLNav can provide a significant improvement in multiple key metrics, such as a 10x reduction in collision checks when navigating real Martian terrains, despite being trained with synthetic terrains; and (ii) MLNav can successfully navigate highly challenging terrains where the baseline ENav fails to find a feasible path before timing out.

I. INTRODUCTION

A. Motivation

NASA’s new rover Perseverance successfully landed on Mars in February 2021 with a new autonomous driving algorithm called Enhanced AutoNav (ENav) \cite{enav}. ENav, a classic tree-based path planner at its core, brought substantial upgrades to its predecessor. As of the writing of this article, ENav has driven the rover for approximately half of the 2.2 km driving distance, a remarkable achievement given that Curiosity rover has been driven autonomously for only \sim 6.4\% of its driving distance to date \cite{curiosity}. However, ENav’s ability is still far behind human rover drivers, particularly on complex terrains where it often fails to find a feasible path even if one exists. In many cases, ground operators must manually drive the rover through challenging scenarios before transferring control to ENav as a way to extend the drive distance beyond the visible terrain. The highly stringent safety requirements (zero-tolerance for mechanical damage or unrecoverable situations) and the limited on-board computational resource are the major roadblocks for further improvements of extraterrestrial autonomous driving.

As mobile robots break out of controlled laboratory settings and are increasingly deployed in a real and complex environment like Mars, planning modules have to balance multiple contextual aspects, including computing resources, responsiveness, system-level performance, and most importantly, safety. This presents a challenge to classic motion planning pipelines as their computational complexity increases exponentially with the dimensionality of the motion planning problem. Furthermore, traditional methods are heavily model-based (in the sense that it requires models of the robot, the world and hand-designed heuristic functions) and hence take little advantage of the contextual structure of the environment; thereby having to typically plan every path from scratch. This has a substantial influence on the finite-time planning performance \cite{finite_time_planning}.

A key motivation for our approach is to study how best to effectively integrate learning-based methods with model-based planning frameworks, by carefully breaking down the traditional navigation stack, identify key components that are bottlenecked and integrate ML only where it is most useful. While there has been a recent trend towards integrating learning into path planning and navigation \cite{learning_in_navigation}, the use of learning methods also presents key challenges for real-world, safety-critical planning problems. First, learning-based methods are inherently unpredictable \cite{learning_based_planning} and lack the necessary mathematical framework to provide guarantees on correctness, a strict necessity for path planning of safety-critical systems like our Mars rovers \cite{safety_critical_systems}. Second, they are sample-inefficient to train, sub-optimal for goal-directed navigation tasks, and have so far been validated only in simple domains. Finally, they cannot exploit the compute vs performance trade-off of fine-grained planning to improve the real-time computational efficiency of the system.
In this paper, we present MLNav – a learning-enhanced framework for real-time motion planning that takes the best of the two worlds: the ability to exploit the contextual structure of the environment from learning-based methods and the ability of guaranteeing predictable and safe behaviors from classic search methods. In a nutshell, MLNav is a search-based path planner that uses learned heuristics, where the safety of the chosen path is guaranteed by running a model-based collision checker. The use of learned heuristics within search-based robotic planning has previously been studied [7]–[11], including showing favorable comparisons versus hand-craft heuristics for rover path planning [12]. Our main contribution is proposing a general system design principle for effectively integrating ML-based approaches into existing navigation pipelines of safety-critical robotic systems, as well as a concrete instantiation for Mars rover navigation.

Our system design is driven by a key observation that, in many safety-critical real-world planning setting, collision checking accounts for vast majority of the total computation time [13]. This is because collision checking often involves model-based computation of the 3D geometry of all objects in the scene; furthermore, it has to run numerous times on path/motion options until finding the optimal (or at least feasible) one, particularly in a complex environment. This prompts the question: can we learn a proxy collision heuristic based on a planner’s past experience? This learned heuristics, used as a subroutine, would then be able to guide the motion planner to select paths that have a high probability of being collision-free. In this way, the actual collision checks only need to be performed once on the final selected path for ensuring safety, saving significant computation.

B. Statement of Contribution

Our contributions are as follows:

- We propose a holistic framework for high-stakes planning in complex environments, that makes judicious use of machine learning to enhance planning effectiveness (while guaranteeing safety) in a way that reduces computational and memory overhead. (Sec. III)
- We ground our framework in a real-(out-of-this-)world path planning algorithm, ENav, where the goal is to efficiently find collision-free paths in hazardous terrains. We identify key components in the planning framework that are bottle-necked by poorly-informed reasoning and can benefit from using machine learning (Sec. IV).
- We evaluate using the flight software simulator (ENav Sim). We use both a realistic environment model that was used for the development and testing of ENav for the Perseverance Rover (Sec. VII), as well as a real Martian terrain data collected by the Rover (Sec. VI).
- Our experiments on real Mars data demonstrate a significant improvement in performance over the baseline (ENav), particularly in the number of collision check to run until finding a feasible solution, without any compromise in safety. In challenging cases, ENav fails to return a feasible path whereas MLNav succeeds.

II. RELATED WORK

Planning and navigation have a rich and varied history. In this review of related work, we focus primarily on contemporary machine-learning-based approaches.

A. ML in Planning and Navigation

The success of deep learning has made the use of ML in planning approaches attractive, as it opens up the possibility of learning high-capacity models to reason about complex, real-world environments. Several prior work proposed systems that use of end-to-end navigation approaches that attempt to either replace the entire navigation pipeline from perception to control using black-box image-to-action mappings [14]–[16] or attempt to learn an end-to-end cost mapping based on large amounts of expert demonstrations [17]–[20]. While considerable progress is being made in this direction, they are several challenges towards adoption in real-world safety-critical robotic applications like Mars rovers. In contrast, a few have attempted to replace only particular navigation subsystems such as global planning [21], local planning [22], [23] or improve individual components such as world representation [24]. Our work falls in this latter category, where we try to systematically “unwrap” our Mars rover system and identify collision checking as a bottleneck component that can be improved using ML. We also refer the readers to [4] for a comprehensive review on this topic.

B. ML for Collision Checking

Several related work have also tried to use machine learning to overcome the computational bottleneck of collision-checking. Examples include learning a distribution of promising regions [11], learning heuristics for collision distance metrics such as swept volume [25], employing a lazy approach to evaluate only the most promising edges based on predicted energy costs [10], and learning to accelerate collision checking itself by modeling the configuration space of the robot [26]–[29]. While similar in theme, in the sense they all use learning to improve the bottlenecks of collision checking, all the above approaches, vary considerably in system design. The key design choices are: (1) to operate directly from raw sensing data or a processed representation (e.g., a configuration space); (2) to directly predict the probability of collision or some other measure such as which paths are more “promising”; and (3) the interplay with the model-based collision checker and overall system which ultimately guarantees safety. Which design choices work best (including safety guarantees, and computational efficiency) depends on the system and its goals. To the best of our knowledge, our approach is the first to be validated on a mature system with real-world safety-critical needs.

III. MLNav Framework

A. Overview and Problem Formulation

In this section, we provide a general formulation of our proposed MLNav framework (as shown in Figure 3). Traditionally, the motion planning pipeline for goal-directed real-time autonomous navigation is done in hierarchical receding
horizon manner, with a global planner at the top level driving the robot in a general direction of the goal while a local planner uses the immediate perceived environment (up to a finite sensing horizon) to make sure that the robot avoids obstacles while making progress towards the goal.

For dynamically constrained systems operating in high-dimensional complex environments, a library of candidate trajectories is usually computed offline by sampling from a much larger (possibly infinite) set of feasible trajectories. Such libraries effectively discretize a large control space and enable tasks to be completed with reasonable performance while still respecting computational constraints. Such library-based model predictive approaches have been widely used in state-of-the-art robotic systems [30]. We follow a similar paradigm as well for rovers on Mars [31], and use it for our MLNav framework. Note that the proposed architecture does also generalize to any sampling-based motion planning problem where edge-evaluation is expensive. Let us define a robot at time $t$ with state $\phi(x_t, m)$, where $x_t \in R$ is the pose of the rover operating in a 2.5D static local heightmap $m \in \mathcal{M}$, sampled from a distribution of terrains $p(m)$. Also, let us assume a library $\mathcal{L}$ of $N$ trajectories is given, such that $\mathcal{L} = \{\xi_j\}$, $j = 1$: $N$, $\xi_j \in \Xi$, where $\Xi$ spans the space of all possible trajectories. At each planning cycle the local planner picks the trajectory that yields the least cost $C(\cdot)$ for traversal. We formulate this as an optimization problem:

$$\phi_o \mapsto \xi^* = \arg \min_{\xi \in \mathcal{L}} C(\xi_j) \quad (1)$$

where $\phi_o$ is the represents the initial state of the robot in the map. The cost function being optimized is then defined as:

$$C(\xi_j) = \alpha \cdot C_{\text{goal}}(\xi_j, \phi_o) + \beta \sum_{t=1}^{T} C_{\text{collision}}(\xi_j, \phi(x_t, m)) \quad (2)$$

where $C_{\text{goal}}$ is the cost for path execution, such as the time to get to the final goal, consisting of the cost within the planning horizon and the cost-to-go beyond the horizon, which usually comes from a separate global planner. This part is fast to compute but does not account for vehicle safety. $C_{\text{collision}}$ is the expected collision cost for each trajectory, computed by estimating the clearance of the robot with the local terrain features over the planning horizon of $T$ time steps. In a deterministic environment $C_{\text{collision}} \in \{0, \infty\}$; in an uncertain environment, it represents the probability of collision. The collision cost is typically computed by repeatedly running a collision checking algorithm at a certain interval over the candidate trajectories. The computation time of collision cost grows proportionally with both the number of trajectory options $N$ and the length of the horizon $T$.

**B. Learned Proxy Collision Heuristics**

We propose to alleviate this bottleneck by leveraging the planner’s past experience to learn a proxy collision heuristic:

$$C_{\text{proxy,collision}}(\mathcal{L}) = f_h(\phi(x, m)) \quad (3)$$

where the function $f_h$ maps the rover’s state and local terrain features, $\phi(x, m)$, to a probability of collision. While any

**IV. Deployment for Mars Rover Navigation**

Next, we ground our MLNav framework in a specific problem domain and study it in the context of path planning for Mars rovers, based on the ENav algorithm.
A. Overview of ENav

ENav is a tree-based planner that considers a parametrized tree of candidate trajectories, represented by $L$ in Eq. 1. At each planning cycle, it generates a 2.5D terrain height map $m$ from stereo imagery. $C_{\text{goal}}$ is the estimated time to reach the final goal, with an additional penalty based on terrain roughness. $C_{\text{collision}}$ is computed by a collision checking algorithm called Approximate Clearance Evaluation (ACE) [32]. The computation of $C_{\text{collision}}$ is by far more complex in comparison to $C_{\text{goal}}$. Therefore, to save the precious on-board computational resource, ENav gives up finding the optimal path and instead makes the following modifications to optimal path planning in Eq. 1. First, before running ACE, ENav computes $C_{\text{goal}}$ for all candidate trajectories in $L$ and sort them by $C_{\text{goal}}$. Then, ENav greedily evaluates $C_{\text{collision}}$ of the trajectories from the top of the sorted list. It cuts off the search if at least one feasible path is found and a pre-specified threshold on the number of ACE execution is reached, even if some trajectories in $L$ remain unevaluated. Finally, ENav chooses the best trajectory among the ones that have evaluated before the cut off.

The vast majority of the computation time of ENav is for computing $C_{\text{collision}}$ with ACE. In the worst case where a sole feasible path is ranked at the bottom of the sorted list, ENav needs to run ACE on all candidates in $L$; if there are multiple feasible paths but the optimal one is located below the cut off threshold, a suboptimal path may be chosen. This is why ENav performance could be enhanced by introducing a heuristic to better sort $L$.

B. MLNav implementation on ENav

In order to more effectively sort ENav’s list of candidate rover paths, such that the most highly ranked paths are feasible and near-optimal, we use the learned proxy collision heuristics. The updated cost function then is as follows:

$$C_{\text{total}}(\xi_j) = \alpha \cdot C_{\text{goal}}(\xi_j, \phi_o) + \beta \cdot C_{\text{proxy},\text{ACE}}(L)$$

where $C_{\text{proxy},\text{ACE}}$ is computed as a look-up using the predicted heuristic map.

In our particular implementation for MLNav, we use a model based on a U-Net [33], and learn it in a supervised manner. The input is a height-map $m$ and the output is a proxy collision heuristic map, such that the value for each pixel in the heuristic map corresponds to the predicted collision probability. The collision cost at any given spatial point in the height-map depends not only on the terrain features but also on the rover heading. We encode the rover heading as part of the learning problem itself by extending the model output to have a multi-channel representation such that each channel represents a cardinal heading angle (see Figure 3). Note that the granularity of the discretization depends on the specific instantiation of the MLNav framework. For this implementation, we found a discretization of 8 heading angles (at 45 degree intervals) to be sufficient. Sigmoid activation is then applied to each channel to give a value in the range [0, 1]. Training data is collected by running ACE on synthetic terrains that are representative of Martian terrain. Note that the training set does not include real Mars terrains although our test does, as described in Section VI.

V. PERFORMANCE EVALUATION

A. Experimental Setup

For all our experiments presented in this section, we used a ROS-based, high-fidelity simulation environment called ENav Sim [34] that was originally developed for prototyping and testing of Perseverance’s ENav algorithm. ENav Sim is capable of generating a rich set of varied 2.5D terrains representative of the candidate Mars landing sites, producing synthetic stereo images for simulating onboard heightmap generation, and simulating the rover’s motion. At its core, the simulator wraps a flight software implementation of ENav and a software library called HyperDrive Sim (HDSim), which has been used for terrain simulation for multiple Mars rover missions. HDSim provides an image rendering capability based on the rover’s navigation cameras, rover-terrain settling and a realistic slip model. Furthermore, ENav Sim provides a method for running a large-scale Monte Carlo simulation in parallel and automatically generate reports that capture the key ENav performance metrics. The learned heuristic model was implemented in Tensorflow and setup as a separate subroutine, that could be invoked by the ENav algorithm on-demand as a ROS service call.

B. Training and Validation of Learned Heuristics

We first evaluated the fidelity of our proposed model to learn the proxy collision heuristics. Training data was gathered by running Monte Carlo simulations of the baseline ENav algorithm on 1500 terrains and randomly sampling 8 heightmaps from each trial. For each cell in each sampled height-map, the ACE costs were estimated for eight fixed rover heading values at 45$^\circ$ intervals, resulting in a 8-channel “ACE map” such that each channel corresponded eight heading-specific ACE values. This ACE-map was then used as the training signal for our modified U-Net model. A
of 7% or less, and (2) Complex - terrains with greater slope or CFA. For ENav, the default tree of paths used in our experiments is composed of 14 candidate turns-in-place, followed by 11 3-meter arcs of various curvatures, followed by another set of 11 3-meter arcs. Thus, at each step the rover is planning about 6 meters ahead of its current position, and considering 1694 potential paths. Note that the hardware of the Perseverance rover does not allow steering while driving, and thus the rover can only move in fixed-curvature arcs. For evaluation, the following performance metrics were used:

- **Success Rate**: Defined as the percentage of trials that result in the rover reaching the goal without timing out or violating safety constraints. Higher values are better.
- **Path Inefficiency**: Defined as the average excess length of the path taken by the rover, as compared to a straight line path from straight to goal, expressed as a percentage. For example, if the rover drives 110 m to reach a goal that is 100 m in straight line distance, the path inefficiency is 10%. Lower values are better.
- **Number of Collision Checks**: Defined as the average number of planning cycles required above a minimum number of ACE checks (default: 275), defined in percentage. When the number of ACE checks exceed a threshold, it indicates that the highest ranked paths were all deemed unsafe, and the rover may need to stop and “overthink” until a solution is found. Lower values are better.
- **Overthink Rate**: Defines as the percentage of trials that result in the rover reaching the goal without timing out or violating safety constraints. Higher values are better.

Next, we performed an ablation study to justify the design choice towards using the U-Net model architecture. In particular, the MLNav framework was designed with computational efficiency as the primary focus. However, one might ask the question: does using a larger state-of-the-art models from semantic segmentation help? Table II shows the performance of different models on learning the proxy collision heuristics. We observe that U-Net provides the best performance even though it is the smallest model. Our hypothesis is that while bigger models have a much larger capacity to learn generic representations required for visual recognition tasks, they also require larger training sets. In contrast, U-Net was specifically designed for biomedical applications, where it could be trained with very few images.

### C. Benefits to Mars Rover Navigation

In this section, we evaluate the potential benefits to the overall pipeline, in the context of the rover navigation. Monte Carlo simulations were run for both the baseline ENav and MLNav, using the same set of terrains. Note that the terrains used for these experiments are a separate set from the set of terrains used to train the learned model, to ensure that the observed performance is not biased. The Monte Carlo simulation consisted of 1500 terrains with various slope and rock density, which is quantified by CFA (cumulative fraction of area) [35]. We report out results on two categories: (1) Benign - terrains with slope less than 15° and CFA value of 7% or less, and (2) Complex - terrains with greater slope or CFA. For ENav, the default tree of paths used in our experiments is composed of 14 candidate turns-in-place, followed by 11 3-meter arcs of various curvatures, followed by another set of 11 3-meter arcs. Thus, at each step the rover is planning about 6 meters ahead of its current position, and considering 1694 potential paths. Note that the hardware of the Perseverance rover does not allow steering while driving, and thus the rover can only move in fixed-curvature arcs. For evaluation, the following performance metrics were used:

- **Success Rate**: Defined as the percentage of trials that result in the rover reaching the goal without timing out or violating safety constraints. Higher values are better.
- **Path Inefficiency**: Defined as the average excess length of the path taken by the rover, as compared to a straight line path from straight to goal, expressed as a percentage. For example, if the rover drives 110 m to reach a goal that is 100 m in straight line distance, the path inefficiency is 10%. Lower values are better.
- **Number of Collision Checks**: Defined as the average number of planning cycles required above a minimum number of ACE checks (default: 275), defined in percentage. When the number of ACE checks exceed a threshold, it indicates that the highest ranked paths were all deemed unsafe, and the rover may need to stop and “overthink” until a solution is found. Lower values are better.
- **Overthink Rate**: Defines as the average number of planning cycles required above a minimum number of ACE checks (default: 275), defined in percentage. When the number of ACE checks exceed a threshold, it indicates that the highest ranked paths were all deemed unsafe, and the rover may need to stop and “overthink” until a solution is found. Lower values are better.

Table II shows the results from the MC simulations. We observe that as using the learned heuristics there is a significant improvement in the overall efficiency of the ENav algorithm. In particular for complex terrain, there was a 20% reduction in path inefficiency, 25% reduction in number of collision checks and a 65% reduction in overthink rate using the MLNav framework, as compared to the baseline ENav algorithm. This improvement in computational and performance efficiency comes at almost no cost; the success rate of MLNav is comparable to the baseline, and within the experimental margin of error. Furthermore, one of the primary objectives of this work was to design a framework that could leverage the benefits of ML without compromising on safety guarantees of the vehicle. Based on our extensive MC simulations, we observe that using the learned model provides a significant improvement in overall efficiency, while maintaining safety guarantees. For the cases where the default ENav does not lead to a solution within the allowed timeframe, the MLNav framework is able to find a solution within the given constraints.
simulations, our assertion holds. In all our experiments, no trial failures were reported due to violated safety constraints; all failures are due to either timeouts or failures to find paths to the goal. This is the most significant result of this paper.

D. Sensitivity & Ablation Analyses

Next, we performed ablation studies on the sensitivity of the MLNav framework to two key parameters:

1) Performance vs. minimum ACE threshold: For all our experimental trials above, we define a threshold value of 275 for the minimum number of ACE checks. In practice, this number can be thought of as the computational budget allocated to the planner and also protects against choosing high-risk paths with finite but poor ACE cost. However, if the learned heuristics is working optimally, one could imagine that further evaluation beyond the first safe trajectory is no longer needed and this computation might be better used elsewhere. In order to study this trade-off we set the minimum ACE thresholds to 0 and repeated the above experiments. We call this version MLNav($\dagger$), and tabulate the results in Table I. For comparison, we also run ENav with the minimum ACE threshold set to 0, shown as Baseline($\dagger$) in the table. This intuitively means that MLNav($\dagger$) and Baseline($\dagger$) always choose the first feasible path found and does not search any further. Interestingly, the difference in path inefficiency between MLNav and MLNav($\dagger$) (also between Baseline and Baseline($\dagger$)) is within the error margin. Furthermore, MLNav($\dagger$) substantially improved on the number of collision checks and overthink rate. Since ACE is run every 25 cm, evaluating a single 6 m path requires 24 ACE runs. Therefore, the average number of ACE checks being 39 and 90 for benign and complex terrains respectively means that MLNav($\dagger$) finds a feasible path only after evaluating 1.6 and 3.7 options. These results imply that the learned heuristics almost always ranks the optimal path near the top of the list.

2) Performance vs. Tree Design: The above experiments demonstrated that MLNav can significantly reduce computation time in the ENav planning. We next ask: Can this surplus computation be put to use back in planner to improve the success rate? We evaluate three different approaches to increase the complexity of the tree. First, by increasing the number of candidate trajectories in the library, which can increase the probability of finding a safe and efficient path towards goal. Here, we increase the branching factor of possible actions at each tree-depth by 4, leading to 4050 candidate trajectories. Note: this could increase the computation time by 2-3x for the baseline ENav algorithm. The results in Table I - MLNav($\dagger$) shows a small improvement in overall success rate with further reduction in computation. Second, we use a deeper tree by adding a set of 11 arcs to previous leaf nodes, extending the planning horizon to 9m. This led to poor results (tabulated at MLNav($\dagger$)DT), where the success rate dropped to 59.9% without any improvement in other metrics. We believe this is due to other system-level uncertainties such as stereo, slip, etc which may be significantly worse that far out. Finally, we use a more complex tree design with variable length arcs at different depth layers - the first layer was 11 turn-in-place, followed by 15 1-meter arcs, 11 2-meter arcs and 7 3-meter-arcs. The results using this tree design (tabulated as MLNav($\ddagger$) (VLT)) show an increase in success rate to 78.8%. As the average number of collision checks for MLNav($\ddagger$) was still lower compared to the baseline, we can conclude that with MLNav, now we can indeed use a more complex tree design to achieve a higher success rate in complex terrains without increasing the computational budget.

VI. MLNAV ON REAL MARS DATA

A. Experimental Setup

We now validate MLNav on ENav Sim using real data from Mars collected by the Perseverance rover. A challenge to this approach is that we do not usually have the complete set of onboard images for reconstructing the terrain due to the limitation in communication data volume between the two planets. In particular, only the last and the penultimate images are typically transmitted for manual drives and, as of the writing of this paper, there have only been 17 occasions where the rover drove fully autonomously. Within these limitations, we chose two data sets acquired on the following Sols (Martian days since landing) for our test venues:

a) Sol 122: The drive on Sol 122 was the last of a series of first-time activities (FTAs) for commissioning ENav on Perseverance. On this Sol, the rover was commanded to drive fully autonomously over ~30 m northwards. As the “final test” for ENav, the goal was intentionally set behind a rock, seen in Figure 4-left, such that the rover had to deviate from the straight-line path to get to the goal. Due to the need for detailed assessment of this particular drive, all of the on-board stereo image pairs were transmitted to Earth, which allowed us to reconstruct the terrain completely. ENav completed the drive successfully.

b) Sol 178: It was one of the most challenging drives of Perseverance to date. The ~85 m drive started with ~15 m of manual driving, followed by fully autonomous driving along a ridge that concluded by climbing a slope to reach a science target. There were large rocks and exposed bedrock along the
Fig. 4. Terrains used for experiments. Images taken by Perseverance on Sol 122 (left) and 178 (right). Image: NASA/JPL-Caltech

Fig. 5. Paths planning on real Martian terrain (Perseverance Sol 122) with Baseline (ENav; top) and MLNav (bottom).

ridge and on each side of the slope as seen in Figure 4-right. Image pairs and onboard heightmaps were downlinked from the last ~15 m segment of this drive, allowing us to recreate the corresponding portion of the terrain in simulation. ENav completed the drive successfully.

For each Sol, we first ran stereo processing to produce DEMs (digital elevation models). The DEMs were then mosaiced using the onboard pose updates to reconstruct the 3D terrain with a 0.05 m resolution. In simulation, the start and goal were chosen to closely match Perseverance’s actual drive path. We used the MLNav† (VLT) setting. Other parameters were identical to the ones we used in the actual Sol 122 and 178 drives.

B. Results

The results on both Sols agree with the results on synthetic terrains presented in the previous section. Both ENav and MLNav found feasible paths to the goal as expected, and the paths are qualitatively similar. This is consistent with the result in Table I that MLNav gives relatively minor improvement in path inefficiency, particularly on benign terrains. Since both of Sol 122 and Sol 178 terrains fall under the “benign” category in Table I, it is expected that the baseline algorithm can find a path as good as MLNav. Notable differences were observed when the rover avoided obstacles. For example, Figure 5 is the visualization of Sol 122 drives by ENav and MLNav when the rover was avoiding a rock. The blue and pink lines seen on the ENav visualizations are the paths on which ENav ran ACE evaluations and ended up with not choosing. Observe that there is only one blue line in the MLNav drive at the same location. This means that the second-ranked path based on ML heuristics turned out to be feasible, hence MLNav ran ACE only on two paths in this planning cycle. Of course, MLNav occasionally needed to ran ACE on many path options until finding a feasible one, but overall MLNav ran substantially smaller number of ACE collision checks and, in the majority of the planning cycles, it evaluates only a single path. The complete movies of Sol 122 and 178 drives are attached as supplemental materials.

Table III shows the quantitative results. On both sols, there was only trivial changes in path inefficiency, indicating that MLNav resulted in a qualitatively similar path as discussed above. In contrast, a substantial improvement in the number of ACE checks is observed. Again, this is because the top-ranked path evaluated by the ML-based heuristics was often feasible even in the presence of obstacles. While the number of test cases with real Martian terrain was limited for practical reasons mentioned above, this experiment demonstrated the ability of MLNav to improve the performance of path planning in a real environment. This result is particularly remarkable because, as explained in Section IV, the training data that we used for this experiment was produced solely with synthetic terrains before the landing of the rover.

VII. Conclusion

In this paper we presented MLNav – a holistic framework for high-stakes planning that allows resource-constrained robotic systems to effectively navigate in complex environments while guaranteeing safety. Our main contribution was a general system design principle for effectively integrating ML methods into existing navigation pipelines of safety-critical robotic systems. We studied the efficacy of our framework through a concrete case study on Mars rover navigation,
and demonstrated substantial improvements across several key performance metrics, using high-fidelity simulations with both real Martian terrain collected by the Perseverance rover and a suite of challenging synthetic terrains. In future work, we plan to further validate the performance of MLNav through end-to-end demonstration on an analog Mars rover.

VIII. ACKNOWLEDGEMENT

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and California Institute of Technology under a contract with the National Aeronautics and Space Administration, and supported in part by Raytheon. The authors would like to thank Olivier Toupet, Mitchell Ingham and Ravi Lanka for valuable discussions and problem formulation, and the JPL Research and Technology Development (R&TD) program for supporting this research.

REFERENCES

[1] M. McHenry, N. Abcouwer, J. Biesiadecki, J. Chang, T. Del Sesto, A. Johnson, T. Litwin, M. Maimone, J. Morrison, R. Rieber, O. Toupet, and P. Two, “Mars 2020 autonomous rover navigation,” in Meeting of the American Astronomical Society, 2020.

[2] A. Rankin, M. Maimone, J. Biesiadecki, N. Patel, D. Levine, and O. Toupet, “Mars curiosity rover mobility trends during the first 7 years,” Journal of Field Robotics, vol. 38, no. 5, pp. 759–800, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.22011.

[3] D. Hsu, J.-C. Latombe, and R. Motwani, “Path planning in expansive configuration spaces,” in Proceedings of International Conference on Robotics and Automation, vol. 3. IEEE, 1997, pp. 2719–2726.

[4] X. Xiao, B. Liu, G. Warnell, and P. Stone. “Motion control for mobile robot navigation using machine learning: a survey,” arXiv preprint arXiv:2011.13112, 2020.

[5] V. Dhiman, S. Banerjee, B. Griffin, J. M. Siskind, and J. J. Corso, “A critical investigation of deep reinforcement learning for navigation,” arXiv preprint arXiv:1802.02274, 2018.

[6] N. Hunt, N. Fulton, S. Magliacane, N. Hoang, S. Das, and A. Solar-Lezama. “Verifiably safe exploration for end-to-end reinforcement learning,” arXiv preprint arXiv:2007.01223, 2020.

[7] J. S. Smith, J.-H. Hwang, F.-J. Chu, and P. A. Vela. “Learning to navigate: Exploiting deep networks to inform sample-based planning during vision-based navigation,” arXiv preprint, 2018.

[8] B. Riviere, W. Hoening, M. Anderson, and S.-J. Chung. “Neural tree expansion for multi-robot planning in non-cooperative environments,” arXiv preprint arXiv:2104.09705, 2021.

[9] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, “Motion planning networks: Bridging the gap between learning-based and classical motion planners,” IEEE Transactions on Robotics, vol. 37, no. 1, pp. 48–66, 2020.

[10] S. Choudhury, M. Bhardwaj, S. Arora, A. Kapoor, G. Ranade, S. Scherer, and D. Dey. “Data-driven planning via imitation learning,” The International Journal of Robotics Research, vol. 37, no. 13-14, pp. 1632–1672, 2018.

[11] B. Ichter, J. Harrison, and M. Pavone. “Learning sampling distributions for robot motion planning,” in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 7087–7094.

[12] N. Abcouwer, S. Daftry, T. del Sesto, O. Toupet, M. Ono, S. Venkatraman, R. Lanka, J. Song, and Y. Yue. “Machine learning based path planning for improved rover navigation,” in 2021 IEEE Aerospace Conference (50100), 2021, pp. 1–9.

[13] M. Elbanhawi and M. Simic. “Sampling-based robot motion planning: A review,” IEEE access, vol. 2, pp. 56–77, 2014.

[14] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” The Journal of Machine Learning Research, vol. 17, no. 1, pp. 1334–1373, 2016.

[15] S. Daftry, J. A. Bagnell, and M. Hebert. “Learning transferable policies for monocular reactive mav control,” in International Symposium on Experimental Robotics. Springer, 2016, pp. 3–11.

[16] M. Pfeiffer, M. Schaebble, J. Nieto, R. Siegwart, and C. Cadena. “From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots,” in IEEE International Conference on Robotics and Automation. IEEE, 2017.

[17] P. Abbeel and A. Y. Ng. “Apprenticeship learning via inverse reinforcement learning,” in Proceedings of the twenty-first international conference on Machine learning, 2004, p. 1.

[18] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. “Maximum entropy inverse reinforcement learning.” in Aaai, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[19] M. Wulfmeier, D. Z. Wang, P. Onodruga, and I. Posner. “Large-scale cost function learning for path planning using deep inverse reinforcement learning,” The International Journal of Robotics Research, vol. 36, no. 10, pp. 1073–1087, 2017.

[20] M. Wulfmeier, D. Z. Wang, and I. Posner. “Watch this: Scalable cost-function learning for path planning in urban environments,” in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2016, pp. 2089–2095.

[21] X. Yao, J. Zhang, and J. Oh. “Following social groups: Socially compliant autonomous navigation in dense crowds,” arXiv preprint arXiv:1911.12063, 2019.

[22] W. Gao, D. Hsu, W. S. Lee, S. Shen, and K. Subramanian. “Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation,” in Conference on Robot Learning. PMLR, 2017.

[23] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson. “Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning,” in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 5113–5120.

[24] C. Richter and N. Roy. “Safe visual navigation via deep learning and novelty detection,” 2017.

[25] H.-T. L. Chiang, A. Faust, S. Sugaya, and L. Tapia. “Fast swept volume estimation with deep learning,” 2018.

[26] N. Das and M. Yip, “Learning-based proxy collision detection for robot motion planning applications,” IEEE Transactions on Robotics, vol. 36, no. 4, pp. 1096–1114, 2020.

[27] Y. Han, W. Zhao, J. Pan, and Y.-J. Liu. “Configuration space decomposition for learning-based collision checking in high-dof robots,” in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2020, pp. 5678–5684.

[28] J. Hub and D. D. Lee. “Learning high-dimensional mixture models for fast collision detection in rapidly-exploring random trees,” in 2016 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2016, pp. 63–69.

[29] J. C. Kew, B. Ichter, M. Bandari, T.-W. E. Lee, and A. Faust. “Neural collision clearance estimator for fast robot motion planning,” arXiv preprint arXiv:1910.05379, 2019.

[30] D. Dey, K. S. Shankar, S. Zeng, R. Mehta, M. T. Agcayazi, C. Erikson, S. Daftry, M. Hebert, and J. A. Bagnell. “Vision and learning for deliberative monocular cluttered flight,” in Field and service robotics. Springer, 2016, pp. 391–409.

[31] J. Carsten, A. Rankin, D. Ferguson, and A. Stentz. “Global path planning on board the mars exploration rovers,” in 2007 IEEE Aerospace Conference. IEEE, 2007, pp. 1–11.

[32] K. Orsu, G. Matheron, S. Ghosh, O. Toupet, and M. Ono. “Fast approximate clearance evaluation for rovers with articulated suspension systems,” Journal of Field Robotics, vol. 37, no. 5, pp. 768–785, 2020.

[33] O. Ronneberger, P. Fischer, and T. Brox. “U-net: Convolutional networks for biomedical image segmentation,” in International Conference on Medical image computing and computer-assisted intervention. Springer, 2015, pp. 234–241.

[34] O. Toupet, T. Del Sesto, M. Ono, S. Myint, J. Vander Hook, and M. McHenry. “A ros-based simulator for testing the enhanced autonomous navigation of the mars 2020 rover,” in 2020 IEEE Aerospace Conference. IEEE, 2020, pp. 1–11.

[35] M. P. Golombek, A. Haldemann, N. Forsberg-Taylor, E. N. Dimaggio, R. Schroeder, B. Jakosky, M. Mellon, and J. Matijevic. “Rock size-distance correlation searching in non-cooperative environments,” in Proceedings of the twenty-first international conference on Robotics and Automation (ICRA). IEEE, 2003, p. 1.