Embodied Visual Navigation with Automatic Curriculum Learning in Real Environments

Steven D. Morad†, Roberto Mecca†, Rudra P.K. Poudel†, Stephan Liwicki†, and Roberto Cipolla†,§

Abstract—We present NavACL, a method of automatic curriculum learning tailored to the navigation task. NavACL is simple to train and efficiently selects relevant tasks using geometric features. In our experiments, deep reinforcement learning agents trained using NavACL in collision-free environments significantly outperform state-of-the-art agents trained with uniform sampling – the current standard. Furthermore, our agents are able to navigate through unknown cluttered indoor environments to semantically-specified targets using only RGB images. Collision avoidance policies and frozen feature networks support transfer to unseen real-world environments, without any modification or retraining requirements. We evaluate our policies in simulation, and in the real world on a ground robot and a quadrotor drone. Videos of real-world results are available in the supplementary material.

Index Terms—Visual-based navigation, reinforcement learning, autonomous agents

I. INTRODUCTION

Navigation forms a core challenge in embodied artificial intelligence (embodied AI) [1], [2]. Typical tasks involve point [3]–[5], object [3], [6], [7] or area-driven [8], [9] navigation in synthetic and real environments [10]–[14]. In our work, we focus on semantic object-driven navigation in unknown indoor scenes. Since semantic object-driven navigation uses object-class labels alone (not specific instances), agents can directly be deployed to novel environments and duties; suitable for post disaster recovery robots or embodied assistance technology with a wide range of task scenarios.

Before the embodied AI renaissance, approaches such as active vision [15] and active visual simultaneous localization and mapping (active VSLAM) [16] were popular methods for building autonomous agents. They combined classical motion planning [17], [18] with non-learned exploration policies such as frontier expansion [19] to direct the agent. Active VSLAM and active vision work well in ideal circumstances, but are brittle and lack generalization ability in real world situations.

Deep reinforcement learning (DRL) gained traction in the landmark paper [20], where DRL agents outperformed humans – all be it in relatively simple arcade games. Since then, the scope of DRL has expanded to real-world applications. In robotics and navigation, DRL shows promise as an alternative to classical models due to its surprising robustness and ability to generalize to real-world uncertainties. [6] trained visual navigation agents in a video-game maze, showing that over time, DRL agents can memorize the layout of a maze from vision alone. Since then, there has been an explosion of DRL-based visual navigation, fuelled by the abundance of indoor photo-realistic simulators and datasets [10]–[14]. Evidence suggests DRL outperforms traditional methods in such cluttered, realistic indoor environments [5].

Recent pushes in indoor visual navigation with DRL have focused on point-driven navigation (e.g. [3]–[5]) – finding the most efficient path given visual observations, noise-free agent coordinates, and the goal coordinates. In 2019, [21] produced near-perfect visual point navigation agents by training on a large set of experience over six GPU-months. However, the requirements of point-driven navigation are less practical in real-world settings, as prior setup of an indoor localization system is required. Nevertheless, point navigation is an important stepping stone towards more natural and difficult forms of indoor navigation. Specifically, [3] reformulates a point navigation agent to find an object using visual cues alone. While promising, retraining is required for new target objects. In alternative approaches an image of the goal position is provided for targeting, which opens the door to generalizing to multiple and novel object types [8], [22]. In particular, these methods store representations of the scene in memory, and match target observations to memorized representations. Therein lies a caveat – scene memorization requires retraining a portion of the policy on novel scenes before it can operate in them. Instead, we choose to use semantic object-driven navigation which generalizes to never-before-seen scenes and target object categories.

A. Contributions

In this work, we focus on indoor object-driven navigation using embodied AI and semantic targeting. We are interested in generalization to new environments and targets, including simulator-to-real-world (Sim2Real) generalization, without any retraining.

Navigating to targets specified by semantic label can generalize across multiple targets and to unknown environments without data refinement or retraining. We call this generalization ability about instances and classes zero-shot semantic navigation. We emphasize, by leveraging large segmentation datasets like COCO [23], we can use zero-shot semantic navigation to navigate to object instances, and even object classes, never seen before in training simulations.

Furthermore, we try to bridge the Sim2Real generalization gap. Note, while DRL visual navigation has proven itself time
and time again in simulation, performance rarely transfers to the real world. One challenge in Sim2Real transfer is overfitting to simulator-rendered images [24]. Using frozen feature encoders trained on real images, [3] shows compelling generalization ability across multiple simulators. In our work, we demonstrate frozen feature networks and collision avoidance help bridge the Sim2Real gap by showcasing our policies on real robots in real environments. Another issue present in almost every navigation simulator is collision modeling exploitation [1]. Agents drive into a wall at an angle and slide along it, covering the perimeter of a simulated building (Fig. 1b). [25] demonstrates collision-avoidance policies trained entirely in simulation can transfer to the real world, but stop short of investigating longer-term navigation policies.

Enforcing collision-free paths for agents results in increased reward sparsity, making training more difficult with state-of-the-art navigation tools (Fig. 1a). We mitigate this elevated sparsity using automatic curriculum learning. The essence of curriculum learning is selecting and ordering training data in a way that produces desirable characteristics in the learner, such as generality, accuracy, and sample efficiency. Automatic curriculum learning (ACL) is the process of generating this curriculum without human in the loop. Curriculum for neural networks was proposed by [26], and [27] affords a thorough overview of ACL applied to DRL. For navigation, tasks can be represented using low-dimensional Cartesian start and goal states. Some ACL methods that produce tasks of this form are asymmetric self play [28] and GoalGAN [29]. Asymmetric self play requires collecting distinct episodes for two separate policies, which is computationally expensive using 3D simulators. GoalGAN trades performance for generality. It can generate tasks for arbitrary problems, but uses a generative adversarial network (GAN) which is notoriously unstable and difficult to train. Instead, we trade generality for efficiency and propose a simple classification-based ACL method termed NavACL specifically for navigation.

In summary, we cast the visual navigation problem setup as follows:

i. the agent’s observations consist of RGB images from an agent-mounted camera and the semantic label of the target (e.g. “football”, “vase”),
ii. the agent’s actions consist of discrete, position-based motion primitives (i.e. move forward, turn left or right), without explicit loop closure outside of said primitives,
iii. upon reaching the target, collision with the environment, or exceeding a preset time limit, the episode ends and contribute:

i. a simple and efficient method to automatically generate curriculum for visual navigation agents,
ii. zero-shot semantic navigation – finding objects and object classes never seen during training,
iii. a collision-free navigation policy for complex, unseen environments that bridges the Sim2Real gap without any sort of retraining

II. APPROACH

A. NavACL

Motivated by evidence that intermediate difficulty tasks provide more learning signal for policy improvement than random tasks [27], [29] and that replaying easy tasks alleviates catastrophic forgetting [30]–[32], we formulate our ACL method, termed NavACL. NavACL filters down uniform random tasks to those that provide the most learning signal to the agent using predicted task success, described below. Since our navigation problem has well-defined termination scenarios (agent reached the goal or not), we use binary task success as the signal metric. Let task \( h = (s_0, s_g) \), with agent start position \( s_0 \) and goal position \( s_g \). \( f^*_\pi(h) \) denotes the probability of navigation policy \( \pi \) solving task \( h \), zero
Adaptive filtering

Now that we can estimate the difficulty (GOID) [29], selecting tasks bounded between two success thresholds \( \mu \) and \( \sigma \) using a fully-connected deep neural network we call \( f_\pi \). Before each forward pass, \( f_\pi \) preprocesses \( h \) into geometric properties (Tab. I), allowing \( f_\pi \) to generalize across scenes. \( f_\pi \) is updated alongside \( \pi \) in the training loop (Alg. 1,2). We define task difficulty as the complement of the estimated success probability, \( 1 - f_\pi(h) \).

In contrast to [29] that formulates scenario generation with GANs for general frameworks, we optimize NavACL to generate scenarios efficiently for the navigation task using simple log loss.

**Adaptive filtering** Now that we can estimate the difficulty of tasks, which tasks should we feed the agent? In one implementation, we produce goals of intermediate difficulty (GOID) [29], selecting tasks bounded between two success thresholds. This ensures we never select tasks that are too easy or too hard. However, GOID does not explicitly deal with catastrophic forgetting of easy tasks. Furthermore, the bounds do not change as the agent improves and task distribution shifts (Fig. 3). Instead, we provide a mixture of task types, where certain tasks adapt to the learner. Easy tasks provide adequate learning signal early in the training process and prevent catastrophic forgetting. Frontier tasks teach the agent to solve new tasks at its current ability. Uniformly sampled random tasks inject entropy and prevent the learner from overfitting to specific task types. Initially, easy and frontier tasks form the majority of the task mixture. The mixture decays into random sampling as the learning agent learns to generalize.

We draw many random tasks and estimate their difficulty using \( f_\pi \), producing a difficulty estimate across the task space. We fit a normal distribution \( \mu_f, \sigma_f \) to this distribution (Alg. 1). \( \mu_f, \sigma_f \) form an adaptive boundary in task space, partitioning it into easy and hard regions, predicated on policy \( \pi \). In particular, task \( h \) is considered an easy task if \( f_\pi(h) > \mu_f + \beta \sigma_f \) and a frontier task if \( \mu_f - \gamma \sigma_f < f_\pi(h) < \mu_f + \gamma \sigma_f \), where \( \beta, \gamma \) are hyperparameters. In other words, task difficulty is relative to the current ability of the agent – if we expect \( \pi \) to do better on task \( h \) than an average task, it is easy. If \( h \) is near the difficulty of the average task, straddling the adaptive boundary, we call it a frontier task (Alg. 3). Intuitively, this should provide a more conservative mixture of tasks than pure random sampling, promoting stable learning in difficult environments. The full algorithm is detailed in Alg. 1-3.

![Algorithm 1: Training loop with \( f_\pi \) update](image)

**Algorithm 1:** Training loop with \( f_\pi \) update

```python
input : \emptyset
output : \pi
\pi, f_\pi, \mu_f, \sigma_f \leftarrow \text{Init}();
for i \leftarrow 0 \text{ to numEpochs} do
\quad tasks, successes, states, actions, rewards \leftarrow \text{Rollouts}(\pi, f_\pi, \mu_f, \sigma_f);
\quad \pi \leftarrow \text{PPO}(\pi, states, actions, rewards);
\quad f_\pi \leftarrow \text{Train}(f_\pi, tasks, successes);
\quad randomTasks \leftarrow \text{GetRandomTasks}();
\quad \mu_f, \sigma_f \leftarrow \text{FitNormal}(f_\pi, randomTasks);
return \pi;
```

**Algorithm 2:** Rollouts

```python
input : \pi, f_\pi, \mu_f, \sigma_f;
output : rollouts
for i \leftarrow 0 \text{ to batchSize} do
\quad task \leftarrow \text{GetDynamicTask}(f_\pi, \mu_f, \sigma_f);
\quad rollouts[i] \leftarrow \text{RunEpisode}(\pi, task);
return rollouts;
```

Fig. 2 presents a flowchart of our contributions, which includes NavACL, the reward function for collision-avoidance, and the frozen feature networks. We discuss each piece in the following subsections.

**B. Reward Shaping**

Our reward function provides negative rewards to discourage collision and intrinsic rewards for exploration and to encourage movement. We define it as:

\[
r(s) = \mathbb{1}_{\text{succ}} + \delta(-\mathbb{1}_{\text{coll}} + \mathbb{1}_{\text{expl}}) + 0.01d.
\]  

![Fig. 2. The agent training pipeline, and how our contributions fit within it. Observations from the environment are compressed into latent features before being passed to the policy network. NavACL trains on navigation episodes and serves tasks back to the simulator.](image)

**TABLE I**

| NAVACL GEOMETRIC PROPERTIES | Properties                                |
|-----------------------------|------------------------------------------|
| Geodesic Distance          | The shortest-path distance from \( s_0 \) to \( s_g \) |
| Path Complexity            | The ratio of euclidean distance to geodesic distance of \( s_0, s_g \) |
| Turn Angle                 | The angle between the starting orientation and \( s_0s_g \), represented as sine and cosine components |
| Agent/Goal Clearance       | Distance from \( s_0 \) and \( s_g \) respectively to the nearest obstacle |
| Agent/Goal Island          | Radius of the traversable area at \( s_0 \) and \( s_g \) respectively |
D. Semantic Target Network

The semantic target feature is produced using a Mask R-CNN with an FPN backbone trained on the COCO dataset [23], [35]. We introduce a small postprocessing layer that enables swapping target classes without retraining. Given an image, the Mask R-CNN predicts a binary mask $M$ for each object class, along with the prediction confidence. We extract the mask with target label $l$ and do scalar multiplication of the binary mask with the prediction confidence to get output $O$.

$$O(x, y) = P(M(x, y)_{label=l}).$$

We can change $l$ at runtime to search for different target classes. Pixels of $O$ still contain shape information on the target object (e.g. a ball mask will be round but a box mask will be square). To prevent the downstream policy from overfitting to one specific object shape, as well as reduce latent size, we apply a max-pool operation to $O$ which is then stacked along the other features into a latent representation, which is fed to the policy network.

III. MODEL DESCRIPTION

Our learner model consists of an actor-critic model with policy $\pi(s)$ and value function $V^\pi(s)$ optimized using proximal policy optimization (PPO) with clipping (Tab. II) [36], [37]. The policy network and value function take latent representations from the feature encoders as input, and produce an action and value estimate as output. The policy networks consists of feature-compression and memory sections. The feature-compression section compresses spatially-coherent latent features into a more compact representation using convolutional layers. Receiving features instead of full RGB images reduces time to train and the likelihood of overfitting to the simulator.

To keep the navigation problem Markovian, the state must contain information on where the agent has been, and if it has previously seen the target. The purpose of the memory section is to store this information. The memory section uses long short-term memory (LSTM) [38] cells to represent state in the partially-observable Markov decision process (POMDP) [39]. With this, we aim to reduce the likelihood of revisiting previously explored areas and to remember the target location if it leaves the view. Note, obstacle circumvention may lead to significant turns, losing sight of the target.

Algorithm 3: GetDynamicTask

```plaintext
input : Training timestep $t$; $f_e$, $\mu_f$, $\sigma_f$; Hyperparameters $\beta$, $\gamma$
output : Task $h$
taskType $\leftarrow$ GetTaskType($t$);
while true do
    $h$ $\leftarrow$ RandomTask($t$);
    switch taskType do
        case easy do
            if $f_e(h) > \mu_f + \beta \sigma_f$ then
                return $h$;
        case frontier do
            if $\mu_f - \gamma \sigma_f < f_e(h) < \mu_f + \gamma \sigma_f$ then
                return $h$;
        case random do
            return $h$;
    end
end
```

The binary indicator $I_{succ}$ is true upon reaching the target, and false otherwise. $I_{coll}$ is true upon collision and false otherwise. The hyperparameter $0 < \delta < 1$ controls the agent’s affinity for learning exploration and motor skills compared to target-seeking behavior. $I_{expl}$ is an intrinsic reward for exploration. We keep a buffer of past agent positions over an episode, and provide a reward if the current position of the agent is some distance from all previous positions. We find that without the intrinsic exploration term, the agent falls into a local maxima of spinning in place to avoid the negative reward from collisions, which is difficult to escape. $d$ is the distance traveled in the current step, expressing the prior that the agent should be trying to cover as large a search area as possible.

C. Frozen Feature Networks

Traditional visual DRL agents use an autoencoder to transform input RGB images into a latent representation, where the autoencoder is trained end-to-end with the policy network [33], [34]. End-to-end training can overfit the policy network to simulation artifacts, and hurt real-world transfer [24]. We use spatial autoencoders pretrained on real images [3] and freeze their weights to prevent overfitting to simulation renders during training. High-polygon meshes scanned by [10] and photorealistic renders provided by [5] produce detailed enough visualizations to work with encoders trained on real-world datasets. Freezing also speeds up policy convergence, as the gradient backpropagates through fewer layers.

Fig. 3. Visualization of $f_e(h)$ estimation of task space across two geometric properties, at various training epochs $E$. Over time, the task distribution shifts. Adaptive NavACL accounts for this shift.
Fig. 4. (a) Validation episode success rate over three trials on a single test-train environment. (b) As the policy improves over time, NavACL increases the distance from start to goal – ratcheting up the task difficulty.

### TABLE II

| Parameter                  | Value          |
|----------------------------|----------------|
| # of Minibatches           | 1              |
| Learning Rate              | 0.005          |
| Clipping Range ($\epsilon$) | 0.10           |
| Discount Factor ($\gamma$) | 0.99           |
| Value Function Coef. ($c_1$) | 0.5            |
| Entropy Coef. ($\beta$ or $c_2$) | 0.01        |
| Timesteps per Update       | 4000           |
| Rollout Workers            | 12             |
| Inner-Loop Epochs          | 4              |
| GAE $\lambda$              | 0.95           |

### IV. EXPERIMENTS

We present three experiments: an ablation study of NavACL, a simulation benchmark of our model on unseen environments and target objects, and a benchmark of our agent operating in the real world.

#### A. Evaluating NavACL

Our first experiment compares the impact of NavACL on visual navigation to GoalGAN as well as the current standard of uniform task sampling. We evaluate NavACL with GOID (NavACL-GOID) and with adaptive filtering (NavACL-Adaptive). We hold all policy parameters the same, and run three navigation trials of five million samples on the Cooperstown environment from the Gibson dataset [10]. Uniform sampling uses Habitat’s built-in task generator to generate tasks [5]. GoalGAN uses an intermediate difficulty value between $0.1$ and $0.9$, used in their MazeAnt navigation experiment. For NavACL-GOID, we filter uniformly random tasks using our $f_\pi$ framework, and target tasks with an intermediate difficulty value of $0.4 \leq f_\pi(h) \leq 0.6$. NavACL-Adaptive uses hyperparameters $\beta = 1$, $\gamma = 0.1$. Both NavACL variants significantly outperform uniform sampling as well as GoalGAN, with NavACL-Adaptive showing an improvement over NavACL-GOID (Fig. 4). Therefore, we use NavACL-Adaptive in remaining evaluations.

#### B. Evaluating Model Performance

Using our methodology, we train a policy over twenty million timesteps using the Habitat 2019 challenge split of the Gibson dataset. Policies are evaluated over ten trials of thirty episodes spread across three unseen test environments, held out from the Habitat split (Fig. 6). The test tasks are generated randomly using the same uniform sampling as the Habitat challenge datasets [5]. The target object is an 11cm radius football (soccer ball). All policies are limited to 150 timesteps, and all tasks have a maximum start to goal distance of 10m. We find increasing the number of timesteps beyond 150 results in little improvement. The action space consists of a rotation of $\pm 30^\circ$ and forward translation of 0.2m.

The random policy selects random actions to provide a lower bound on performance. The NavACL policy is trained using depth, reshading (de-texturing and re-lighting), and semantic features, along with NavACL and intrinsic rewards. NavACL Zero-Shot is identical to NavACL, but during testing we change the target from the ball to a large vase to evaluate zero-shot semantic generalization to unseen targets of different shapes and sizes. We recruit ten volunteers from varying backgrounds in order to establish an upper-bound on performance. The human policies are trained and tested just like the agent policies. The volunteers played the training set until they were comfortable with the controls, receiving the same RGB observations and action space as the agents. Once comfortable, the volunteers played through the same test set as the agents. We use the SPL metric defined by [40]

$$\frac{1}{N} \sum_{i=1}^{N} \frac{1}{\text{success}_{i}} \max \left( \frac{l_i}{p_i} \right)$$

where $l$ is the length of the agent’s path, $p$ is the length of the shortest path from start to goal, and $N$ is the number of episodes. Results are presented in Fig. 5 and Tab. III. Our agents are able to find semantically specified targets in simulation, performing drastically better than random. Agents perform slightly worse on unseen semantic classes, but still exhibit zero-shot semantic generalization capability. On average, humans are able to outperform the agents on unseen environments. However, humans can memorize test scenes during the first few episodes, giving them an advantage over agents during later episodes. On Cooperstown, agents are within one standard deviation of human-level performance.
in success rate, suggesting they outperform some humans in some cases. We found agents had trouble navigating to new spaces in larger, unseen environments. Agents did not have as much trouble when navigating in large, previously-seen environments. Memory for embodied visual agents is an active area of research [8], [41]–[43], and we expect leveraging these memory modules will improve performance in larger environments. Another limitation was model throughput – it took roughly a week to train twenty million timesteps on a GPU machine, and previous experiments were still showing policy improvement at sixty million timesteps. With memory improvements and distributed computing, we believe our models could approach human performance.

C. Sim2Real Transfer

We transfer our policy without modification to a Turtlebot3 wheeled robot (AGV) and a DJI Tello quadrotor (UAV). The AGV uses wheel encoders for closed-loop control for motion primitives (single actions), but does not estimate odometry across actions. We tested the AGV on seven tasks spanning three environments and three objects, one being an unseen object and one being from an unseen semantic class. We use wheel odometry to measure SPL for the AGV (Tab. IV). The AGV did not experience a single collision over the 29m it traveled during tests and was robust to actuator noise as well as wheel slip caused by terrain (hardwood, carpet, and rugs).

The UAV uses IR sensors to determine height and an IMU to obtain very noisy position estimates for motion primitives and hovering stability. We did not train a separate model for the UAV. We used the model trained with the AGV height and camera field of view, which drastically differ from the UAV (0.2m vs. ~0.75 – 1.5m and 68° vs. 47° respectively). Policies trained for the AGV seemed surprisingly effective on the UAV, suggesting greater model generalization than we anticipated. The UAV was able to fly in-between legs of a camera tripod, through doorways, and even around moving people on many occasions without collision. This surprising generalization performance implies it may be possible to train a single navigation model for use on diverse types of robots that implement similar motion primitives. We provide video results of both the AGV and the UAV in the supplementary.

![Fig. 5. (5a) Training and validation results of our model (5b) As the policy improves, NavACL produces harder tasks. At the beginning, shorter paths (geodesic distance) and narrow corridors (agent clearance) help guide the agent. As the policy improves, the agent navigates without corridor guidance, and even reaches goals near obstacles (goal clearance).](image)

![Fig. 6. Simulation test scenes, left to right: Cooperstown (40m²), Avonia (60m²), Hometown (61m²)](image)

![TABLE III](image)

| Policy          | Scene      | Success Rate | SPL |
|-----------------|------------|--------------|-----|
| Random          | All        | 0.03 0.02    | 0.03 0.03 |
| Cooperstown     | All        | 0.00 0.06    | 0.00 0.08 |
| Avonia          | All        | 0.01 0.01    | 0.01 0.01 |
| Hometown        | All        | 0.00 0.00    | 0.00 0.00 |
| NavACL Zero-Shot| All        | 0.36 0.21    | 0.17 0.10 |
| Cooperstown     | All        | 0.55 0.25    | 0.08 0.10 |
| Avonia          | All        | 0.43 0.20    | 0.02 0.02 |
| Hometown        | All        | 0.09 0.06    | 0.03 0.03 |
| NavACL          | All        | 0.42 0.24    | 0.09 0.10 |
| Cooperstown     | All        | 0.63 0.31    | 0.09 0.10 |
| Avonia          | All        | 0.50 0.21    | 0.04 0.05 |
| Hometown        | All        | 0.12 0.09    | 0.06 0.06 |
| Human           | All        | 0.79 0.56    | 0.15 0.10 |
| Cooperstown     | All        | 0.76 0.54    | 0.17 0.10 |
| Avonia          | All        | 0.89 0.63    | 0.13 0.13 |
| Hometown        | All        | 0.72 0.51    | 0.13 0.13 |

![TABLE IV](image)

| Scene | Target       | Task Dist. (m) | SPL |
|-------|--------------|----------------|-----|
| Office 1 | Football    | 2.95           | 0.92 |
| Office 1 | Football    | 3.09           | 0.53 |
| Office 1 | Football    | 2.92           | 0.42 |
| Office 1 | Football    | 1.67           | 0.18 |
| Office 2 | Football    | 9.10           | 0.65 |
| House   | Orange Football | 4.19 | 0.92 |
| House   | Vase         | 5.14           | 0.54 |
Fig. 7. (7a) The AGV navigating to the vase target in the house scene, with several never-before-seen obstacles littering the path to the target. Previous semantic targets (football and pink star ball) are present to emphasize zero-shot semantic navigation capability. The agent turned $360^\circ$ (1) to evaluate its options, then took the path between the desk and tent (2), adjusted the trajectory towards the wide-open area in front of the blue tent (3), and rotated $360^\circ$ (4). The agent explored the areas surrounding the bike, bookshelves, and the blue tent (5,6,7). Target detection occurred at (8), and the AGV made a beeline for the target (9,10). (7b) While flying down a hallway (1), the UAV notices an empty cubicle (2). It threads the needle, flying between the chair wheels and seat (3). After exploring the cubicle (4,5), it leaves and heads into the adjacent open office without collision (6).

material,\(^2\) and illustrations in Fig. 7.

V. CONCLUSIONS

We introduce NavACL and present two variants (NavACL-GOID) and (NavACL-Adaptive) for task generation in navigation. Both methods significantly improve upon uniform sampling (the current standard approach) as well as GoalGAN in sparse-reward settings. Combining NavACL with frozen feature networks and collision-free policies produces agents capable of zero-shot semantic navigation in both simulation and the real world.

A. Future Work

We found LSTMs had issues with generalization to new environments with the compute power available to us. Future work will focus on integrating more structured and efficient memory modules \(^8\)\(^-\)\(^{--}\)\(^\text{[8], [41]--[43]}\) into our learning pipeline.

The unexpected real-world generalization ability between mobility types warrants further investigation. Training an agent with an actuator abstraction layer allows transfer to disparate, never-before-seen robots. It may be prudent to invest computational resources in training a single model with abstract actuation that can be applied to drones, wheeled robots, walking robots, blimps, etc., rather than training each model individually.

We evaluated NavACL using on-policy reinforcement learning, but NavACL may be useful for selecting which episodes to replay when using off-policy methods. It may also prove useful in selecting and ordering training episodes for imitation learning.

\(^2\)Also available at https://www.youtube.com/playlist?list=PLkG_dDkoI9pPdOy7ec-sSuZ0pB9layC

REFERENCES

[1] A. Kadian, J. Truong, A. Gokaslan, A. Clegg, E. Wijmans, S. Lee, M. Savva, S. Chernova, and D. Batra, “Are we making real progress in simulated environments? measuring the sim2real gap in embodied visual navigation,” ArXiv preprint arXiv:1912.06321, 2019.

[2] F. Xia, W. B. Shen, C. Li, P. Kasimbeg, M. E. Tchapmi, A. Toshev, R. Martín-Martín, and S. Savarese, “Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments,” IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 713–720, 2020.

[3] A. Sax, B. Emi, A. R. Zamir, L. J. Guibas, S. Savarese, and J. Malik, “Mid-level visual representations improve generalization and sample efficiency for learning visuomotor policies.” 2018.

[4] D. S. Chaplot, D. Gandhi, S. Gupta, A. Gupta, and R. Salakhutdinov, “Learning to navigate using active neural slam,” ArXiv preprint arXiv:2004.05155, 2020.

[5] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, \textit{et al.}, “Habitat: A platform for embodied ai research,” in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 9339–9347.

[6] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, \textit{et al.}, “Learning to navigate in complex environments,” ArXiv preprint arXiv:1611.03673, 2016.

[7] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, “Deep reinforcement learning with successor features for navigation across similar environments,” in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2017, pp. 2371–2378.

[8] L. Mezghani, S. Sukhbaatar, A. Szlam, A. Joulin, and P. Bojanowski, “Learning to visually navigate in photorealistic environments without any supervision,” ArXiv preprint arXiv:2004.04954, 2020.
