MA3: Model-Accuracy Aware Anytime Planning with Simulation Verification for Navigating Complex Terrains

Manash Pratim Das,[1] **Damon M Conover,**[2] **Sungmin Eum,**[2,3] **Heesung Kwon,**[2] **Maxim Likhachev**[1]
[1] Carnegie Mellon University
[2] DEVCOM Army Research Laboratory (ARL)
[3] Booz Allen Hamilton
mpratimd@andrew.cmu.edu, damon.m.conover.civ@army.mil, eum_sungmin@bah.com, heesung.kwon.civ@army.mil, maxim@cs.cmu.edu

Abstract

Off-road and unstructured environments often contain complex patches of various types of terrain, rough elevation changes, deformable objects, etc. An autonomous ground vehicle traversing such environments experiences physical interactions that are extremely hard to model at scale and thus very hard to predict. Nevertheless, planning a safely traversable path through such an environment requires the ability to predict the outcomes of these interactions instead of avoiding them. One approach to doing this is to learn the interaction model offline based on collected data. Unfortunately, though, this requires large amounts of data and can often be brittle. Alternatively, models using physics-based simulators can generate large data and provide a reliable prediction. However, they are very slow to query online within the planning loop. This work proposes an algorithmic framework that utilizes the combination of a learned model and a physics-based simulation model for fast planning. Specifically, it uses the learned model as much as possible to accelerate planning while sparsely using the physics-based simulator to verify the feasibility of the planned path. We provide a theoretical analysis of the algorithm and its empirical evaluation showing a significant reduction in planning times.

1 Introduction

Search-based motion planning problems aim to search for a feasible path in a graph from a vertex defined as ”start” to a vertex defined as ”goal”. A path consists of consecutive edges. While an edge connects a pair of vertices, a vertex might have multiple edges connected with it. Moreover, each edge will have a non-negative cost associated with it, and the cumulative sum of the costs associated with the edges that form a path in the graph refers to the cost of the path. The cost of an edge can also be infinite, which indicates that the edge is not valid, and a feasible path refers to a path that does not contain any such invalid edges. In this work, we focus on the class of planning problems, where it is time-consuming to determine the cost of an edge. Consequently, it is expensive w.r.t. time to determine if an edge is invalid. As a motivating example, the path planning problem for an autonomous ground robot that traverses over a complex off-road environment is one such problem.

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

In an unstructured off-road environment, the robot can encounter a combination of environmental elements that generates a complex interaction. For example, the environment may contain tall grass, bushes, low-hanging tree branches, rocks, gravel, mud, ridges, rough terrain, etc. The robot would encounter very complex physical interactions with them, making the interaction practically infeasible to predict in closed-form. To plan a path for the vehicle through this terrain, we can set up a state-lattice graph that spans the region to be traversed. The vertices in this graph can represent the states of the robot over the terrain, and the edges can represent a sequence of actions validated by an “expert” that will make the robot traverse between the states.

We use the term “expert” to refer to an entity that can predict traversability and, in particular, whether an edge is valid in the context of the graph we defined above. Historically analytical models (Al-Milli, Althoefer, and Seneviratne 2007; Gennery 1999), machine learning models (Chavez-Garcia et al. 2017; Hirose et al. 2018), and simulation-based (Chavez-Garcia et al. 2018; Al-Milli, Seneviratne, and Althoefer 2010) models have been popular choices to serve as “experts”. (Nampoothiri et al. 2021) provides a recent survey of models used in traversability prediction. In general, there is a trade-off between the time required to query a model and its accuracy. Consider the following example: A neural network model can look at the scene in front of the vehicle using onboard sensors to predict traversability very quickly. However, their prediction is based on interpolation, which does not reason about the complex physics of interaction. On the other hand, a high-fidelity simulator can reason about physics and provide a more reliable prediction.

We propose an anytime planning algorithm MA3[1] that can utilize a learning-based and a simulation-based “expert” to evaluate edges. The learning-based evaluator is approximate but quicker, while the simulation-based evaluator is accurate. MA3 runs two parallel threads. The key idea is to use the quicker edge-evaluator to guide the search in one thread and use the accurate edge-evaluator “minimally” to verify the solution or correct mistakes in another thread. Accounting for the mistakes helps MA3 ensure completeness and bounded-suboptimality guarantees.

[1] Source code and demo: manashpratim.com/ma3-planner

2 Problem Formulation

As an example, let us use the off-road navigation problem. Consider a 4-wheel drive robot with Ackermann steering. The off-road environment contains only one type of terrain with no obstacles such as rocks, trees, etc. In particular, this terrain is traversable everywhere except for where the elevation is very rough, and the robot cannot traverse over it, this may be either because it does not have enough power or because it gets stuck due to physical limitations.

Setup: Let the state $(x, y, \theta) \in \mathbb{R}^2 \times S^1$ of the robot be defined by the 2D location of the robot on the surface $x, y \in \mathbb{R}^2$ and the heading angle $\theta \in S^1$. Let $(.)_w$ and $(.)_b$ be used to define the coordinates relative to a world frame and the body frame of the vehicle, respectively. Let $\mathcal{G}_{\mathcal{V},\mathcal{E},W}$ represent a graph parameterized by the vertices $\mathcal{V}$, edges $\mathcal{E}$ and a map $W : \mathcal{E} \to \mathbb{R}^+$ which gives the cost of an edge. We generate this graph offline with optimistic edges. During online planning, only the edge-costs are updated to reflect non-traversable edges. The vertices are generated by discretizing the state-space uniformly. The edge connections between edges are generated as follows. We first define a set of local targets M', where each target $m = (dx_b, dy_b, d\theta_w, dt)$ specifies a small-change in state-space. We then check if the vehicle starting from any state say $s = (x_b, y_b, \theta_w)$ can reach a new state determined by this relative change $s' = (x_b + dx_b, y_b + dy_b, \theta_w + d\theta_w)$ within a time-limit dt based on its dynamics constraints and under "optimistic conditions". In our case, this condition refers to a flat world $\mathcal{W}^*$ without any elevation variation, which is traversable everywhere. We store such relative and local targets m that are traversable under optimistic conditions in a set M. This set can be used to generate successors for an implicit graph from any state during planning. Again, note that, in a real-world $\mathcal{W}$, the edge between a state and its successor may not be traversable due to the elevation variations (Fig. 1).

Let $W_{\text{optimistic}}$ and $W_{\text{sim},\mathcal{W}}$ map the cost of traversing an edge in $\mathcal{W}^*$ and $\mathcal{W}$ respectively, such that $W_{\text{optimistic}}(e) \leq W_{\text{sim},\mathcal{W}}(e)$ holds true for every edge. Let $\mathbb{M}_{\text{sim}}$ and $\mathbb{M}_{\text{ml}}$ denote the simulator and the ML model, respectively. The cost of an edge e in $W_{\text{sim},\mathcal{W}}$ that connects the states say (s, s') determined by the target m can only be revealed by simulating the robot in the world $\mathcal{W}$ at s to see if it can reach s' within dt. This call to the simulator is denoted by QUERYSIMMODEL(e) and returns ∞ if s' could not be reached. In contrast, a query to $\mathbb{M}_{\text{ml}}$ is denoted by the function QUERYMLMODEL : $\mathcal{E} \to \{\text{true}, \text{false}\} \times [0, 1]$ and is expected to return a binary decision on the validity of the queried edge, and a confidence score about its prediction. $\mathbb{M}_{\text{ml}}$ can use local features $f(s, m)$ which depend on s and m to make its prediction. For example, we use an ensemble of convolutional neural network (CNN) classifiers which takes a two-channel image as the input. The first channel contains the elevation map of the region around the robot centered and oriented w.r.t. s_b, and the second channel is a binary image which contains information about m (Please refer to Fig. 5 in Section 5.2).

Finally, we are given 1) a world $\mathcal{W}$, 2) a graph $\mathcal{G}_{\mathcal{V},\mathcal{E},W}$, 3) a pair of user-specified start and goal states in the graph $(s_{\text{start}}, s_{\text{goal}})$, and 4) two edge-evaluators $\mathbb{M}_{\text{ml}}$, $\mathbb{M}_{\text{sim}}$. The problem statement is to finds a path $\pi = \{e_1, e_2, \ldots\}$ such that the cost of the path $c_\pi = \sum_{e_i \in \pi} W_{\text{sim},\mathcal{W}}(e_i)$ satisfies 1) $c_\pi \neq \infty$, and 2) $c_\pi \leq \omega_{\text{sub}} c_{\pi^*}$, where c_{π^*} is the cost of the optimal path, and $\omega_{\text{sub}} \in [0, \infty)$ is a user-defined parameter.

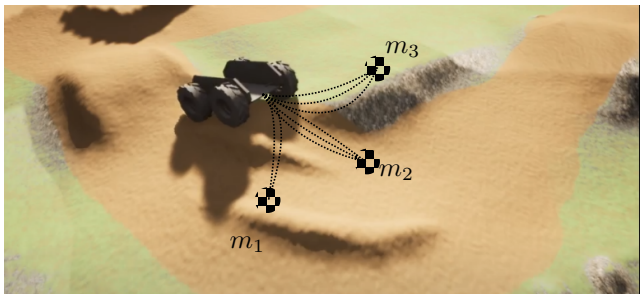

Figure 1: Three targets are shown here. The dotted connections indicates that the controller of the vehicle can take any local path to reach the targets. Given the terrain elevation, whether the controller in simulation can reach a target within the time-limit specifies if the edge is valid.

3 Related Work

For search-based planning problems where edge evaluations dominate the run-time of the planner, LAZYSP (Mandalika et al. 2019) class of algorithms are proven to evaluate the minimum number of edges (edge-optimal) (Haghtalab et al. 2018) required to find the optimal path. The main idea behind this class of algorithms is to generate path candidates in order of their potential to be the optimal path and to keep eliminating them if they are invalid until you find the first valid path. Enforced by the order in which these candidates are considered, the first valid path is guaranteed to be optimal. These algorithms are edge-optimal as they use an optimistic edge-costs $W_{\text{optimistic}}$ to search the graph for the candidate paths. And query the time consuming evaluator $W_{\text{evaluator},\mathcal{W}}[e]$ only to eliminate candidates. LAZYSP utilizes a shortest path planner like A* (Hart, Nilsson, and Raphael 1968) internally. Variants of A* which aims at reducing edge-evaluations are LAZYA* (Cohen, Phillips, and Likhachev 2015) and rational-LAZYA* (Karpas et al. 2018).

Our work builds upon the LAZYSP framework such that in addition to the expensive but error-free edge-evaluator, the planner can also utilize an error-prone but relatively faster edge-evaluator. While (Cohen, Phillips, and Likhachev 2015; Haghtalab et al. 2018; Mandalika et al. 2019) are designed to search directly for the optimal path, our Algorithm 1 is an Anytime algorithm. Note that PSMP (Hou et al. 2020) is also based on LAZYSP and is an anytime algorithm. However, it is not related to our problem. PSMP utilizes additional global information about the map to accelerate the elimination of path candidates. However, MA3 does not require global information; rather, the additional edge-evaluator only uses local information around the edge. Moreover, PSMP only affects the LAZYEDGESELECTOR function (see Algorithm 1) which we borrowed from LAZYSP, and hence MA3 can be extended like PSMP to utilize global information.

4 Approach

The key idea in MA3 is to delay edge-evaluations by $\mathbb{M}_{\text{sim}}$ and carry out the regular LAZYSP-like search using $\mathbb{M}_{\text{ml}}$. Ideally, the simulator is queried only to verify a path found valid by $\mathbb{M}_{\text{ml}}$. However, some additional simulation queries are made to evaluate 1) edges where the model $\mathbb{M}_{\text{ml}}$ is not confident about its prediction, and 2) edges that have been marked as invalid by $\mathbb{M}_{\text{ml}}$ with high confidence. Fig. 2 shows an example run-through of MA3.

Notations: First we introduce some additional notations used in Algorithm 1. Relevant line numbers are shown as $\{\dots\}$. Multiple priority values $(p_1, p_2, \dots)$ ordered in descending order of their significance can be used to determine the priority of an element in a priority queue. Lesser significant priority is only used to break ties. Finally, the procedure $\text{SP}(s_{\text{start}}, s_{\text{goal}}, W_{\text{current}})$ calls A* or its variants (like LPA* (Koenig, Likhachev, and Furcy 2004)) to return 1) a least-cost path π based on the edge costs W_{current}, and 2) the cost of the path c_π. While $W_{\text{current}}[e]$ gives the cost of an edge e, $W_{\text{current}}[e]$.model contains the information about the source of the edge cost, which can take either of the following values {init, temp, ml, sim, ppe}.

- init: cost are based on $W_{\text{optimistic}}$ $\{7\}$
- temp: if cost is set to ∞ temporarily $\{35\}$
- ml: cost updated based on $\mathbb{M}_{\text{ml}}$ evaluation
- sim: cost updated based on $\mathbb{M}_{\text{sim}}$ evaluation
- ppe: potentially pessimistic edge (explained in 4.2)

4.1 Lazier Simulation Evaluations

First, let us relate the parts of our algorithm similar to LazySP. Like LazySP, A* always uses a edge-cost-map W_{current} (which is initialized with $W_{\text{optimistic}}$) to compute the least-cost path {9 and 41}. During the search, this data structure is updated by calling the OVERWRITE function to contain the most-relevant cost for all the edges. Specifically, we store two information about an edge, 1) its cost in $W_{\text{current}}[e]$, and 2) the latest model that contributed the edge-cost in $W_{\text{current}}[e]$.model. OVERWRITE also sets the boolean variable is_new_W to "true" when the edge-cost being overwritten is different from its previous value. Doing this allows the search to run A* only when the graph has changed from the previous iteration. {15-36} does very similar work like the main-loop of LazySP, i.e., 1) it considers one path candidate after the other (π) sorted in increasing order of their potential path cost (c_π) {16}, 2) selects edges E from that path based on the LAZYEDGESELECTOR (Mandalika et al. 2019) for evaluation, and 3) repeats this process until a path is found where no new edges require evaluation. Let us denote such a path as "evaluated path".

Here is how MA3 is different. Firstly, it runs two threads, one that performs the graph-search {1-48}, and the other {49-60} that uses $\mathbb{M}_{\text{sim}}$ to evaluate sets of edges that are scheduled in a priority queue $\mathcal{P}_{\text{sim}}$. Secondly, for every edge that has been selected by the LAZYEDGESELECTOR, instead of using only one edge-evaluator, MA3 chooses between $\mathbb{M}_{\text{ml}}$ and $\mathbb{M}_{\text{sim}}$. The more expensive $\mathbb{M}_{\text{sim}}$ is used if the confidence of the approximate $\mathbb{M}_{\text{ml}}$ is below a user-defined threshold ϵ_{conf} {21-36}. SCHEDULEFORSIM is a non-blocking function; thus, MA3 would not wait for the evaluation of an edge in $\mathbb{M}_{\text{sim}}$ to be available. Instead, it will temporarily disable that edge by setting its cost to ∞ {35} and continue with the search to find another path candidate. The idea is that, while the simulator is busy evaluating edges from potentially optimal paths, it might be worth exploring the graph and finding a potentially sub-optimal "evaluated path" quickly. Additionally, MA3 does extra work to maintain bounded-suboptimality (marked with bold line numbers in 1) and completeness (marked with underlined line numbers in 1) properties.

Once an "evaluated path" is found, LazySP will return it as a solution. However, in MA3, this path may consist of edges that were evaluated by only the $\mathbb{M}_{\text{ml}}$ and thus, may be invalid. Therefore, MA3 will verify those edges using $\mathbb{M}_{\text{sim}}$ before returning it as a solution. Again, instead of waiting for verification, these edges will only be scheduled. However, the edges being scheduled for path verification are set to a higher priority (see SCHEDULEFORSIM) {61-67}.

GETSIMDATA processes the evaluations completed by the simulator. Once an "evaluated-path" is verified to be valid in simulation, the upper bound on path-cost ub is updated {78-80} and the path is added to $\mathcal{P}_{\text{valid}}$. It is still not ready to be returned as a solution yet. Within the search, when CHECKGAP {89-97} is called, MA3 can now compare the upper bound ub with the lower-bound lb on path cost to see if the sub-optimality condition is satisfied. This ensures the bounded-suboptimality property of our algorithm, and having found such a path, the anytime solution stored in $\mathcal{P}_{\text{valid}}$ can finally be returned {45-48}.

Now we explain how we maintain the lower-bound lb on path-cost. A* generates least-cost path candidates based on W_{current} which initially contains optimistic costs; thus, the path candidates have the potential to set the lower bound. We add these paths to LB {10 and 43}. A path in LB will lose the potential to set the lower bound if it contains at least one invalid edge. Thus, when discovered, such paths are removed from LB {76}. Alternatively, if a path candidate consists of potentially pessimistic edges (explained in 4.2), it can incorrectly raise the lower bound. Thus, in {30} we disallow these candidates to increase the lower bound.

4.2 Handle ML Model Inaccuracies

The error-prone model $\mathbb{M}_{\text{ml}}$ can make two types of errors: 1) false-positive errors, which leads to optimistic edge costs, and 2) false-negative errors, which leads to pessimistic edge costs. Optimistic edge costs do not violate completeness guarantees. We handle them lazily, i.e., we let those edges be considered during the search until they are a part of an "evaluated path". In this case, MA3 will reveal their actual edge cost when verifying the "evaluated path".

In contrast, the pessimistic edges may lead the search not to discover a valid path. An edge that is invalidated by $\mathbb{M}_{\text{ml}}$ with high confidence is potentially pessimistic. We keep track of these edges in O_e {31}. Now, it is a matter of selecting edges from this list and scheduling them for simulation to reveal their actual cost. We do this in CHECKGAP {92-95}. We choose to schedule these edges for simulation after the search has exhausted all possible path candidates to find

Algorithm 1: Pseudocode for MA3

procedure MA3($s_{\text{start}}, s_{\text{goal}}, \omega_{\text{sub}}, \epsilon_{\text{conf}}$)
  $\mathcal{P}_{\text{valid}} \leftarrow \emptyset$ ▷ List of Sim Validated Paths
  $\mathcal{P} \leftarrow \emptyset$ ▷ Path candidates ordered by path cost
  $\mathcal{P}_{\text{sim}} \leftarrow \emptyset, \mathcal{P}_{\text{data}} \leftarrow \emptyset$ ▷ Edge-sets waiting for Sim
  $E_{\text{conf}} \leftarrow \emptyset$ ▷ Edges with high confidence
  $O_{\text{e}} \leftarrow \emptyset$ ▷ Potentially pessimistic edges
  W_{current} ▷ Latest edge costs. Initialized with $W_{\text{optimistic}}$
  is_new_$W \leftarrow$ false
  $(c_\pi, \pi) \leftarrow \text{SP}(s_{\text{start}}, s_{\text{goal}}, W_{\text{current}})$
  $LB \leftarrow \{(\pi, c_\pi)\}$ ▷ Paths that determine lower-bound
  $ub \leftarrow \infty$ ▷ upper bound on path cost
  Insert π in $\mathcal{P}$ with priority c_π
  Start SIMULATIONWORKER Thread
  while ($|\mathcal{P}| > 0$ or $|\mathcal{P}_{\text{sim}}| > 0$) **do**
    if $|\mathcal{P}| > 0$ **then**
      $(c_\pi, \pi) \leftarrow \text{POP}(\mathcal{P})$
      $E \leftarrow$ LAZYEDGESELECTOR(π)
      if E is $\emptyset$ **then**
        SCHEDULEFORSIM(π, c_π)
      else
        for all $e \in E$ **do**
          $c_e \leftarrow W_{\text{current}}[e]$
          (is_valid, conf) $\leftarrow$ QUERYMLMODEL(e)
          if is_valid = false **then**
            $c_e \leftarrow \infty$
          if conf $> \epsilon_{\text{conf}}$ **then**
            $W_{\text{current}} \leftarrow$ OVERWRITE(e, c_e, 'ml')
            Insert e in E_{conf}
            if $c_e = \infty$ **then**
              Update the path-cost of π in LB with $\min_{(\hat{\pi}, \hat{c}_\pi) \in LB \setminus \{(\pi, c_\pi)\}} \hat{c}_\pi$
              Insert (e, c_π) in O_{e}
              Break **for** loop
          else
            SCHEDULEFORSIM($\{e\}, c_\pi$)
            $W_{\text{current}} \leftarrow$ OVERWRITE(e, ∞, 'temp')
            Break **for** loop
    else
      Wait until $|\mathcal{P}_{\text{data}}| > 0$
    GETSIMDATA()
    if is_new_W **then**
      $(c_\pi, \pi) \leftarrow \text{SP}(s_{\text{start}}, s_{\text{goal}}, W_{\text{current}})$
      Insert π in $\mathcal{P}$ with priority c_π
      Insert (π, c_π) into LB
      is_new_$W \leftarrow$ false
    if CHECKGAP() **then**
      Break **while** loop or improve solution
  Terminate SIMULATIONWORKER Thread
return Least cost path from $\mathcal{P}_{\text{valid}}$
procedure SIMULATIONWORKER
  while Thread not terminated **do**
    (priorities $(p_1, p_2), E$) $\leftarrow \text{POP}(\mathcal{P}_{\text{sim}})$
    continue if $p_1 = 1$ and $p_2 > ub$
    $C \leftarrow \emptyset$
    for all $e \in E$ **do**
      if $W_{\text{current}}[e]$.model = 'sim' **then**
        $c_e \leftarrow W_{\text{current}}[e]$
      else
        $c_e \leftarrow$ QUERYSIMMODEL(e)
      Insert c_e in C while preserving order
    Insert (p_1, p_2, E, C) in $\mathcal{P}_{\text{data}}$

Algorithm 1: Pseudocode for MA3 (continued)

function SCHEDULEFORSIM(E, c)
  if the set of edges E represent a path **then**
    $\pi \leftarrow E$ ▷ A path is a set of edges
    Insert π in $\mathcal{P}_{\text{sim}}$ with priority $(1, c)$
  else
    $\{e\} \leftarrow E$
    Insert $\{e\}$ in $\mathcal{P}_{\text{sim}}$ with priority $(2, c)$

function GETSIMDATA
  $\mathcal{P}_{\text{data}} \leftarrow$ Get the edge-sets that have been evaluated
  for all $(p_1, p_2, E, C) \in \mathcal{P}_{\text{data}}$ **do**
    path_verified $\leftarrow$ true
    for all $e \in E$ and the corresponding $c_e \in C$ **do**
      $W_{\text{current}} \leftarrow$ OVERWRITE(e, c_e, 'sim')
      Insert e in E_{conf}
      if $c_e = \infty$ **then**
        Remove from LB path π that generates e
        path_verified $\leftarrow$ false
    if $p_1 = 1$ and path_verified = true **then**
      Insert $(E, \sum_{c_e \in C} c_e)$ in $\mathcal{P}_{\text{valid}}$
      $ub \leftarrow \min(ub, \sum_{c_e \in C} c_e)$
  $\mathcal{P}_{\text{data}} \leftarrow \emptyset$

function LAZYEDGESELECTOR(π)
return At least one edge from π that is not in E_{conf} or $\emptyset$ if all edges of π are in E_{conf} .

function OVERWRITE(e, c_e, model)
  if $W_{\text{current}}[e] \neq c_e$ **then**
    $W_{\text{current}}[e] \leftarrow c_e$
    is_new_$W \leftarrow$ true
  $W_{\text{current}}[e]$.model $\leftarrow$ model

function CHECKGAP
  $lb \leftarrow \min_{(\pi, c_\pi) \in LB} c_\pi$
  if $(ub/lb) > \omega_{\text{sub}}$ **then**
    if no paths left to consider in $\mathcal{P}$ and $\mathcal{P}_{\text{sim}}$ **then**
      $(e, c_\pi) \leftarrow \arg\min_{O_e} c_e$ s.t. $W_{\text{current}}[e]$.model = 'ml'
      $W_{\text{current}}[e]$.model $\leftarrow$ 'ppe'
      SCHEDULEFORSIM($\{e\}, c_{\text{est}}$)
    return false
  else return true

a valid path that satisfies the suboptimality gap. And we select them in increasing order of the path-cost corresponding to the path that contains the edge. Once scheduled, we mark the model of these edges to be 'ppe' such that MA3 does not add them again to O_e.

4.3 Theoretical Analysis

Claim: If there exists an optimal path π^* that is valid according to $\mathbb{M}_{\text{sim}}$, then given an admissible heuristic function, Algorithm 1 is guaranteed to return a path $\hat{\pi}$ such that its cost $c_{\hat{\pi}}$ is no more than ω_{sub} times the cost c_{π^*} of π^*, and such that $\hat{\pi}$ is valid according to $\mathbb{M}_{\text{sim}}$.

Lemma 4.1. *All paths returned by Algorithm 1 are valid according to the given $\mathbb{M}_{sim}$.*

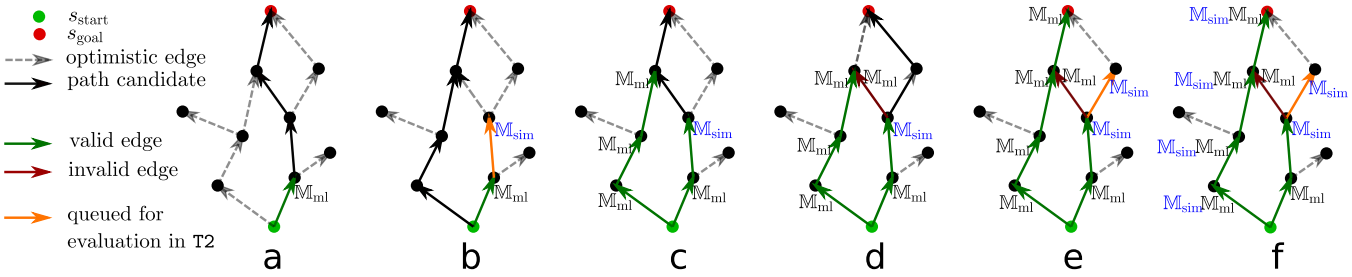

Figure 2: MA3 uses two threads say T1 and T2. The search and all $\mathbb{M}_{ml}$ evaluations takes place in T1, whereas all the $\mathbb{M}_{sim}$ evaluations take place in T2. The results generated in T2 are periodically synced with T1 in batches. Assuming that $\mathbb{M}_{ml}$ makes no mistakes, a typical run of MA3 would look like this. In (a) a shortest-path-candidate is found and its first edge is selected for evaluation. Since $\mathbb{M}_{ml}$ was confident about this edge, MA3 accepts its evaluation. In (b) the next edge is selected for evaluation, however, the $\mathbb{M}_{ml}$ is not confident and hence $\mathbb{M}_{sim}$ is used. Let us assume that querying $\mathbb{M}_{sim}$ takes 3x the time than that of $\mathbb{M}_{ml}$. Thus, while $\mathbb{M}_{sim}$ is evaluating that edge in T2 MA3 will attempt to continue the search in T1 through the next shortest-path-candidate (marked in black). In (c) we see that $\mathbb{M}_{ml}$ is confident and hence has evaluated three edges until $\mathbb{M}_{sim}$ evaluates its edge. Since $\mathbb{M}_{sim}$ evaluated this edge to be valid, the search can now resume along the previous shortest-path-candidate. In (d) $\mathbb{M}_{ml}$ invalidated an edge and similarly the next shortest-path-candidate is considered. Again in (e) as we wait for $\mathbb{M}_{sim}$ evaluation, MA3 considers the next shortest-path-candidate and employs $\mathbb{M}_{ml}$ to evaluate the last edge that connects with the goal. At this point, we have an "evaluated-path" and thus in (f) the simulator is made to verify this path "with high priority" i.e. over usual edge-evaluations. Since a valid path is found MA3 will return this solution if it meets the suboptimality criteria.

Proof. This theorem is enforced by construction as GETSIMDATA function only inserts paths π marked for verification (distinguished by priority $p_1 = 1$) when all the edges in the path have non-infinite costs. Note that some of these edge costs (for edges with model 'sim') are being taken from storage {55,56} to prevent duplicate evaluations. Because MA3 is a multi-threaded program, we ensure that the model is updated strictly after the correct edge costs are overwritten in W_{current} {84-88}. This also ensures that MA3 corrects false-positive errors made by $\mathbb{M}_{ml}$. □

Theorem 4.2 (Completeness)**.** *Algorithm 1 is complete if the heuristic function is admissible.*

Proof. Given an admissible heuristic function, A* and LazySP are proven to be complete. Hence, given an admissible heuristic function, MA3 will generate path candidates that are optimal given the current edge costs W_{current}. Only false-negative errors in W_{current} made by $\mathbb{M}_{ml}$ will prevent MA3 to find a feasible path. Suppose that there are no path-candidates left {92}, then by construction, in {93-95} MA3 ensure that all the potentially pessimistic edges get an opportunity to get scheduled for $\mathbb{M}_{sim}$. Note that the condition specified in {93} skips some potentially pessimistic edges from O_e. However, these edges are those which 1) have already been evaluated by $\mathbb{M}_{sim}$ or 2) have appeared as potentially pessimistic edge before and thus are already scheduled for $\mathbb{M}_{sim}$. Fig. 3 shows all possible ways in which the model of an edge evolves. An edge with source 'ml' will only upgrade to 'sim' either directly or via 'ppe' {94}. Thus, the condition $W_{\text{current}}[e]$.model = 'ml' in {93} will never miss to schedule a potentially pessimistic edge. If all potentially pessimistic edges are eventually evaluated in $\mathbb{M}_{sim}$, all the false-negative errors in W_{current} will eventually be corrected to find the optimal path. Hence MA3 is complete. □

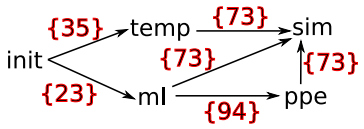

Figure 3: Flow Diagram For Model Sources

Theorem 4.3 (Bounded Suboptimality)**.** *The path $\hat{\pi}$ returned by Algorithm 1 is guaranteed to satisfy $c_{\hat{\pi}} \leq \omega_{sub} c_{\pi^*}$.*

Proof. The best valid path found by $\mathbb{M}_{sim}$ is used to update the upper bound ub {78-80} on the path-cost as once a path is found, we only want to find a better path. Thus, the upper bound informs about the best valid path found so far. Next, we discuss the lower bound on path cost. The first path π_1 found by A* when $W_{\text{current}} = W_{\text{optimistic}}$ contains only optimistic edge costs. The cost of a valid path can only be greater than or equal to c_{π_1}. Hence, if $lb \leftarrow c_{\pi_1}$, then any path π that satisfies $c_\pi / lb \leq \omega_{\text{sub}}$ is bounded-suboptimal. However, if we discover that an edge in π_1 is invalid, then the lower bound as described above will only be a loose bound. Thus, we raise the lower bound by considering the next optimal least-cost path. Therefore, it now remains to prove that the lower bound is not raised incorrectly. As described in Section 4.2, if $\mathbb{M}_{ml}$ only makes false-positive (optimistic) errors in W_{current}, the path costs returned by A* will also be optimistic. However, if a path candidate returned by A* contains an edge that is potentially pessimistic, the path cost may be higher than the actual path cost, in which case, this path cost should not be used to determine the lower bound. Hence, by ignoring such paths {30}, MA3 ensures that the lower bound is never raised incorrectly. □

5 Experiments

We are interested in three primary metrics, namely:

1. Planning times: MA3 is primarily designed to reduce calls to the time expensive edge-evaluator $\mathbb{M}_{\text{sim}}$ as in general the time spent in search and querying the approximate evaluator $\mathbb{M}_{\text{ml}}$ is negligible when compared to the time required to query the $\mathbb{M}_{\text{sim}}$. Therefore planning time is proportional to the number of calls.
2. Rate of success in finding a valid solution: False-positive errors made by $\mathbb{M}_{\text{ml}}$ can invalidate edges that are required for the solution. So, we are interested in knowing how many times does the planner fail to find a solution.
3. Path cost: This metric allows us to empirically compare the solution quality of the planner to the optimal solution.

5.1 Baselines

All our baselines are based on LazySP (Haghtalab et al. 2018) as it is proven to be edge-optimal with one edge-evaluator, which means that there is no algorithm that can evaluate a lesser number of edges compared to LazySP for the same planning problem with the same information. So, we only generate baselines that gradually employ smarter techniques to employ two edge-evaluators with the properties as discussed above. We run MA3 with parameters $\omega_{\text{sub}} = 2$, $\epsilon_{\text{conf}} = 0.6$.

1. LSP w SIM Evaluation: This is the primary baseline which is simply LazySP with the accurate edge evaluator $\mathbb{M}_{\text{sim}}$. Since $\mathbb{M}_{\text{sim}}$ is accurate, this baseline is also guaranteed to be complete. Also, note that LazySP is not an anytime, so this will always return the optimal solution.
2. LSP w ML Evaluation: One might choose to not use $\mathbb{M}_{\text{sim}}$ with LazySP but rather use an approximate edge-evaluator $\mathbb{M}_{\text{ml}}$. While the edge-evaluation time for these baselines will be very low, it does not guarantee completeness. We verify the path returned by this baseline in $\mathbb{M}_{\text{sim}}$ to compute the success rate.
3. LSP w ML + SIM Verification: This baseline is very similar to Baseline 2 where $\mathbb{M}_{\text{ml}}$ is only used during the search, but before a path is returned, it is verified in $\mathbb{M}_{\text{sim}}$. If the path is found to be invalid, the search will resume until a valid path is verified. Note that it also does not guarantee completeness as this baseline will fail to correct false-negative errors. It will return the first valid path without any guarantees to return a bounded-suboptimal path. This is analogous to running MA3 with $\epsilon_{\text{conf}} = 0$, $\omega_{\text{sub}} = \infty$ and without inserting any edge to O_e in $\{31\}$.
4. LSP w ML + SIM Evaluation & Verification: This baseline builds on Baseline 3, in that it makes a choice between $\mathbb{M}_{\text{sim}}$ and $\mathbb{M}_{\text{ml}}$ depending on whether the model $\mathbb{M}_{\text{ml}}$ is confident about its prediction (similar to $\{26\}$ in MA3). This is analogous to running MA3 with $\epsilon_{\text{conf}} =$ same as MA3, $\omega_{\text{sub}} = \infty$ and without inserting to O_e.
5. Single Thread MA3 (LSP w ML + SIM Evaluation, Verification & Correction): Baseline 2,3, and 4 lack completeness guarantees and hence would not have a way to guarantee bounds on the solution quality. Thus, they return the first solution they find. Baseline 3 and 4 also use two threads like MA3. Moreover, MA3 also returns the first solution that meets the bounded suboptimality criteria and utilizes two parallel threads. In contrast, Baseline 1 is designed to run on a single thread and return the optimal solution. Therefore, we add this baseline to compare how would MA3 perform if it were to not use the benefits of two parallel threads and return the optimal solution. This is analogous to running MA3 by replacing line $\{38\}$ with "Wait until $|\mathcal{P}_{\text{sim}}| = 0$" and setting $\omega_{\text{sub}} = 1$.

5.2 Off-Road Navigation Domain

We use three maps with varying sizes, which directly dictates the size of the graph. Our simulation environment is developed on Unreal Engine 4, and the elevation map is a direct replica of the Budds Creek Motocross Track, Maryland, USA, obtained from publicly available laser-scans Fig. 4. The three maps are taken from various parts of this motocross park. While the three maps simply vary in surface area size, the "medium" map is from an area where many valid paths may exist from any start and goal pair. The vertex count for the graph generated from each map in increasing order are $76,716$, $86,076$ and $411,048$. The maximum branching factor for each vertex was $|M| = 11$. We simulate a 1.5x scaled-up model of the Clearpath Husky robot using the Nvidia PhysX engine. The low-level controller for the vehicle that accepts a target $m \in M$ (as shown in Fig. 1) is based on Carla's (Dosovitskiy et al. 2017) implementation of lateral and longitudinal PID controllers.

Machine Learning Model: We use an ensemble of CNN binary-classifiers to provide us with an estimate for the prediction confidence. An example input to the model corresponding to an edge in the graph is shown in Fig. 5. Note that this model is not trained on any parts of the Budds Creek test environment. It is trained offline by collecting data obtained after rolling out simulations on a series of maps with varying elevations. These elevation maps are not from the real world but generated based on the Perlin noise model (Perlin 1985) with a manually selected range of parameters that generate reasonable variation in elevation roughness and amplitude. We believe that this is a valid process to prepare the model for a real-world map, and pre-training with this data does not require any real-world elevation map. Once the model is trained, it is not changed in the course of the experiments described in the following sections. We employ a trick to make the predictions more robust at no additional ML inference time. Instead of querying the model for a target $m \in M$ and for a vehicle location given by the vertex, we additionally query the ML model parallelly in the GPU for a set of neighboring vehicle locations for the same target m and then take a vote inversely weighted by the deviation. Finally, with TensorRT optimization (Vanholder 2016), the inference time on an Nvidia Titan V GPU is around $3ms$.

Table 1, Table 2 and Table 3 present the performance comparison for MA3 and the baselines for various metrics over 100 planning episodes each for the three maps. It is important to note that the metrics in Table 1 and Table 3 are computed only if all the planner found a path which

Maps		LSP w SIM	LSP w ML + SIM V	LSP w ML + SIM E&V	Single Thread MA3 (Optimal)	MA3
Small	Avg : 95% CI	1	0.99 : [0.87, 1.11]	1.07 : [0.85, 1.29]	0.74 : [0.59, 0.90]	1.60 : [0.70, 2.51]
	Min, Max	1, 1	0.62, 1.61	0.62, 2.24	0.13, 1.31	0.62, 8.24
Medium	Avg : 95% CI	1	2.93 : [1.40, 4.46]	2.57 : [1.31, 3.84]	1.38 : [1.10, 1.66]	2.54 : [1.28, 3.80]
	Min, Max	1, 1	0.96, 11.04	1.00, 9.34	0.93, 2.45	0.97, 8.53
Large	Avg : 95% CI	1	4.54 : [3.15, 5.93]	3.69 : [2.66, 4.73]	1.04 : [0.95, 1.13]	3.48 : [2.56, 4.40]
	Min, Max	1, 1	0.66, 18.60	0.75, 15.03	0.49, 1.77	0.79, 12.93

Table 1: Speed-up in planning times w.r.t. LSP w SIM (Factor x)

Maps	LSP w SIM	LSP w ML	LSP w ML + SIM V	LSP w ML + SIM E&V	Single Thread MA3 (Optimal)	MA3
Small	0	26.31	10.52	5.26	0	0
Medium	0	31.25	0	0	0	0
Large	0	57.64	2.35	1.17	0	0

Table 2: Failure Rate of the Planners (%)

Maps		LSP w SIM	LSP w ML	LSP w ML + SIM V	LSP w ML + SIM E&V	MA3
Small	Avg : 95% CI	1	1.01 : [1.00, 1.03]	1.01 : [1.00, 1.03]	1.00 : [1.00, 1.02]	1.00 : [1.00, 1.02]
	Min, Max	1, 1	1.00, 1.07	1.00, 1.07	1.00, 1.07	1.00, 1.07
Medium	Avg : 95% CI	1	1.01 : [1.00, 1.03]	1.00 : [1.00, 1.00]	1.00 : [1.00, 1.03]	1.01 : [1.00, 1.03]
	Min, Max	1, 1	1.00, 1.08	1.00, 1.00	1.00, 1.08	1.00, 1.08
Large	Avg : 95% CI	1	1.00 : [1.00, 1.00]	1.00 : [1.00, 1.00]	1.00 : [1.00, 1.00]	1.00 : [1.00, 1.01]
	Min, Max	1, 1	1.00, 1.01	1.00, 1.01	1.00, 1.00	1.00, 1.03

Table 3: Suboptimality in path cost w.r.t LSP w SIM

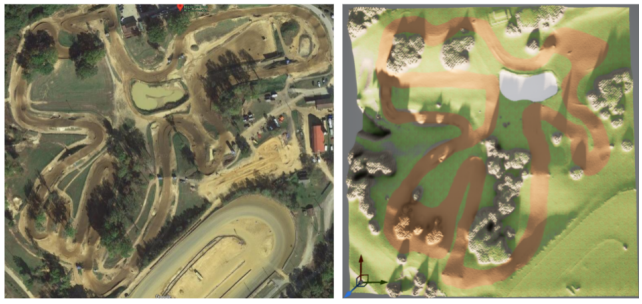

Figure 4: Left: Google Earth View of Budds Creek Motorcross Park. Right: Replicated terrain elevation map in UE4.

is valid in $\mathbb{M}_{\text{sim}}$. For the planning episodes, a random goal state was selected once, and then 100 random start locations were selected. Note that the planners are given the same start-goal pair per episode. We then ran a backwards Dijkstra's search (Dijkstra et al. 1959) to solve the single-source shortest-path problem with edge-costs from $W_{\text{optimistic}}$. This allowed us to compute the cost-to-go values once, which can then be used as a consistent and admissible heuristic in all the 100 planning episodes. MA3 and all the baselines are implemented in C++. The average query time for $\mathbb{M}_{\text{sim}}$ is around $450ms$ while that of $\mathbb{M}_{\text{ml}}$ is around $3ms$. We could not manage to make the simulation faster without decreasing the fidelity. We chose Nvidia PhysX Engine as it utilizes GPU parallelization, and we disabled UE4 rendering to speed up physics simulation.

5.3 Speed-up in Planning Times

For each planning episode, we compute the pairwise speed-up achieved in planning time for a planner as compared to that of LazySP w SIM. Table 1 shows the various statistics over 100 planning episodes. We present 1) average with 95% confidence interval (within []), and 2) the minimum and maximum values observed. We did not present the speed-up for **LSP w ML** due to space-constraints as it is trivially very fast since it does not query $\mathbb{M}_{\text{sim}}$ and is around 10x. We observe that the speed-up achieved correlates with the size of the graph as expected. **MA3** achieved an average speed-up of 3.48x in the "Large" map. Although **LSP w ML + SIM V** and **LSP w ML + SIM E&V** baselines achieves a higher speed-up, as we will observe in Section 5.4, these baselines did not solve all the planning problems. In a particular case, **MA3** was around two times slower (speed-up of 0.62) which is because $\mathbb{M}_{\text{ml}}$ made significant errors. False-positive errors waste simulation efforts for path verification and false-negative errors lead the search away from valid

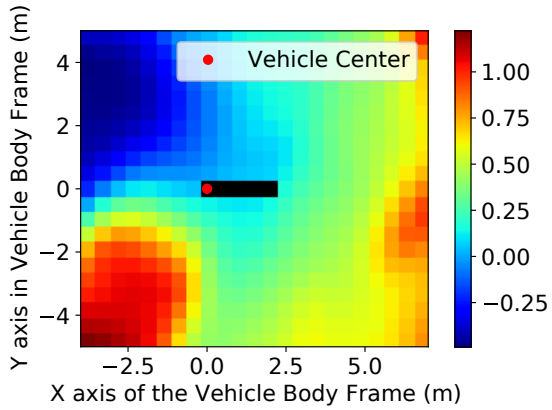

Figure 5: An example 2-channel input to $\mathbb{M}_{\text{ml}}$. The first channel encodes the height around the vehicle (in meter). The second channel is a binary image with 'black' pixels encoding the ideal path between the center of the vehicle to a target m. In this case, the target is $2.5m$ along the x-axis (vehicle forward). In this visualization, only the black pixels from the second channel are overlaid on top of the first channel.

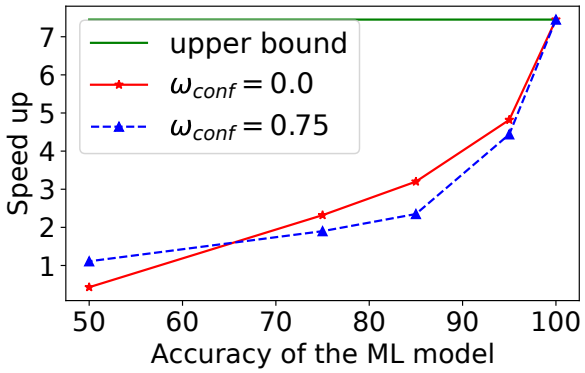

Figure 6: Speed-up with varying model accuracy and confidence thresholds

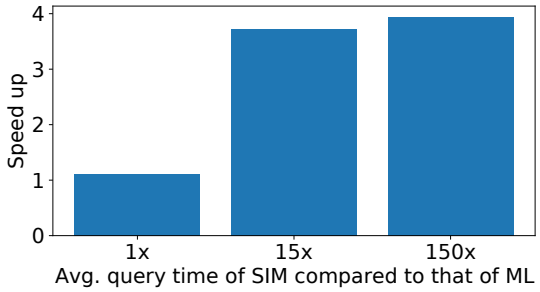

Figure 7: Speed-up with simulation query time

solutions requiring the evaluation of potentially-pessimistic edges. Section 5.5 presents how the errors in $\mathbb{M}_{\text{ml}}$ affect performance of **MA3**.

5.4 Success Rate and Solution Cost

Table 2 presents the success rate for each planner. MA3 achieves a 100% success rate thanks to its completeness properties. The "medium" map provides a lot of alternate routes, which is why Baseline 3 and 4 did not fail in the random planning episodes we tested them on.

Table 1 presents the speed-up for the planners which might return sub-optimal solution. Recall that Baselines 2,3 and 4 return the first solution, Baseline 5 returns optimal solution and MA3 returns bounded-suboptimal solution $\omega_{\text{sub}} = 2$. Therefore, compared to **LazySP w SIM**, and except for Baseline 5, the planners trade optimality for planning times. Thus, we present the ratio of the path cost found by the planner as compared to that of the optimal path in Table 3. We observe that a significant speed-up can be achieved for certain planning problems without sacrificing too much on the solution quality. We use the bootstrap method (DiCiccio and Efron 1996) to compute the confidence intervals as suboptimality cannot be lower than 1.

5.5 Ablation Studies

The ML model is error-prone, while the simulator is relatively slower to query. Therefore, in these experiments, we wanted to measure how the performance of the planners is affected by the characteristics of the edge-evaluators.

Performance vs ML model accuracy In order to control the model accuracy, we employed the following trick. We used the same ML model trained as described in Section 5.2 which provides a binary prediction and its confidence score, but instead of using the prediction, we flip the ground-truth prediction with a probability to emulate a learned model with a certain rate of accuracy in the test environment. Fig. 6 shows the speed-up achieved by **MA3** in the "Medium" map with various ω_{conf}. Setting $\omega_{\text{conf}} = 0$ makes MA3 rely more on $\mathbb{M}_{\text{ml}}$ for initial edge evaluation, thus we observe that MA3 has higher speed-ups when the $\mathbb{M}_{\text{ml}}$ is more accurate. However, as the accuracy reduces, the speed-up also reduces. Interestingly the speed-up with an ML model with 50% accuracy is lower when $\omega_{\text{conf}} = 0$ as compared to when $\omega_{\text{conf}} = 0.75$. From a user's perspective, ω_{conf} should be set based on the accuracy of $\mathbb{M}_{\text{ml}}$.

Performance vs Simulation Query Time As expected we observe in Fig. 7 that MA3 shows significant speed-up when $\mathbb{M}_{\text{sim}}$ is relatively slower than $\mathbb{M}_{\text{ml}}$. For this experiment, we cached the results from the simulator so that they can be made available with any custom delay to emulate any simulation query time. The intuition behind this is that if SIMULATIONWORKER has high throughput, it may perform additional simulations before the main thread reaches GETSIMDATA and realizes that all the evaluations required to return a valid path are available.

6 Conclusion and Future Work

In this work, we looked at the off-road navigation-planning problem. We attempt to determine traversability using a simulator, assuming that it is accurate, and use a learned model trained with simulation data to approximate traversability quickly during online planning. To that end, we provide a novel planning algorithm, MA3, which can exploit the complementary properties of these models to find bounded-suboptimal paths in significantly less time. In the future, we want to evaluate our system on a real robot.

Acknowledgments

This work was supported by the ARL-sponsored A2I2 program, contract W911NF-18-2-0218.

References

Al-Milli, S.; Althoefer, K.; and Seneviratne, L. D. 2007. Dynamic analysis and traversability prediction of tracked vehicles on soft terrain. In *2007 IEEE International Conference on Networking, Sensing and Control*, 279–284. IEEE.

Al-Milli, S.; Seneviratne, L. D.; and Althoefer, K. 2010. Track–terrain modelling and traversability prediction for tracked vehicles on soft terrain. *Journal of Terramechanics*, 47(3): 151–160.

Chavez-Garcia, R. O.; Guzzi, J.; Gambardella, L. M.; and Giusti, A. 2017. Image classification for ground traversability estimation in robotics. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, 325–336. Springer.

Chavez-Garcia, R. O.; Guzzi, J.; Gambardella, L. M.; and Giusti, A. 2018. Learning ground traversability from simulations. *IEEE Robotics and Automation letters*, 3(3): 1695–1702.

Cohen, B.; Phillips, M.; and Likhachev, M. 2015. Planning single-arm manipulations with n-arm robots. In *International Symposium on Combinatorial Search*, volume 6.

DiCiccio, T. J.; and Efron, B. 1996. Bootstrap confidence intervals. *Statistical science*, 11(3): 189–228.

Dijkstra, E. W.; et al. 1959. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1): 269–271.

Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; and Koltun, V. 2017. CARLA: An open urban driving simulator. In *Conference on robot learning*, 1–16. PMLR.

Gennery, D. B. 1999. Traversability analysis and path planning for a planetary rover. *Autonomous Robots*, 6(2): 131–146.

Haghtalab, N.; Mackenzie, S.; Procaccia, A.; Salzman, O.; and Srinivasa, S. 2018. The provable virtue of laziness in motion planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, 106–113.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Hirose, N.; Sadeghian, A.; Vázquez, M.; Goebel, P.; and Savarese, S. 2018. Gonet: A semi-supervised deep learning approach for traversability estimation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3044–3051. IEEE.

Hou, B.; Choudhury, S.; Lee, G.; Mandalika, A.; and Srinivasa, S. S. 2020. Posterior sampling for anytime motion planning on graphs with expensive-to-evaluate edges. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 4266–4272. IEEE.

Karpas, E.; Betzalel, O.; Shimony, S. E.; Tolpin, D.; and Felner, A. 2018. Rational deployment of multiple heuristics in optimal state-space search. *Artificial Intelligence*, 256: 181–210.

Koenig, S.; Likhachev, M.; and Furcy, D. 2004. Lifelong planning Astar. *Artificial Intelligence*, 155(1-2): 93–146.

Mandalika, A.; Choudhury, S.; Salzman, O.; and Srinivasa, S. 2019. Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 745–753.

Nampoothiri, M.; Vinayakumar, B.; Sunny, Y.; and Antony, R. 2021. Recent developments in terrain identification, classification, parameter estimation for the navigation of autonomous robots. *SN Applied Sciences*, 3(4): 1–14.

Perlin, K. 1985. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3): 287–296.

Vanholder, H. 2016. Efficient inference with tensorrt. In *GPU Technology Conference*, volume 1, 2.