# Learning Object-conditioned Exploration using Distributed Soft Actor Critic

**Ayzaan Wahid**
Robotics at Google
ayzaan@google.com

**Austin Stone**
Robotics at Google
austinstone@google.com

**Kevin Chen**
Stanford University
kchen92@stanford.edu

**Brian Ichter**
Robotics at Google
ichter@google.com

**Alexander Toshev**
Robotics at Google
toshev@google.com

**Abstract:** Object navigation is defined as navigating to an object of a given label in a complex, unexplored environment. In its general form, this problem poses several challenges for Robotics: semantic exploration of unknown environments in search of an object and low-level control. In this work we study object-guided exploration and low-level control, and present an end-to-end trained navigation policy achieving a success rate of 0.68 and SPL of 0.58 on unseen, visually complex scans of real homes. We propose a highly scalable implementation of an off-policy Reinforcement Learning algorithm, distributed Soft Actor Critic, which allows the system to utilize 98M experience steps in 24 hours on 8 GPUs. Our system learns to control a differential drive mobile base in simulation from a stack of high dimensional observations commonly used on robotic platforms. The learned policy is capable of object-guided exploratory behaviors and low-level control learned from pure experiences in realistic environments.

**Keywords:** Robotics, Reinforcement Learning, Embodied Vision

## 1 Introduction

Visual Navigation is a long-standing problem at the intersection of Robotics and Computer Vision. Much of the existing effort in the field has been focused on safely and efficiently navigating a robot to a geometrically specified goal, commonly known as point goal navigation. However, in many practical settings the goal location is unknown, such as in solving the problem: "Where are my keys?" This class of problems, in which the goal is specified semantically, is known as Semantic Visual Navigation. Tackling this problem involves overcoming numerous challenges, such as generating intelligent plans to efficiently explore the space and controlling the robot in a safe and efficient manner. The vision community has studied visual navigation using high-level discrete control with a known goal in an unknown environment, whereas the robotics community has investigated using low-level continuous control to reach a *known* goal in a *known* environment. We seek to combine these, and learn to navigate to a target of unknown location with low-level continuous control in an unknown environment.

In this work we focus on object-conditioned exploration and low-level control, for the purpose of navigating to objects in realistic simulated settings (see Fig. 1). We assume no prior knowledge of the environment and no localization information. Further, we aim towards a simulation setup which mimics many of the challenges present in real world robotics. We use environments based on scans of real homes [1] to provide maximum visual realism and complexity, and simulate the physics of a differential drive robot to address the challenges of low-level control. While we primarily tackle the *exploration* and *control* components of this problem, we show results that demonstrate the challenges of learning policies which can perform exploration, recognition, and control in search of an object. We leave real world deployment for future work, as this would require more work in sim-to-real transfer or training in the real, which are challenging due to the sim-to-real gap and limited ability to collect large amounts of data on a robot. However, the more realistic setup we provide in simulation is a step towards better real world performance compared to prior methods.

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

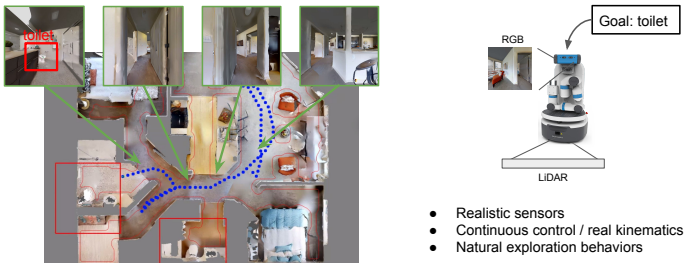
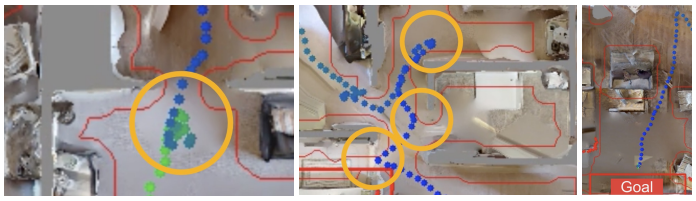



Figure 1: Navigating to an object requires object-conditioned exploration (e.g., efficiently exploring a space using contextual cues) and low level control (e.g., controlling a differential drive). We present a scalable distributed SAC system to learn an object navigation policy in realistic environments, addressing all of the above challenges.




(a) Circle (b) Peek (c) Beeline

Figure 2: (2a) The agent learns to move straight and periodically turn in place to gather information. (2b) The agent learns to "peek" into rooms and backtrack if the object is unlikely to be present. (2c) The agent learns to move directly to the goal once it is seen.

For the above setup of semantic navigation, we revisit end-to-end learned navigation policies. We introduce a distributed version of the Soft Actor Critic (SAC) algorithm [2] which allows us to train a policy from experience. SAC, as an off-policy reinforcement learning algorithm, is amenable to scalable parallelization, where we parallelize both the model optimization and the experience collection. This allows us to train a conventional Neural Net (ResNet50 + LSTM) as a policy to output continuous actions from rich, high dimensional, realistic robot observations. We also note that we specifically choose an off-policy algorithm, since these are more sample efficient than on-policy alternatives. Sample efficiency is important in the robotics domain, where data collection is expensive and maximal data re-use is highly desired. Using this system, we achieve a success rate of 0.68 and SPL of 0.58 for navigating to 9 different object types by training over 98 million experience steps per 24 hours on 8 GPUs.

We demonstrate that a single end-to-end model can learn several important skills in the realistic and visually rich simulation environments required for object navigation. First the agent learns natural exploration behaviors, such as turning in place when the object is not visible, backtracking when failing to find the goal object, and going straight to the target object when visible (Fig. 2). These behaviors are then conditioned on the type of object, so that the agent performs different exploration for different types of objects. It is important to note that these emerge from experience without any explicit supervision.

Finally, we demonstrate the challenges of using low-level control, and the utility of having a comprehensive set of sensors, e.g., range sensing, which allows for more collision free object navigation.

## 2 Related Work

The proposed work is related to different approaches towards visual navigation. Classical techniques for navigation typically start with map construction, followed by planning and execution of the planned trajectories. The mapping stage uses visual observations to simultaneously localize and construct maps, either topological or semantic and metric (see for an extensive review [3, 4] of SLAM).

**Learned Navigation System** In the last few years a large body of work on learned navigation policies has emerged [5, 6, 7, 8]. This body of work has laid the groundwork for learning-based navigation, where a sensors-to-control policy is learned from experience or demonstrations. Researchers

have studied a wide variety of navigation problems, such as metric navigation [9, 10, 11, 12], target-driven semantic navigation [7, 13, 14], and exploration [15, 12, 16, 17, 18, 19].

Most relevant to our problem setup, Yang et al. [13] train Graph Convolutional Networks for object navigation with discrete commands. Mousavian et al. [14] address a similar problem using learning from demonstrations.

More recently, we have seen two different developments which facilitate learning realistic navigation policies. First, reinforcement learning (RL) algorithms have become more stable and mature, e.g.,[20, 21]. Second, we have seen the development of navigation simulation environments, which simulate realistic visuals and physics at scale, e.g. [22, 23]. In this paper, we leverage these new simulations and learn a navigation policy which tackles both complex visuals and physics.

These advances have already been utilized, to a degree, for point navigation. Chaplot et al. [23] demonstrate a hierarchical learned navigation system for point navigation with very strong results on the Habitat challenge [23]. Sax et al. [10] apply PPO on top of mid-level image representations for strong results on the same benchmark. For the same problem, Wijmans et al. [11] demonstrate even stronger results using a distributed version of PPO.

While point navigation [23, 9] has made remarkable progress, its applicability to real-world settings remains limited. In most applications, the precise metric location of the target is unknown, which is the case in object navigation. This important difference introduces serious challenges as the agent has to efficiently plan and explore the unknown environment. Furthermore, real-world robots operate using continuous control rather than the commonly used discrete left/right/forward actions. These important details must be taken into account in the choice and design of the algorithm. In contrast to prior works, which have tackled some of these challenges individually (Table 1), we propose a method to handle them altogether.

**Motion Planning and Obstacle Avoidance** In this work we emphasize having a realistic setup, meaning continuous control. The main challenge is collision-free movement respecting the geometry of the robot and obstacles. Classical approaches include Artificial Potential Fields [24] or Dynamic Window Approach [25]. These require an explicit model of the environment, which we do not have at test time.

Deep Learning has found traction in Motion Planning research as well. In more detail, control policies have been trained using supervised learning [26] and DDPG [27] for point navigation. However, instead of point navigation, we achieve this in the context of semantic, long-range navigation.

## 3 Object-conditioned Exploration

Object Navigation (ObjectNav) is defined as the problem of navigating to an object, specified by a label, in unstructured and unknown environments [9]. In this work, we want to tackle object-conditioned exploration in its most challenging form:

**Realistic complex visuals:** the agent navigates in realistic environments. We propose to use GibsonEnv [1] as a set of scans of real homes.

**Realistic physics:** the agent has realistic embodiment. We suggest using a model of a robotic system – more precisely a differential drive with mounted RGB, depth, and 1-D LiDAR modeled after a Fetch robot.

**Semantics:** We use 9 object types to navigate to. The starting location can be up to 15m away, and as such the robot explores multiple rooms before getting to the target.

The problem of object navigation brings together a range of skills – visual recognition, object-conditioned exploration of unknown environments, and low-level control. As such, in this study we mostly assume that the goal object detection is given by an oracle.

## 4 Learning of General Navigation Policy

Consider the case where the robot starts in the dining room and is looking for an oven. The robot may perform rotational motion to explore the environment, spot the kitchen bar counter in the distance, and use this cue to move towards it, hypothesizing that there is a kitchen area with the goal behind the counter. This type of reasoning process may benefit from end-to-end neural network policies which can tightly integrate these skills. Furthermore, these policies may benefit from learning via

| | No Goal Vector | Cont. Control | Visual Realism | Physics Realism | Obj.-cond. exploration |
|---|---|---|---|---|---|
| Zhu et al. [7] | ✓ | | | | |
| Gupta et al. [6] | | | ✓ | | ✓ |
| Mousavian et al. [14] | ✓ | | ✓ | | ✓ |
| Sax et al. [10] | ✓ | | ✓ | ✓ | |
| Yang et al. [28] | ✓ | | | | ✓ |
| Chaplot et al. [12] | ✓ | | ✓ | | |
| Wijmans et al. [11] | | | ✓ | | |
| Ours | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 1: Comparison with prior learning-based visual navigation approaches.

experience in realistic environments, since explicit supervision for the above reasoning is hard to define.

### 4.1 Distributed Soft Actor-Critic

As commonplace, we model the problem as a Partially Observable Markov Decision Process (POMDP) [5] [6]. In particular, we apply Soft Actor-Critic to learn a policy for the MDP and use an LSTM to handle the partial observability (see [1], [21], and Supplement). As SAC is an off-policy RL algorithm, it lends itself well to a distributed and parallel application (see Fig. 3).

**Collect** The data collection can be parallelized over multiple workers, on which we run our policy in the simulation engine. In our implementation we use a simulator that renders GibsonEnv worlds [1] and simulates a LoCoBot differential drive base [29]. Note that most of the running time is spent in stepping through the environment. Our simulator steps at 0.6 - 1.2 secs per step on CPU. Many other navigation simulation engines have GPU specific renderers [1, 23] to speed up graphic rendering. In our design we opted for running 100 CPUs in parallel.

**Training** We use a rack of 8 GPUs to perform synchronous SGD. We use the SAC implementation from the Tensorflow Agents library within TensorFlow [30].

**Experience Replay** To mix experiences from multiple collect workers, we use an Experience Replay Buffer (ERB). The ERB receives experiences generated by all collect jobs and sends them to the training worker. In more detail, each collect worker sends an unroll (of a length of up to 100 steps in our implementation) to the ERB . The ERB sends batches of samples (of length 20 each) to the workers. Each sample is generated as a random crop from an unroll. In this way, the ERB guarantees uniform sampling of the data. Note that we use a distributed implementation of ERB due to its large memory requirements.

**Performance Analysis** The computational bottleneck in the above setup is the training, as the training time scales approximately logarithmically w. r. t. the number of workers [31], while the collect process scales linearly. Therefore, it is important to reduce additional computational burden on the training workers. We do this by moving experience collect on the separate collect workers. In our experiments we can process 98 million experience steps per 8 GPUs per day. The policy converges in 2.5 days using 250M experience steps.

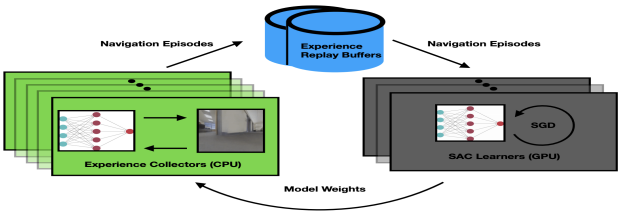



Figure 3: Diagram of distributed Soft Actor Critic architecture.

### 4.2 Reward Definition for Object Navigation

The only aspect specific to object navigation in the above POMDP is the reward definition used at train time. We deliberately opt for a simple reward, which provides clear supervision for clearly desired and undesired behaviors, such as reaching goal or colliding respectively. At the same time, we provide no supervision for behaviors that are hard to quantify, such as efficient exploration behaviors or effective low level controls.

In more detail, the reward consists of four components:

$$R(s,a) = \text{collision}(s) + \text{step}(s) + \text{progress}(s,a) + \text{success}(s)$$

where $\text{collision}(s) = -0.05$ if $s$ denotes a state of collision, otherwise 0. $\text{step}(s) = -0.01$ which is used to encourage efficiency (penalizes long episodes). The third term measures the distance progress towards the goal, without taking into account rotations. More precisely, if $s'$ denotes the state after taking action $a$ at $s$ and $d(s)$ denotes geodesic distance to goal, then $\text{progress}(s,a) = 0.1(d(s) - d(s'))$. The final term, $\text{success}(s) = 1.0$ if state $s$ denotes success of reaching the goal object (see Sec. 5 for detailed definition for object navigation).

### 4.3 Observations, Actions, and Network Architecture

**Observations** Our agent runs with three different observation modalities: forward facing RGB image, ground level LiDAR, and depth image, as commonly found on robots. The `RGB` observation is a frontal camera view from the agent's perspective at 0.88m height, perpendicular to the ground plane with a horizontal FOV of 79 degrees. This setup mimics the setup of a LoCoBot with an elevated Intel RealSense Camera. It gives the agent the opportunity to see the world directly and use semantic cues. However, the camera at this height has a blind spot immediately in front of the robot base, which makes obstacle avoidance challenging. In order to perceive geometry, we use a depth image called `DEPTH` of the same size and extrinsics as `RGB`.

Lastly, the `LiDAR` observation is a 1-dimensional array of 222 agent-relative (x, y, z) Cartesian coordinates, which indicate the ray-distance to occupied space in the environment. It is mounted at the base of the robot, has a wider field of view than `DEPTH` at 220 degrees, and is motivated by the blind spot of `RGB`. It is used by real world robotic systems such as Fetch, and can be modelled on a LoCoBot by using a base mounted Hokyuo Rangefinder.

**Auxiliary Observations** In addition to the above observations, we feed into our policy two other auxiliary observations shown to benefit learning: the action from the previous time step, a binary collision bit designating whether the action at the previous step was successful. These observations help the agent learn collision avoidance, and empirically leads to faster convergence.

**Oracle Observations** We conduct experiments with auxiliary oracle observation to understand the importance of semantics. This oracle observation is defined as a binary mask of only the goal object, called `Det`. This observation helps break the problem into two distinct phases: exploration (when the object is not visible) and direct navigation to the object while avoiding obstacles (once it becomes visible).

**Action Space** The action space of the agent is motivated by commonly used differential drive robots such as LoCoBot or Fetch. We use twist commands, an angular and linear velocity.

**Architecture** We use a standard NN architecture, whereby the different observations get embedded into vectors. These get concatenated, and subsequently fed into an LSTM, which outputs actions and critic values. Further details regarding the exact architecture and training details can be found in supplementary material.

## 5 Empirical Analysis

### 5.1 Setup

We evaluate our system in an environment which simulates both the visuals and physics of the real world. We use the worlds from Gibson [1], which consists of high quality scans of real homes, and simulate a differential drive robot in a custom environment for performance reasons. We use a subset of the worlds for which object segmentations and labels are available, as provided in [32]. This dataset comes with 25 homes for training, 5 for validation, and 5 for testing. Note that many of the homes have multiple floors, which we treat as disjoint spaces. For example, the test set has 12 floors total. We test navigating to 9 objects: 'bed', 'chair', 'microwave', 'refrigerator', 'table', 'toilet', 'oven', 'tv', 'sofa'.

We model the kinematics of a commonly used differential drive robot, i.e., LoCoBot or Fetch. Its base is a 0.18m radius circle and its camera height is 0.88m. To evaluate each approach we compute Success weighted by Path Length (SPL) [9].

The success criterion, needed to compute $S_i$, is motivated by the idea that the robot should be able to physically reach the object. In particular, we expect that the agent is looking at the object and

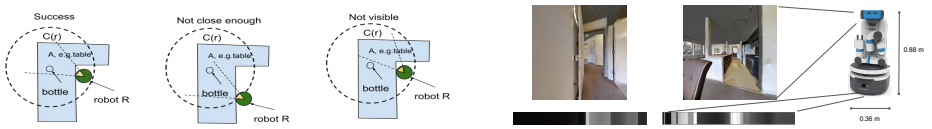



Figure 4: Left: A robot is considered to have reached the target bottle placed on corner desk if it is looking at the bottle and is within $1m$ from bottle surface. Right: `RGB` (top) and `LiDAR` observations (bottom) from the environment. Note that the camera is at a height of $0.88$m with 79 degrees FOV which makes collision detection challenging. Using range sensing close to the base makes the control problem easier, e. g. dark means close and it demonstrates that the robot is quite close to the wall in the left image and the chair in the right image.

that the object is within reaching distance (see Fig. 4, Left). More precisely, $S_i = 1$ if agent is at a navigable pose within $1m$ from the object 3D bounding box and the object is visible (within the central $20\%$ of the `RGB` observation), otherwise $S_i = 0$.

### 5.2 Experiments

In this section we present an empirical analysis of the learned policy. To focus on understanding exploration and low-level control, we assume that we are given an oracle goal object detection in most of the analysis below (see Sec. 4.3).

We report results over $120$ episodes, $10$ episodes for each of the $12$ house floors. Each episode runs for up to $500$ steps. The robot is spawned randomly within 15m of the goal at a random orientation. Given that we use scans of standard homes (rooms are of size approx 5 x 5 m), this distance is large enough that the goal object is usually not visible from the starting point. This is also aided by the small camera FOV of 79 degrees. In our test set, the object is not visible at start in $90\%$ of the episodes.

Using modalities `RGB+LiDAR+Det` our navigation policy achieves an SPL of $0.58$ and success rate of $0.68$ on Gibson Test environments (see Table 2, left). In order to provide context to this result, we compare with several baselines. Because the environment is only partially observable, it is not reasonable for any agent to achieve 1.0 SPL. Thus, to get a sensible upper bound on performance, we ask $12$ human raters to perform navigation using `RGB` observation by virtually traversing $12$ environments (see appendix for details). The average SPL for our raters is $0.80$ with a success rate of $0.86$. The challenges raters face are: not being able to identify the correct direction, which leads to a suboptimal path; running out of time; getting stuck in narrow spaces, where the head camera view is not allowing for collision free traversals.

We compare against two scripted baselines with varying degrees of exploration intelligence to quantify how much learning exploration helps. For all of these baselines, the agent goes straight and collision-free to the goal whenever the object becomes visible.

**Roomba:** the policy moves straight until collision. When it collides, it picks a random rotation direction, performs rotations until collision free, and repeats.

**Topo Graph Traversal (TGT):** the policy follows a topological graph of the environment, defined as the skeleton of the floorplan [33]. We perform a depth-first traversal of this graph. This guarantees an efficient coverage of the space, however the space is explored without taking into account the goal label. This baseline is by definition collision free.

Both scripted baselines achieve similar SPL, however Roomba has a lower success rate as it struggles exploring far-away spaces. TGT can do this, however often times the path taken is not very efficient. Our navigation policy achieves almost double the SPL. Note that humans are not perfect either.

**Observation Modalities** Oftentimes it is assumed that `RGB` and `Depth` are sufficient observation modalities. However, on real world robotic systems a stronger emphasis is put on range sensing or wider field of view as they usually give complementary information. For example, when mounted on top of a robot, both `RGB` and `Depth` have a blind spot right in front of the robot (see Fig. 4, Right). This blind spot makes it harder to maintain a collision free path. Therefore, in this work we mount a `LiDAR` sensor to the robot base and investigate its utility.

We present results over combinations of modalities in Table 2, right. In all cases we use `Det`, so that the policy can focus on exploration and continuous control. We see that adding depth sensing leads

| | SPL | SR |
|---|---|---|
| Human Navigator | 0.80 | 0.86 |
| RGB + LiDAR + Det | 0.58 | 0.68 |
| TGT | 0.32 | 0.50 |
| Roomba | 0.30 | 0.35 |

| Modalities | stuck at collision | | sliding at collision | |
|---|---|---|---|---|
| | SPL | SR | SPL | SR |
| RGB + Det | 0.14 | 0.14 | 0.28 | 0.30 |
| Depth + Det | 0.22 | 0.25 | 0.39 | 0.45 |
| RGB + Depth + Det | 0.26 | 0.30 | 0.46 | 0.55 |
| RGB + LiDAR + Det | 0.58 | 0.68 | 0.68 | 0.79 |
| RGBD + LiDAR + Det | 0.57 | 0.66 | 0.67 | 0.75 |

Table 2: Success weighted by Path Length (SPL) and object reached Success Rate(SR). Left: comparison of our approach with scripted and human baselines. Right: different combinations of observation modalities. We have two behaviors of the simulator: 'stuck', proper physics dynamics whereby the agents usually gets stuck at collision, and 'sliding', where the agent slides upon collision as used in [23].

to a boost, and in particular LiDAR has the larger boost in performance. We conjecture that this is due to the ability to avoid collisions, in which the agent otherwise gets stuck.

**Low level continuous control** We use a simulator for our differential drive robot model. Empirically, the most challenging aspect in that respect is collisions, especially in narrow passages, which lead to a halt.

In order to assess how much of challenge this presents, we evaluate our policies in a modified version of our simulator where, at collision, the agent is allowed to slide along the obstacle, as used in [23]. The results, shown in Table 2, indicate a $0.10$ to $0.20$ boost in SPL across all policies in this more forgiving regime. This suggests that properly modeling dynamics, especially in the physically realistic continuous control case, is of high importance.

**Starting Distance to Goal** As the starting distance to the goal increases, the problems become harder (see Fig. 5). It is noteworthy that LiDAR-based policies tend to perform better over larger distances, which is due to better collision avoidance (usually narrow doorways in our data).

**Visual Recognition** In most of the experiments we assume that an oracle gives us goal object recognition via Det, in order to focus on exploration. To demonstrate that our system is capable of learning recognition in addition to exploration and continuous control, we train our policy without Det (see Table 3). Det helps across the board, in particular when the object is far away and small as a result. Some objects are large and centrally placed, e.g. sofa and bed. These are easily captured in RGB and thus better navigable. Others, such as toilet, are in small enclosed spaces, so that they are rarely immediately visible and thus are more challenging.

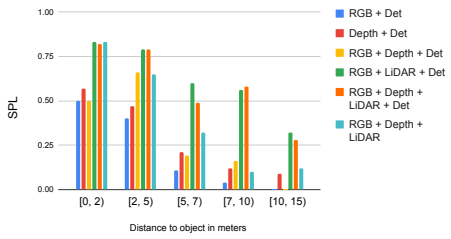
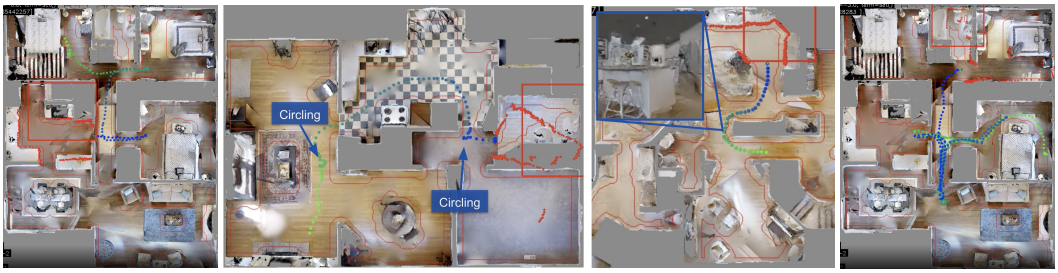



Figure 5: SPL for different starting distances to goal and all modality combinations.

| object | RGBDL | RGBDLDet |
|---|---|---|
| bed | 0.40 | 0.73 |
| chair | 0.41 | 0.53 |
| oven | 0.41 | 0.39 |
| fridge | 0.65 | 0.48 |
| sofa | 0.36 | 0.77 |
| toilet | 0.04 | 0.28 |
| mean | 0.28 | 0.52 |

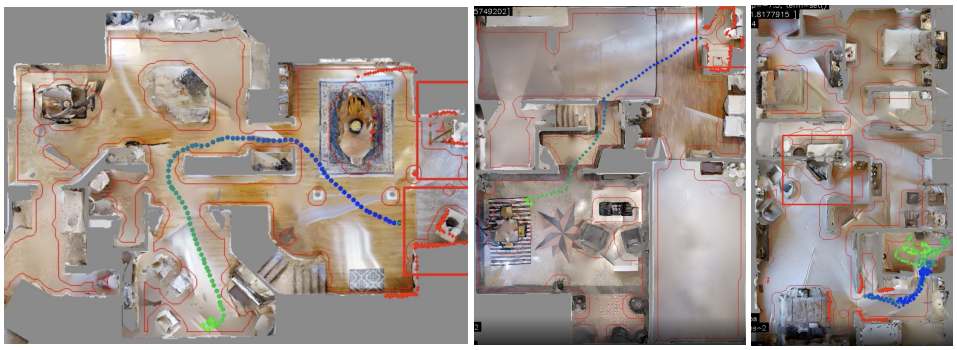
Table 3: SPL for our model with and without Det, also split per object.

**Emergent Behaviors** To solve object navigation, several exploration and navigation behaviors must emerge. Towards exploration, an agent must be able to gather information about the environment efficiently and comprehensively. We note that several of these behaviors naturally emerge through the learning process, as shown in Fig. 6. The agent learns not only to only cover the space, but to leverage the minimal required amount of exploration, e.g., the agent *peeks* into rooms only the necessary amount to see if the object is present. The agent also learns to efficiently gather information in regions that must be traversed, e.g., by repeatedly moving in straight lines, followed by *circling* to get a 360° view. Finally, the agent learns to leverage semantic *context* of the surrounding scene to make decisions even when the object is not in view, e.g., recognizing that a kitchen likely has an oven or that a bathroom has a toilet. We do however find suboptimal behaviors emerging, such as *revisiting* regions or heavily exploring single regions. This is likely a result of the limited memory of LSTMs and indicates a need for further research into long-term memory for navigation.




(a) Peeking (b) Circling (c) Context (d) Fail: Revisits

Figure 6: For each figure, the trajectory is colored from green to blue (start to end), and the goal location(s) are indicated via a red box. (6a) The agent efficiently explores the environment, exploring rooms only the amount necessary to confirm the lack of the the target object. (6b) The agent alternates between directly moving through the environment, followed by circling behavior to gather information. (6c) The agent leverages context of the scene. Even without seeing the oven, the agent recognizes the kitchen and approaches the area. (6d) At times the agent doubles back and revisits previously explored regions, indicating a need for longer-term memory.



(a) Beelining (b) Narrow Passages (c) Fail: Stuck

Figure 7: (7a) The agent first establishes itself through circling and then moves through the environment in direct, straight lines. Furthermore, once the object is identified, the agent takes the most direct route. (7b) The agent is able to navigate efficiently through narrow passages. (7c) The agent occasionally gets stuck within regions of the map or in collision with objects outside its field of view.

Towards navigation, several challenging problems and emergent behaviors become apparent when considering a formulation of the navigation problem with continuous control and an embodied agent (Fig. 7). To maximize its ability to both cover the space and to reach known goals, the agent learns to move with direct, smooth trajectories. This behavior is particularly visible once the object has been found, in which case, the agent *beelines* to the goal. The agent also learns to identify and traverse *narrow passages*, e.g., doors. We find that while the agent is capable of entering doorways, it at times gets *trapped* in regions either by never deciding to exit a region or getting stuck on objects that are outside its field of view. These challenges exist particularly for domains like robotics, with embodied agents and continuous control, and warrant further research.

## 6 Conclusion

In this paper we study object-conditioned exploration and low-level continuous control in the context of object navigation. We present a generic and scalable RL setup which trains navigation policies with emerging natural exploration behaviors. Further, we show challenges with low-level continuous control, and demonstrate how to address them. For future work, we plan to apply memory-based models such as Transformers to aid long horizon exploration, and investigate reward shaping to improve precision control.

## References


[1] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018.

[2] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[3] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. D. Reid, and J. J. Leonard. Simultaneous localization and mapping: Present, future, and the robust-perception age. *IEEE Transactions on Robotics*, 2016.

[4] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3D semantic parsing of large-scale indoor spaces. In *CVPR*, 2016.

[5] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell. Learning to navigate in complex environments. *ArXiv arXiv:1611.03673*, 2017.

[6] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2017.

[7] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

[8] D. Mishkin, A. Dosovitskiy, and V. Koltun. Benchmarking classic and learned navigation in complex 3d environments. *arXiv preprint arXiv:1901.10915*, 2019.

[9] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

[10] A. Sax, J. O. Zhang, B. Emi, A. Zamir, S. Savarese, L. Guibas, and J. Malik. Learning to navigate using mid-level visual priors. In *International Conference on Learning Representations*, 2020.

[11] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra. DD-PPO: Learning near-perfect pointgoal navigators from 2.5 billion frames. *arXiv*, pages arXiv–1911, 2019.

[12] D. S. Chaplot, S. Gupta, D. Gandhi, A. Gupta, and R. Salakhutdinov. Learning to explore using active neural mapping. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/pdf?id=HklXn1BKDH.

[13] W. Yang, X. Wang, A. Farhadi, A. Gupta, and R. Mottaghi. Visual semantic navigation using scene priors, 2019.

[14] A. Mousavian, A. Toshev, M. Fišer, J. Košecká, A. Wahid, and J. Davidson. Visual representations for semantic target driven navigation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8846–8852. IEEE, 2019.

[15] J. Zhang, L. Tai, J. Boedecker, W. Burgard, and M. Liu. Neural slam: Learning to explore with external memory. *arXiv preprint arXiv:1706.09520*, 2017.

[16] T. Chen, S. Gupta, and A. Gupta. Learning exploration policies for navigation. In *International Conference on Learning Representations*, 2019.

[17] E. Parisotto and R. Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017.

[18] N. Savinov, A. Dosovitskiy, and V. Koltun. Semi-parametric topological memory for navigation. *arXiv preprint arXiv:1803.00653*, 2018.

[19] K. Fang, A. Toshev, L. Fei-Fei, and S. Savarese. Scene memory transformer for embodied agents in long-horizon tasks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 538–547, 2019.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[21] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[22] A. Zamir, F. Xia, Z.-Y. He, and S. Sax. Gibson environment for embodied real-world active perception. *CVPR (to appear)*, 2018.

[23] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9339–9347, 2019.

[24] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[25] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[26] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[27] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson. PRM-RL: long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5113–5120. IEEE, 2018.

[28] J. Yang, Z. Ren, M. Xu, X. Chen, D. Crandall, D. Parikh, and D. Batra. Embodied visual recognition. *arXiv preprint arXiv:1904.04404*, 2019.

[29] C. Wögerer, H. Bauer, M. Rooker, G. Ebenhofer, A. Rovetta, N. Robertson, and A. Pichler. LOCOBOT-low cost toolkit for building robot co-workers in assembly lines. In *International conference on intelligent robotics and applications*, pages 449–459. Springer, 2012.

[30] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[31] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting distributed synchronous sgd. In *International Conference on Learning Representations Workshop Track*, 2016.

[32] I. Armeni, Z.-Y. He, J. Gwak, A. R. Zamir, M. Fischer, J. Malik, and S. Savarese. 3D scene graph: A structure for unified semantics, 3d space, and camera. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5664–5673, 2019.

[33] B. J. Kuipers and T. S. Levitt. Navigation and mapping in large scale space. *AI magazine*, 9 (2):25–25, 1988.

[34] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[35] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

# 7 Appendix

# 8 Soft Actor-Critic

As commonly done, we formulate the problem as a Partially Observable Markov Decision Process (POMDP) $(\mathcal{O}, \mathcal{S}, \mathcal{A}, P, R)$. The observation set $\mathcal{O}$ consists of sensory measurements such RGB images, depth images, and LiDAR readings. The state space $\mathcal{S}$ captures the current and past poses of the robot and environment. $\mathcal{A}$ is the action space, which in our case consists of twist commands (linear and angular velocities). $P$ represents the state transition probability of the next state $s_{t+1} \in \mathcal{S}$ given the current state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$. Lastly, $R$ is a reward.

We learn a navigation policy $\pi(a_t \mid o_t)$ using Soft Actor-Critic (SAC) [21, 34], an off-policy form of maximum entropy reinforcement learning. The idea behind maximum entropy reinforcement learning is to learn a policy that maximally accomplishes the task while simultaneously being maximally random, with the idea that this randomness aids both exploration and robustness. To accomplish this, SAC learns a policy that maximizes the expected sum of rewards along with the $\alpha$-temperature weighted entropy $\mathcal{H}$ over the policy [21],

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau \sim \pi} \sum_{t=1}^{T} R(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot \mid o_t))$$

with $\tau = (s_0, o_0, a_1, s_1, o_1, \cdots, a_T, s_T, o_T)$ sampled from $\pi$. The above policy is learned in an Actor-Critic setting, where the policy is optimized to be consistent with a critic, a state-action function. The latter is learned via a 'soft' Bellman residual optimization.

The optimization iterates between two steps. In the *experience collect* step, sequences of state transitions and associated actions are sampled from the environment using a recent policy. These transitions are used to perform the aforementioned optimization in the *training* step.

# 9 Implementation Details

**NN Architecture** We use the same NN architecture for all function approximators used in the SAC implementation, the policy, state action function, and value function. This architecture consists of observation embedders whose outputs are concatenated and fed into a single layer LSTM of dimension 512.

As an observation embedder we use a ResNet50 [35], whose channel number is scaled down by a factor of 4. All image observations, (`RGB`, `DEPTH`, `DET`), are concatenated along their channel dimension and fed into the network. The last observation, `LiDAR`, is 1-dimensional and is embedded using a 3 layer ConvNet, with a final fully connected layer. All resulting observation embeddings are of dimension 128 and are added together as a final observation embedding.

The auxiliary observations, denoting the previous action and its success, are independently embedded using two 2-layer MLP, both layers of dimension 128.

In addition to the above observation, the policy is conditioned by the label of the target object. This label is represented by a 1-hot vector, which is embedded by a similar 2-layer MLP, both layers of dimension 128.

**Training Details** We use the Adam optimizer with a learning rate of 0.000316. The LSTM unroll length during training is 20, which is substantially smaller than the maximum step length of 500 during evaluation and of 100 during collection. Finally, SAC uses lagging weights for the target state action function in its Bellman error loss to stabilize training. In our implementation we gradually update these weights every step with a Polyak update of weight 0.005 every 1 step. We use a gamma discount factor of 0.99. We performed a grid search over only the learning rate, in the range [1e-4, 1e-3].

# 10 Evaluation Details

**Metric Definitions** (SPL), defined over $N$ navigation episodes, is the average of the success indicators $S_i$ scaled by navigation efficiency, expressed in terms of the optimal path length $o_i$ to goal

and the length $l_i$ of the actual path taken: SPL $= \frac{1}{N}\sum_{i=1}^{N} S_i \frac{o_i}{\max\{l_i, o_i\}}$. We note that a perfect SPL of 1.0 is not realistic to expect because navigation decisions in unexplored environments can be ambiguous.

**Human Raters** We provide a virtual setup where a rater can perform 'forward', 'turn left', 'turn right' discrete actions using the keys 'W', 'A', and 'D'. The rater has not seen the environments before, and is allowed a single navigation episode per home. We use the same criteria for starting point and episode length as for the policy evaluation. The maximum velocity and turn rate are kept consistent between the policy and human action spaces, so that that the maximum distance possible to cover is similar in both cases. In particular, the linear displacement for human rater is set at 0.25m; for robot the max possible displacement per action is 0.2m. Thus, the policy is a bit disadvantaged.