Deep DAgger Imitation Learning for Indoor Scene Navigation

Tariq Patanam Eli Shayer Younes Bensouda Mourri

1. Abstract

The ability for robots to successfully navigate indoors is a critical step to bring the benefit of robotics to the general population. We use deep imitation learning on the THOR dataset to demonstrate the potential for deep imitation learning, in which an expert provides the learner the best routes and expects the learner to extrapolate to new situations to teach robots how to navigate indoors. Using the DAGGER algorithm, we train a neural network on one indoor scene and validate its effectiveness in a different indoor scene. We show that the DAGGER algorithm is able to substantially improve the average trajectory length compared to other potential navigation algorithms when generalizing to navigation in previously unexplored indoor scenes.

2. Introduction

Robot navigation has a wide range of applications. In this paper, we explore a specific technique of teaching robots how to navigate indoor scenes. However, the same approach could be used in self driving cars, drones, and all kinds of moving autonomous machines. In this project we use deep imitation learning to teach a robot how to navigate from a starting location to a target location in the context of an indoor scene.

Navigating indoors is harder than navigating outdoors due to several reasons. First of all, it is harder to get an accurate location of where the objects are. Household furniture such as chairs, tables, and other things, is not always in the same place unlike roads and stop signs. Second of all, guessing how much distance there is between any two objects is a harder task because small distance errors could cause collisions as the space is tighter and the need for accuracy is higher. [1]

Navigating indoors requires scene recognition, a route from each scene, and a way to associate each scene to its route. Each route needs to have certain rewards based on different target locations and starting points. Being able to do all of this is a very hard task and is one of the most challenging things to do in the field of robotics.

One possible way to teach a robot how to navigate indoor scenes uses imitation learning [2]. Imitation learning could be thought of as having a teacher who teaches an infant to do something and as time goes by, the child learns to do it by remembering his previous actions. Eventually the child will not have to consult with the teacher anymore. Similarly, in this paper, we train a robot to make decisions using the expert policy, which acts like a teacher, and eventually the robot would use its history to predict the next move.

Imitation learning and inverse reinforcement learning are two methods that are very common in this type of setting. In imitation learning, the only goal is to copy the teacher without even knowing the long term reward. On the other hand, inverse reinforcement learning tries to infer the goal of the teacher. It is more commonly associated with a reward function.

In this project we use imitation learning and inverse reinforcement learning to train robots to navigate indoor scenes. Ideally, after the robot is trained, it should be able to go from any starting location to any target point without consulting the teacher. Since it is very costly to keep bothering a human being every time, the goal is to minimize making calls to the expert or human being. We will see how these two types of algorithms perform on our data set and how likely they are to help us navigate indoors.

3. Related Work

Imitation learning is one form of Learning from Demonstration [3]. This machine learning technique relies on the presence of an expert who tells the 'student' what to do.

Unlike other papers, the method we used in here does not require a human being to train the robot in a new environment. Conversely, the robot is trained in a completely different scene and could perform well in another unseen environment. Other papers require a human being to teach the robot to navigate at first which is very costly, and may reflect human biases.[4] 3D matching, or local volumetric patch descriptors that matches partial 3D data together, have also been used to improve indoor navigation. [5] Those kind of algorithms would, for example, retain visual maps in real time views and form connections between the features found in each image. In short, it uses some sort of dictionary to make the connections between each image. However, other than just having to use manual training, this algorithm does not perform well on closed loops, since it just keeps learning the same thing over and over again.

On the other hand, supervised learning is sometimes used to recognize furniture in a house and in other indoor scenes which helps reduce collision rates. [6] The paper, by McManus, which focused on furniture recognition shed some new light on the field of robot navigation by providing an algorithm that is robust to extreme weather conditions and lighting. Rather than just looking at the features and patterns within each image, McManus has shown that it is possible to specifically search for corners and "scene signatures" to perform classification tasks. This type of algorithm is thus immune to weather conditions. However, there is a robustness - precision trade off problem since one could not precisely determine where the object is located under blurry situations. Other papers which focused on object detection to make predictions in Atari games have also proven to be very effective. [7]

The reinforcement learning (RL) algorithm also has a variety of applications other than just indoor navigation and there are numerous papers that have used it in the field of robotics. Although they have each taken a slightly different approach, in the overall, they all use some sort of reward driven algorithm. Recently however, people started merging RL with deep learning (DL) and got some very promising results, such those described in the Atari paper. These kinds of papers allow for target driven robots to reach a destination by using visual navigation.

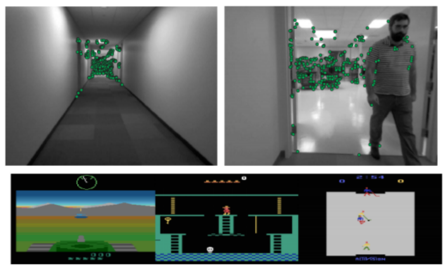


Figure 1: Examples from the ALE (left to right: Battle Zone, Montezuma's Revenge, Ice Hockey).

4. Method

4.1. Generating Expert Policies and Costs

In our task, these expert policies are generated by finding the shortest path to different target objects in the THOR framework while avoiding collisions completely. The shortest distance between any two given locations is provided within the dataset, which makes the generation of an ideal path trivial.

4.2. Supervised Imitation Learning

One of the biggest drawbacks of imitation learning is that the algorithm assumes IID actions which is not the case all the time. This means that the robot does not move from one place to another with an equally distributed probability. This violation of an assumption of the algorithm can decrease performance in practice [10]. To solve this problem, we have used a supervised learning approach where our loss is defined as:

$$\pi_{sup} = \arg\min_{\pi \in \prod} E_{s \sim d_{\pi^*}} [l(s, \pi)]$$

Which is proven to satisfy the following theorem:

$$E_{s \sim d_{\pi^*}} [l(s, \pi)] = \epsilon, then J(\pi) = J(\pi^*) + T^2 \epsilon$$

As a result, we are guaranteed to obtain a poor performance due to the quadratic term T^2. However, the supervised learning algorithm used with imitation learning provided us with an error almost linear in T and ϵ. Hence, we implemented a feed forward algorithm.

4.3. Forward Training Algorithm

We implemented the following version of the forward training algorithm [10]: For a given number of iterations and a given target location, we first randomly choose a start location in the given scene. When choosing the random start location we make sure to avoid locations from which the end location is unreachable. From this start location, we get the image that the robot is observing from our THOR scene. This is the X sample for a given step which we feed into our neural network.

From the expert policy we determine which choices in movement would maintain the possibility of an ideal-length path. The cross-entropy loss comes from this knowledge, as we penalize any step that does not allow for the possibility of an ideal path.

This process is repeated for *N* iterations, where at first each iteration has a different start step and takes only one step towards the target location. Then, each iteration took all steps necessary until we reached the target object and we took the loss and backpropagation at each step *and* iteration.

4.4. The DAGGER Algorithm

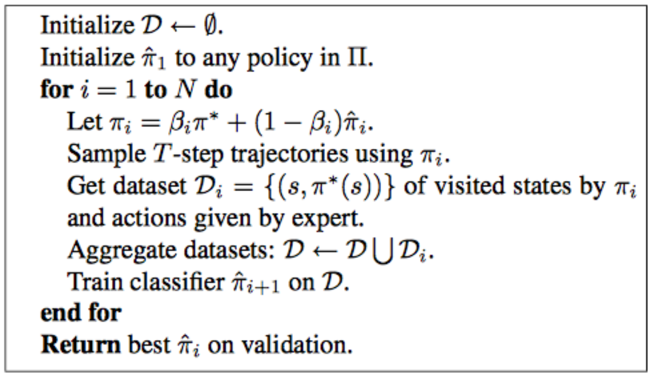
The most obvious problem with the forward feed algorithm is that it will not perform well when the state space is large and therefore the number of iterations (the number of policies to try out) must be large to converge. This is proven by the bounds demonstrated by Ross, Gordon, and Bagnell [10]. The DAGGER algorithm avoids both the problems of supervised learning and the forward training algorithm by keeping the expert policy continuously at hand. It is as seen in Figure 1. We modified the DAGGER algorithm to take into account our problem of having a number of different start locations and a number of target locations. We initially experimented with training each target location sequentially over a number of fixed start locations.

Algorithm 1 Initial DAGGER Training

```
1: procedure TRAIN
2:     for <each target location> do
3:         for <each start location> do
4:             DAgger loop as presented in Figure 1
```

However, this presented two problems. First, the model overfit on the target locations trained at a later stages, and second it was unable to generalize to different start locations. Therefore we adjusted our DAGGER implementation to train over randomly sampled (target location, start location) pairs. Now for n target locations and m start locations we have N as defined in Figure 1 to be $N = n * m$ and we avoid overfitting on both later target locations and fixed start locations.

We make one additional (commonly used) modification to the present DAGGER algorithm and that is to decay the β value after each iteration by the β decay factor we call, γ . After each trajectory within an iteration *i*, we set the β value as $\beta = \beta * \gamma$. This increasingly weans the learning model off the expert policy. Section 5.5 discusses the consequences of not setting γ correctly.

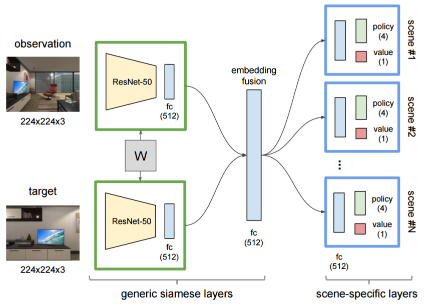


```
Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in $\Pi$.
for $i = 1$ to $N$ do
    Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
    Sample $T$-step trajectories using $\pi_i$.
    Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by $\pi_i$
    and actions given by expert.
    Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
    Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation.
```

Figure 1. The DAGGER algorithm

4.5. Deep DAGGER Architecture

The deep DAGGER architecture utilizes a siamese network layered on top of a pretrained ResNet-50 model. We feed both the ResNet features of the target image and the of the view from the current location. Each is run through a 512 neuron fully connected layer and then fused together in one layer before passing on the features to a 512 scene specific layer. This siamese architecture allows our model with a target specific layer before fusing to generalize to different targets.



Yuke's Siamese Architecture Model

In here we see that the model is concatenating two inputs, an observation and a target, and processes them through some scene specific layers which has the layouts and object arrangements. In order for the algorithm to generalize properly, all the targets in the scenes share the same scene specific layer. Each scene specific layer has a fully connected layer, a policy and a value. Based on the current inputs, the algorithm learns to decide on the right move.

4.5.1 Loss

Imitation learning does not attempt to maximize a reward function or its Q-value as in reinforcement learning because it does not have access to the reward function of its environment. Therefore we resort to minimizing a loss function comparing the expert policy's action to our model's action defined as:

$$\epsilon_N = \min_{\pi \in \prod \frac{1}{N} \sum_{i=1}^{N}} E_{s \sim d_{\pi_i}}[l(s, \pi)]$$

To provide the loss with an upper bound, we could use the following bound:

$$J(\hat{\pi}) \leq T_{\epsilon_N} + O(1)$$

where T_{ϵ_N} is the best error possible over T iterations. ϵ_n is the ideal error and O(1) could be viewed as some extra constant which is the difference between the ideal loss and the current loss. The proof for the linear error, rather than quadratic error of supervised imitation learning could be further seen in the Ross's paper [10]. We also used a normalized exponential function where:

$$y_c = \frac{e^{zi}}{\sum_{i=1}^{N} e^{z_i}}$$

The cost was optimized my minimizing the following equation: $y_i - t_i$ where t_i was our predicted move versus the expert action.

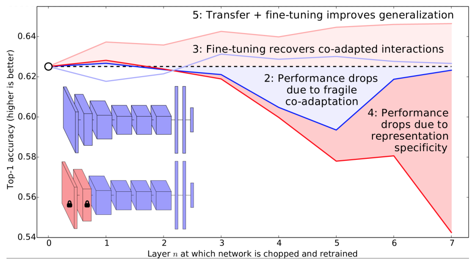


Figure 2. The demonstrated trade offs in fine tuning demonstrate why we pretrained our top layers to prevent representation specificity

4.6. Transfer Learning for Deeper Siamese Networks

We hypothesized that although a ResNet model trained on ImageNet captures a large amount of features present in THOR scenes, this ResNet model is not fine tuned for the indoor scene data nor is it fine tuned for correlating image data to four possible action ouputs. Therefore, we should be able to achieve superior results by retraining the last layer of ResNet-50 and forming a deeper siamese network as seen in .

We further hypothesized that if we were to use our untrained top Siamese layers on top of ResNet, we would backprop large gradients in the initial stages of training. Such large gradients would destroy the existing ResNet layers (the unfrozen ResNet layers) rather than fine-tuning them. In order to prevent destabilization of the ResNet, then, we first trained our existing top model before layering it on top the ResNet model.

We successfully implemented this model using a pretrained (on ImageNet) ResNet-50 from Keras. However due to the incredibly slow training using the Keras framework, we were not able to finish training in time to report these results.

4.7. Model Strengths and Weaknesses

The model we have used does well when tested on the same scene with different targets and generalizes really well. This is probably the biggest advantage of our model because we do not need a teacher every time the robot has to navigate in a different environment. This is a big improvement because in the real world, being able to generalize is extremely important since no robot is likely to be 'tested' on a previously seen environment. As a result, it will be less costly to use since the need of the expert will not be there anymore.

The disadvantage of this model is that it does not reach the target all the time. Although it generalizes well to new environments, it is not completely immune to collisions. Sometimes the model gets stuck in some sort of loop which makes it harder to reach the target. Another disadvantage of this model is that it takes a very long time to train prior to getting some good results.

4.8. Training

Our model is trained on each scene individually. Traditional DAGGER training as detailed above operates from one starting point to one end point and trains for N iterations as shown above. However, in indoor scenes there is no defined starting and end point. The robot should be able to navigate from any start to location to any target location. To generalize to different starting locations, for each target location in the scene we generate 15 random starting points within the scene. Therefore, for m targets we iterate $N = m * 15$ times in the DAGGER algorithm. At each iteration we sample T trajectories where $T = 50$. Here we make another slight modification to the DAGGER algorithm. Rather than training only after sampling all T trajectories, we aggregate our dataset after *each* trajectory and then retrain our model. We found this approach along with increasing the number of trajectories and moderating β decay to converge towards the target much more successfully than the traditional DAGGER approach. Our approach is much more depth-first rather than breadth-first. Retraining after each trajectory means it is much less likely to explore the state space initially. The emphasis is put on reaching the target quickly rather than learning how to recover as it explores the state space. We train after each trajectory for 10 epochs. Overall, our training method greatly reduces, by several orders of magnitude, the amount of frames needed to train on as compared to the target driven reinforcement algorithm. The target driven model was trained over 100M frames or scene images [8] while our model required only 250,000 frames to train and achieve similar results. However unlike in A3C we are aggregating an increasingly large dataset and the training process is slow for an algorithm that has the expert policy at hand. We address methods for speeding up training while keeping similar results in Future Work.

5. Experiments

We carried out a number of experiments to measure how best to optimize DAGGER for indoor navigation, namely in THOR simulations, and to compare its results to the current state of the art performance in THOR. In particular we ran experiments to find the ideal β decay and episode values. Our metrics for evaluating the model include average trajectory length to target locations and success rates in generalizing to new targets. Our success rate is as defined by Zhu et. al[8] to be the proportion of times the model reaches its target in under 500 steps.

5.1. THOR Framework

One of the key challenges in indoor navigation learning, especially in the context of the large datasets needed for deep learning, is data collection. Dividng the real world into coordinate points, having a physical robot collect data for all those coordinates, and generating an expert policy can be a cumbersome process. Therefore we utilized the THOR data framework developed by Yuke et. al. The THOR framework has a number of detailed indoor scenes like bedrooms, bathrooms, and living rooms with target objects such as desks, chairs, and beds in each scene. Each scene is divided into a grid of locations and the shortest path distance between any two locations is provided. We utilize the shortest path distance as our expert policy. The clearest drawback of THOR is that such a simulation does not fully model real world physics. Nevertheless, it simulates real world interactions such as collisions with objects and it will allow us to judge different models before training in the real world.

Figure 3. Four of the targets trained in the THOR dataset. Objects near the target can be collided into

5.2. Collision-Indifferent versus Collision-Avoiding Models

As aforementioned the THOR framework allows for rich physical interactions such as collisions with objects during a trajectory. In the real world avoiding such collisions is optimal. However, we hypothesized that allowing the model to collide with objects during training alone and continue without restarting, would allow for more exploration in less iterations and increase its success rate in reaching the target. Therefore we trained two models. For one model during training, we sampled a trajectory until it successfully reached the target, hit the max step limit of 1000 during training (the 500 step limit is set only during testing of the model), *or* collided with an object. For the second model we allowed it to continue after a collision until it either reached the target or ran out of steps. We then tested the models from 5 different random starting points within a scene and the same 5 targets from training. During testing we count collisions as failures. Our results presented in 5 actually prove our hypothesis incorrect. The model that was cut short during collisions, the collision-avoiding model successfully reached the target more often than the collision-indifferent model. This was true for both the targets trained on and the new (untrained-on) targets we attempted to generalize for (discussed in the Target Generalization section 5.4. The unexpected results were probably because the collision-indifferent model was not penalized for its collisions during training so it was not trained to avoid collisions during testing. This shows that although the expert policy repeatedly indicated the collision-free path (positive reinforcement), penalizations for colliding were an important factor in training (negative reinforcement).

5.3. Trajectory Length Reduction

The most basic task at hand is to successfully reach the target and to do so in a realistic fashion. Decreasing trajectory length from a start location to the end location ensures that the robot will minimize collisions, act in a manner natural to humans, and substantially increase efficiency. We tested trajectory length by choosing five random start locations within the scene and sampled 10 trajectories of the models attempt to reach the target. We capped the maximum amount of steps that the virtual robot could take to 500 to avoid situations where the robot became stuck looping between the same state distribution.

Our results in Table 1 show that our Deep DAgger model is able to outperform excellent methods such as A3 CL in reaching the target at a reasonable pace. Interestingly the collision-indifferent trained version of Deep DAgger was able to significantly undercut both the collision-avoiding Deep DAgger and current state-of-the-art target-driven RL approach. There is one caveat. The results were normalized to take into account different success rates in the Deep DAgger models but not in regards to other models. Hence, the reason for such a dramatic improvement as compared to the Target-driven RL approach may simply be because it reaches closer targets.

However, there is one important reason to believe that collision-indifferent model outperforms both our collision-avoiding model and all others. It marries the best of both the target-driven RL approach which has much more extensive training (as aforementioned Zhu et. al trained for over 100 million frames) and imitation learning which has the expert policy in hand. Perhaps to overcome the problem of not penalizing collisions (mentioned in our section on Collision-Avoiding versus Collision-Indifferent) more positive reinforcement is needed and we should train the model for much more than 250,000 frames. More testing needs to be done.

Average Trajectory Length by Algorithm				
Random Walk	A3C RL (Multi-threaded)	Target-driven RL	Deep DAgger Collision-Avoiding	Deep DAgger Collision-Indifferent
2744.3	723.5	210.7	233.4	148.9

Table 1. Deep DAgger shows substatiantial improvement over existinig models in trajectory length

5.4. Target Generalization

Even before deep learning's recent popularity, DAgger has been shown to work well with traditional machine learning techniques. There is nothing new there. One new aspect we presented and mentioned before was giving our model the ability to start from any location post-training. However, we also wanted to test the model's ability to generalize to different scene-related targets. For instance, if we taught our indoor navigation model to find a fridge in the kitchen, it should also be able to find the toaster adjacent to the fridge. In two experiments, we trained on our model five and eight targets. We then tested them with a number of targets one step away, two steps away, four steps away, and eight steps away from the trained on target. We hypothesized that targets closer to trained on targets would be more easily found because of scene similarity.

Unlike in the original work of Zhu et. al we only consider collision free trajectories during validation (Note: this is separate from collision-indifference during training) to be successes. Therefore it is no surprise, a random baseline model (that is a model that chooses one of the four actions of left, right, forward, or backward randomly) was unable to reach any targets successfully.

As shown in **Figure bla** our DAgger trained model was able to generalize to targets near the targets that it trained on, indicating that the model was able to learn nearby regions well. This demonstrates some particularly unique results in imitation learning compared to foundational works such as by Ross et. al [10] where no generalization was attempted. It demonstrates that in the training process, our model was taught by the expert policy not only how to reach the specific targets it was training on, but it was also taught the *process* of finding and proceeding to targets as well. As we hypothesized closer targets were more easily found probably both due to scene similarity with the trained on targets and because of those regions of the scene were more explored during training.

5.5. Beta Decay Optimization

The DAGGER algorithm introduces three new hyperparameters into our model: β, γ (the β decay factor) and T, the number of trajectory samples taken at each iteration with a set start location to target location. Although β is typically initialized to 1 [12], we found that this prevents necessary exploration of the state distribution. Therefore, we began with $\beta = 0.9$ in order to explore a wider space of states early in training.

The more careful consideration is in optimizing T, the number of sampled trajectories and γ. If T is low and γ is high the model seems deceptively good reaching the target consistently during training. However at validation time the model performs poorly because it seriously *underfits* the data. Why? With a high γ and a low T, the β - where β is reduced as $\beta = \beta * \gamma$ after each of the T trajectories - never goes low, and the model never fully weans itself off the expert policy. Therefore at test time, the model is not able to recover from unseen distributions. With a higher T and lower $gamma$, we can clearly see that the model is actually learning and the converging to a high success rate versus having a consistently high success rate as seen in figure

We also discovered that there is a trade off in having a high T and higher γ versus having a lower T and lower γ. Notice in both situations the model weans itself off the expert policy by sending β low. However, the tradeoff is in the first the model learns the expert shortest path to target too well and sticks to it. Therefore it does not explore how to recover from unseen states. In the second case, the model veers off the shortest path to target and learns a suboptimal path. We found, through validation that with a starting β value of 0.9, the optimal values are $\gamma = 0.95$ and $T = 50$.

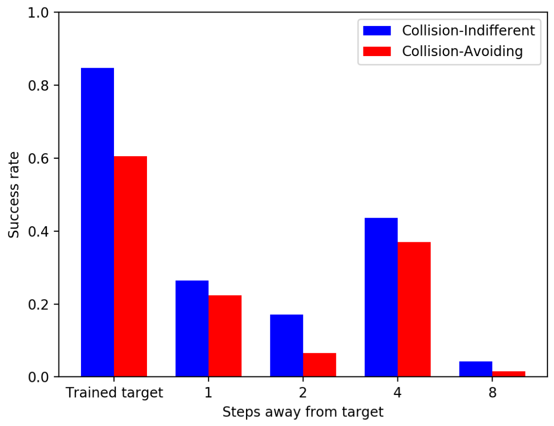


Figure 4. A comparison of the collision-indifferent and collision-avoiding trained models on both their trained targets and to untrained targets

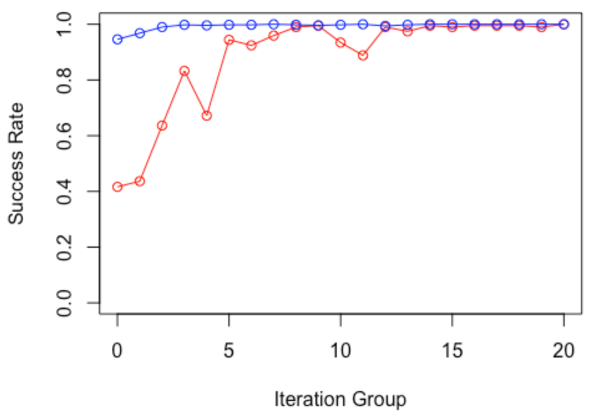


Figure 5. With $T = 50$ and $\gamma = 0.95$, the model learns to fit the data versus the blue line where the model performs deceptively well but only because of its excessive reliance on the expert policy

6. Conclusion and Future Work

We applied a novel approach to indoor scene navigation in the form of Deep DAgger. In addition, we experimented with a number of factors in tuning such a model. Our experiments shows that Deep DAgger is able to learn to reach targets successfully. It also is able to learn the process of reaching targets and because of this, able to generalize to other targets well.

Deep DAgger, especially with collision-avoidance training, still trails target driven RL as presented by Zhu et. al [8] in both trajectory length and target generalization. However, our experimentation with the collision-indifferent training shows that Deep DAgger is promising because it may learn shorter trajectories and generlization as well as, if not better than, target driven RL, if it is trained over many more frames than our 250,000.

6.1. Future Work

Although DAGGER greatly reduces the amount of frames needed to train on as compared to traditional deep reinforcement learning, it must, on every episode, train on an increasingly large dataset which is expensive at train time. Additionally, the form in which we apply DAGGER, in a grid environment, would be difficult to replicate in real life. This is because a real scene would be difficult to discretize into locations, as we were able to do in the context of the THOR dataset.

Other possible improvements include further optimizing our neural network architecture and implementing the DART algorithm. DART would particularly be beneficial, because we would explore a larger sample of spaces, which would provide a more general set of data with which to train.[13]

Nevertheless, we were able to show the efficacy of the DAGGER algorithm in the context of the THOR dataset, and demonstrate its ability to reduce the trajectory length between in starting location and a target location. This represents progress towards training a robot to navigate indoors while avoiding collisions.

References

[1] Ross et. al. Learning Monocular Reactive UAV Control in Cluttered Natural Environments.

[2] Stefan Schaal. Is imitation learning the route to humanoid robots?.

[3] Brenna Argall, et. al. A survey of robot learning from demonstration. 2009.

[4] K. Kidono, J. Miura, and Y. Shirai, Autonomous visual navigation of a mobile robot using a human guided experience, Robotics and Autonomous Systems, 2002.

[5] K. Konolige, J. Bowman, J. Chen, P. Mihelich, M. Calonder, V. Lepetit, and P. Fua, View-based maps, Intl. J. of Robotics Research, 2010.

[6] C. McManus, B. Upcroft, and P. Newman, Scene signatures: Localised and point-less features for localisation, in RSS, 2014.

[7] Y. Liang, M. C. Machado, E. Talvitie, and M. Bowling, State of the art control of atari games using shallow reinforcement learning, in AAMAS, 2016.

[8] Zhu et. al. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning 2016.

[9] Jonathan Ho, Stefano Ermon. Generative Adversarial Imitation Learning 2016.

[10] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning 2010.

[11] Iacoboni, Marco; Dapretto, Mirella (2006). ”The mirror neuron system and the consequences of its dysfunction”. Nature Reviews Neuroscience. 7 (12): 94251. PMID 17115076. doi:10.1038/nrn2024.

[12] Z. Richard, K. Andrew. ”Imitation Learning” in CS 159 Lecture at Caltech. Slide 42.

[13] Laskey et. al. Iterative Noise Injection for Scalable Imitation Learning. March 2017.