# Indoor Target-driven Visual Navigation Using Imitation Learning

Xin Zheng
Stanford University
xzheng3@stanford.edu

Yu Guo
Stanford University
yuguo68@stanford.edu

## Abstract

*We investigate visual navigation in a virtual household environment using imitation learning, where the robot learns the expert's policy to navigate to given targets. For targets whose expert's policy is known to the robot, our model achieves great performance on navigating to those targets from randomly chosen starting states. Taking both the current state observation and the target as input, our model is able to generalize to new targets without retraining. We also implement DAGGER (Dataset Aggregation) and safeDAGGER, and compare their performance to the base imitation learning model. We find that DAGGER can bring robot's and expert's trajectory distributions closer and safeDAGGER can dramatically reduce queries to the expert. However, we also observe that DAGGER does not generalize well to new tasks, which needs to be addressed in future work.*

## 1. Introduction

Robots play significant roles in improving people's lives. Some home robots have already been used to make domestic life easier, such as robot vacuum clearer. However, for robots to complete more complicated task, e.g. grasping a cup of coffee from kitchen, a good capability of indoor navigation is necessary for a robot to successfully move to the target. Inspired by this requirement, in our project, we focus on the problem of navigating a robot to find a given target in common household environments using only visual inputs.

In our setup, a robot searches for a given target in an environment consisting of 3D scenes. The visual observation data we will use for action and reaction in environments is from The House Of inteRactions (AI2-THOR) framework [1], which provides an environment with high-quality 3D scenes and a physics engine. Our scene set is composed of 20 scenes belonging to 4 common scene types in a household environment: kitchen, living room, bedroom, and bathroom (Fig. 1). The robot's action space includes four actions: moving forward, moving backward, turning left, and turning right. Upon every action, the robot will observe a new current state image, as illustrated in Fig. 1. At each time step, taking the target image and the current state image as input, the robot decides which action to take to reach the target (Fig. 2).

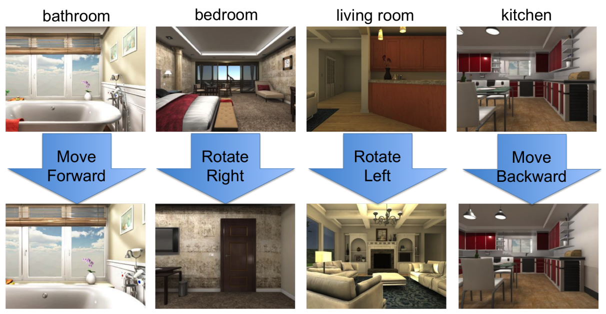

Figure 1. Dataset: 20 scenes belong to 4 common scene types, kitchen, living room, bedroom, and bathroom; Action Space: moving forward, moving backward, turning left, and turning right.

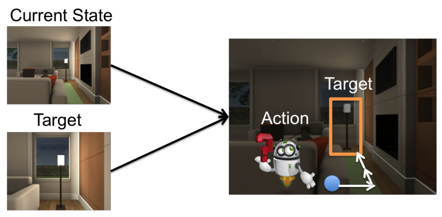

Figure 2. The goal of our model is to navigate towards a visual target with a minimum number of steps. Our model takes the current state image and target image as the input and outputs a sequence of actions taken by robot to reach the target.

In this work, we use imitation learning [2, 3] to train our action policy. The base imitation learning model can have difficulty dealing with states that the agent never experiences before. The DAGGER model [4] aims to improve upon base imitation learning by letting a primary policy collect training examples referring to expert policy simultaneously. In our setting, it can bring robots and experts trajectory distributions closer by labeling additional data points resulting from applying the current policy. In DAGGER, every states experienced when applying the current policy need to be labeled, so it can be very expensive. The safeDAGGER model [5] aims to ease this problem by designing a safe policy that determines whether to query the expert or not. Here we implement both DAGGER and safeDAGGER, and compare their performance to the base model.

The metric of our results will be the average trajectory length for the robot to navigate from a random starting state to the target in a scene. The best possible performance is provided by the shortest path came with each scene. We evaluate our method for the following tasks: (1) Navigate to trained targets, where the goal is to navigate from random starting states to targets that have been used during training within a scene. (2) Navigate to new targets, where the goal is to navigate to targets that have not been used during training within a scene.

The structure of our report is as following: In Section 2 we review the background and related work on indoor navigation. In Section 3 we present our imitation learning network architecture and DAGGER models. In Section 4 we discuss and analyze our results. We conclude and point out future work in Section 5.

## 2. Background/Related work

The mathematical description of visual navigation task is as following: Let $o(s)$ denote an observation for state s. The policy is specified as a parametrized function $\pi_\theta(a|o(s))$ mapping observations to a probability over actions, where $\pi_\theta(a|o(s))$ can be generated with neural network with $\theta$ denoting the weights. The goal is to achieve the optimal policy $\pi^*$ to navigate the agent to the target.

There exists multiple approaches for visual navigation. In general, the approaches can be generalized into two different categories: (1) mapping, which decomposing the navigation task into two stages: mapping the environment and generating the path through constructed map (e.g. [6, 7, 8, 9, 10, 11, 12]). For example, in [12], the main architecture contains a mapper to integrate first-person images into a top-down 2D representation of the environment, follows by a planner which outputs the actions to take. The planner in [12] is based on value iteration networks(VIN) proposed by [13]. VIN model[13] provides a novel differentiable approximation of the value-iteration algorithm, which can be represented as a convolutional neural network (CNN), and trained end-to-end using standard back propagation. It has been shown that such a model leads to better generalization in a diverse set of tasks [13]. (2) mapless, which does not need a prior map of the environment. (e.g. [1, 14, 15, 16]). The research most directly relevant to our work is the work of Zhu et al. [1], which also study first-person view target-driven visual navigation. The implementation in [1] uses a deep reinforcement learning model that takes as input an RGB image of the current observation and another RGB image of the target. The output of the model is an action. Zhu et al. also parallelize the training process using a method similar to actor-critic (A3C) algorithm [17] but with each thread running with a different navigation target. Our approach is based on [1] but using imitation learning instead of reinforcement learning.

## 3. Approach

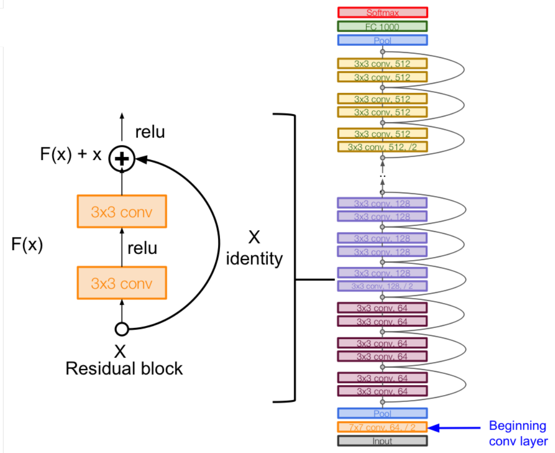

Figure 3. ResNet architecture to extract 2048-d features from each observation image, taken from [18]

### 3.1. Dataset

We use [hdf5](http://www.h5py.org/) dumps of the simulated scenes [1]. Each dump contains the agent's first-person observations sampled from a discrete grid in four cardinal directions. To be more specific, each dump stores the following information :

- observation: 300x400x3 RGB image (agent's first-person view)
- ResNet feature: 2048-d extracted feature of the observations
- location: (x, y) coordinates of the sampled scene locations on a discrete grid with 0.5-meter offset
- rotation: agent's rotation in one of the four cardinal directions, 0, 90, 180, and 270 degrees
- graph: a state-action transition graph. e.g. graph[i][j] is the location id of the destination by taking action j in location i.
- shortest path distance: a square matrix of shortest path distance (in number of steps) between pairwise locations, where -1 means two states are unreachable from each other.

In simulation, we choose the targets based on their appearance in the RGB images. We use ResNet to extract features from these RGB images. The architecture of ResNet is shown in Fig. 3. We do imitation learning by cloning the behavior of shortest path. The shortest path contains a sequence of state observations and corresponding optimal actions $(o_1, a_1, ..., o_N, a_N)$ to move an agent from its current location to a target in the least steps. We recover the shortest path between pairwise locations using 'shortest path distance' information as listed above. At each step on the shortest path $o_k$, the 'shortest path distance' from current location to the target is denoted as $dist(o_k)$. We can recover the action taken by the agent at step $k$ through searching in the action space, looking for the action $a_k$ that can lead to the next location $o_(k+1)$ with $dist(o_{k+1}) = dist(o_k) - 1$, where $dist(o_{k+1})$ is the shortest path distance from next location to the target. We repeat this process until target is reached. The sequence of actions $(a_1, a_2, ..., a_N)$ forms a shortest path.

## 3.2. Network Architecture

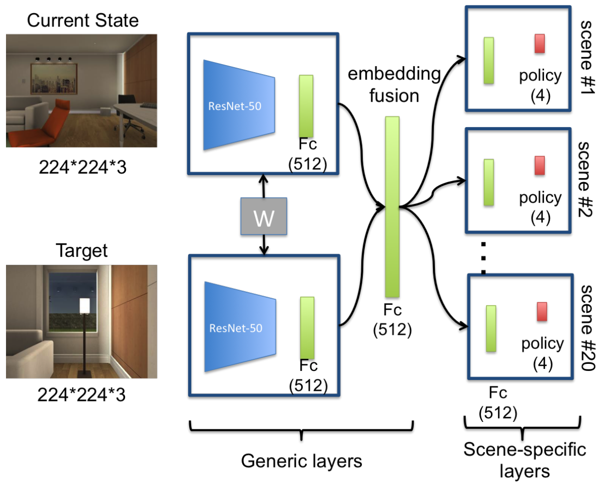

Figure 4. Our network architecture consists of generic layers and scene-specific layers; The numbers in parentheses show the output dimensions.

Our network consists of two parts as in Fig. 4: the first part is generic siamese layers where targets across all the scenes share; The second part is scene-specific layers which are used to capture the special characteristics of each scene. The inputs to the networks are two images representing the agent's current state and the target respectively. Two-streams of weight-shared siamese layers are used to transform the two images into the same embedding space. To be more detailed, the bottom part of the siamese layers are ResNet-50[19] (Fig. 3)which are pretrained and frozen during our network training. The outputs of ResNet-50 are two 8192-d vectors which are both projected into a 512-d embedding space, where they are fused and projected to a 512-d joint representation. The second part of the network use two fully-connected layers to transform the 512-d joint representation into a 4 policy output and a single value output. The two fully-connected layers are unique to each scene. We minimize the cross entropy loss between our predicted policy and the shortest path labeling using stochastic gradient descent method.

## 3.3. DAGGER (Dataset Aggregation)

```
Result: best $\pi^*$ on validation
Initialize $D_0$ and $\pi_1$;
for $i \leftarrow 1$ to $N$ do
    Step1: train policy $\pi_\theta(a_t|o_t)$ from human data
      $D_{\pi^*} = (o_1, a_1, ..., o_N, a_N)$;
    Step2: run policy $\pi_\theta(a_t|o_t)$ to get observations
      $D_\pi = (o_1, ..., o_M)$;
    Step3: ask human to label dataset $D_\pi$ with optimal
      actions $a_t$;
    Step4: Aggregate $D_{\pi^*} \leftarrow D_{\pi^*} \cup D_\pi$;
end
```

**Algorithm 1:** DAGGER Algorithm [4]

DAGGER[4] fine-tunes the policy trained with the imitation learning. With the policy trained with imitation learning as the initial policy, in each iteration of DAGGER, the robot follows the current policy and undergo a path, which might be different from the shortest path. For each step on this path, we label the state with the optimal action given by shortest path. As a result, we get a new dataset of state-action pairs. This new dataset is aggregated with the previous dataset. This aggregated dataset is then used to retrain the policy. The algorithm of Dagger is shown in Algorithm 1. Note that here we choose the $\beta_i$ parameter in the DAGGER model to be $\beta_i = Indicator(i = 1)$.

## 3.4. safeDAGGER

In DAGGER, we need to label every state that the robot undergoes under the current primary policy in each iteration. This is very expensive and sometimes impossible because the the human labeling might not be available in a timing manner. To make DAGGER more efficient in terms of querying the expert, we also implement safe DAGGER [5]. The basic idea of safe DAGGER is to construct a safe policy that determines whether an action by the robot is safe or not without querying the expert, along with the primary policy that determines which action the robot should take. Also, in safeDagger, a so called safe strategy [5] is used when running the current policy based on the safe policy. For a predicted move by the primary policy, if the safe policy predicts that it is safe, then we go with the primary policy, otherwise we query the export policy and take the expert's move. The algorithm of safeDagger is is shown in Algorithm 2. In

```
Result: $\pi_M$ and $\pi_{safe,M}$
Initialize $D_0$ using a reference policy $\pi^*$;
Initialize $D_{safe}$ using a reference policy $\pi^*$;
$\pi_0 = argmin_{\pi} l_{supervised}(\pi, \pi^*, D_0)$;
$\pi_{safe,0} =$
  $argmin_{\pi_{safe}} l_{safe}(\pi_{safe}, \pi_0, \pi^*, D_{safe} \cup D_0)$;
for $i \leftarrow 1$ to $M$ do
    Step1: Collect $D'$ using the safety strategy using
      $\pi_{i-1}$ and $\pi_{safe,i-1}$;
    Step2: Subset selection:$D_i \leftarrow \{\phi(s) \in$
      $D' | \pi_{safe,i-1}(\pi_{i-1}, \phi(s)) = 0)\}$ ;
    Step3: $D_i \leftarrow D_{i-1} \cup D'$;
    Step4: $\pi_i = argmin_{\pi} l_{supervised}(\pi, \pi^*, D_i)$;
    Step5: $\pi_{safe,i} =$
      $argmin_{\pi_{safe}} l_{safe}(\pi_{safe}, \pi_i, \pi^*, D_{safe} \cup D_i)$;
end
```

**Algorithm 2:** SafeDAgger Algorithm [5]

our implementation, we use the 512-d scene specific layer as the input and a network with one fully connected layer to train the safe policy.

## 4. Experiment

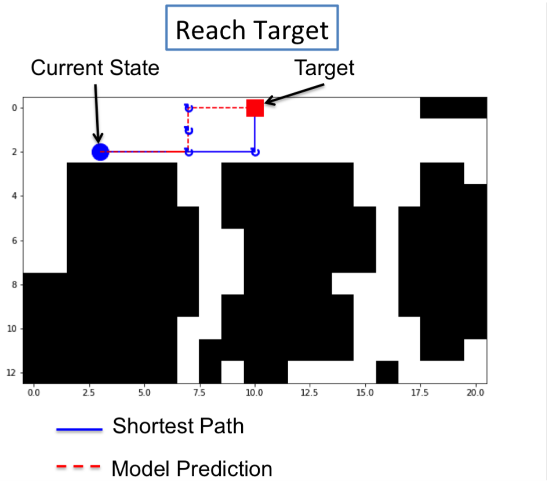

Figure 5. Representative Success Case for our model: Topview of the living room scene, with the model-predicted paths and shortest paths between random start and target.

All our models are implemented in Tensorflow [20] and trained on an NVIDIA Tesla K80 GPU. We train our model on two sets of scenes. The first one contains 4 scenes in total, coming from 4 different environments, with 5 targets in each scene. The larger set contains in total 10 scenes(3 scenes for bedroom, 3 scenes for bathroom, 3 scenes for living room and 1 scene for kitchen), with 8 targets in each scene. During training, we randomly choose 50 starting states for each target and collect the shortest path of each starting state and target pair to form the training dataset.

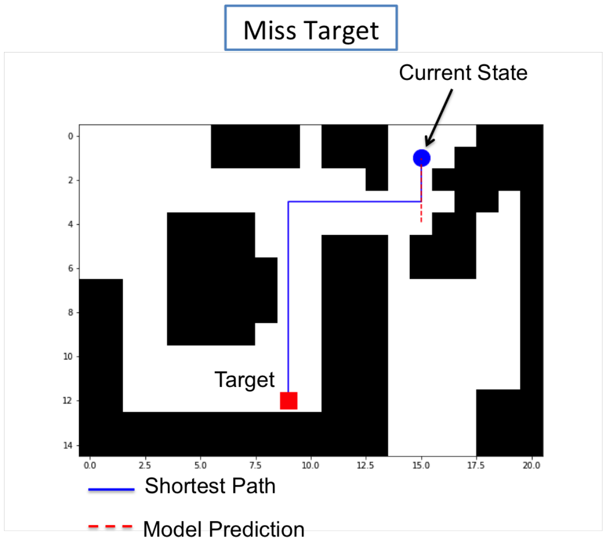

Figure 6. Representative Failure Case for our model: Topview of the kitchen scene with the model-predicted paths and shortest paths between random start and target.

| | Bed. | Bath. | Living | Kitchen |
|---|---|---|---|---|
| imitation learning | | | | |
| success rate | 0.80 | 0.93 | 0.90 | 0.96 |
| avg. shortest length | 13.04 | 6.58 | 12.78 | 17.60 |
| avg. predicted length | 76.76 | 37.57 | 67.68 | 33.92 |
| avg. successful length | 22.11 | 18.82 | 40.91 | 21.86 |
| DAGGER (1) | | | | |
| success rate | 0.89 | 0.96 | 0.88 | 0.93 |
| avg. shortest length | 12.90 | 6.47 | 14.10 | 17.67 |
| avg. predicted length | 45.34 | 19.94 | 51.04 | 41.23 |
| avg. successful length | 14.93 | 9.28 | 18.16 | 21.75 |
| DAGGER (2) | | | | |
| success rate | 0.98 | 0.98 | 0.92 | 0.92 |
| avg. shortest length | 12.44 | 6.76 | 13.59 | 17.71 |
| avg. predicted length | 20.98 | 15.29 | 37.38 | 41.56 |
| avg. successful length | 15.29 | 9.48 | 15.58 | 20.10 |

Table 1. Navigation Results for a small scene set with 4 scenes and 5 targets for each scene. Here all the paths get cut off at 300 steps. success rate: the portion of success cases; avg. shortest length: average shortest length between pairwise starting points and targets; avg. predicted length: average path length predicted by our model; avg. successful length: average path length in success cases predicted by our model. The numbers in parentheses represent the numbers of iteration in DAGGER.
.

Our goal is to navigate the robot from the current location to the target. For testing, we report the performance as the number of steps it takes to reach a target from a random starting point. Here we set a maximum number of steps(e.g. 200 steps) for the agent to reach target, since in our experiment either the robot reaches the target quickly or it cannot reach the target. We consider the navigation as successful if the agent can reach the target within maximum number of steps, and failure otherwise. For each target, we randomly generate 50 starting points to evaluate the model performance and compute the average statistics. We compare our model with the shortest path.

## 4.1. Test on trained targets

Here we evaluate our model on the trained targets, where the goal is to navigate from random starting states to targets that have been used during training within a scene. Note that although we test on the targets that are used for training, the starting states for test are chosen purely randomly, so they are different from the starting states for training.

We first show some qualitative results. Fig. 5 and Fig. 6 visualizes representative success and failure cases for our model, respectively, where we simplify our environment to a 2D grid and the black regions denote where the robot does not have access to. In the success case, the agent is able to figure out actions to take and reach the target image. Fig. 5 shows that the path taken by the agent in success case can be different from the given shortest path. In the failure case(Fig. 6), the agent cannot reach the target from the starting state. It collides into the wall and stays there until reaching the maximum steps.

We now present the quantitative results on the smaller scene set with 4 scenes and 5 targets in each scene, as shown in Table 1. Firstly, the baseline, i.e., imitation learning without DAGGER already achieves reasonably good performance, with success rate all above 80% for the 4 scenes. However, the average length of successful path are much larger than the shortest path. For example, in the living room scene, the average length of successful path is about 3 times of the average short path length. Yet with DAGGER, we observe that not only the success rate approves, but also the average length of successful length is much closer to shortest path length. The latter observation makes sense because DAGGER collects more optimal state-action pairs during training. If the robot deviates from the shortest path, the extra data points collected by DAGGER will bring it closer to the optimal path.

We also show the results on the larger scene set with 10 scenes and 8 targets in each scene in Table 2. Here again, all models achieve great performance in terms of success rate. Also, with DAGGER, the predicted path for robot to reach the targets is closer to shortest path than the basic imitation model. However, the improvement is not as prominent as the case with the smaller scene set.

## 4.2. DAGGER VS safeDAGGER

The results presented in Table 2 do not suggest much difference between DAGGER and safeDAGGER. The main purpose of safeDAGGER is to reduce the number of queries that the robot makes to the expert on the optimal action corresponding to shortest path in our case. To see the effects of the safe policy implemented in safeDAGGER, we record the total number of queries during training for DAGGER and safeDAGGER with iteration number of 1 and 2 in Table 3. The results demonstrate that, in our implementation, with the same iteration number, the total number of queries of safeDAGGER is about one order of magnitude less than that of DAGGER, which proves the principle of safeDAGGER.

| | Bed. | Bath. | Living | Kitchen |
|---|---|---|---|---|
| imitation learning | | | | |
| success rate | 0.90 | 0.97 | 0.82 | 0.80 |
| avg. shortest length | 10.82 | 7.25 | 16.31 | 17.45 |
| avg. predicted length | 37.19 | 15.58 | 55.95 | 66.34 |
| avg. successful length | 18.76 | 9.55 | 24.69 | 32.92 |
| DAGGER (1) | | | | |
| success rate | 0.95 | 0.96 | 0.82 | 0.78 |
| avg. shortest length | 10.87 | 7.11 | 15.36 | 16.02 |
| avg. predicted length | 23.35 | 18.41 | 53.83 | 59.82 |
| avg. successful length | 13.40 | 9.85 | 21.02 | 19.12 |
| DAGGER (2) | | | | |
| success rate | 0.94 | 0.93 | 0.84 | 0.87 |
| avg. shortest length | 10.71 | 7.23 | 15.98 | 15.45 |
| avg. predicted length | 30.10 | 24.31 | 49.64 | 43.37 |
| avg. successful length | 19.58 | 10.41 | 20.64 | 19.97 |
| safeDAGGER (1) | | | | |
| success rate | 0.93 | 0.88 | 0.78 | 0.84 |
| avg. shortest length | 10.84 | 7.08 | 15.58 | 16.21 |
| avg. predicted length | 26.88 | 34.45 | 59.00 | 48.88 |
| avg. successful length | 14.52 | 11.88 | 20.00 | 20.10 |
| safeDAGGER (2) | | | | |
| success rate | 0.95 | 0.95 | 0.88 | 0.91 |
| avg. shortest length | 11.05 | 7.23 | 16.21 | 17.27 |
| avg. predicted length | 22.70 | 17.98 | 42.37 | 38.92 |
| avg. successful length | 13.69 | 9.40 | 19.85 | 22.99 |

Table 2. Navigation Results for a larger scene set with 10 scenes and 8 targets for each scene. Here all the paths get cut off at 200 steps. success rate: the portion of success cases; avg. shortest length: average shortest length between pairwise starting points and targets; avg. predicted length: average path length predicted by our model; avg. successful length: average path length in success cases predicted by our model. The numbers in parentheses represent the numbers of iteration.

| Model | DA 1 | DA 2 | sDA 1 | sDA 2 |
|---|---|---|---|---|
| Total queries | 69915 | 114898 | 3478 | 14212 |

Table 3. Total number of queries made during training for DAGGER with iteration 1 (DA 1), DAGGER with iteration 2 (DA 2), safeDAGGER with iteration 1 (sDA 1), and safeDAGGER with iteration 1 (sDA 1).

## 4.3. New target generalization

In addition to navigating the robots from a random starting point to the targets used during training, our model is also capable of generalization across targets, which means it

| Distance | 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| imitation learning | | | | |
| success rate | 0.49 | 0.18 | 0.17 | 0.07 |
| avg. shortest length | 11.03 | 11.49 | 11.32 | 14.60 |
| avg. predicted length | 119.13 | 174.98 | 174.98 | 189.58 |
| avg. successful length | 33.94 | 60.21 | 55.36 | 57.95 |
| DAGGER (1) | | | | |
| success rate | 0.31 | 0.07 | 0.06 | 0.01 |
| avg. shortest length | 10.90 | 11.62 | 11.64 | 14.83 |
| avg. predicted length | 143.62 | 189.85 | 190.85 | 197.63 |
| avg. successful length | 19.88 | 50.80 | 38.09 | 22.31 |

Table 4. Navigation Results for new targets with 1, 2, 4 and 8 steps from the nearest trained targets. Here all the paths get cut off at 200 steps.

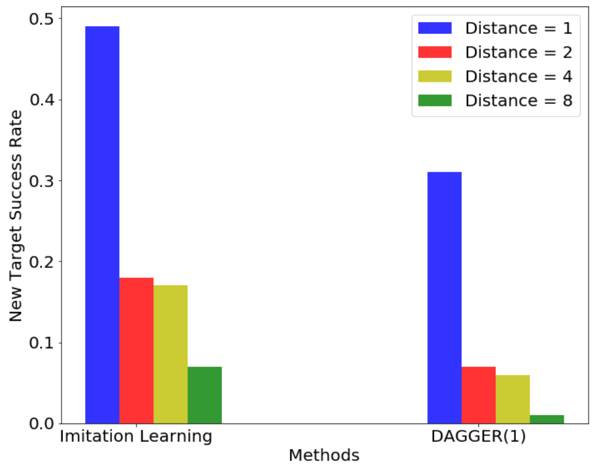

Figure 7. Target generalization performance of imitation learning and DAgger with iteration 1. The four bars in each group indicate the impact of distance between the new targets and nearest trained targets on generalization performance.

can navigate to targets that have not been used during training within a scene. To test our model's generalization ability, for each scene in the larger scene set with 10 scenes, we compare the success rate of navigation to new targets with 1, 2, 4 and 8 steps from the nearest trained targets. The results are summarized in Table 4. First, the success rate decreases as the distance to the nearest trained target increases(Fig. 7), which is expected because at longer distance, the features of images can be very different. Second, the successful path length is much larger than shortest path length, which suggests that for new targets, the robot do much more exploration than that for trained targets. Third, the performance with DAGGER is worse than that of basic imitation learning(Fig. 7), which is possibly caused by excessive training on the trained targets. In DAGGER, much more data points are collected for training, so the model is trained specifically and extensively for the chosen targets. Therefore, it is harder to generalize to new targets.

## 5. Conclusion

In this report, we investigate indoor target-driven navigation with imitation learning. We also implement the DAGGER and safeDAGGER model. We evaluate our model on both trained targets and new targets. On trained targets, our models achieve great performance in terms of success rate, which is the ratio of successful tasks to total navigation tasks. We also show that both DAGGER and safeDAGGER model, by collecting new data points and fine tuning the model, achieve better results than the base model, in the sense that the predicted path is closer to the ground truth shortest path. On the other hand, for new targets that are close to trained targets, we observe that the DAGGER model generalizes worse than the base imitation learning model. We also show that the performance of our model degrades as the new targets are further from the nearest trained targets.

For future work, we plan to tackle the problem that DAGGER model does not generalize well to new targets. We can also investigate other model structures such as deep reinforcement learning or generalize value iteration network to high dimensional inputs.

## Acknowledgement

We sincerely thank Yuke Zhu for helpful advice and providing the data framework for this project. We also refer to Yuke's code on the network and environment setup on `https://github.com/yukezhu?tab=repositories`.

## References

[1] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *arXiv preprint arXiv:1609.05143*, 2016.

[2] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[3] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[4] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 1, page 6, 2011.

[5] Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end simulated driving. 2017.

[6] Kurt Konolige, James Bowman, JD Chen, Patrick Mihelich, Michael Calonder, Vincent Lepetit, and Pascal Fua. View-based maps. *The International Journal of Robotics Research*, 29(8):941–957, 2010.

[7] Feras Dayoub, Timothy Morris, Ben Upcroft, and Peter Corke. Vision-only autonomous navigation using topometric maps. In *Intelligent robots and systems (IROS), 2013 IEEE/RSJ international conference on*, pages 1923–1929. IEEE, 2013.

[8] Robert Sim and James J Little. Autonomous vision-based exploration and mapping using hybrid maps and rao-blackwellised particle filters. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2082–2089. IEEE, 2006.

[9] David Wooden. A guide to vision-based map building. *IEEE Robotics & Automation Magazine*, 13(2):94–98, 2006.

[10] Andrew J Davison. Real-time simultaneous localisation and mapping with a single camera. In *ICCV*, volume 3, pages 1403–1410, 2003.

[11] Masahiro Tomono. 3-d object map building using dense object models with sift-based recognition features. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 1885–1890. IEEE, 2006.

[12] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. *arXiv preprint arXiv:1702.03920*, 2017.

[13] Aviv Tamar, Sergey Levine, Pieter Abbeel, Yi Wu, and Garrett Thomas. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2146–2154, 2016.

[14] H Haddad, Maher Khatib, Simon Lacroix, and Raja Chatila. Reactive navigation in outdoor environments using potential fields. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1232–1237. IEEE, 1998.

[15] Anthony Remazeilles, François Chaumette, and Patrick Gros. Robot motion control from a visual memory. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 5, pages 4695–4700. IEEE, 2004.

[16] Avik De, Karl S Bayer, and Daniel E Koditschek. Active sensing for dynamic, non-holonomic, robust visual servoing. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6192–6198. IEEE, 2014.

[17] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[18] Fei-Fei Li, Justin Johnson, and Serena Yeung. Cs231n. Stanford University, 2017.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[20] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.