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

Gregory Kahn, Pieter Abbeel, *Fellow, IEEE*, and Sergey Levine

***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 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 reinforcement learning approach, which we call BADGR , is an end-to-end learning-based mobile robot navigation system that can be trained with autonomously-labeled 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**

***Index Terms*—Big Data in robotics and automation, reinforcement learning, autonomous agents.**

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 Fig. 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. In this letter, 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 reinforcement 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 an automated 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 [4], [5], 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 reinforcement learning-based mobile robot navigation system that can be trained entirely with autonomously-labeled 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

Manuscript received October 15, 2020; accepted January 28, 2021. Date of publication February 3, 2021; date of current version February 22, 2021. This letter was recommended for publication by Associate Editor T. Kulvicius and Editor T. Asfour upon evaluation of the reviewers' comments. This work was supported in part by ARL DCIST CRA W911NF-17-2-0181, in part by the National Science Foundation via IIS-1700697, and in part by the DARPA Assured Autonomy Program, and Berkeley Deep Drive. The work of Gregory Kahn was supported by an NSF Graduate Research Fellowship. *(Corresponding author: Gregory Kahn.)*

The authors are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley 94710 USA (e-mail: gkahn@berkeley.edu; pabbeel@berkeley.edu; svlevine@berkeley.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2021.3057023, provided by the authors.

Digital Object Identifier 10.1109/LRA.2021.3057023

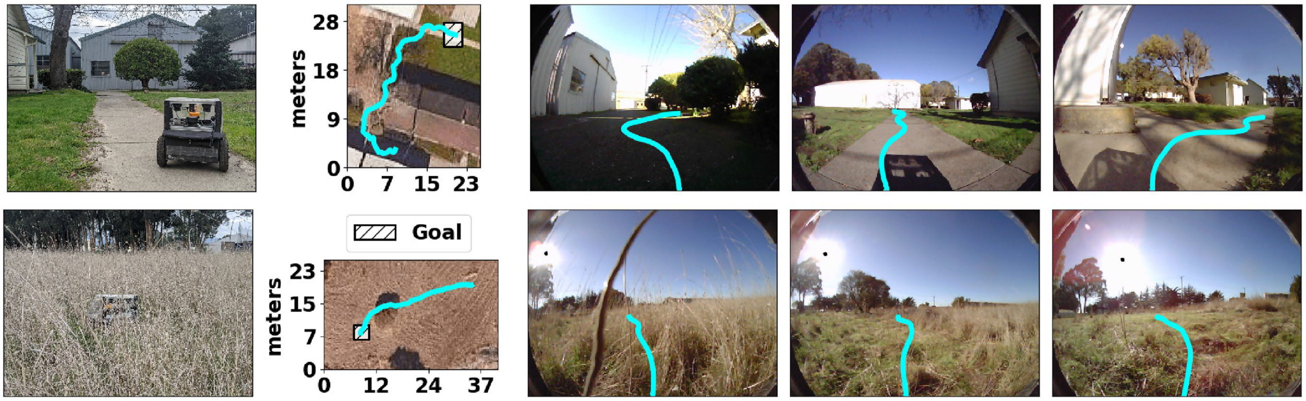


Fig. 1. BADGR is an end-to-end learning-based mobile robot navigation system. BADGR works by gathering off-policy data—such as from a random control policy—in real-world environments, and uses this data to train a neural network model that predicts future relevant events—such as collision, position, or terrain properties—given the current sensor readings and the recorded executed future actions. Using only this model, RGB images, and GPS, BADGR can plan into the future and execute actions that follow sparse GPS waypoints without colliding while (top row) preferring smooth concrete paths and (bottom row) ignoring geometrically distracting obstacles such as tall grass. While the particular components that BADGR is built on draw from prior work, BADGR demonstrates 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.

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], [6], [7] to outdoor driving [3], [8]. 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. However, these approaches still have limitations, including not being able to reason about affordances of the environment and—most importantly—do not get better as the robot acts in the world [1].

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 [9], [10]. 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 [5], [11]–[13] or humans [14], [15] 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 [16], [17], 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 [4], [18]–[20]; bridging the sim-to-real gap remains a challenging open research problem. Prior works have investigated learning navigation policies directly from real-world experience, but typically require a person [18], [21] or SLAM algorithm [22] to gather the data, assume access to the ground-truth robot state [23], learn using low-bandwidth sensors [24], or only perform collision avoidance [4], [25]. 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 an automated data labelling mechanism that does not depend on humans or SLAM algorithms.

The most similar works to our BADGR system are GCG [4], CAPs [5], and DeepDriving [26]. However, GCG only learned to avoid collisions, and CAPs and DeepDriving required human supervision in order to learn non-collision avoidance behaviors; in contrast, BADGR is able to reason about both geometric and non-geometric navigational without any human supervision in complex, real-world environments.

To the best of our knowledge, there are no prior learning-based systems that satisfy the same, similarly weak assumptions as BADGR.

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 autonomously gathers large amounts of off-policy data in real-world environments. Using this data, BADGR labels relevant events—such as collisions or bumpy terrain—in an automated manner, and adds these labelled events back into the dataset. BADGR then trains a predictive model that takes as input the current camera images and a future sequence of actions, which correspond to linear and angular velocity commands, and predicts the relevant future events. When deploying the trained BADGR system, the user specifies 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. BADGR is similar to model-based reinforcement learning methods, which learn dynamics models from state transition data; however, BADGR predicts future events instead of state, which enables it to scale to high-dimensional inputs, such as images.

In order to build an autonomous learning-based navigational system, BADGR uses *retrospective* self-supervision. 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. In order to learn using retrospective self-supervision, the robot can only learn about events it has experienced and that can be measured using the onboard sensors, and that experiencing these events, even undesirable events such as colliding, is acceptable. We believe this approach to autonomous self-supervised robot learning is 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.

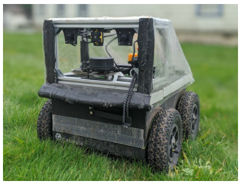


Fig. 2. The mobile robot.

The robot we use is the Clearpath Jackal, shown in Fig. 2. The Jackal measures 508 mm × 430 mm × 250 mm and weighs 17 kg, making it ideal for navigating in both urban and off-road environments. The Jackal is controlled by specifying the desired linear and angular velocity, 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 1.3 GB per minute streaming in from the sensors. Experiments were monitored remotely via a 4 G 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.

Fig. 3. While collecting data, the robot will periodically require a manual intervention to reset from catastrophic failures, though recovery is usually automatic.

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 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. 3), 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 4 Hz.

C. Autonomous 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 an automated fashion.

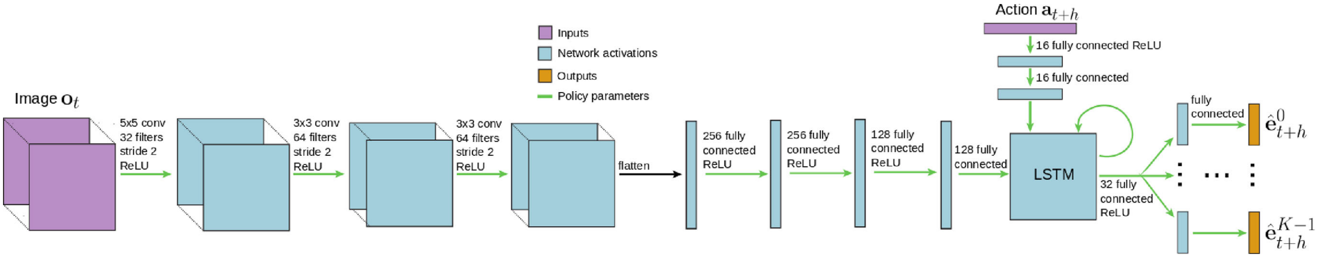


Fig. 4. Illustration of the deep neural network predictive model at the core of our learning-based navigation policy. The neural network takes as input the current RGB image and processes it with convolutional and fully connected layers to form the initial hidden state of a recurrent LSTM unit [27]. 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.

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.

Importantly, while these events are calculated using human-engineered functions, we still call this labelling scheme self-supervised because, once these simple labelling functions are created, all current and future data can be labelled with zero additional human effort. The labeling is automatic, and does not require any complex models, only simple evaluation with onboard sensors. Our approach stands in contrast to more standard methods for off-road navigation, which might require manual labeling of images to, for example, segment out traversable and impassable areas in an image [14].

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, similarly to [4], [5], 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 $f_\theta(\mathbf{o}_t, \mathbf{a}_{t:t+H}) \rightarrow \hat{\mathbf{e}}^{0:K}_{t:t+H}$, 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 $\mathbf{a}_{t:t+H} = (\mathbf{a}_t, \mathbf{a}_{t+1}, \ldots, \mathbf{a}_{t+H-1})$, and predicts K different future events $\hat{\mathbf{e}}^k_{t:t+H} = (\hat{\mathbf{e}}^k_t, \hat{\mathbf{e}}^k_{t+1}, \ldots, \hat{\mathbf{e}}^k_{t+H-1})\ \forall k \in \{0, \ldots, K-1\}$.

The model we learn is an image-based, action-conditioned predictive deep neural network, shown in Fig. 4. The input observation $\mathbf{o}_t$ is the current 128×96 RGB image, the inputted future actions $\mathbf{a}_{t+h} \in \mathbb{R}^2$ are the commanded linear and angular velocities, and the predicted navigation events are collision and bumpiness. The network first processes the input observation 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 $\hat{\mathbf{e}}^{0:K}_{t+h}$.

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

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

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 $R(\hat{\mathbf{e}}^{0:K}_{t:t+H})$ 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

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

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 [28]. 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 $\tilde{\mathbf{a}}^{0:N}_{0:H}$ are sampled that are time-correlated and centered around this running action sequence estimate

$$\epsilon^n_h \sim \mathcal{N}(0, \sigma \cdot \mathbf{I})\ \forall n \in \{0 \ldots N-1\}, h \in \{0 \ldots H-1\}$$
$$\tilde{\mathbf{a}}^n_h = \beta \cdot (\hat{\mathbf{a}}_{h+1} + \epsilon^n_h) + (1-\beta) \cdot \tilde{\mathbf{a}}^n_{h-1} \text{ where } \tilde{\mathbf{a}}_{h<0} = 0, \qquad (3)$$

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 $\tilde{R}^n = R(f_\theta(\mathbf{o}_t, \tilde{\mathbf{a}}^n_{0:H}))$. Given each action sequence and its corresponding reward, we update the running estimate of the optimal action sequence via a reward-weighted average

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

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, 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.

Algorithm 1: Training BADGR.

```
initialize dataset $\mathcal{D} \leftarrow \emptyset$
**while** not done collecting data **do**
  get current observation $\mathbf{o}_t$ from sensors
  get action $\mathbf{a}_t$ from data collection policy
  add $(\mathbf{o}_t, \mathbf{a}_t)$ to $\mathcal{D}$
  execute $\mathbf{a}_t$
  **if** in collision **then**
    execute reset maneuver
  **end if**
**end while**
**for each** $(\mathbf{o}_t, \mathbf{a}_t) \in \mathcal{D}$ **do**
  calculate event labels $\mathbf{e}_t^{0:K}$ using self-supervision
  add $\mathbf{e}_t^{0:K}$ to $\mathcal{D}$
**end for**
use $\mathcal{D}$ to train predictive model $f_\theta$ by minimizing
Eqn. 1
```

Algorithm 2: Deploying BADGR.

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

F. Algorithm Summary

During training (Alg. 1), 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 (Alg. 2), 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.

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]

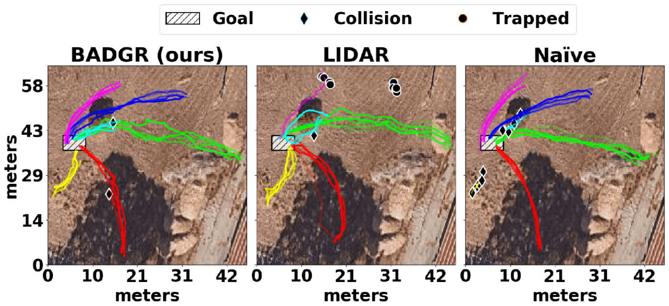
We performed our evaluation in a real-world outdoor environment consisting of both urban and off-road terrain. BADGR autonomously gathered 34 hours (98 km) of data in the urban terrain and 8 hours (23 km) in the off-road terrain. Although the amount of data gathered may seem significant, the total dataset consisted of 720 000 off-policy datapoints, which is smaller than currently used datasets in computer vision [29] and significantly smaller than the number of samples often used by deep reinforcement learning algorithms [30].

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 as such

$$\begin{aligned} R(\hat{\mathbf{e}}^{0:K}_{t:t+H}) = -\sum_{t'=t}^{t+H-1} & R^{\text{COLL}}(\hat{\mathbf{e}}^{0:K}_{t'}) \\ & + \alpha^{\text{POS}} \cdot R^{\text{POS}}(\hat{\mathbf{e}}^{0:K}_{t'}) + \alpha^{\text{BUM}} \cdot R^{\text{BUM}}(\hat{\mathbf{e}}^{0:K}_{t'}) \\ R^{\text{COLL}}(\hat{\mathbf{e}}^{0:K}_{t'}) =\ & \hat{\mathbf{e}}^{\text{COLL}}_{t'} \\ R^{\text{POS}}(\hat{\mathbf{e}}^{0:K}_{t'}) =\ & (1 - \hat{\mathbf{e}}^{coll}_{t'}) \cdot \frac{1}{\pi} \angle(\hat{\mathbf{e}}^{\text{POS}}_{t'}, \mathbf{p}^{\text{GOAL}}) + \hat{\mathbf{e}}^{coll}_{t'} \\ R^{\text{BUM}}(\hat{\mathbf{e}}^{0:K}_{t'}) =\ & (1 - \hat{\mathbf{e}}^{coll}_{t'}) \cdot \hat{\mathbf{e}}^{\text{BUM}}_{t'} + \hat{\mathbf{e}}^{coll}_{t'}, \end{aligned} \tag{5}$$

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

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



Method	Successfully Reached Goal
BADGR (ours)	**23/25 (92%)**
LIDAR	15/25 (60%)
Naïve	15/25 (60%)

Fig. 5. 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.

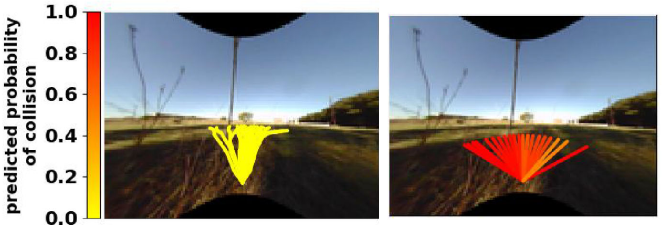
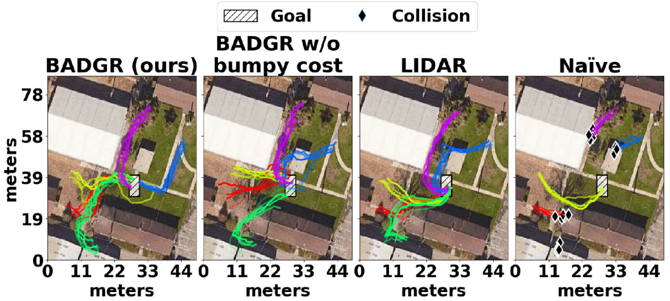


Fig. 6. Comparison of our BADGR policy (left) versus the LIDAR policy (right). Each image shows the candidate paths each policy considered during planning, and the color of each path indicates if the policy predicts the path will result in a collision. The LIDAR policy falsely predicts the paths driving 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× faster.



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. 7. 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.

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. This reward function took 3 hours to design and tune.

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. Each LIDAR range measurement is assigned a cost based on the associated range measurement plus the angle between the goal location the associated LIDAR range measurement angle; the minimum cost angle is then used as the setpoint for a PID controller that steers the robot towards that direction. 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. The LIDAR policy was specifically tuned to maximize performance in the deployment environments. We did not compare to other learning-based methods because, to our knowledge, there are no prior learning-based navigation systems that satisfy the same, similarly weak assumptions as BADGR.

Our evaluation consisted of 208 separate trials, in which the robot drove for a total of 2 hours and covered 5.8 km. 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.

Off-road environment. We first 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. 5 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. BADGR almost always succeeded in reaching the goal by avoiding collisions and getting stuck, while not falsely predicting that all grass was an obstacle.

Additionally, even when the LIDAR approach succeeded in reaching the goal, the path it took was sometimes suboptimal. Fig. 6 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× faster on average compared to the LIDAR policy.

Urban environment. Next, we evaluated all the approaches for the task of reaching a goal GPS location while avoiding collisions and bumpy terrain in an urban environment. Fig. 7

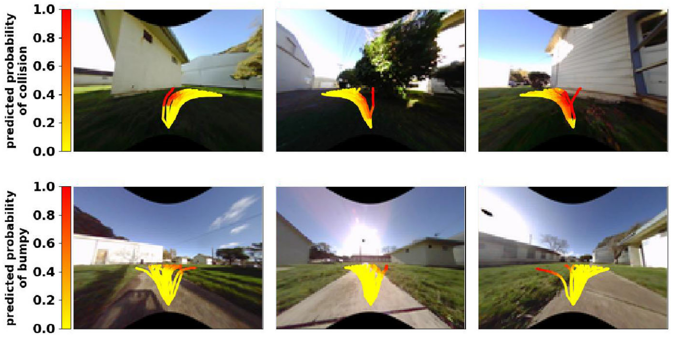
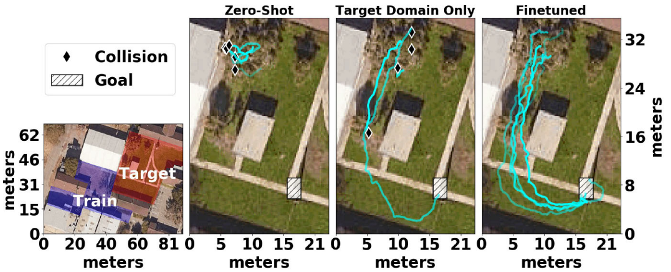

predicted probability of collision
1.0
0.8
0.6
0.4
0.2
0.0
predicted probability of bumpy
1.0
0.8
0.6
0.4
0.2
0.0

Fig. 8. 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	Zero-shot	Target Domain Only	Finetuned (ours)
Successfully reached goal	0/5	1/5	5/5

Fig. 9. Experimental demonstration of our BADGR approach improving as it gathers more experience.

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

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 Fig. 9. 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 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. 10 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 autonomously-labeled, off-policy data gathered in real-world environments, without any simulation or human supervision, and can improve as it gathers more data. Although BADGR can autonomously gather data with minimal human supervision, BADGR needs to experience an event—such as collision—in order to learn about that event. While this requirement may be acceptable for remote, off-road applications such as search and rescue, agriculture, mining, and landscaping, this requirement could be unacceptable for safety-critical applications such as

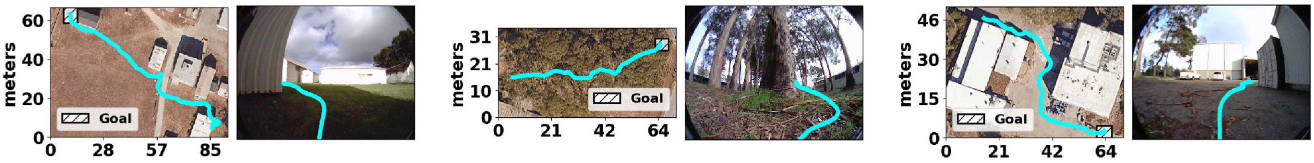


Fig. 10. Our BADGR policy can generalize to novel environments not seen in the training data. Each row shows the BADGR policy executing in a different novel environment. The first column shows the approximate path followed by the BADGR policy. The remaining columns show sampled images from the onboard camera while the robot is navigating, with the future path of the robot overlaid onto the image.

autonomous driving. Investigating methods which leverage human supervision that is already in place, such as safety driver disengagements, could enable learning for safety-critical applications. Also, while BADGR was able to navigate in static environments, future work building upon the recent advances in multi-agent path prediction could be integrated into BADGR's core predictive model to enable successful navigation in dynamic scenes. We believe that solving these and other challenges is crucial for real world mobile robot navigation, and that BADGR is a promising step towards this goal.

References

[1] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J.M Rendón-Mancha, "Visual simultaneous localization and mapping: A survey," *Artif. Intell. Rev.*, vol. 43, pp. 55–81, 2015.

[2] N. J. Nilsson, C. A. Rosen, B. Raphael, G. Forsen, L. Chaitin, and S. Wahlstrom, "Application of Intelligent Automata to Reconnaissance," Stanford Res. Inst.,Tech. Rep. Project 5953 Final Report, Dec. 1968. [Online]. Available: https://www.ai.sri.com/pubs/files/nilsson68-p5953-final.pdf

[3] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer, "Vision and navigation for the Carnegie-Mellon Navlab," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 10, no. 3, pp. 362–373, May 1988.

[4] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 5129–5136.

[5] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Composable action-conditioned predictors: Flexible off-policy learning for robot navigation," in *Proc. Conf. Robot Learn.*, 2018, *arXiv:1810.07167*.

[6] J. P. How, B. Behihke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *IEEE Control Syst. Mag.*, vol. 28, no. 2, pp. 51–64, Apr. 2008.

[7] S. Shen, N. Michael, and V. Kumar, "Autonomous multi-floor indoor navigation with a computationally constrained MAV," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 20–25.

[8] S. Thrun *et al.*, "Stanley: The robot that won the darpa grand challenge," *J. Field Robot.*, vol. 23, no. 9, pp. 661–692, 2006.

[9] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, "Deep ordinal regression network for monocular depth estimation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2002–2011.

[10] J.-R. Chang and Y.-S. Chen, "Pyramid stereo matching network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 5410–5418.

[11] R. Hadsell *et al.*, "Learning long-range vision for autonomous off-road driving," *J. Field Robot.*, vol. 26, no. 2, 120–144, 2009.

[12] C. Richter and N. Roy, "Safe visual navigation via deep learning and novelty detection," in *Robot.: Sci. Syst.*, 2017, doi: 10.15607/RSS.2017.XIII.064.

[13] L. Wellhausen, A. Dosovitskiy, R. Ranftl, K. Walas, C. Cadena, and M. Hutter, "Where should I walk? Predicting terrain properties from images via self-supervised learning," *IEEE Robot. Automat. Lett.*, vol. 4, no. 2, pp. 1509–1516, Apr. 2019.

[14] A. Valada, J. Vertens, A. Dhall, and W. Burgard, "Adapnet: Adaptive semantic segmentation in adverse environmental conditions," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 4644–4651.

[15] N. Hirose, A. Sadeghian, M. Vázquez, P. Goebel, and S. Savarese, "Gonet: A semi-supervised deep learning approach for traversability estimation," in *Proc. IROS IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 3044–3051.

[16] S. Ross *et al.*, "Learning monocular reactive uav control in cluttered natural environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 1765–1772.

[17] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 4693–4700.

[18] J. Bruce, N. Sünderhauf, P. Mirowski, R. Hadsell, and M. Milford, "Learning deployable navigation policies at kilometer scale from a single traversal," in *Proc. Conf. Robot Learn.*, 2018, *arXiv:1807.05211*.

[19] X. Meng, N. Ratliff, Y. Xiang, and D. Fox, "Neural autonomous navigation with riemannian motion policy," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 8860–8866.

[20] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors End-to-End with autorl," *IEEE Robot. Automat. Lett.*, vol. 4, no. 2, pp. 2007–2014, Apr. 2019.

[21] A. Loquercio, A. I. Maqueda, C. R. del-Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," in *IEEE Robot. Automat. Lett.*, vol. 3, no. 2, pp. 1088–1095, Apr. 2018.

[22] D. Gandhi, L. Pinto, and A. Gupta, "Learning to fly by crashing," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 3948–3955.

[23] M. Riedmiller, M. Montemerlo, and H. Dahlkamp, "Learning to drive a real car in 20 minutes," in *FBIT Front. Convergence Biosci. Inf. Technol.*, 2007, pp. 645–650.

[24] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra, "Benchmarking reinforcement learning algorithms on real-world robots," in *Proc. Conf. Robot Learn.*, 2018, pp. 561–591.

[25] A. Kendall *et al.*, "Learning to drive in a day," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 8248–8254.

[26] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proc. Int. Conf. Comput. Vis. Workshop*, 2015, pp. 2722–2730.

[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[28] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar, "Deep dynamics models for learning dexterous manipulation," in *Proc. Conf. Robot Learn.*, 2019, pp. 1101–1112.

[29] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[30] M. Hessel *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2018, *arXiv:1710.02298*.