BADGR: An Autonomous Self-Supervised Learning-Based Navigation System

Gregory Kahn, Pieter Abbeel, Sergey Levine
Berkeley AI Research (BAIR), University of California, Berkeley

Fig. 1: BADGR is an end-to-end learning-based mobile robot navigation system that can be trained with self-supervised off-policy data gathered in real-world environments, without any simulation or human supervision. Using only RGB images and GPS, BADGR can follow sparse GPS waypoints without colliding while (top row) preferring smooth concrete paths and (bottom row) ignoring geometrically distracting obstacles such as tall grass.

Abstract—Mobile robot navigation is typically regarded as a geometric problem, in which the robot's objective is to perceive the geometry of the environment in order to plan collision-free paths towards a desired goal. However, a purely geometric view of the world can can be insufficient for many navigation problems. For example, a robot navigating based on geometry may avoid a field of tall grass because it believes it is untraversable, and will therefore fail to reach its desired goal. In this work, we investigate how to move beyond these purely geometric-based approaches using a method that learns about physical navigational affordances from experience. Our approach, which we call BADGR, is an end-to-end learning-based mobile robot navigation system that can be trained with self-supervised off-policy data gathered in real-world environments, without any simulation or human supervision. BADGR can navigate in real-world urban and off-road environments with geometrically distracting obstacles. It can also incorporate terrain preferences, generalize to novel environments, and continue to improve autonomously by gathering more data. Videos, code, and other supplemental material are available on our website https://sites.google.com/view/badgr

I. INTRODUCTION

Navigation for mobile robots is often regarded as primarily a geometric problem, where the aim is to construct either a local or global map of the environment, and then plan a path through this map [1]. While this approach has produced excellent results in a range of settings, from indoor navigation [2] to autonomous driving [3], open-world mobile robot navigation often present challenges that are difficult to address with a purely geometric view of the world. Some geometric obstacles, such as the tall grass in Figure 1, may in fact be traversable.

Different surfaces, though geometrically similar, may be preferred to differing degrees—for example, the vehicle in Fig. 1 might prefer the paved surface over the bumpy field. Conventionally, these aspects of the navigation problem have been approached from the standpoint of *semantic* understanding, using computer vision methods trained on human-provided traversability or road surface labels [4, 5, 6, 7]. However, traversability, bumpiness, and other mobility-relevant attributes are *physical* attributes of the environment, and in principle can be learned by the robot through experience, without human supervision. In this paper, we study how autonomous, self-supervised learning from experience can enable a robot to learn about the *affordances* in its environment using raw visual perception and without human-provided labels or geometric maps.

Instead of using human supervision to teach a robot how to navigate, we investigate how the robot's own past experience can provide *retrospective* self-supervision: for many physically salient navigational objectives, such as avoiding collisions or preferring smooth over bumpy terrains, the robot can autonomously measure how well it has fulfilled its objective, and then retrospectively label the preceding experience so as to learn a *predictive* model for these objectives. For example, by experiencing collisions and bumpy terrain, the robot can learn, given an observation and a candidate plan of future actions, which actions might lead to bumpy or smooth to terrain, and which actions may lead to collision. This in effect constitutes a self-supervised multi-task reinforcement learning problem.

Based on this idea, we present a fully autonomous, self-

improving, end-to-end learning-based system for mobile robot navigation, which we call BADGR—Berkeley Autonomous Driving Ground Robot. BADGR works by gathering off-policy data—such as from a random control policy—in real-world environments, and uses this data to train a model that predicts future relevant events—such as collision, position, or terrain properties—given the current sensor readings and the recorded executed future actions. Using this model, BADGR can then plan into the future and execute actions that avoid certain events, such as collision, and actively seek out other events, such as smooth terrain. BADGR constitutes a fully autonomous self-improving system because it gathers data, labels the data in a self-supervised fashion, trains the predictive model in order to plan and act, and can autonomously gather additional data to further improve itself.

While the particular components that we use to build BADGR draw on prior work [8, 9], we demonstrate for the first time that a complete system based on these components can reason about both geometric and non-geometric navigational objectives, learn from off-policy data, does not require any human labelling, does not require expensive sensors or simulated data, can improve with autonomously collected data, and works in real-world environments.

The primary contribution of this work is an end-to-end learning-based mobile robot navigation system that can be trained entirely with self-supervised off-policy data gathered in real-world environments, without any simulation or human supervision. Our results demonstrate that our BADGR system can learn to navigate in real-world environments with geometrically distracting obstacles, such as tall grass, and can readily incorporate terrain preferences, such as avoiding bumpy terrain, using only 42 hours of autonomously collected data. Our experiments show that our method can outperform a LIDAR policy in complex real-world settings, generalize to novel environments, and can improve as it gathers more data.

II. RELATED WORK

Autonomous mobile robot navigation has been extensively studied in many real-world scenarios, ranging from indoor navigation [2, 10, 11, 12] to outdoor driving [3, 13, 14, 15, 16]. The predominant approach for autonomous navigation is to have the robot build a map, localize itself in the map, and use the map in order to plan and execute actions that enable the robot to achieve its goal. This simultaneous localization and mapping (SLAM) and planning approach [1] has achieved impressive results, and underlies current state-of-the-art autonomous navigation technologies [17, 18]. However, these approaches still have limitations, such as performance degradation in textureless scenes, requiring expensive sensors, and—most importantly—do not get better as the robot acts in the world [19].

Learning-based methods have shown promise in addressing these limitations by learning from data. One approach to improve upon SLAM methods is to directly estimate the geometry of the scene [20, 21, 22]. However, these methods are limited in that the geometry is only a partial description

of the environment. Only learning about geometry can lead to unintended consequences, such as believing that a field of tall grass is untraversable. Semantic-based learning approaches attempt to address the limitations of purely geometric methods by associating the input sensory data with semantically meaningful labels, such as which pixels in an image correspond to traversable or bumpy terrain. However, these methods typically depend on existing SLAM approaches [4, 5, 23, 9, 24] or humans [6, 7] in order to provide the semantic labels, which consequently means these approaches either inherit the limitations of geometric approaches or are not autonomously self-improving. Methods based on imitation learning have been demonstrated on real-world robots [25, 26, 27], but again depend on humans for expert demonstrations, which does not constitute a continuously self-improving system. End-to-end reinforcement learning approaches have shown promise in automating the entire navigation pipeline. However, these methods have typically focused on pure geometric reasoning, require on-policy data, and often utilize simulation due to constraints such as sample efficiency [28, 8, 29, 30, 31, 32]. Prior works have investigated learning navigation policies directly from real-world experience, but typically require a person [30, 33, 34] or SLAM algorithm [35] to gather the data, assume access to the ground-truth robot state [36], learn using low-bandwidth sensors [37], or only perform collision avoidance [8, 38]. Our approach overcomes the limitations of these prior works by designing an end-to-end reinforcement learning approach that directly learns to predict relevant navigation cues with a sample-efficient, off-policy algorithm, and can continue to improve with additional experience via a self-supervised data labelling mechanism that does not depend on humans or SLAM algorithms.

The most similar works to our BADGR system are GCG [8] and CAPs [9]. However, GCG only learned to avoid collisions and CAPs required human supervision in order to learn non-collision avoidance behaviors, while BADGR is able to reason about both geometric and non-geometric navigational without any human supervision in complex, real-world environments.

III. BERKELEY AUTONOMOUS DRIVING GROUND ROBOT

Our goal is to enable a mobile robot to navigate in real-world environments. We therefore developed BADGR, an end-to-end learning-based mobile robot navigation system that can be trained entirely with self-supervised, off-policy data gathered in real-world environments, without any simulation or human supervision, and can improve as it gathers more data.

BADGR operates by autonomously gathering large amounts of off-policy data in real-world environments. Using this data, BADGR labels relevant events—such as collisions or bumpy terrain—in a self-supervised manner, and adds these labelled events back into the dataset. BADGR then trains a predictive model that takes as input the current observation, such as camera images, and a future sequence of actions, such as linear and angular velocity commands, and predicts the relevant future events. When deploying the trained BADGR system,

the user designs a reward function that encodes the task they want the robot to accomplish in terms of these relevant events—such as to reach a goal while avoiding collisions and bumpy terrain—and the robot autonomously plans and executes actions that maximize this reward.

In order to build a self-supervised learning-based navigational system, we must assume that *retrospective* supervision is available to the robot. This means that the robot must be able to experience events, such as collisions, and then learn to avoid (or seek out) such events in the future. We therefore make the following assumptions: (1) the robot can only learn about events it has experienced and that can be measured using the onboard sensors and (2) experiencing these events, even undesirable events such as colliding, is acceptable. We believe these assumptions are realistic for many real-world autonomous mobile robot applications.

In the following sections, we will describe the robot, data collection and labelling, model training, and planning components, followed by a summarizing overview of the entire system.

A. Mobile Robot Platform

The specific design considerations for the robotic platform focus on enabling long-term autonomy with minimal human intervention.

The robot we use is the Clearpath Jackal, shown in Fig. 2. The Jackal measures $508\text{mm} \times 430\text{mm} \times 250\text{mm}$ and weighs 17kg, making it ideal for navigating in both urban and off-road environments. The Jackal is controlled by specifying the desired linear and angular ve-

Fig. 2: The mobile robot platform.

locity, which are used as setpoints for the low-level differential drive controllers. The default sensor suite consists of a 6-DOF IMU, which measures linear acceleration and angular velocity, a GPS unit for approximate global position estimates, and encoders to measure the wheel velocity. In addition, we added the following sensors on top of the Jackal: two forward-facing $170°$ field-of-view 640×480 cameras, a 2D LIDAR, and a compass.

Inside the Jackal is an NVIDIA Jetson TX2 computer, which is ideal for running deep learning applications, in addition to interfacing with the sensors and low-level microcontrollers. Data is saved to an external SSD, which must be large and fast enough to store 1.3GB per minute streaming in from the sensors. Experiments were monitored remotely via a 4G smartphone mounted on top of the Jackal, which allowed for video streaming and, if needed, teleoperation.

The robot was designed primarily for robustness, with a relatively minimal and robust sensor suite, in order to enable long-term autonomous operation for large-scale data collection.

B. Data Collection

We design the data collection methodology to enable gathering large amounts of diverse data for training with minimal human intervention.

The first consideration when designing the data collection policy is whether the learning algorithm requires on-policy data. On-policy data collection entails alternating between gathering data using the current policy, and retraining the policy using the most recently gathered data. On-policy data collection is highly undesirable because only the most recently gathered data can be used for training; all previously gathered data must be thrown out. In contrast, off-policy learning algorithms can train policies using data gathered by any control policy. Due to the high cost of gathering data with real-world robotic systems, we choose to use an off-policy learning algorithm in order to be able to gather data using any control policy and train on all of the gathered data.

The second consideration when designing the data collection policy is to ensure the environment is sufficiently explored, while also ensuring that the robot execute action sequences it will realistically wish to execute at test time. A naïve uniform random control policy is inadequate because the robot will primarily drive straight due to the linear and angular velocity action interface of the robot, which will result in both insufficient exploration and unrealistic test time action sequences. We therefore use a time-correlated random walk control policy to gather data, which is visualized in Fig. 3.
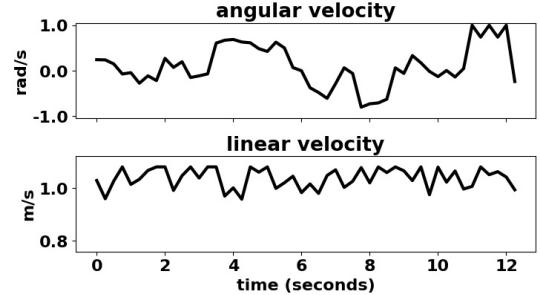

Fig. 3: An example plot of the commanded angular and linear velocities from our time-correlated random control policy that is used to gather data.

As the robot is gathering data using the random control policy, it will require a mechanism to detect if it is in collision or stuck, and an automated controller to reset itself in order to continue gathering data. We detect collisions in one of two ways, either using the LIDAR to detect when an obstacle is near or the IMU to detect when the robot is stuck due to an obstacle. We used the LIDAR collision detector in urban environments in order to avoid damaging property, and the IMU collision detector in

Fig. 4: While collecting data, the robot will periodically require a manual intervention to reset from catastrophic failures, though recovery is usually automatic.
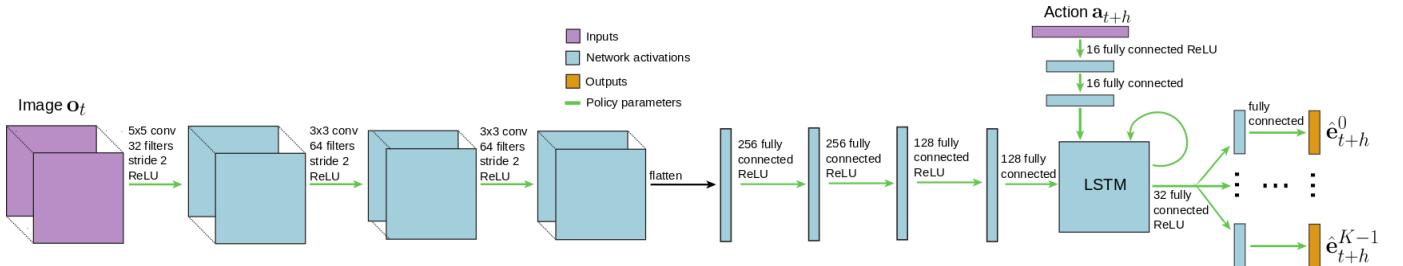
Fig. 5: Illustration of the deep neural network predictive model at the core of our learning-based navigation policy. The neural network takes as input the current RGB image and processes it with convolutional and fully connected layers to form the initial hidden state of a recurrent LSTM unit [39]. This recurrent unit takes as input H actions in a sequential fashion, and produces H outputs. These outputs of the recurrent unit are then passed through additional fully connected layers to predict all K events for all H future time steps. These predicted future events, such as position, if the robot collided, and if the robot drove over bumpy terrain, enable a planner to select actions that achieve desirable events, such reaching a goal, and avoid undesirable events, such as collisions and bumpy terrain.

off-road environments because the LIDAR detector was overly pessimistic, such as detecting grass as an obstacle. Once a collision is detected, a simple reset policy commands the robot to back up and rotate. However, sometimes the reset policy is insufficient, for example if the robot flips over (Fig. 4), and a person must manually reset the robot.

As the robot collects data, all the raw sensory data is saved onboard. After data collection for the day is complete, the data is copied to a desktop machine and subsampled down to 4Hz.

C. Self-Supervised Data Labelling

BADGR then goes through the raw, subsampled data and calculates labels for specific navigational events. These events consist of anything pertinent to navigation that can be extracted from the data in a self-supervised fashion.

In our experiments, we consider three different events: collision, bumpiness, and position. A collision event is calculated as occurring when, in urban environments, the LIDAR measures an obstacle to be close or, in off-road environments, when the IMU detects a sudden drop in linear acceleration and angular velocity magnitudes. A bumpiness event is calculated as occurring when the angular velocity magnitudes measured by the IMU are above a certain threshold. The position is determined by an onboard state estimator that fuses wheel odometry and the IMU to form a local position estimate.

After BADGR has iterated through the data, calculated the event labels at each time step, and added these event labels back into the dataset, BADGR can then train a model to predict which actions lead to which navigational events.

D. Predictive Model

The learned predictive model takes as input the current sensor observations and a sequence of future intended actions, and predicts the future navigational events. We denote this model as $f_\theta(\mathbf{o}_t, \mathbf{a}_{t:t+H}) \rightarrow \hat{\mathbf{e}}_{t:t+H}^{0:K}$, which defines a function f parameterized by vector θ that takes as input the current observation $\mathbf{o}_t$ and a sequence of H future actions $\mathbf{a}_{t:t+H} = (\mathbf{a}_t, \mathbf{a}_{t+1}, ..., \mathbf{a}_{t+H-1})$, and predicts K different future events $\hat{\mathbf{e}}_{t:t+H}^k = (\hat{\mathbf{e}}_t^k, \hat{\mathbf{e}}_{t+1}^k, ..., \hat{\mathbf{e}}_{t+H-1}^k) \ \forall k \in \{0, ..., K-1\}$.

The model we learn is an image-based, action-conditioned predictive deep neural network, shown in Fig. 5. The network first processes the input image observations using convolutional and fully connected layers. The final output of the these layers serves as the initialization for a recurrent neural network, which sequentially processes each of the H future actions $\mathbf{a}_{t+h}$ and outputs the corresponding predicted future events $\hat{\mathbf{e}}_{t+h}^{0:K}$.

The model is trained—using the observations, actions, and event labels from the collected dataset—to minimize a loss function that penalizes the distance between the predicted and ground truth events

$$\mathcal{L}(\theta, \mathcal{D}) = \sum_{(\mathbf{o}_t, \mathbf{a}_{t:t+H}) \in \mathcal{D}} \sum_{k=0}^{K-1} \mathcal{L}^k(\hat{\mathbf{e}}_{t:t+H}^k, \mathbf{e}_{t:t+H}^k) : \quad (1)$$
$$\hat{\mathbf{e}}_{t:t+H}^{0:K} = f_\theta(\mathbf{o}_t, \mathbf{a}_{t:t+H}).$$

The individual losses $\mathcal{L}^k$ for each event are either cross entropy if the event is discrete, or mean squared error if the event is continuous. The neural network parameter vector θ is trained by performing minibatch gradient descent on the loss in Eqn. 1.

E. Planning

Given the trained neural network predictive model, this model can then be used at test time to plan and execute desirable actions.

We first define a reward function $R(\hat{\mathbf{e}}_{t:t+H}^{0:K})$ that encodes what we want the robot to do in terms of the model's predicted future events. For example, the reward function could encourage driving towards a goal while discouraging collisions or driving over bumpy terrain. The specific reward function we use is specified in the experiments section.

Using this reward function and the learned predictive model, we solve the following planning problem at each time step

$$\mathbf{a}_{t:t+H}^* = \arg \max_{\mathbf{a}_{t:t+H}} R(f_\theta(\mathbf{o}_t, \mathbf{a}_{t:t+H})), \quad (2)$$

execute the first action, and continue to plan and execute following the framework of model predictive control.

We solve Eqn. 2 using the zeroth order stochastic optimizer from Nagabandi et al. [40]. This optimizer works by maintaining a running estimate $\hat{\mathbf{a}}_{0:H}$ of the optimal action sequence. Each time the planner is called, N action sequences $\tilde{\mathbf{a}}_{0:H}^{0:N}$ are sampled that are time-correlated and centered around this running action sequence estimate

$$\epsilon_h^n \sim \mathcal{N}(0, \sigma \cdot \mathbf{I}) \ \forall n \in \{0...N-1\}, h \in \{0...H-1\} \quad (3)$$
$$\tilde{\mathbf{a}}_h^n = \beta \cdot (\hat{\mathbf{a}}_{h+1} + \epsilon_h^n) + (1-\beta) \cdot \tilde{\mathbf{a}}_{h-1}^n \text{ where } \tilde{\mathbf{a}}_{h<0} = 0,$$

in which the parameter σ determines how close the sampled action sequences should be to the running action sequence estimate, and the parameter $\beta \in [0,1]$ determines the degree to which the sampled action sequences are correlated in time.

Each action sequence is then propagated through the predictive model in order to calculate the reward $\tilde{R}^n = R(f_\theta(\mathbf{o}_t, \tilde{\mathbf{a}}_{0:H}^n))$. Given each action sequence and its corresponding reward, we update the running estimate of the optimal action sequence via a reward-weighted average

$$\hat{\mathbf{a}}_{0:H} = \frac{\sum_{n=0}^{N} \exp(\gamma \cdot R^n) \cdot \tilde{\mathbf{a}}_{0:H}^n}{\sum_{n'=0}^{N} \exp(\gamma \cdot R^{n'})}, \quad (4)$$

in which $\gamma \in \mathrm{R}^+$ is a parameter that determines how much weight should be given to high-reward action sequences.

Each time the planner is called, new action sequences are sampled according to Eqn. 3, these action sequences are propagated through the learned predictive model in order to calculate the reward of each sequence, the running estimate of the optimal action sequence is updated using Eqn. 4, and the robot executes the first action $\hat{\mathbf{a}}_0$.

This optimizer is more powerful than other zeroth order stochastic optimizers, such as random shooting or the cross-entropy method [41], because it warm-starts the optimization using the solution from the previous time step, uses a soft update rule for the new sampling distribution in order to leverage all of the sampled action sequences, and considers the correlations between time steps. In our experiments, we found this more powerful optimizer was necessary to achieve good planning.

F. Algorithm Summary

We now provide a brief summary of how our BADGR system operates during training (Alg. 1) and deployment (Alg. 2).

During training, BADGR gathers data by executing actions according to the data collection policy and records the onboard sensory observations and executed actions. Next, BADGR uses the gathered dataset to self-supervise the event labels, which are added back into the dataset. This dataset is then used to train the learned predictive model.

When deploying BADGR, the user first defines a reward function that encodes the specific task they want the robot to accomplish. BADGR then uses the trained predictive model, current observation, and reward function to plan a sequence of actions that maximize the reward function. The robot executes the first action in this plan, and BADGR continues to alternate between planning and executing until the task is complete.

Algorithm 1 Training BADGR

1: initialize dataset $\mathcal{D} \leftarrow \emptyset$
2: **while** not done collecting data **do**
3:     get current observation $\mathbf{o}_t$ from sensors
4:     get action $\mathbf{a}_t$ from data collection policy
5:     add $(\mathbf{o}_t, \mathbf{a}_t)$ to $\mathcal{D}$
6:     execute $\mathbf{a}_t$
7:     **if** in collision **then**
8:         execute reset maneuver
9:     **end if**
10: **end while**
11: **for each** $(\mathbf{o}_t, \mathbf{a}_t) \in \mathcal{D}$ **do**
12:     calculate event labels $\mathbf{e}_t^{0:K}$ using self-supervision
13:     add $\mathbf{e}_t^{0:K}$ to $\mathcal{D}$
14: **end for**
15: use $\mathcal{D}$ to train predictive model f_θ by minimizing Eqn. 1

Algorithm 2 Deploying BADGR

1: **input**: trained predictive model f_θ, reward function R
2: **while** task is not complete **do**
3:     get current observation $\mathbf{o}_t$ from sensors
4:     solve Eqn. 2 using f_θ, $\mathbf{o}_t$, and R
      to get the planned action sequence $\mathbf{a}_{t:t+H}^*$
5:     execute the first action $\mathbf{a}_t^*$
6: **end while**

IV. EXPERIMENTS

In our experimental evaluation, we study how BADGR can autonomously learn a successful navigation policy in real-world environments, improve as it gathers more data, generalize to unseen environments, and compare it to purely geometric approaches. Videos, code, and other supplemental material are available on our website[1].

We performed our evaluation in a real-world outdoor environment consisting of both urban and off-road terrain. BADGR autonomously gathered 34 hours of data in the urban terrain and 8 hours in the off-road terrain. Although the amount of data gathered may seem significant, the total dataset consisted of 720,000 off-policy datapoints, which is smaller than currently used datasets in computer vision [42] and significantly smaller than the number of samples often used by deep reinforcement learning algorithms [43].

Our evaluations consist of tasks that involve reaching a goal GPS location, avoiding collisions, and preferring smooth over bumpy terrain. In order for BADGR to accomplish these tasks, we design the reward function that BADGR uses for planning
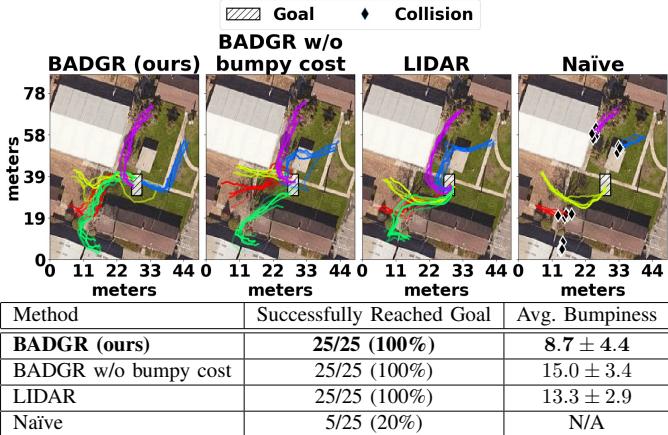
---

[1]https://sites.google.com/view/badgr

Fig. 7: Visualization of BADGR's predictive model in the urban environment. Each image shows the candidate paths that BADGR considers during planning. These paths are color coded according to either their probability of collision (top row) or probability of experiencing bumpy terrain (bottom row) according to BADGR's learned predictive model. These visualizations show the learned model can accurately predict that action sequences which would drive into buildings or bushes will result in a collision, and that action sequences which drive on concrete paths are smoother than driving on grass.

Method	Successfully Reached Goal	Avg. Bumpiness
BADGR (ours)	**25/25 (100%)**	8.7 ± 4.4
BADGR w/o bumpy cost	25/25 (100%)	15.0 ± 3.4
LIDAR	25/25 (100%)	13.3 ± 2.9
Naïve	5/25 (20%)	N/A

Fig. 6: Experimental evaluation in an urban environment for the task of reaching a specified goal position while avoiding collisions and bumpy terrain. Each approach was evaluated from 5 different start locations—each color corresponding to a different start location—with 5 runs per each start location. The figures show the paths of each run, and whether the run successfully reached the goal or ended in a collision. The table shows the success rate and average bumpiness for each method. Our BADGR approach is better able to reach the goal and avoid bumpy terrain compared to the other methods.

as such

$$R(\hat{\mathbf{e}}_{t:t+H}^{0:K}) = -\sum_{t'=t}^{t+H-1} R^{\text{COLL}}(\hat{\mathbf{e}}_{t'}^{0:K}) + \qquad (5)$$
$$\alpha^{\text{POS}} \cdot R^{\text{POS}}(\hat{\mathbf{e}}_{t'}^{0:K}) + \alpha^{\text{BUM}} \cdot R^{\text{BUM}}(\hat{\mathbf{e}}_{t'}^{0:K})$$

$$R^{\text{COLL}}(\hat{\mathbf{e}}_{t'}^{0:K}) = \hat{\mathbf{e}}_{t'}^{\text{COLL}}$$

$$R^{\text{POS}}(\hat{\mathbf{e}}_{t'}^{0:K}) = (1 - \hat{\mathbf{e}}_{t'}^{coll}) \cdot \frac{1}{\pi} \angle(\hat{\mathbf{e}}_{t'}^{\text{POS}}, \mathbf{p}^{\text{GOAL}}) + \hat{\mathbf{e}}_{t'}^{coll}$$

$$R^{\text{BUM}}(\hat{\mathbf{e}}_{t'}^{0:K}) = (1 - \hat{\mathbf{e}}_{t'}^{coll}) \cdot \hat{\mathbf{e}}_{t'}^{\text{BUM}} + \hat{\mathbf{e}}_{t'}^{coll},$$

where α^{POS} and α^{BUM} are user-defined scalars that weight how much the robot should care about reaching the goal and avoiding bumpy terrain. An important design consideration for the reward function was how to encode this multi-objective task. First, we ensured each of the individual rewards were in the range of $[0, 1]$, which made it easier to weight the individual rewards. Second, we ensured the collision reward always dominated the other rewards: if the robot predicted it was going to collide, all of the individual rewards were assigned to their maximum value of 1; conversely, if the robot predicted it was not going to collide, all of the individual rewards were assigned to their respective values.

We evaluated BADGR against two other methods:

- *LIDAR*: a policy that drives towards the goal while avoiding collisions using the range measurements from an onboard 2D LIDAR. Note that our method only uses the camera images, while this approach uses LIDAR.
- *Naïve*: a naïve policy that simply drives straight towards the specified goal.

We compare against LIDAR, which is a common geometric-based approach for designing navigation policies, in order to demonstrate the advantages of our learning-based approach, while the purpose of the naïve policy is to provide a lower bound baseline and calibrate the difficulty of the task.
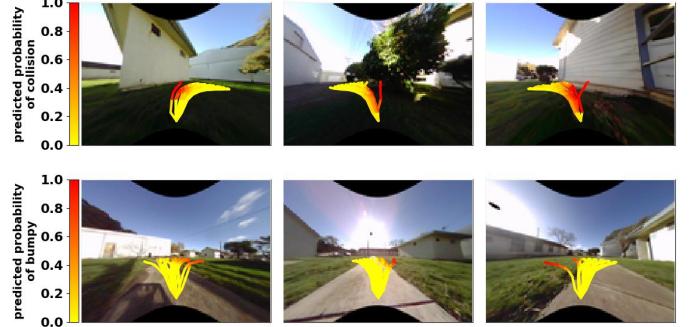
Note that for all tasks, only a single GPS coordinate—the location of the goal—is given to the robot. This single GPS coordinate is insufficient for successful navigation, and therefore the robot must use other onboard sensors in order to accomplish the task.

Urban environment. We first evaluated all the approaches for the task of reaching a goal GPS location while avoiding collisions and bumpy terrain in an urban environment. Fig. 6 shows the resulting paths that BADGR, LIDAR, and the naïve policies followed. The naïve policy almost always crashed, which illustrates the urban environment contains many obstacles. The LIDAR policy always succeeded in reaching the goal, but failed to avoid the bumpy grass terrain. BADGR also always succeeded in reaching the goal, and—as also shown by Fig. 6—succeeded in avoiding bumpy terrain by driving on the paved paths. Note that we never told the robot to drive on paths; BADGR automatically learned from the onboard camera images that driving on concrete paths is smoother than driving on the grass.

While a sufficiently high-resolution 3D LIDAR could in principle identify the bumpiness of the terrain and detect the paved paths automatically, 3D geometry is not a perfect indicator of the terrain properties. For example, let us compare tall grass versus gravel terrain. Geometrically, the tall grass is bumpier than the gravel, but when actually driving over these terrains, the tall grass will result in a smoother ride. This example underscores the idea that there is not a clear mapping between geometry and physically salient properties such as whether terrain is smooth or bumpy.

BADGR overcomes this limitation by directly learning about physically salient properties of the environment using the raw onboard observations—in this case, the IMU readings—to determine if the terrain is bumpy. Our approach does not make assumptions about geometry, but rather lets the predictive model learn correlations from the onboard sensors;
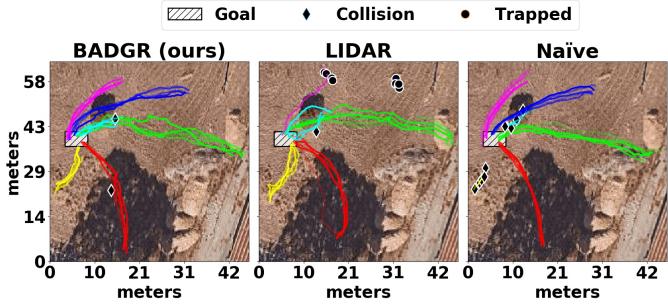
Fig. 8: Experimental evaluation in an off-road environment for the task of reaching a specified goal location while avoiding collisions. Each approach was evaluated from 5 different start locations—each color corresponding to a different start location—with 5 runs per each start location. Each run terminated when the robot collided, failed to make progress and was trapped, or successfully reached the goal. Our BADGR policy is the only approach which can consistently reach the goal without colliding or getting trapped.

Fig. 9: Comparison of LIDAR (top) versus our BADGR approach (bottom) in a tall grass portion of the off-road environment. The LIDAR policy incorrectly labels the grass as untraversable, and therefore rotates in-place in an attempt to find a traversable path; after completing a full rotation and failing to detect any traversable path, the LIDAR policy determines the robot is trapped. In contrast, our BADGR approach has learned from experience that some tall grass is indeed traversable, and is therefore able to successfully navigate the robot towards the goal.

Fig. 7 shows our predictive model successfully learns which image and action sequences lead to collisions and bumpy terrain and which do not.

Off-road environment. Next, we evaluated all the approaches for the task of reaching a goal GPS location while avoiding both collisions and getting stuck in an off-road environment. Fig. 8 shows the resulting paths that BADGR, LIDAR, and the naïve policies followed. The naïve policy sometimes succeeded, but oftentimes collided with obstacles such as trees and became stuck on thick patches of grass. The LIDAR policy nearly never crashed or became stuck on grass, but sometimes refused to move because it was surrounded by grass which it incorrectly labelled as untraversable obstacles (Fig. 9a). BADGR almost always succeeded in reaching the goal by avoiding collisions and getting stuck, while not falsely predicting that all grass was an obstacle (Fig. 9b).

Additionally, even when the LIDAR approach succeeded in reaching the goal, the path it took was sometimes suboptimal. Fig. 10 shows an example where the LIDAR policy labelled a few strands of grass as untraversable obstacles, and therefore decided to take a roundabout path to the goal; in contrast, BADGR accurately predicted these few strands of grass were traversable, and therefore took a more optimal path. BADGR reached the goal $1.2\times$ faster on average compared to the LIDAR policy.
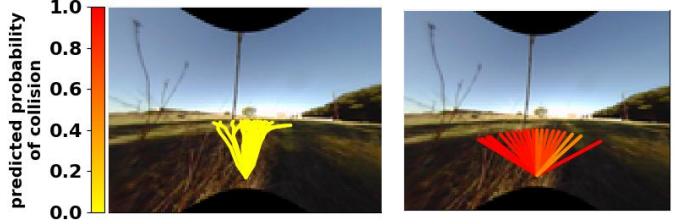

Fig. 10: Comparison of our BADGR policy (left) versus the LIDAR policy (right). Each image shows the candidate paths each policy considered during planning, and the color of each path indicates if the policy predicts the path will result in a collision. The LIDAR policy falsely predicts the paths driving left or straight will result in a collision with the few strands of tall grass. In contrast, our BADGR policy correctly predicts that the grass is traversable and will therefore drive over the grass, which will result in BADGR reaching the goal $1.5\times$ faster.

Self-improvement. A practical deployment of BADGR would be able to continually self-supervise and improve the model as the robot gathers more data. To provide an initial evaluation of how additional data enables adaptation to new circumstances, we conducted a controlled study in which BADGR gathers and trains on data from one area, moves to a new target area, fails at navigating in this area, but then eventually succeeds in the target area after gathering and training on additional data from that area.

In this experiment, we first evaluate the performance of the original model trained only in the initial training domain, labeled as 'zero-shot' in Figure 11. The zero-shot policy fails on every trial due to a collision. We then evaluate the performance of a policy that is finetuned after collecting three more hours of data with autonomous self-supervision in the target domain, which we label as 'finetuned.' This model succeeds at reaching the goal on every trial. For completeness, we also evaluate a model trained *only* on the data from the target domain, without using the data from the original training domain, which we label as 'target domain only.' This model is better than the zero-shot model, but still fails much more frequently than the finetuned model that uses both sources of
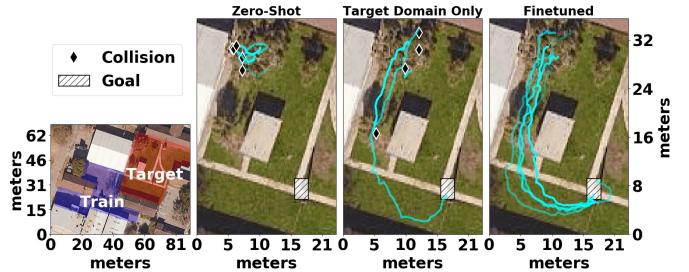

Fig. 11: Experimental demonstration of our BADGR approach improving as it gathers more experience. The robot's task was to reach a goal in the new target area without colliding. The zero-shot policy trained with only the initial training data always failed. The finetuned policy trained only using data from the target domain travelled farther before colliding, but still predominantly failed. The policy trained using both the initial training data and three hours of autonomously gathered, self-supervised-supervised data in the target domain always succeeded in reaching the goal. This result demonstrates that BADGR improves as it gathers more data, and also that previously gathered data from other areas can actually accelerate learning.

Fig. 12: Our BADGR policy can generalize to novel environments not seen in the training data. Each row shows the BADGR policy executing in a different novel environment. The first column shows the approximate path followed by the BADGR policy. The remaining columns show sampled images from the onboard camera while the robot is navigating, with the future path of the robot overlaid onto the image.

experience.

This experiment not only demonstrates that BADGR can improve as it gathers more data, but also that previously gathered experience can actually accelerate policy learning when BADGR encounters a new environment. From these results, we might reasonably extrapolate that as BADGR gathers data in more and more environments, it should take less and less time to successfully learn to navigate in each new environment; we hope that future work will evaluate these truly continual and lifelong learning capabilities.

Generalization. We also evaluated how well BADGR— when trained on the full 42 hours of collected data—navigates in novel environments not seen in the training data. Fig. 12 shows our BADGR policy successfully navigating in three novel environments, ranging from a forest to urban buildings. This result demonstrates that BADGR can generalize to novel environments if it gathers and trains on a sufficiently large and diverse dataset.

V. Discussion

We presented BADGR, an end-to-end learning-based mobile robot navigation system that can be trained entirely with self-supervised, off-policy data gathered in real-world environments, without any simulation or human supervision, and can improve as it gathers more data. We demonstrated that our approach can learn to navigate in real-world environments with geometrically distracting obstacles, such as tall grass, and can readily incorporate terrain preferences, such as avoiding bumpy terrain, using only 42 hours of data. Our experiments showed that BADGR can outperform a LIDAR policy in complex real-world settings, generalize to novel environments,

and can improve as it gathers more data.

Although BADGR can autonomously gather data with minimal human supervision, the robot periodically requires human assistance, for example if the robot flips over. These periodic resets can be a significant burden, especially if the robot is operating in a remote location. Investigating methods that train policies which are cautious in novel environments could further decrease the amount of human supervision needed while collecting data. Also, while BADGR can improve as it gathers more data, this improvement requires gathering a non-negligible amount of data and retraining from scratch. Approaches that adapt in an online fashion could improve BADGR's performance when deployed. Finally, our experiments solely evaluated BADGR in static environments, without moving agents such as pedestrians and cars. Gathering and training with data from non-static environments could prove challenging due to biases in the data from the interactions between the data collection policy and the other agents in the environment. We believe that solving these and other challenges is crucial for enabling robot learning platforms to learn and act in the real world, and that BADGR is a promising step towards this goal.

VI. Acknowledgments

REFERENCES

[1] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2008.

[2] C Rosen and N Nilsson. Application of intelligent automata to reconnaissance. Technical report, SRI, 1968.

[3] Charles Thorpe, Martial H Hebert, Takeo Kanade, and Steven A Shafer. Vision and navigation for the Carnegie-Mellon Navlab. *TPAMI*, 1988.

[4] Jeff Michels, Ashutosh Saxena, and Andrew Y Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *ICML*, 2005.

[5] Raia Hadsell, Pierre Sermanet, Jan Ben, Ayse Erkan, Marco Scoffier, Koray Kavukcuoglu, Urs Muller, and Yann LeCun. Learning long-range vision for autonomous off-road driving. *Journal of Field Robotics*, 2009.

[6] Abhinav Valada, Johan Vertens, Ankit Dhall, and Wolfram Burgard. Adapnet: Adaptive semantic segmentation in adverse environmental conditions. In *ICRA*, 2017.

[7] Noriaki Hirose, Amir Sadeghian, Marynel Vázquez, Patrick Goebel, and Silvio Savarese. Gonet: A semi-supervised deep learning approach for traversability estimation. In *IROS*, 2018.

[8] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *ICRA*, 2018.

[9] Gregory Kahn, Adam Villaflor, Pieter Abbeel, and Sergey Levine. Composable Action-Conditioned Predictors: Flexible Off-Policy Learning for Robot Navigation. In *CoRL*, 2018.

[10] Jonathan P How, Brett Behihke, Adrian Frank, Daniel Dale, and John Vian. Real-time indoor autonomous vehicle test environment. *IEEE Control Systems Magazine*, 2008.

[11] Shaojie Shen, Nathan Michael, and Vijay Kumar. Autonomous multi-floor indoor navigation with a computationally constrained MAV. In *ICRA*, 2011.

[12] Liam Paull, Jacopo Tani, Heejin Ahn, Javier Alonso-Mora, Luca Carlone, Michal Cap, Yu Fan Chen, Changhyun Choi, Jeff Dusek, Yajun Fang, et al. Duckietown: an open, inexpensive and flexible platform for autonomy education and research. In *ICRA*, 2017.

[13] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 2006.

[14] Sebastian Scherer, Sanjiv Singh, Lyle Chamberlain, and Mike Elgersma. Flying fast and low among obstacles: Methodology and experiments. *The International Journal of Robotics Research*, 2008.

[15] Chris Urmson and et. al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 2008.

[16] Paul Furgale and Timothy D Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 2010.

[17] Waymo. URL https://waymo.com.

[18] Skydio. URL https://www.skydio.com.

[19] Jorge Fuentes-Pacheco, Jose Ruiz-Ascencio, and Juan Manuel Rendon-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, 2015.

[20] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In *CVPR*, 2018.

[21] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *CVPR*, 2018.

[22] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *CVPR*, 2019.

[23] Charles Richter and Nick Roy. Safe Visual Navigation via Deep Learning and Novelty Detection. In *RSS*, 2017.

[24] Lorenz Wellhausen, Alexey Dosovitskiy, René Ranftl, Krzysztof Walas, Cesar Cadena, and Marco Hutter. Where should I walk? Predicting terrain properties from images via self-supervised learning. *RA-L*, 2019.

[25] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive uav control in cluttered natural environments. In *ICRA*, 2013.

[26] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv:1604.07316*, 2016.

[27] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *ICRA*, 2018.

[28] Fereshteh Sadeghi and Sergey Levine. (CAD)2 RL: Real Single-Image Flight without a Single Real Image. *RSS*, 2017.

[29] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *ICLR*, 2018.

[30] Jake Bruce, Niko Sünderhauf, Piotr Mirowski, Raia Hadsell, and Michael Milford. Learning deployable navigation policies at kilometer scale from a single traversal. In *CoRL*, 2018.

[31] Xiangyun Meng, Nathan Ratliff, Yu Xiang, and Dieter Fox. Neural Autonomous Navigation with Riemannian Motion Policy. In *ICRA*, 2019.

[32] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end-to-end with autorl. *RA-L*, 2019.

[33] Antonio Loquercio, Ana I Maqueda, Carlos R Del-Blanco, and Davide Scaramuzza. Dronet: Learning to fly by driving. In *RA-L*, 2018.

[34] Noriaki Hirose, Fei Xia, Roberto Martín-Martín, Amir Sadeghian, and Silvio Savarese. Deep Visual MPC-Policy Learning for Navigation. In *RA-L*, 2019.

[35] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. In *IROS*, 2017.

[36] Martin Riedmiller, Mike Montemerlo, and Hendrik Dahlkamp. Learning to drive a real car in 20 minutes. In *FBIT*, 2007.

[37] A Rupam Mahmood, Dmytro Korenkevych, Gautham Vasan, William Ma, and James Bergstra. Benchmarking reinforcement learning algorithms on real-world robots. In *CoRL*, 2018.

[38] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *ICRA*, 2019.

[39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[40] Anusha Nagabandi, Kurt Konoglie, Sergey Levine, and Vikash Kumar. Deep Dynamics Models for Learning Dexterous Manipulation. In *CoRL*, 2019.

[41] Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.

[42] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[43] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018.