TASI: Terrain-Aware System Identification for Autonomous Navigation of Wheeled Robots

Ryan Adolf*, Tarun Amarnath*, Jeremy Hughes* and Kaushik Shivakumar*
Department of Electrical Engineering and Computer Sciences, University of California at Berkeley
*equal contribution

I. Abstract

Terrain-Aware System Identification (TASI) enables robots to autonomously find an optimal path in an environment using only odometry and visual inputs. TASI creates a feature map of the environment by stitching camera images transformed based on a Turtlebot's odometry. It estimates dynamics from movement and visual features using kernel regression, accounting for uncertainties, and uses them for path planning. Results show consistent navigation in simulation and that leveraging visual commonality in terrains may lead to faster convergence on an optimal path (sometimes by 50% or more). Real-world tests show promising results, though non-idealities in physical systems complicate analysis.

II. Introduction

Path planning is a well-known problem in the field of robotics. Autonomous vehicles constantly face the issue of navigating through novel scenarios, and while roads remain fairly consistent, obstacles that come up change constantly. Other planning problems arise in novel environments. For example, in search and rescue scenarios, different regions might end up more navigable than others, and a robot would have to autonomously determine the best course of action to avoid obstacles, avoid (or conquer) rough terrain, and reach its destination.

Navigating the real world often takes a lot of guesswork. In mission critical scenarios, minimizing the uncertainty of a chosen path is often crucial to safety. A poor navigational decision can cripple a robot, so it must avoid areas that might be seen as more dangerous. While no system is born with knowledge about safety and drivability, making such an identification and learning process autonomous and uncertainty-aware can improve performance during eventual deployment.

The goal of TASI is to perform this kind of terrain-aware learning and navigation. In TASI, the robot drives over various regions and creates a map of the environment. Simultaneously, it updates a feature vector with information about the regions it has taken an image of to inform the control algorithm about the terrains present in different areas of the map, and it also builds a model mapping visual features to driveability characteristics. This model not only contains information about driveability, but also about aleatoric (environmental) and epistemic (modeling) uncertainties of each terrain. Finally, to close the loop, this information is used to inform the path planner, which then chooses the most navigable route.

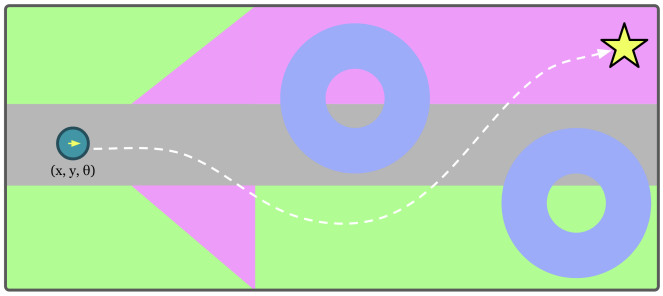

Fig. 1: A graphic of the problem we tackle in this work: navigation through visually distinct terrains with initially unknown driveability characteristics.

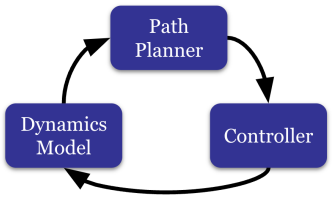

Fig. 2: The overall feedback loop TASI uses.

III. Related Work

A. Terrain-Based Control

Recent work in adaptive control has proven the convergence of concurrently learned parameter estimates up to some error term. The method presented in Kamalapurkar et. al. maintains robustness to Gaussian error during online estimation [6]. A more specialized focus on terrains by Coyle et al. works on updating control modes based on terrain classification. Their approach performs identification on the surface the robot drives over and switches to a pre-determined control mode as necessary. They develop an update rule to decide when to change control modes and test empirically. [2]. Neither makes use of visual data as an input into the learning and classification algorithms. We tune the dynamics of our system as we traverse different terrains and autonomously learn a mapping from visual features to driveability characteristics, which is then used for path planning.

B. Visual Navigation

A large body of work deals with understanding safe and effective navigation for an autonomous robot dealing with unfamiliar elements. The original theory behind control barrier functions ensures safety, particularly in unknown and unstructured environments [1]. Lyapunov functions are adapted to incorporate these restrictions, accounting for the stronger conditions involved in implementation of the ideas in realistic situations. Testing of the theory is more rigorously accomplished by Xiao et al. through implementation using autonomous vehicles. Their work constructs an end-to-end pipeline for learning visual markers and avoiding obstacles [9]. Procuring and processing the images themselves is developed further through BADGR system, which outperforms LIDAR on more undrivable terrains, such as tall grass [5]. Rather than exclusively using vision for obstacle avoidance, we propose to extract visual features to inform our path planner and augment existing data about differing terrains. Our work takes information from an image to identify similar regions and predict aleatoric uncertainties.

IV. Methods

A. Controller Design

We use a unicycle model robot in simulation and real to track trajectories in the 2D coordinate plane.

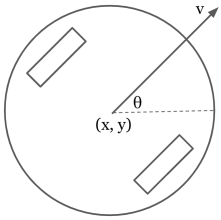

Fig. 3: Unicycle model robot.

The dynamics of this robot can be framed as the following, where a and w are inputs, and d and k are terrain-dependent velocity scaling factors.

$$\begin{aligned} \dot{x} &= vd\cos\theta \\ \dot{y} &= vd\sin\theta \\ \dot{v} &= a \\ \dot{\theta} &= kw \end{aligned} \tag{1}$$

By using feedback linearization [7], we convert the system to a second-order one where we have direct control over $\ddot{x}$ and $\ddot{y}$. We then provide true inputs a and w to the robot based on the following equation.

$$\begin{bmatrix} a \\ w \end{bmatrix} = \begin{bmatrix} d\cos\theta & -kv\sin\theta \\ d\sin\theta & kv\cos\theta \end{bmatrix}^{-1} \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} \tag{2}$$

We use a PD controller to steer this system along a trajectory with open loop trajectory targets for $\ddot{x_d}$ and $\ddot{y_d}$.

$$\ddot{x} = \ddot{x_d} + K_p(x_d - x) + K_d(\dot{x_d} - \dot{x})$$
$$\ddot{y} = \ddot{y_d} + K_p(y_d - y) + K_d(\dot{y_d} - \dot{y})$$

B. Path Planning

In order to perform path planning, we divide the world as we know it into a grid, where each vertex is a corner of a grid cell. Edges equidistant from two different terrains take the higher of the two weights. Edge weights correspond to the inverses of speed scaling factor d and are also proportional to the distance (diagonals are $\sqrt{2}$ units) longer than horizontal and vertical segments. We then run Dijkstra's shortest paths algorithm on this graph [3]. Areas that are less driveable tend to be planned around due to their higher edge weights. Figure 4 provides an illustration of how we may perform path planning in the presence of different terrains with different visual features.

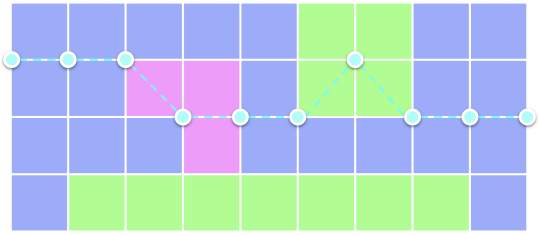

Fig. 4: Path planning across terrains with Dijkstra's algorithm.

C. Dynamics Estimation

1) Estimator: We use kernel regression, specifically the Nadaraya-Watson estimator [8], for learning the dynamics of terrains given visual features associated with them. Kernel regression is a non-parametric technique to fit nonlinear functions to data, and we adapt it to also provide us with uncertainty information. Intuitively, this estimator uses a weighted combination of nearby datapoints (according to some function K) to inform the prediction at a test terrain vector $\mathbf{x}$. We use a separate estimator for predicting d and k values.

$$\hat{E}[y|X = \mathbf{x}] = \frac{\sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x_i}) y_i}{\sum_{j=1}^{N} K(\mathbf{x}, \mathbf{x_j})} \tag{3}$$

2) Uncertainty: We focus on quantifying and estimating two types of uncertainty in our dynamics models, as has been done in much prior work [4]. Aleatoric uncertainty U_a refers to noise and randomness present in the environment for a given terrain input $\mathbf{x}$. With more data near $\mathbf{x}$, U_a converges to a fixed value. For example, gravel would have high U_a, whereas pavement would have low U_a with its smoother, more consistent, surface. Epistemic uncertainty (U_e) is the model's inherent uncertainty in its own predictions of dynamics for $\mathbf{x}$, and it decreases with more data closer to $\mathbf{x}$. Unseen terrains would have high U_e, and familiar ones would have low U_e.

In order to measure aleatoric uncertainty, we use a cascaded approach as in figure 5. Specifically, while our first kernel regressor predicts the dynamics values directly as $\hat{y}_i$, the second one uses the results to formulate its own ground truth data, $U_{ai} = (y_i - \hat{y}_i)^2$, and fit a model to it such that $\hat{U_{ai}}$ estimates can be formed for unseen but similar terrain vectors.

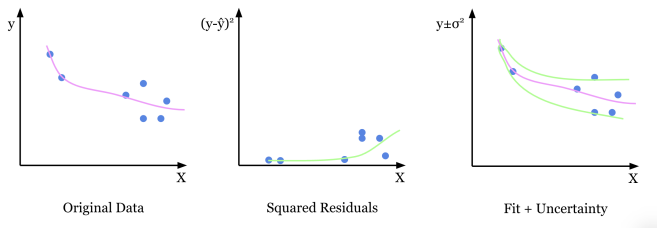

Fig. 5: A cascaded approach to fitting data and measuring aleatoric uncertainty.

To measure epistemic uncertainty, we use a notion of "pseudo-counts" in the denominator of an "uncertainty kernel" K_u to measure the support for a particular test point in the training dataset.

$$U_e = \text{Var}[\hat{E}[y|X = x]] \approx \frac{1}{\sum_{j=1}^{N} K_u(\mathbf{x}, \mathbf{x_j})}$$

3) Priors: We find that kernel regression as written has one weakness: for highly novel terrains different from those in the training dataset, we make predictions based on the closest terrains, even if they are far apart. Instead, we wish to predict the driveability characteristics terrains as perfect ($d = k = 1$) for novel terrains. While this may seem like an arbitrary choice, we prove that such an "optimistic" exploration strategy has certain favorable properties.

We define a prior with a value v_p and w_p. When predicting a dynamics parameter for a visual feature $\mathbf{x}$, we add as a synthetic datapoint $(\mathbf{x}, v_p)$ with weight w_p. Intuitively, if there are not many other datapoints near this point, the prior will dominate; otherwise, it will not and the prediction will largely be based on the nearby datapoints.

D. Optimistic Exploration Theory

Theorem IV.1. *Assume that the estimated edge length for an unseen edge is always lower than the true edge length, and that the true edge length is estimated exactly once an edge is traversed once. This leads to the optimal path between two points eventually being discovered.*

Proof. Suppose the algorithm has converged on a path φ, with true cost $C(\varphi) = \hat{C}(\varphi)$, and that there exists a path φ' such that $C(\varphi') < C(\varphi)$. Because of the assumptions made about estimations, $\hat{C}(\varphi') \leq C(\varphi')$, and thus $\hat{C}(\varphi') < \hat{C}(\varphi)$. But by definition of the exploration algorithm, we would choose to traverse path φ', meaning we have not in fact converged on φ. Thus, no such path φ' may exist. We must also show that we do indeed converge, which must happen as estimated edge lengths eventually converge (they cannot continually increase), which means the shortest path we compute must also converge. □

However, we must consider the case of estimation error. If estimation error can be arbitrarily large after a single traversal of an edge, then we may converge on arbitrarily suboptimal shortest paths. If we bound the maximum error after one or more traversals to e, then we can claim that if we converge on path φ, $C(\varphi) \leq (1 + e)C(\varphi*)$. More interestingly, even without bounds on exploration error, we may be able to mitigate this issue with an Epsilon-Greedy exploration strategy.

Theorem IV.2. *Using an Epsilon-Greedy strategy (sampling a random path with probability $\epsilon > 0$), under the assumption that estimation error converges almost surely to 0 in the number of edge traversals, the path φ we converge on is guaranteed to be optimal.*

Proof. Every edge is visited with nonzero probability over time, so as $t \to \infty$, $\forall e \in E, \hat{C}(e) - C(e) \to 0$ almost surely. Due to this convergence property, our estimate of all edge lengths approach the true ones almost surely, meaning that $\hat{C}(\varphi*) \to C(\varphi*)$, so the length of the path we converge on will be optimal. □

We also note that while it may be tempting to believe that using visual features only speeds up the rate of convergence towards the optimal path, we note that such a claim is false using the greedy strategy we employ here. While it may be possible to devise a more complex exploration strategy that guarantees faster convergence in expectation when using terrain information, under the assumptions of certain priors on dynamics, we defer this to future work.

E. Visual Terrain Mapping

We initially apply the depth clouds procured through the TurtleBot's Kinect to identify obstacles and segment the ground. Reading this information with prior knowledge of the walls of the room allows us to recreate obstacles in the robot's path and plan around them. However, using the depth maps to segment the ground is extremely noisy. Additionally, with compuational limitations, read time is significant, preventing real-time analysis. We then pivot to using homography transforms and occupancy grids.

The images taken by the TurtleBot come from the perspective of its camera, mounted low to the ground. This skew makes constructing an accurate map difficult, so a homography transform is first performed with a matrix calibrated on test points to warp the image to an overhead perspective, as in Figure 7. The map used is broken down into grid cells for path planning and featurization. This straightened picture is positioned in this world grid in the appropriate location. The TurtleBot publishes its position through ROS, allowing us to locate the appropriate cells to modify, like with the corners in Figure 6.

For each region in the image captured, some data for terrain identification is calculated and recorded in a corresponding feature grid to help inform estimation about the dynamics of different environments the robot has not traversed. This is constantly updated. The information includes the average R, G, and B values in each voxel, the hue, saturation, and value averages, and the RMS contrast, calculated using the standard deviation of the intensities. To expedite the process of feature updates especially with higher voxel resolution, this process is vectorized.

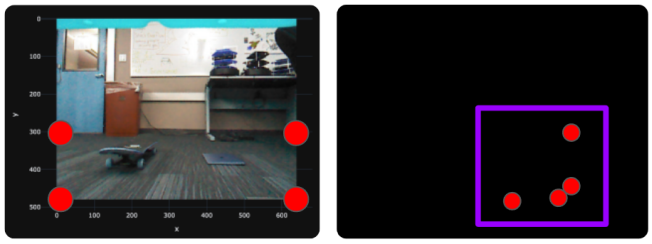

Fig. 6: Points in the coordinate space of the robot's base are calculated to account for the horizon line. The scene in the image is transformed to locations on the global map based on the robot odometry. A bounding box rounded to the voxel grid is calculated for feature updates.

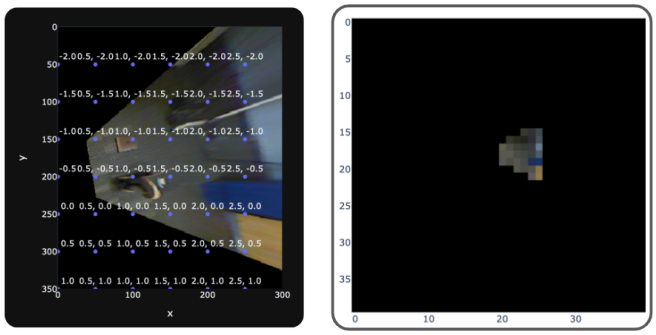

Fig. 7: Using homography, the points are projected to the perspective of an overhead camera. The terrain map is updated with both the image and the featurization.

F. Overall Approach

We describe the overall TASI exploration algorithm in algorithm 1 and figure 2.

Algorithm 1 TASI overview

dyn ← initial dynamics model
while not converged **do**
  $\phi \leftarrow$ odometry data and history
  $I \leftarrow$ camera data and history
  map = updateMap(map, ϕ, I)
  dyn_map = dyn(map)
  path = plan(dyn_map)
  buf $\leftarrow (x_t, u_t, x_{t+1})$ from executing path
  dyn ← refit(dyn, buf)

V. Experimental Results

A. Simulation

There are 2 main methods of our design demonstrated in simulation: the planning algorithm and the resulting state estimation. These work in tandem. The first experimental setup consists of a 10×10 grid that includes mock dynamics realized with k and d scaling values randomly sampled between 0.2 and 1. These mock dynamics are a simulation of what the feature extraction from the visual input would give to the planner. Each grid cell has a unique one-hot encoded terrain vector. The planner is then commanded to navigate from (1,1) to (9,9) and back repeatedly until it converges on its conclusion for the optimal path based on the dynamics of the environment. Figure 8 shows the paths taken on reaching convergence for this setup. The planner leverages its explorative disposition, since all undriven cells in the grid are given dynamics values of 1, to learn more about its environment before settling on the true optimal path.

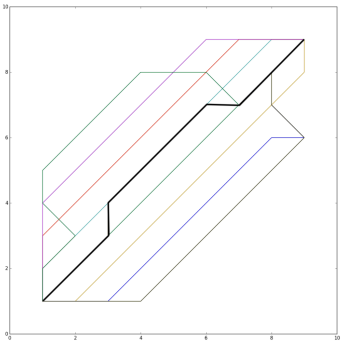

Fig. 8: Chosen path for 10×10 environment with no common features. Takes 8 iterations (4 there and back) of navigating to goal and returning to converge to optimal path (black).

It should be noted that after each iteration the estimated cost of the new plan chosen rises as shown in Figure 9 because for every cell it determines the dynamics are less than what it optimistically thought previously, and that this estimated cost converges on the true cost.

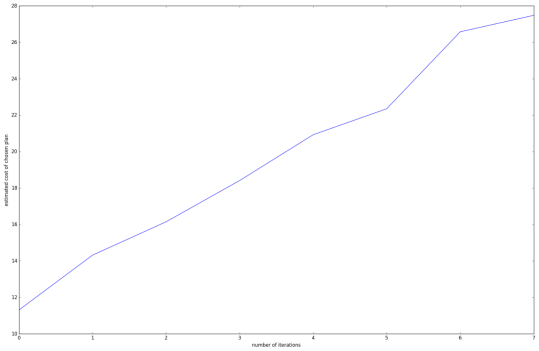

Fig. 9: Estimated cost of the chosen path for each iteration for the first experiment.

The following experiment demonstrates the importance of leveraging the common features in the robot's visual environment such as observing a large patch of grass. The second simulation experiment instead randomly samples the k and d values from a set number of values (10-20) between 0.2 and 1. The planner is commanded to navigate to the same

goal, and is now shown in Figure 11 to converge in 2-4 iterations. The impact can be seen from the measures of epistemic uncertainty from the first iteration in the previous run versus this, Figure 10, and the quick drop in average error of dynamics estimations shown in Figure 12. This allows the planner to get an estimate for the environment much more quickly and then choose the optimal path much sooner. This can be seen with an equivalent 20×20 grid that the first experiment failed to converge on, as every iteration presents a longer compute times with more data to fit. The second experiment saw the planner converge in just 3 iterations, Figure 13.

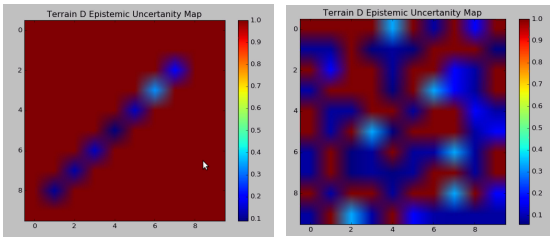

Fig. 10: Left: Epistemic uncertainty map of first simulation experiment after 1 iteration, 0.1 is most certain. Right: Epistemic uncertainty map of second simulation experiment after 1 iteration, 0.1 is most certain.

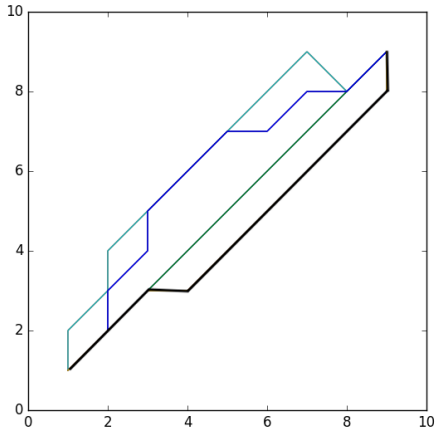

Fig. 11: Chosen path for 10×10 environment with common features. Takes 4 iterations (2 there and back) of navigating to goal and returning to converge to optimal path, shown in black.

B. Real-World

We run tests on the TurtleBot hardware. The first tests include testing the performance of the homography grid. We configure a small voxel size using average color as a feature to create an image representing composited homography images. Results show that there exist slight variations in lighting and perspective when combining homographies. However, the

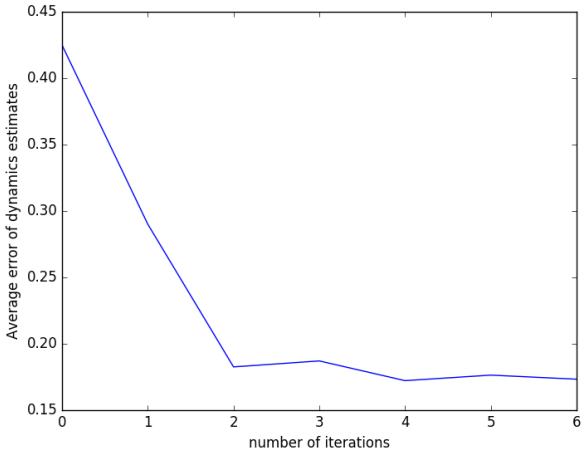

Fig. 12: Average error of dynamics estimates of all cells in the environment for each iteration, on the 20×20 environment with common features.

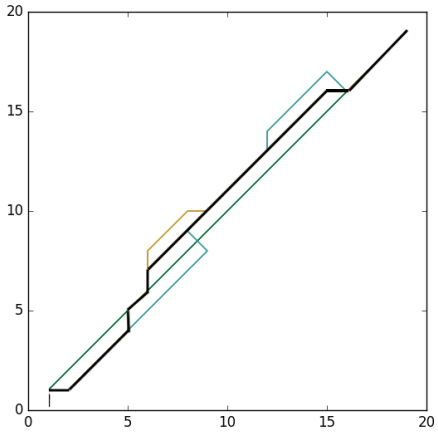

Fig. 13: Chosen path for 20×20 environment with common features. Takes 3 iterations (2 there and back) of navigating to goal and returning to converge to optimal path, shown in black.

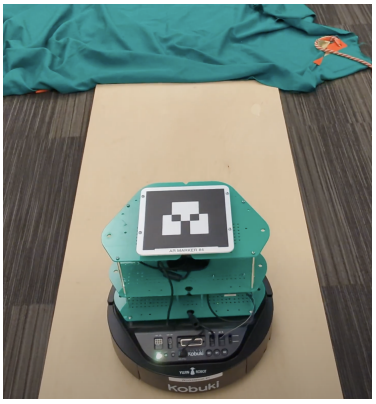

Fig. 14: The gravel terrain.

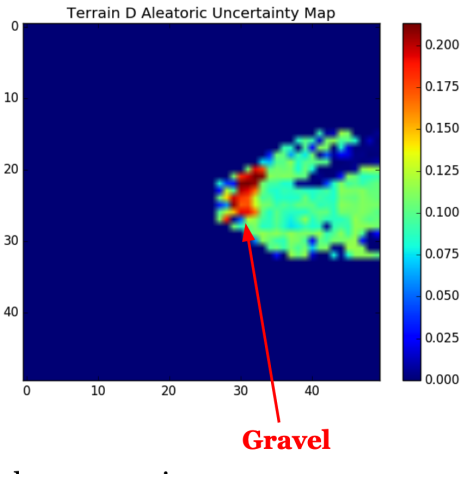

Fig. 15: The aleatoric uncertainty map after running on the gravel terrain.

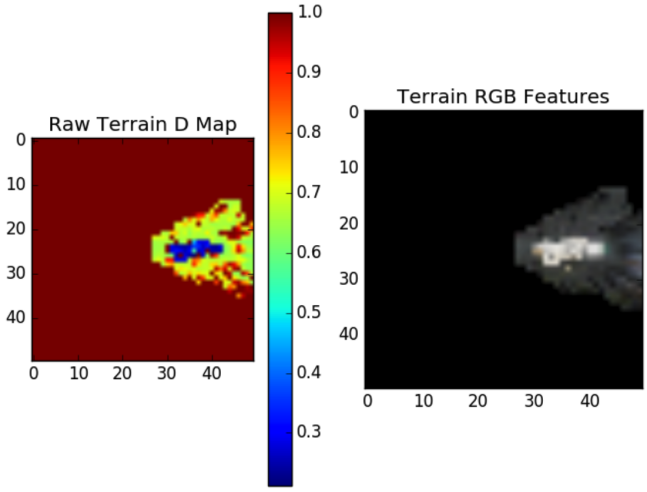

Fig. 16: The d value estimates after running on the plank.

results appear high-quality enough to map out various terrains on the ground.

In our second test, we run the TurtleBot over a wooden terrain. We choose this texture to be markedly different than the grey carpet in the lab. As the robot rolls over the plank, we hold back the robot to simulate a terrain with high resistance to motion (such as mud or snow). We verify that the robot estimates the d parameter to be low on this wooden surface and is able to automatically generalize this fact to all points on the wood surface (figure 16).

The third test involves running the TurtleBot over a terrain simulating gravel. We spread pebbles on the floor and cover them with a green tarp for color (figure 14). We found that the robot estimates the area to have high aleatoric uncertainty, as seen in figure 15. However, while we expect the TurtleBot to also report low d values, we found that the d values in this location are higher. This may be because the TurtleBot is driving from a higher platform or because the odometry becomes more unreliable in such terrains.

Finally, we test that the controller can learn from past terrrains and successfully plan new routes in the real world. We instruct the TurtleBot to alternate navigating between its initial and goal configurations. When the robot navigate to the goal for the second time, we qualitatively compare its travel time to the previous iteration. While the results of terrain mapping should have led the robot to reach its goal at least as fast as the first time, the TurtleBot took at least several seconds longer. The path generated by the optimizer has many turns while the initial path was a straight line. The robot may have underestimated the d or k parameters and aggressively turns and brakes, which slows it down in the test. Videos are available on the website linked in the appendix.

VI. Conclusions and Future Work

In both sim and real, we are able to accomplish terrain-aware system identification and planning, as the robot finds areas with undesired dynamics or higher uncertainty and plans around them. Both feature mapping and path planning happen simultaneously, permitting Dijkstra's to continuously use information about the dynamics of the regions the robot travels over. Imaging does not have the accuracy we seek, as the homography map suffers from noise coming from the shaking of the robot itself. Additionally, transformation to the world space does not happen reliably due to the RealSense camera being loose, causing varying skews in the generation of the grid. Control also needs more tuning for less jerky navigation along a desired trajectory. Further, robot odometry is unreliable on many terrains as it comes from actively powered wheels. Still, this project provides a proof of concept for this method, especially in sim. While it is tested with wheeled robots, the principles involved are generalizable to other systems that need to have context-aware dynamics models.

The dynamics estimates in simulation often have errors near the transitions between different terrains, due to the presence of acceleration. TASI typically still converges to a solution but resolving issues like these will increase robustness.

Further, Dijkstra's algorithm, when applied to grid cells, produces trajectories with sharp corners. The PD controller smoothens these during execution, causing the robot to deviate from the chosen path, a safety concern near hazardous regions. Future implementations can rely on Control Barrier Function (CBF) lane-keeping to ensure the controller never leaves the set of safe states while navigating to the goal [1].

Future work could involve more rigorous mapping and System ID testing with different real terrains like sand and grass. Further, the ideas behind TASI must be shown to work with other robotic platforms. Currently, our work assumes flat surfaces, meaning mapping can be done easily. Testing in areas where homography-based mapping is less realistic, such as open spaces and sloped terrains, opens the avenue to utilizing other sensors, such as LIDAR. Finally, with more data and compute, we may be able to use neural networks as opposed to kernel regression for dynamics estimation.

VII. Acknowledgements

We would like to thank the 106B course staff and ESG for providing the means to carry out this research, as well as Professor Shankar Sastry and Professor Yi Ma for teaching EECS 106B at Berkeley.

VIII. Appendix

Our code is publicly viewable **here**, and our website is available **here**.

References

[1] Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. pages 3420–3431, 2019.

[2] Eric Coyle, Emmanuel G. Collins, and Liang Lu. Updating control modes based on terrain classification. In *2010 IEEE International Conference on Robotics and Automation*, pages 4417–4423, 2010.

[3] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[4] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction. *CoRR*, abs/1910.09457, 2019.

[5] Gregory Kahn, Pieter Abbeel, and Sergey Levine. BADGR: an autonomous self-supervised learning-based navigation system. *CoRR*, abs/2002.05700, 2020.

[6] Rushikesh Kamalapurkar, Benjamin Reish, Girish Chowdhary, and Warren E. Dixon. Concurrent learning for parameter estimation using dynamic state-derivative estimators. *IEEE Transactions on Automatic Control*, 62(7):3594–3601, 2017.

[7] Arthur J. Krener. Approximate linearization by state feedback and coordinate change. *Systems Control Letters*, 5(3):181–185, 1984.

[8] E. A. Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964.

[9] Wei Xiao, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, Ramin Hasani, and Daniela Rus. Differentiable control barrier functions for vision-based end-to-end autonomous driving, 2022.