Supervised Learning Benchmarks for Point Goal Navigation in Photo-realistic indoor cluttered environments

Muhammad Zubair Irshad
mirshad7@gatech.edu

Asawaree Bhide
abhide9@gatech.edu

Shrija Mishra
smishra309@gatech.edu

Shubhangi Upasani
supasani8@gatech.edu

Abstract

*The aim of this work is to solve the point goal navigation task in photo-realistic, indoor environments using Habitat. In this task, a virtual agent (robot) starts at a random position in an unknown environment and is given the coordinates of where it has to navigate to. This is not a trivial task in realistic, cluttered environments as the agent has to traverse an environment while avoiding obstacles in the absence of a map. We present a previously under-explored paradigm in point goal navigation task i.e. using supervised learning for point goal navigation. We implement a benchmark based on RNN that preserves the temporal information present in the trajectories and predicts the most optimal next action given an observation action pair as ground truth. Our experiments reveal that supervised learning shows promise for this task. We evaluated our work against various classical and deep learning baselines and report losses and accuracy along with SPL metric for these baselines. Our imitation learning approach achieved an accuracy of 56%. We have open sourced our code base, which can be accessed here:*https://github.com/zubair-irshad/habitat_imitation_learning/

1. Introduction

1.1. Motivation

Given egocentric images from an RGB-D camera and using agent poses, we tried to solve the task of point goal navigation in an unseen indoor environment (without ground-truth map). We used the 'Success weighted by Path Length' (SPL) metric for our baselines and accuracy as performance metric for our approach. We tried using a behavior cloning approach to this problem. The objective was to train an agent to successfully navigate from its random starting point in the environment to the given target coordinates and learn an optimal mapping from states to actions. Agent is said to have successfully reached if it is within 2 times its own radius of the target position when STOP action is performed.

1.2. Related Work

Indoor navigation has been in discussion since long and has been studied extensively in classical robotics. Recent advancements in deep learning have opened avenues which were unexplored earlier. This problem statement is usually approached using methods broadly divided into two categories: supervised learning (SL) and reinforcement learning (RL). Most of the previous works focus on RL approaches. A major challenge with RL algorithms is to define a reasonable reward function, agent's hidden state and goal state. Another issue with using deep RL for navigation in the real world is that training agents is computationally expensive and requires a huge amount of data. This specific issue has been dealt with in [10]. In this work, the authors have solved the task of target-driven navigation using an actor-critic model and have proposed a new framework (AI2-THOR) which lets agents interact with their environments to collect training samples.

In [14], the authors present a family of policy optimization methods for RL. To perform policy updates, they alternate between sampling data from the policy and performing multiple epochs of SGD on the sampled data.
[12] solve the navigation task without access to a map, directly from an RGB-D camera and a GPS+Compass sensor, with a focus on distributed RL.

In addition to the above works, we studied [3], a useful survey that discusses various learning-based approaches and the applicability of supervised vs reinforcement learning to visual navigation. Inspired by this, we decided to explore some supervised learning approaches like imitation learning and behavioural cloning for our problem objective. In the imitation learning approach, a model is trained to predict expert behavior and agent actions are consequently learnt from this expert.

An interesting work in the field of supervised learning is [8] where the authors work on navigation among humans, which is especially challenging because human movement is unpredictable. They have used a learning-based perception module and a model-based planning module for this work. For the perception module, a supervised learning approach has been used to train a CNN model on RGB images of a photo realistic dataset.

[4] presents DAGGER, an iterative algorithm to train a stationary deterministic policy. DAGGER trains a policy using an aggregation of all datasets collected during past iterations under previous policies. The idea is to collect inputs during training that the policy may come across during test time.

In [2], the authors apply both classical and learning based methods for environment exploration. Using analytical path planners with three learned modules, Neural SLAM, Global Policy and Local Policy, the paper leverages the patterns and regularities present in the layouts around. The authors have successfully transferred this strategy to the Point Goal Navigation Task.

A limitation of imitation learning approach is that it requires the expert to provide action labels without being fully in control of the system. The expert provides actions only during training so the agent can be brought back to the right track if it took a wrong action. In the testing phase however, the agent has to predict its own actions. The errors that creep in because of no supervision at test time are not accounted for in training and are known as exposure bias. This also decreases the safety guarantees and when using humans as experts, is likely to degrade the quality of the collected labels due to perceived actuator lag. HG-Dagger [7] resolves some of these issues. Behavioral cloning also suffers from some other limitations such as data mismatch and compounding error issues.

SL, although not as explored as RL for this task, has tremendous potential and the motivation for our project is to assess the effectiveness of the SL approach.

1.3. Impact

Autonomous robots are gradually becoming indispensable for various industries and some crucial life-saving tasks. Solving the task of visual navigation in simulation is a step towards training physical robots to navigate in the real world (in both indoor and outdoor settings). This task is of great interest in the AI community, where current work is focused on providing agents with human-like capabilities. Studies like [9] show that there are 3 million people with a mobility impairment and such navigational robots aid to their independence by helping them with groceries, medicines, etc. Point Goal navigation is an especially important task in this domain, where a robot could ultimately move through a real environment (with obstacle-handling capabilities [8]) to a target location to perform further tasks including scene understanding or object retrieval.

1.4. Data

We used the Gibson 3D dataset [13]. This is a dataset of indoor spaces collected from real spaces by 3D scanning and reconstruction. The dataset includes a diverse set of environments including households, offices, museums, hospitals, etc. Inorder to incorporate realism, the data comes with no GPS or Compass and has some introduced noisy actuation and sensing. For each space, the dataset also includes:

- 3D reconstruction
- RGB images
- Depth images
- Semantic maps

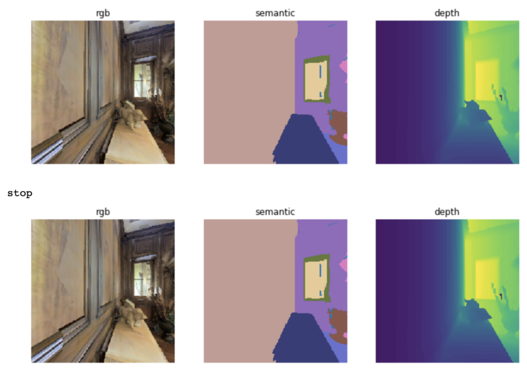

Figure 1. Example for episode start and end

The dataset is comprised of 572 different environments, of which we have used a subset of 4+ rated 72 environments. Prior work[1] suggests these 72 environments provide high quality photo-realistic simulations with no holes and rendering issues. We generated optimal actions and corresponding observations on the fly since saving all the data to a disk comes at a memory cost. Dataset was generated sequentially as follows:

- **Navigable data collection:** A subset of 50k trajectories including start and end points were generated per scene. These trajectories included start and end points randomly selected for each scene. To ensure randomness, we selected a sample of trajectories from each scene to generate batches of training and validation data.

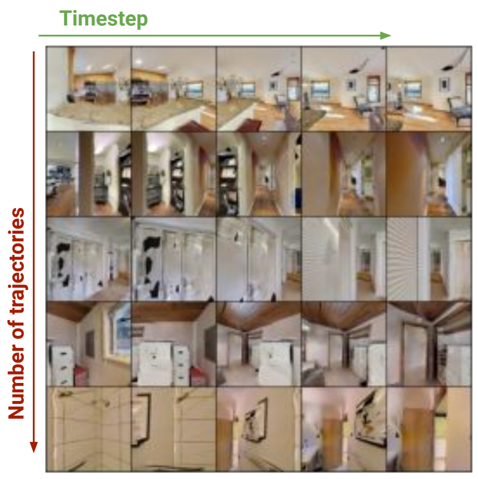

Figure 2. Scaled down representation of RNN Dataloader

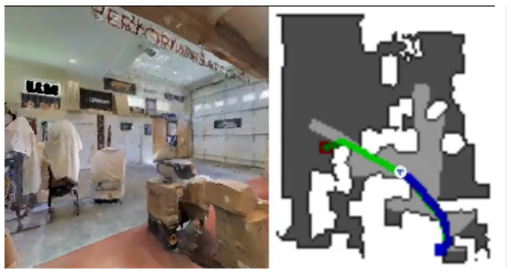

Figure 3. RGB image and top-down-map for shortest path follower

- **Optimal actions and observations:** We used classical path planning algorithm namely, shortest path follower (figure 3) to generate shortest path optimal actions and corresponding observations for each trajectory in a batch.
- **Data-loading:** Pytorch data loader was used along with padding and collating for each batch of training and validation data (figure 1.4). Padding and collating was incorporated to handle variable length sequencing. We padded trajectories shorter than the maximum trajectory length with 0s and subsequent actions with -1s. The final data loader is temporal in nature and comprised of the shape **(N,T,C,H,W)** where N denotes batch size, T captures the time domain temporal information as a trajectory progresses inside an environment, and C,H,W comprise of the RGB channel dimensions.

2. Approach

The use of supervised learning for navigation in photo realistic environments (like those provided in Habitat) has been lesser explored than RL based agents. RL, although ubiquitous, provides sparse rewards to the agent. SL, on the other hand, provides more immediate rewards to the agent and assists in faster learning. Another merit for SL is it uses gradient decent on a loss function. With correct weight solutions, each step of gradient update takes the model one step closer to the optimum. Equipped with above facts, we wanted to see if the supervised approach offers interesting advantages/insights over traditional RL approaches through this work. Additionally, an imitation learning based agent is not part of the Habitat platform currently, which we found an exciting area to venture into.

2.1. Supervised Learning

We formulate the point goal navigation task as a supervised learning problem. Given an observation state (x) and label (y) which is the optimal action that the agent should take at this state, our task is to find a mapping function $y = f(x)$ which takes the robot from an initial configuration to a goal configuration in the most optimal way.

2.1.1 Sequence to Sequence Model:

We further divide this formulation into a sequence to sequence problem. Though prior work [5] has shown great promise of Convolution Neural Networks to get a mapping of actions directly from images, we use a Recurrent Neural Network because our data is temporal in nature and dependencies between observations at each time step are an important aspect of the problem. A CNN fails to capture such dependencies.

Hence we model our architecture (figure 4) using an LSTM[6] based encoder decoder. Our observation space $\tilde{x} = x_1, x_2, ...x_n$ is comprised of images at each time-step along the trajectory.

Image Embedding: Each trajectory t_i is stacked and presented as a batch to Resnet-18 for feature extraction. We aim to get a concise representation of features from dense raw images available as input. The architecture of Resnet-18 was modified to get a final output equivalent to CNN embed dimension. We sequentially process each time step of a batch of trajectories to get a concise feature representation at each time step. We denote the output of encoding as $feature_i = CNN_{enc}(t_i)$. We concatenate this output along the time dimension to get a feature representation of all time steps in the trajectory.

Action Space: Our actions space is discrete and comprises of 6 different actions as follows: `stop, move forward, turn left, turn right, look up, look down`. `move forward` takes a 0.25m forward step in the environment whereas `turn left, turn right, look up, look down` moves the agent's heading by 30 degrees in the environment.

Action prediction using LSTM Decoder: We present

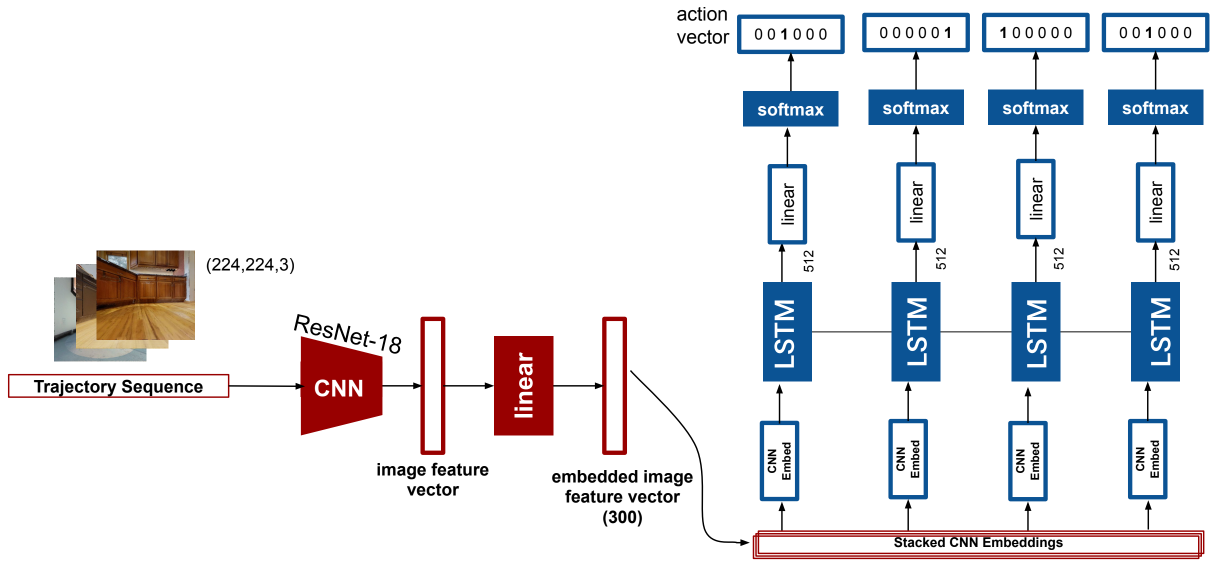

Figure 4. Encoder Decoder Architecture for temporal domain sequence information

the output of encoder as a sequence to an LSTM to get the output as $h_i = LSTM_{dec}(feature_i, h_{i-1})$. Moreover, we use a fully connected layer and a softmax layer $o_i = softmax(linear(h_i))$ at the output of LSTM to get a 6 dimensional output comprising of 6 actions.

Loss and Optimizer: Since our action space is discrete, we use a Negative Loss Likelihood Loss to compute mean loss for all data points using the following equation:

$$loss_i = \sum_{n=1}^{M} y_i \log p_i \tag{1}$$

We backdrop through the loss and perform gradient descent using Adam optimizer to optimize the weights of our neural networks. Since we use Resnet-18 as feature extractor, we only optimize the weights of fully connected layers in our CNN. For RNN, we optimize the weights of our entire model.

2.2. Comparison with Baselines

We also compared our approach against various baselines for this task:

Reinforcement Learning method - Proximal Policy Optimization (PPO) [14] The Proximal Policy Algorithm (PPO) is built on top of actor critic architecture and combats some of the major limitations of actor critic like dealing with outlier data and excessive dependence on hyperparameter tuning in addition to sudden updates to the policy. The algorithm tries to maintain smooth gradient updates to achieve constant improvement in performance. A major improvement in the PPO algorithm is the Generalized Advantage Estimation (GAE) that helps in reducing variance by stabilizing the discounted rewards. Apart from this, PPO also uses a surrogate policy loss which is a ratio of the new probabilities to the old probabilities multiplied by the advantage. Furthermore, this loss is clipped to make sure the updates to the policy are gradual. The trajectories are segregated into random minibatches and the network is updated using these.
We use the PPO alogrithm as our baseline. We wanted to see how much does GPS+compass information contribute towards the agent' performance. Therefore, we started our project implementation with PPO that uses GPS information. The algorithm was evaluated using the SPL metric and final distance to goal. The plots for these have been included in the later sections.

Classical method - SLAM based [15] This is a modular navigation pipeline. The localization module estimates the agent's pose in the environment and has been done using ORB-SLAM2. [16] The mapping module estimates a 2 dimensional obstacle map of the environment. The outputs of these 2 modules are used to plan a trajectory to the goal by the planning module. The planning module uses the D* Lite algorithm, which is identical to A* algorithm when planning for the first time and updates only the costs of the nodes that are affected by newly discovered obstacles for subsequent planning. Finally, the locomotion module generates an action to take. In this implementation, the agent has no prior knowledge about the environment at the start of each episode and has to explore while navigating towards the goal.
The issue with such classical approaches is that hand-engineering is required to a great extent.

Active Neural SLAM An exploration architecture with three modules, Neural SLAM, local policy and a global policy coupled with a path planner and map when transferred to Point Goal Navigation task has proven to perform the best in Habitat 2019 challenge. Motivated by the paper [2], we tried exploring ways to produce free space maps and estimate agents pose. The technique here takes advantage of the various regularities present in the layout in simulator and real world. The paper [2] additionally introduces their custom motion and sensor data collected. This domain of study adapts to a reward based system. Reward here is the maximization of the coverage, i.e. the total area in the map that can be traversed.

Three modules that make up the architecture have their individual roles. Mapper made of ResNet18 followed by 2 fully connected layers, dropout and then 3 deconvolutional layers coupled with a pose estimator made of 3 convolutional layers followed by 3 dense layers make up the Neural SLAM module. It predicts the map and agent pose based on current state and past predictions. This predicted map and pose is used by the Global policy to generate a long term goal which is broken down into short term goals using path planning. This module is 5 layer CNN with 3 dense layers. Local policy made of pretrained ResNet18 CNN followed by dense and GRU layers then takes navigational actions based on current state in order to reach this short-term goal. Figure 2.2 shows a sample run, where blue circles are long term goals, green patches are map predictions, red lines are agent pose predictions and grey areas the ground truth.

Figure 11 is the reward plot for this exploration algorithm. Reward here is proportional to the increase in coverage area. We see that once the models are trained, they perform quite well on unexplored scenes and are able to chart the map intelligently. This not only helps us with this experiment, rather makes it a potential feature to be included in our proposed Imitation learning algorithm too. For this we aim to stack maps with our input images and feed it into our network. We tend to incorporate this into our future work.

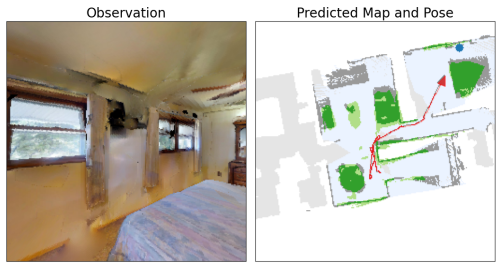

Figure 5. Sample Neural SLAM Run

2.3. Challenges and Issues

We tried to get our dataloader in place for imitation learning as the first step. There were two possible avenues to try for this, one that structured a dataloader as a RNN while the other that used a CNN regime. While using a CNN to encode state-action pairs did seem plausible, a demerit was that all temporal information about trajectories was lost. The CNN rendered state images independent of each other. A workaround for this is using additional feature engineering to encode the temporal information of trajectories. RNN, on the other hand, worked better as it preserved sequences of trajectories. Therefore, we decided to go for RNN dataloader.

Along with this, there were other challenges as well. To start with, while writing the data-loader for the input to training loop was that trajectories were of varying lengths. We had to pad them, which was accompanied with padding the corresponding actions. As the actions were being represented by one hot vectors, padding actions with zeros was problematic. To counter this issue, we padded actions with -1s and images with 0s. We then ignored the padded values while calculating CNN and RNN forward passes, cross entropy loss and accuracy.

A challenge with loading a large image data-set of the form (N,T,C,H,W) is the large GPU memory requirements needed. We encountered several stoppages during our training because of GPU memory overload issue. We worked around this issue by using a small batch size and ignoring trajectories greater than 120 steps in length. Moreover, resource limitations allowed us to train on a subset of training data.

The Matterport 3D scenes were modified on the Habitat repositories while researchers were working on making the meshes take less GPU memory. We believe modifications may have similarly been made for the Gibson scenes, because some scenes were not being rendered while we were running baselines on them. This incompatibility with more recent versions created issues during the course of the project. So we eliminated those scenes and went ahead with the working ones.

Installing and importing ORB-SLAM2 took an unexpectedly long time. There are open Github issues with possible solutions for this and we initially could not find a fix even after trying all of them. However, we were able to create a soft link between the required dependencies and resolve the problem without making changes to the cmake file (as most online solutions suggested). This is a fix that we believe would help other people as well.

2.4. Code repositories

We have used the following external code repositories in our work. We thank the authors of these works for making these publicly available.

- Habitat API [17] [1]
- Active Neural Slam [19]

3. Experiments and Results

For our baselines, we were able to experiment with the models used and the hyper-parameters tuned. For our PPO experiment we had leveraged the pre-trained models from [17]. Post this we did an extensive training and tuning of the hyper-parameters involving learning rates, optimizer, number of episodes, batch size, epochs, and learning decay. On similar lines we have trained our Active Neural SLAM module, where we used the pre-trained models for local policy, global policy and the neural slam module and carried a hyper-parameter tuning. For SLAM we have trained it from scratch and tried various hyper-parameters and have reported the best results.

Evaluation Metrics
We used the Success weighted by Path Length (SPL) metric to assess the performance of our approach. Intuitively, this suggests how accurately the agent was able to traverse a trajectory while staying on an optimal path[18]. For behavior cloning/imitation learning approach, we considered accuracy as a metric.

$$SPL = \frac{1}{N}\sum_{i=1}^{N} S_i \frac{l_i}{\max(p_i l_i)} \quad (2)$$

where l_i = length of shortest path between goal and target for an episode, p_i = length of path taken by an agent in an episode and S_i is the binary indicator of success in an episode.

The agent is considered to have successfully navigated through a trajectory if its position in the environment is within twice its radius of the target coordinates after calling the STOP action. We conducted our experiments and compared against recent baselines for point goal navigation. Experiments and results for each approach are detailed as follows:

3.1. Supervised Learning (Our Approach)

Our experiments were based on teacher forcing criteria. In these experiments, we aim to maximize the likelihood of getting the most optimal action $a^{\star}$ given a previous state-action sequence and the current ground truth label a. Teacher forcing always selects the current optimal ground truth label a for the prediction of future outputs.

3.1.1 Implementation

For encoding, we used Transfer Learning and extracted features from a pre-trained resnet-18 architecture previously trained for Image-net dataset [11]. We performed minimal data pre-processing. We cropped the images to a size of (3,224,224) and used the same transforms previously used while training image-net. We modified the resnet-18 architecture to remove the last fully connected layer. This block was called 'Feature Extractor'. We then added two fully connected layers with ReLU activation and BatchNorm. We named this block 'Linear Block' The purpose of adding a separate Linear Block was to learn the weights of this block and make better predictions about the extracted features.

Parameter	Value
Batch size	20
Trainloader Size	720
No. of epochs	100
Encoder Parameters	
CNN Embed dim	300
Dropout	0.3
Decoder Parameters	
Input size	300
Hidden size	140
No. of layers	3
Dropout	0.3
No. of classes	6

Table 1. Hyperparameters

We collectively learned the parameters of the encoder Linear Block and complete decoder architecture and used an Adam Optimizer for optimization. We used Pytorch to train our model for a fixed epoch of 30. We only trained on a subset of training data due to resource limitations. Our complete batch comprised of 100 batches with 720 trajectories in a single batch. We trained on the first batch for our experiments.

Table 1 presents our choice of hyper parameters for both encoder and decoder. We selected these hyper parameters using prior information available such as the output of resnet feature extractor and size of RNN input block.

3.1.2 Results

Our results 6 show decreasing training and validation loss for 200 epochs. Similarly, an increasing accuracy for training and validation suggests that the promise of supervised learning for point goal navigation. Loss and accuracy trends also reveal that the model is under fitting to the data. This is a reasonable and expected output since we have trained on a subset of the batch due to resource constraints. This also suggests that increasing the batch size would positively effect the training and validation accuracy and losses.

The validation loss and accuracy curve suggests that the model is able to generate good prediction for unseen data set. However, more experiments needs to be conducted to predict the performance on unseen data-set in real-time sim-

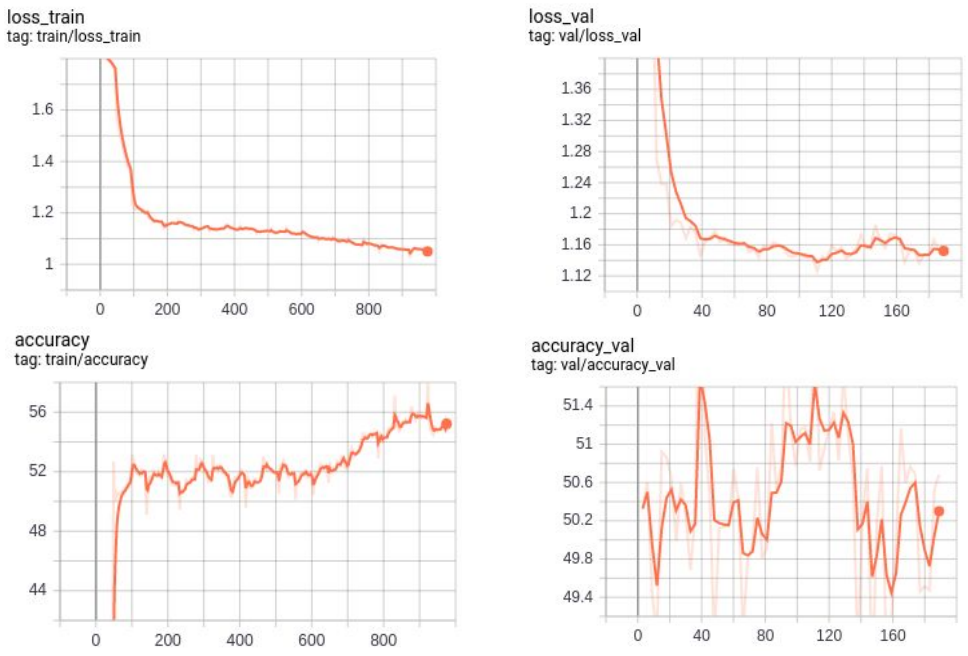

Figure 6. Training Loss/Validation Loss and Training Accuracy/Validation Accuracy for Supervised Learning

ulation environment where one wrong action can start a ripple effect on the overall navigation trajectory and may lead the agent astray.

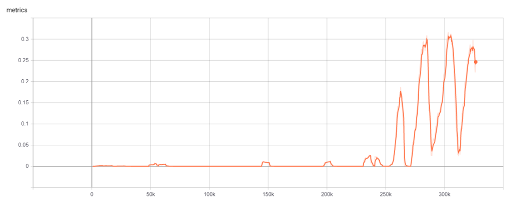

Figure 7. Baseline1: SPL plot for PPO

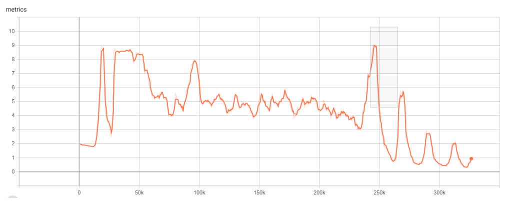

Figure 8. Baseline1: Distance to goal plot for PPO

3.2. Inferences for Baselines

- We see from the plots for PPO (figures 7, 8, 9) that the SPL and distance to goal was initially low but as the number of updates to the policy increased, these metrics on the validation set increased as well. We were

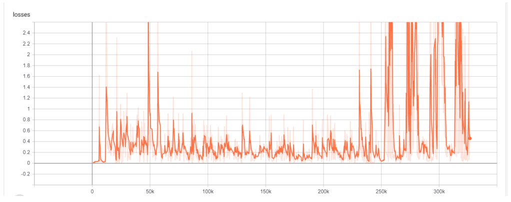

Figure 9. Baseline1: Loss plot for PPO

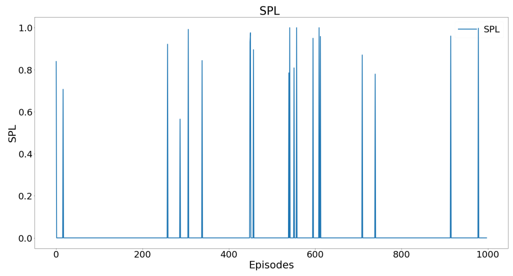

Figure 10. Baseline2: SPL plot for Classical SLAM

Method	Metric	Value
Behavior cloning	Accuracy	56%
PPO	SPL	0.3
SLAM	SPL	0.018
NeuralSLAM	Rewards	38-100

Table 2. Results

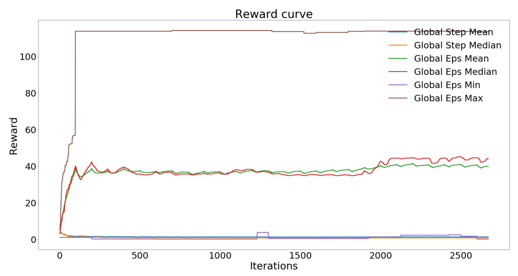

Figure 11. Baseline3: NeuralSLAM reward

able to achieve sharp increases for the SPL and we believe that if we had enough computational resources and time, we would have been able to train for longer and achieve even better results for the validation data.

- For the NeuralSLAM model, we see a sharp initial rise in the rewards (figure 11) but it slowly plateaus to a constant value. One of the possible reasons for this could be noise in the data or the agent hitting an obstacle. We saw that the agent was not able to navigate to the goal. The plot included is for one validation scene. We faced several technical issues in validating this approach on all scenes. Therefore, the results obtained are not conclusive to draw any general trends.

- An issue with SLAM is the reproducibility of results. Some of the episodes are such that the goals are unreachable. In this case, the algorithm fails because tracking is lost due to difficulty of environment mapping. The implementation resets the map in these scenarios, which is why we believe a range of 0 values have been reported for SPL (figure 10). These are indeed hard episodes that should be eliminated before training. Slam approach gave similar results for matterport data as well which has been highlighted here. https://github.com/facebookresearch/habitat-api/issues/270

4. Project Takeaways

Modelling to reflect structure of problem: The navigation task is inherently a time series data of information (in the form of images) as an agent traverses through an environment. This is a sequential task where future observations are dependent on previous actions. Using an RNN was appropriate here as input is handled in time steps and the agent can be equipped with short-term memory. As the agent moves, it needs to create and maintain internal representations of its environment, so it is important to use structures that can work with these representations.

Supervised Learning vs Reinforcement Learning Although Reinforcement learning has proven to be really successful in solving many problems, Supervised Learning can provide great advantages over Reinforcement Learning in our case. Faster time convergence, less sample efficiency and Sim2Real are some of the advantages of Supervised Learning over Reinforcement Learning for Navigation. However, lesser generalization capability makes it much harder for Supervised Learning to be effective in a wide range of tasks and scenarios.

5. Work Division

We all gained a working knowledge of the theory and technical aspects of each component of the project. We decomposed all the project work into parts and each took ownership of a specific task, providing help and insights into each others' work when required. Table 3 highlights these tasks for each member of the team.

6. Future Work

To fit the model better, the work needs to be expanded to train on the complete training dataset available in Habitat. Additionally, student forcing model needs to be developed for this work to make the training and test distribution similar. To improve the generalization performance to unseen data, a data-set aggregation algorithm needs to be developed and tested. Lastly, we propose using maps as additional features along with RGB images to the RNN model. A map provides a representation of an environment with information about obstacles and navigable paths. An example of such a map would be the Bird's eye view images (such as that generated by SPF in figure 3). However, the unavailability of pre-generated maps during test time is a challenge with this approach. Hence, we propose using ground truth maps as labels and using an intermediate architecture to predict local maps as the agent traverses an environment. This intermediate architecture would then feed into the local policy to predict optimal actions.

References

[1] Savva, Kadian, et al. "Habitat: A Platform for Embodied AI Research" arXiv preprint arXiv:1904.01201 (2019).

[2] Chaplot, Gandhi et al. "Learning to Explore using Active Neural SLAM" arXiv:2004.05155 (2020).

[3] Ye, Yang "From Seeing to Moving: A Survey on Learning for Visual Indoor Navigation (VIN)" arXiv:2002.11310 (2020).

[4] Ross, Gordon, Bagnell "A Reduction of Imitation Learning and Structured Predictionto No-Regret Online Learning" arXiv:1011.0686 (2011).

[5] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang et al., End to end learning for self-driving cars, 2016.

[6] Hochreiter, S., Schmidhuber, Jü. (1997). Long short-term memory. Neural computation, 9, 1735–1780.

[7] Kelly, Sidrane, et al. ”HG-DAgger: Interactive Imitation Learning with Human Experts” arXiv:1810.02890 (2019).

[8] Tolani, Bansal, et al. ”Visual Navigation Among Humans with Optimal Control as a Supervisor” arXiv:2003.09354 (2020).

[9] https://ai.googleblog.com/2019/02/long-range-robotic-navigation-via.html

[10] Zhu, Mottaghi, et al. ”Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning” arXiv:1609.05143 (2016).

[11] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248–255).

[12] Wijmans, Kadian, et al. ”DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames” arXiv:1911.00357 (2020).

[13] Xia, Zamir, He, et al. ”Gibson env: real-world perception for embodied agents” arXiv:1808.10654 (2018).

[14] Schulman, et al. ”Proximal Policy Optimization Algorithms” arXiv:1707.06347 (2017).

[15] Mishkin, Dosovitskiy, Koltun. ”Benchmarking Classic and Learned Navigation in Complex 3D Environments” arXiv:1901.10915 (2019).

[16] Artal, Tardos. ”ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras” arXiv:1610.06475 (2017).

[17] https://github.com/facebookresearch/habitat-api

[18] Anderson, Chang, et al. ” On evaluation of embodied navigation agents” arXiv:1807.06757 (2018).

[19] https://github.com/devendrachaplot/Neural-SLAM

Student Name	Contributed Aspects	Details
Asawaree	Implementation of SLAM	Habitat docker setup; Training, hyperparametering tuning for SLAM Testing SLAM Written report
Shrija	Implementation of Neural SLAM	Training, hyperparameter tuning, testing for NeuralSLAM; Analysis and Testing for Neural SLAM Testing SLAM metrics Result visualization for baselines Written report

Shubhangi	Implementation of PPO	Trained and fine-tuned hyperparameters for PPO algorithm;
		Analyzed results obtained;
		Initial data prerocessing for generating trajectories from gibson scenes and getting object annotations
		Written report

Zubair	Implementation of Supervised Learning Benchmark	Written code to get shortest path observations and ground truth actions for a trajectory Implemented variable length data-loader with padding and collating in pytorch Implemented Encoder Decoder Architecture in pytorch Implemented training pipeline and performed experiments to evaluate effectiveness of supervised learning Written report

Table 3. Contributions of team members.