# An A* Curriculum Approach to Reinforcement Learning for RGBD Indoor Robot Navigation

Kaushik Balakrishnan[1], Punarjay Chakravarty[1] and Shubham Shrivastava[1]

***Abstract*— Training robots to navigate diverse environments is a challenging problem as it involves the confluence of several different perception tasks such as mapping and localization, followed by optimal path-planning and control. Recently released photo-realistic simulators such as Habitat [1] allow for the training of networks that output control actions directly from perception: agents use Deep Reinforcement Learning (DRL) to regress directly from the camera image to a control output in an end-to-end fashion. This is data-inefficient and can take several days to train on a GPU. Our paper tries to overcome this problem by separating the training of the perception and control neural nets and increasing the path complexity gradually using a curriculum approach. Specifically, a pre-trained twin Variational AutoEncoder (VAE) [2] is used to compress RGBD (RGB & depth) sensing from an environment into a latent embedding, which is then used to train a DRL-based control policy. A*, a traditional path-planner is used as a guide for the policy and the distance between start and target locations is incrementally increased along the A* route, as training progresses. We demonstrate the efficacy of the proposed approach, both in terms of increased performance and decreased training times for the PointNav task in the Habitat simulation environment. This strategy of improving the training of direct-perception based DRL navigation policies is expected to hasten the deployment of robots of particular interest to industry such as co-bots on the factory floor and last-mile delivery robots.[2]**

## I. INTRODUCTION

To go from point A to B in an indoor environment is challenging for a mobile robot. In the absence of GPS and using the visual/RGBD sensor available on the robot, one has to map an environment & localize in it (SLAM) and then path-plan an obstacle-free route to get from a start to target location. This was the traditional approach to mobile robotics. Recently, Deep Reinforcement Learning (DRL) has shown to provide more robust navigation policies compared to SLAM, if the robot (agent) is trained in simulation and exposed to an order of magnitude more experience [1].

This involves training navigation policies that regress directly from the camera image to a control output. However, splitting this task into two: learning a compact state representation, termed "representation learning" and then using this representation to learn a robust control policy [3], [4], [5], [6], [7], [8], [9] has the following advantages: (1) errors in policy learning will not affect perception as the latter is decoupled from the former, but the vice versa is not true;

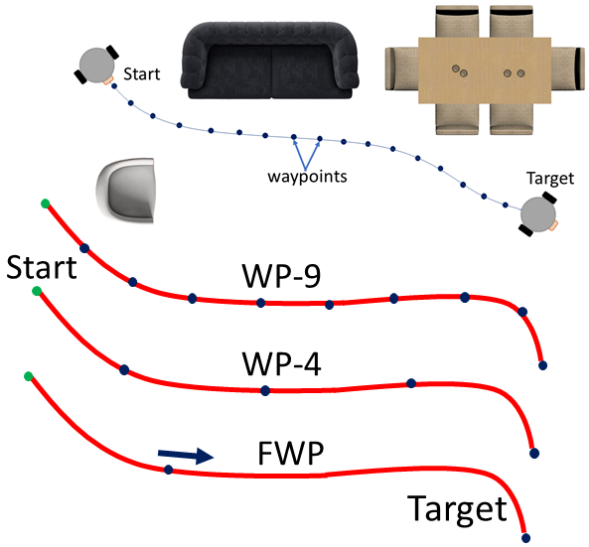


Fig. 1: Top: waypoints generated between desired start and target locations by the A* algorithm. This work looks at assisting the training of a DRL-based robot navigation policy, by incrementally increasing the difficulty of the navigation task in a curriculum. Bottom: 3 curriculum training approaches with 9 and 4 discrete waypoints and a continuously moving waypoint: WP-9, WP-4 & FWP.

(2) once perception is learned, it can be reused to learn multiple policies for different tasks [6], which is not feasible in complete end-to-end training as the perception needs to be re-learned every time a new task is learned. These advantages have the potential to speed-up the overall learning of the task at hand.

The recently released Gibson [10] and Habitat simulators [1] have generated excitement in the field of RGBD vision-based robot navigation in indoor environments. In SplitNet [6], the authors investigated three tasks in the Habitat environment: (1) Point-to-Point Navigation (PointNav); (2) Scene Exploration; (3) Run Away from Location (Flee). They decoupled the perception and policy, and used an Encoder-Decoder architecture, where the visual/perception encoder is trained using auxiliary visual and motion-based tasks, and the policy Decoder is trained on embodied tasks using the visual Encoder. They demonstrated robust learning of both perception and policy on all three tasks, including transfer to new visual environments as well as to new embodied tasks. In [11], the authors undertook Imitation Learning to train a robot to navigate in the Gibson simulator using

[1] The authors are with Ford Greenfield Labs, Palo Alto, CA, USA. kbalak18@ford.com, pchakra5@ford.com, sshriva5@ford.com

[2]More information and videos of robot navigation using our approach can be found at: https://www.towardsautonomy.com/Robot-Navigation-Using-Vision-Embedding

the Dijkstra algorithm and obtained high success rates. In another study, the authors used DRL for target-driven robot navigation in an indoor scenes simulator called AI2THOR [12], where only RGB images of the state and the target are used to train the navigation policy network. All these studies demonstrated robustness of DRL for robot navigation using large amounts of vision data, however techniques to speed-up DRL for vision-based navigation is warranted as most of these techniques are GPU-compute intensive and their speed-up can help in faster learning and deployment of robots in the real world.

In this paper, we build on these past investigations and train DRL agents for the problem of indoor robot navigation in the Habitat environment by separating perception (i.e., representation learning) and control (i.e., navigation policy). We use a VAE to encode RGB and Depth images, and use these latent encodings as well as a reading and heading angle for the target (from the PointGoal sensor), to learn navigation policies.

Additionally, we use a traditional path-planner, A* to assist the DRL agent during training. A* guides the agent by giving it shorter-distance goal locations (waypoints) between the original start and target locations. We experiment with two different curriculum-based training of the DRL agents, one by decreasing the number of intermediate waypoints used (termed the SWP-N agent) or by moving the episodic goal farther away from the start position (termed the FWP agent). We describe the problem and our method in Section III, implementation details in Section IV and an experimental analysis of these methods in Section V.

In summary, our contributions are as follows: (1) a principled approach to compare different navigation-agnostic VAE-based perception embeddings for their usefulness to a DRL in learning a subsequent navigation policy; (2) Using a traditional A* path-planning algorithm in a curriculum fashion to assist in the training process of this navigation policy.

## II. RELATED WORK

### A. Visual perception

Training end-to-end Vision-based DRL navigation policies can be very time consuming as the CNNs used to learn vision-based features involve several matrix operations, and this can particularly require several million images/experiences, accompanied by several days of compute hours, for training the DRL policy [1], [12]. End-to-end learning of DRL policies from RGB images has been very successful for Atari games [13], but for larger images such as the Habitat environment, it is very challenging [1], [14]. For learning robust navigation policies using RGBD images, it is critical to obtain good perception representation that is feature rich and compact enough.

We use the Variational Auto-Encoder (VAE) [2] for perception representation; VAE is a generative version of the vanilla Auto-encoder, that constrains the latent space $z$ to a Gaussian distribution with zero mean and unit variance $p(z) = \mathcal{N}(0, I)$. The encoder is used to produce the latent space encoding and the decoder takes in $z$ to reconstruct the input image. The VAE is trained using a combination of the reconstruction loss (typically, L2) and the KL divergence loss for the embedding to conform it to a unit Gaussian.

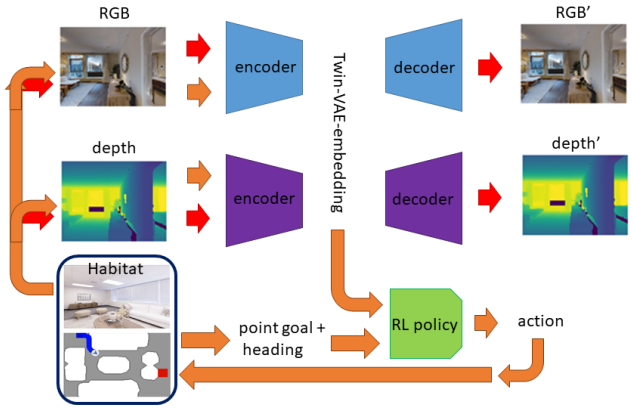


Fig. 2: A twin (RGB-depth) VAE learns an embedded representation of the environment, which is then used to train a navigation policy using DRL. Information flow during the VAE and DRL training are shown in red and orange respectively.

Using VAEs for representation learning, followed by a DRL control policy is not new, see for instance [5], [9]. The VAE is generally pre-trained, and so the perception is learned independent of control. In [6], the authors used a vanilla Autoencoder (AE) to compress the image, without the KL divergence loss. One difference between the two approaches is the that when VAE is used for perception, the resulting embedding is stochastic, i.e., the same input image fed to the encoder multiple times will result in different embeddings as they are sampled from the Gaussian; whereas in the vanilla AE this is not the case. Other approaches such as using shared latent spaces [15], [16] can also be considered in future studies.

Another challenge in vision-based robot navigation is on transfer learning to new targets and/or new scenes. To this end, [12] trained RGB-based DRL navigation policies for one or more scenes and used this to transfer learn (or fine-tune) to newer scenes. They showed that the DRL learns faster and the overall trajectory length is shorter if more scenes are used in the training. This improves the overall data-efficiency of the DRL training on newer scenes/targets. We will also briefly address transfer learning in the experiments conducted in this paper (section V).

### B. Robot navigation with A* and vision

Robot navigation using vision-based sensors is gaining renewed interest in the literature with the advent of state-of-the-art simulators [1], [10], [12], robust datasets [10], [17], and efficient deep learning algorithms. A goal-driven DRL framework for visual robotic navigation was provided by [18]. Robot navigation using a PointGoal, i.e., the position of the goal with respect to the current location of

the robot/agent, was used in [19], [1]; we undertake the PointGoal navigation task in this study, but with a curriculum that gradually increases the difficulty during training. A hierarchical method for navigation combining a sampling based path planning with DRL, called PRM-RL, was proposed in [20]. Their DRL agents were trained for short-range, point-to-point navigation capturing robot dynamics and task constraints without knowledge of the large-scale topology, while Probabilistic Roadmaps (PRMs) as sampling-based planners were used to provide roadmaps which connect robot configurations. Our hybrid path-planning/DRL is similar in spirit to theirs, but we use A* as our path-planner and use a curriculum-based training of the DRL agents.

Specifically, we undertake an investigation of using a small number of waypoints between the start and target locations for the DRL agent to successfully learn to navigate, as well as aiming for longer start-to-target distances as the learning progresses (more details on this in Section IV). Note that we are sequentially increasing the complexity of the navigation problem on the policy learned by the DRL agents, sticking to a pre-determined curriculum. Our approach is not imitation learning as the actions have to be learned by the DRL agents by exploration. Furthermore, despite the use of a two-level hierarchy for the policy, i.e., A* for waypoint determination (in training only) and a DRL policy for the navigation action, we are not undertaking Hierarchical Reinforcement Learning (HRL) for navigation in the spirit of [21]. HRL involves learning at multiple levels, whereas in our case the higher-level A* is a graphical search algorithm, and only the lower-level DRL policy involves learning from data.

## III. Problem Setup

**The PointNav task** We use the Habitat simulator [1] to train our DRL to learn policies for the point-goal navigation task in the Gibson environment [10]. The robot/agent is equipped with an RGBD camera, a point-goal sensor and a heading sensor. The point-goal sensor is like an indoor GPS: it provides the agent with its current position and the relative position of the target location. The heading sensor provides the current global heading angle of the agent. In the point-goal navigation task, the agent is asked to navigate from the initial starting position to the required end position using only its RGBD, heading and point-goal sensors and without a map. These start and target locations are randomly initialized at the beginning of each episode, for which no straight line path is possible. The agent needs to learn navigation strategies that avoid obstacles and negotiate doorways since the start and target locations can be in different rooms.

**Twin-VAE** We pre-train perception in the environment by using a twin-VAE setup as shown in Figure 2. RGBD cameras are initialized randomly in the environment, and at each location, RGB and depth images are collected at angular increments of $10^o$ for a full $360^o$ sweep. These images are used to train the RGB and depth encoder-decoder branches (blue and purple in the figure) with the standard VAE reconstruction and KL divergence losses [2]. Once the VAE is pre-trained, only its encoders are used for training the DRL. RGB and depth images are encoded to their respective embeddings, which are concatenated to provide the final visual embedding from the camera. This embedding is used for training the DRL policy.

**A* Curriculum Learning** The task of learning the DRL policy is assisted by incrementally increasing the difficulty of the PointNav task. We do this during training by using A* to determine an optimal path between start and target locations in the bird's eye view (BeV) map of the environment. A new sub-goal, a point to navigate to, that is on this A* path, is provided to the DRL. This sub-goal is close to the starting location to begin with, and then as training progresses, gets farther and farther away from it. We test the following variants of curriculum learning based on discrete and continuous subdivisions of the path:

1) **WP-N**: In Way-point-N or WP-N, the A* path is divided into N equidistant waypoints (WPs) including the target location. At the beginning of the training episode, the agent is asked to navigate to the first WP. When it reaches within 0.2m of this WP, the goal is revised to the next one and so on till the final target location. We investigate the number of intermediate waypoints required for successful navigation by experimenting with WP-10, WP-8, WP-6. WP-4, WP-3 and WP-2. WP-1 involves no subdivisions of the path and is the same as the original PointNav task.
2) **SWP-N**: Sequential WP-N or SWP-N involves keeping the number of WPs constant for a fixed number (few thousand) episodes. This is the same as WP-N, where the agent is asked to navigate from the 1st to the Nth waypoint within the same episode. However, N decreases episodically. Once the agent has mastered a higher N, requiring a smaller length sub-path traversal, the agent is subjected to a lower N, requiring a larger length sub-path traversal. For instance, the start-to-target path is divided into N waypoints for every 10k episodes, and N is decreased following the set: (10, 8, 6, 4, 3, 2, 1). Note that WP-1 is the same as PointNav.
3) **FWP**: Farther Waypoint involves only one WP that moves farther and farther away from the start in continuous, linear increments, as training progresses. The training is commenced with the WP at 20% distance along the A* path from the start. Over the course of training, this WP is moved farther and farther along the path until it is at 100% of the distance from the start to target after several tens of thousands of episodes, at which point the FWP problem is the same as the PointNav problem.

Note that SWP-N and FWP become PointNav agents by the end of their training. However WP-N is an agent that has only mastered a shorter navigation distance compared to the original PointNav agent. We use WP-N to set up the problem and demonstrate the efficacy of navigating smaller paths and using this to boot-strap the training of longer path (SWP-N and FWP) agents. Hence, SWP-N and FWP do not require A* at test time, whereas WP-N agents require A* at test time

to obtain intermediate goal locations. In this paper, we will focus more on SWP-N and FWP for this reason.

At any time instant $t$, let the sensory readings be denoted as follows: (1) RGB image, $s_t^{RGB}$; (2) Depth map, $s_t^{Depth}$; (3) pointgoal sensor reading, $PG_t$; and (4) heading angle, $H_t$. The pre-trained twin-VAE is used to encode $s_t^{RGB}$ to $z_t^{RGB}$ and $s_t^{Depth}$ to $z_t^{Depth}$. These are concatenated with the other two sensor readings to obtain a compact representation of the state at time t as: $s_t = (z_t^{RGB}, z_t^{Depth}, PG_t, H_t)$.

Once the sensory readings are concatenated into a compact $s_t$, we use Deep Reinforcement Learning (DRL) to learn a policy $\pi_\theta$ that outputs action $a_t$ at time t:

$$a_t = \pi_\theta(s_t), \tag{1}$$

where the actions are one of three: (1) move forward by 0.25 m; (2) turn left by $10^o$; (3) turn right by $10^o$. A fouth action called "Done" is executed whenever the agent is within 0.2 m from the goal position. Note that in [1], they trained an agent to also learn this trivial task, but this was not the case in [11], [6]; we take the latter approach. A schematic of the overall setup is shown in Fig. 2.

## IV. IMPLEMENTATION DETAILS

### A. Training the VAE

The RGB and Depth maps obtained from Habitat are 256×256 and are resized to 192×192 before being fed into the VAE for training. For the Encoder of the VAE, we use 10 layers of 4×4 filters with the number of feature maps per layer varying from 64 to 512. This is followed by fully-connected layers of size $N_z$ each to obtain $\mu_z$ and $\log \sigma_z^2$. The reparameterization trick is used to obtain z [2], the Encoder's embedding vector. For the Decoder, we use a mirror image of the Encoder to scale back to 192×192 size. For the latent code, we will consider a size of $N_z$ = 128 and 256 in this study. All layers use the Relu activation function, except the final output layer of the Decoder which uses a Sigmoid; the intermediate latent layer uses no activation functions to determine $\mu_z$ and $\log \sigma_z^2$ for the Gaussian latent code. We use a batch size of 64 and 50,000 iterations to train each of the VAEs, and use the Adam optimizer [22].

### B. Training the DRL Policy

The Proximal Policy Optimization (PPO) algorithm [23] is used to train the policy network. The input to the DRL algorithm comprises of the visual embedding plus the point-goal and heading data as detailed in the previous section (This state at time $t$ is given by $s_t = (z_t^{RGB}, z_t^{Depth}, PG_t, H_t)$. The policy network consists of two fully connected layers with 512 and 256 units and tanh activation function, followed by a LSTM layer [24] with 256 units. The output of the LSTM branches out into the policy and value streams, each going through a fully connected layer with 256 units and the tanh activation function. This is followed by a Softmax layer with 3 probabilities corresponding to the actions described earlier

For training the DRL agent, the reward function at time $t$, $r_t$, is same as [1] and is given by:

$$r_t = \begin{cases} S + d_{t-1} - d_t + \lambda & \text{goal reached} \\ d_{t-1} - d_t + \lambda & \text{otherwise} \end{cases} \tag{2}$$

where $d_t$ is the distance to the goal from the agent's current location at time $t$, $S$ = 10 is a bonus for reaching the goal, and $\lambda$ = -0.01 to penalize a stationary agent. The Adam optimizer [22] is used to train the policy network as well. Other PPO-related hyper-parameters are selected as follows: discount factor $\gamma$ = 0.95, PPO clip value $\epsilon$ = 0.1, and Generalized Advantage Estimation [25] parameter $\lambda$ = 1.0.

### C. Evaluation metrics

Similar to other Habitat-based navigation work, [1], [6], [11], we use the Success-weighted Path Length (SPL) for evaluating the policy learnt at the end of each episode:

$$SPL = S \frac{l}{\max(l, p)} \tag{3}$$

where $S$ is a binary indicator of success, $l$ is the shortest path length from start to goal position and $p$ is the path length traveled by the agent in the episode. We also evaluate the Mean Success Rate, which is the mean of $S$ over a fixed number of episodes.

## V. EXPERIMENTAL RESULTS

Our primary interest in this study is to undertake indoor robot navigation in the Habitat environment [1]. The *Quantico* indoor scene in the Gibson [10] dataset is used for the experiments, except for the Transfer Learning (TL) experiments, where we TL from *Quantico* to the *Pleasant* environment. Our experiments relate to the VAE latent encoding parameters, followed by quantitative and qualitative evaluations of the baseline PointNav agent compared to the agents (WP-N, SWP and FWP) trained by A* curriculum learning.

### A. Choice of latent encoding

As aforementioned, for representation learning, we use separate VAEs to encode the RGB and Depth images, concatenating the two encodings for subsequent use to train the DRL agents. For this we can use either the latent code $z$ sampled from the Gaussian $z \sim p(z)$, or we can directly use the mean of the Gaussian $\mu_z$. For instance, [5] uses the VAE's $z$ for the RL, whereas [9] uses $\mu_z$. Likewise, the dimension of the latent code can also have an effect on the training of the DRL agents. To better understand the choice of the VAE encoding that would result in a robust encoding of the visual inputs for efficient training of the DRL agents, we consider three cases: (1) $N_z$ = 256; $\mu_z$, (2) $N_z$ = 256; $z$, and (3) $N_z$ = 128; $z$. Separate DRL agents are trained on the PointNav problem using these three different choices and the exponentially-averaged (with degree of weighting $\alpha$ = 0.001) SPL during the training of the agents are presented in Fig. 3. As evident, $N_z$ = 128 performs better than 256, which we believe is due to over-fitting when the latent code dimension is large. Likewise, the training is better when $z$ is used in lieu of $\mu_z$, as the stochasticity involved in $z$ acts as

a regularizer and can be a good source of exploration noise. For the rest of this paper, we will use $N_z = 128$ and $z \sim p(z)$ as the encoding for training the DRL agents.

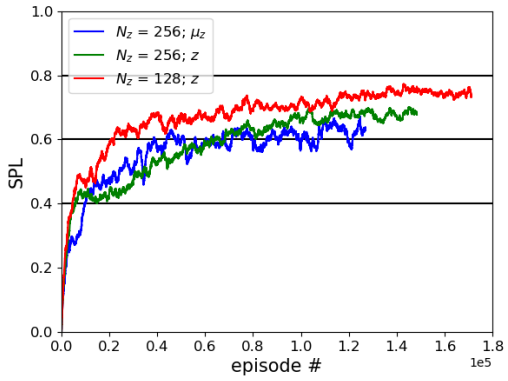


Fig. 3: Effect of the latent encoding on the training of DRL agents.

### B. Navigation results

We will now summarize results from the different cases of interest as identified earlier. The best performing PointNav agent is used as the baseline (this is the agent corresponding to the red curve in Fig. 3). Note that we will only compare this PointNav baseline performance with SWP and FWP agents as these two agents are essentially PointNav at the end of the training. Performance of the WP-N agents are presented only for demonstration that shorter paths lead to improvement in learning; note that WP-N agents are not desired for deployment as they still require the help of A* at test time. Thus, the test time performance of WP-N agents are not presented or compared with other agents as we are only interested in agents that use A* in training but not at test time.

*1) PointNav:* For the PointNav problem, the best performing agent is the $N_z = 128$ and $z$ sampled from the Gaussian $z \sim p(z)$, i.e., the agent corresponding to the red curve in Fig. 3. We test the agent's performance over 500 random test episodes. The only difference between training and test mode is that in training the action is sampled from the policy's softmax output, but in test mode, the greedy action is chosen. The best performing PointNav agent (red curve in Fig. 3) has a test time performance of mean SPL = 0.73 and mean success rate = 0.852. We will use these values as the baseline for comparing the other agents with.

*2) WP-N:* We present the training curves for the different WP-N cases considered, along with the best performing PointNav agent in Fig. 4. For WP-N, the SPL is computed as a piecewise average over the individual sub-goals. The availability of the intermediate waypoints has significantly improved the performance for the WP-N cases vis-á-vis the PointNav case (refer to Fig. 4), as the overall problem of navigation from start to goal has been simplified. With fewer

| Case name | SPL | Success rate |
|---|---|---|
| PointNav | 0.73 | 0.852 |
| WP-10 | 0.92 | 0.998 |
| WP-8 | 0.9 | 0.992 |
| WP-6 | 0.9 | 0.994 |
| WP-4 | 0.88 | 0.984 |
| WP-3 | 0.88 | 0.983 |
| WP-2 | 0.85 | 0.988 |

TABLE I: SPL and success rate for WP-N agents

number of intermediate waypoints, the agents take longer to achieve an average SPL of over 0.8, which is as expected.

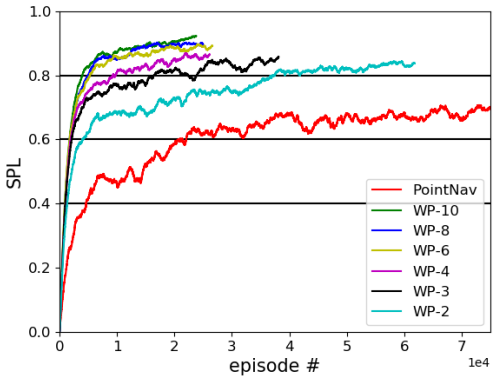


Fig. 4: Training performance for the WP-N agents. The PointNav agent (red curve) was run till $\sim$ 170k episodes (see the red curve from Fig. 3) but is presented here only till 75k episodes for better comparison with the other agents presented in the figure.

The averages of the SPL and the success rate of the agents for 500 test episodes are summarized in Table *I*. The success rate is over 0.98 and the SPL over 0.85 for all the WP-N agents but not the PointNav agent. This investigation proves that only a small number of intermediate waypoints suffices, and a DRL policy can make efficient use of it to learn to navigate much faster than the PointNav cases. This has important implications for robot navigation applications, as a small handful number of intermediate goal points to visit between the start and target can be generated even with a low resolution map, and one can then combine it with a DRL-based navigation policy and still obtain high accuracy. From Fig. 4, we observe that even 1-2 intermediate waypoints (WP-2 and WP-3) between the start and target positions suffices. However, as aforementioned, these WP-N agents are not preferred for deployment/test time as they still require the use of A*. We will now consider the hybrid training practice of using WP-N agents in training and following a curriculum (i.e., SWP and FWP agents) to gradually transition them to the PointNav agents so that they can be deployed without any further use of A* at test time.

*3) SWP-N and FWP:* Having established that a few intermediate waypoints between the start and target locations suf-

| Case name | SPL | Success rate |
|---|---|---|
| PointNav | 0.73 | 0.852 |
| SWP-10 | 0.91 | 0.96 |
| FWP | 0.77 | 0.9 |

TABLE II: Test time SPL and success rate for SWP-10 and FWP agents

fice, we will now investigate the SWP-N and FWP problems, both of which involve curriculum-based training. Note that WP-N agents require waypoints at both training test time, whereas the SWP-N and FWP agents require intermediate waypoints only during training, and are thus preferred. For SWP-10, we follow the curriculum: WP-10($<$10k), WP-8(10-20k), WP-6(20-30k), WP-4(30-40k), WP-3(40-60k), WP-2(60-80k) and WP-1($>$80k), where the number in parenthesis denotes the episode number range. (Note: WP-1 is identical to PointNav). For the FWP agent, we start the training with the revised goal set as the 20-th percentile of the start-to-goal A* path, and gradually increase it linearly over the course of 64k episodes to the 100-th percentile of the A* path. The FWP problem also reduces to the PointNav problem at the end of the training as the waypoint the agent is shooting for coincides with the target location (for episode number $>$ 64k).

The training curves are presented in Fig. 5 for the SWP-10, FWP and the baseline PointNav agents. We observe that SWP-10 agent achieves high SPL values in the early stages of the curriculum, but drops in performance as the curriculum is more difficult at the later stages of the training. On the other hand, the FWP agent has learned to achieve SPL values of $\sim 0.8$ and maintains the same level of performance. Both the SWP-10 and FWP agents (both of which are essentially PointNav at the end of the training) maintain superior performance over the PointNav agent. Furthermore, both SWP-10 and FWP agents also learn much faster than the PointNav agent.

The averages of SPL and success rates over 500 test time episodes for the SWP-10 and FWP agents are summarized in Table *II*. Note that at test time, the greedy action from the policy probabilities is used, instead of sampling. Both SWP-10 and FWP agents maintain a success rate of over 0.9 and the SPL is also higher for these agents compared with the PointNav agent. The SWP-10 agent performs the best among all the agents and achieves an average SPL of 0.91. The SWP and FWP agents also learn much faster than the PointNav agent (see Fig. 5). Thus, learning with a curriculum assists in learning a better policy and also faster (measured in terms of number of training episodes).

### C. Comparison of different agents

We now compare the test time performance of the PointNav, SWP-10 and FWP agents for a few test episodes. The paths traced by the agents, including the start and target positions, for 8 episodes at test time are shown in Fig. 6. The observations reported here are based on performance

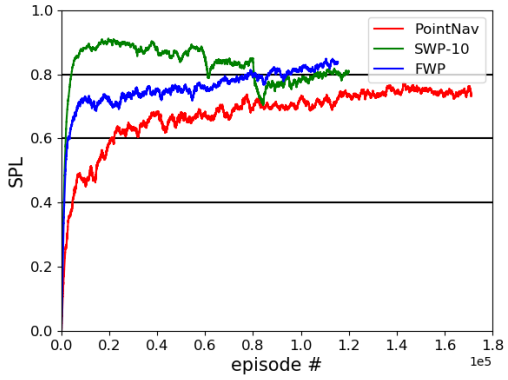


Fig. 5: Training performance for the SWP-10 and FWP agents.

over several more test episodes, but we will discuss only these 8 chosen test episodes for brevity; we will use the term "episode-a" to refer to Fig. 6 (a) and so on for the other episodes as well. In episodes-a and b, the PointNav (shown in red) fails to reach the target position. We noticed many episodes where the PointNav agent failed closer to the start position than otherwise; this agent has starting trouble in such episodes. On the other hand, agents SWP and FWP successfully navigated to the goal for episodes-a and b, with the SWP agent being smoother for episode-a and FWP for episode-b.

In many episodes, we also observed the PointNav agent to reach close to the target, only to overshoot it and get confused, but the agent was still able to reach the target, albeit with an unnecessarily longer path. This is reflective in episodes-c, d, e, and f (notice in the vicinity of the target, shown by the blue square). The SWP and FWP agents perform better in these episodes and successfully reach the target in shorter paths than the PointNav agent.

The FWP agent is coasting more than the SWP, as evident from episodes-a, e, and f, where we observe the FWP agent to stay closer to walls and make near $90^o$ turns. In these episodes, the SWP agent has relatively smoother paths, which we desire for real word deployment. However, in episodes-b and g, the reverse is observed–the SWP agent (shown in yellow) is coasting more than the FWP agent. This is an interesting behavior in that one agent is not "better" than the other agent in all test episodes, and the location of the start and target positions has an influence on the agents' performance. This behavior is observed in other episodes as well (not presented in the Figure), and the choice of the curriculum used in the training has a significant influence in the final test time performance. This has implications to real world robots that are trained using a DRL-based navigation policy following a curriculum.

In episode-h, the PointNav agent actually performs better than the other two agents, which is a very rare outcome. For this episode, the FWP agent (in purple) actually fails to reach the target and is stuck in a corner. Another interesting

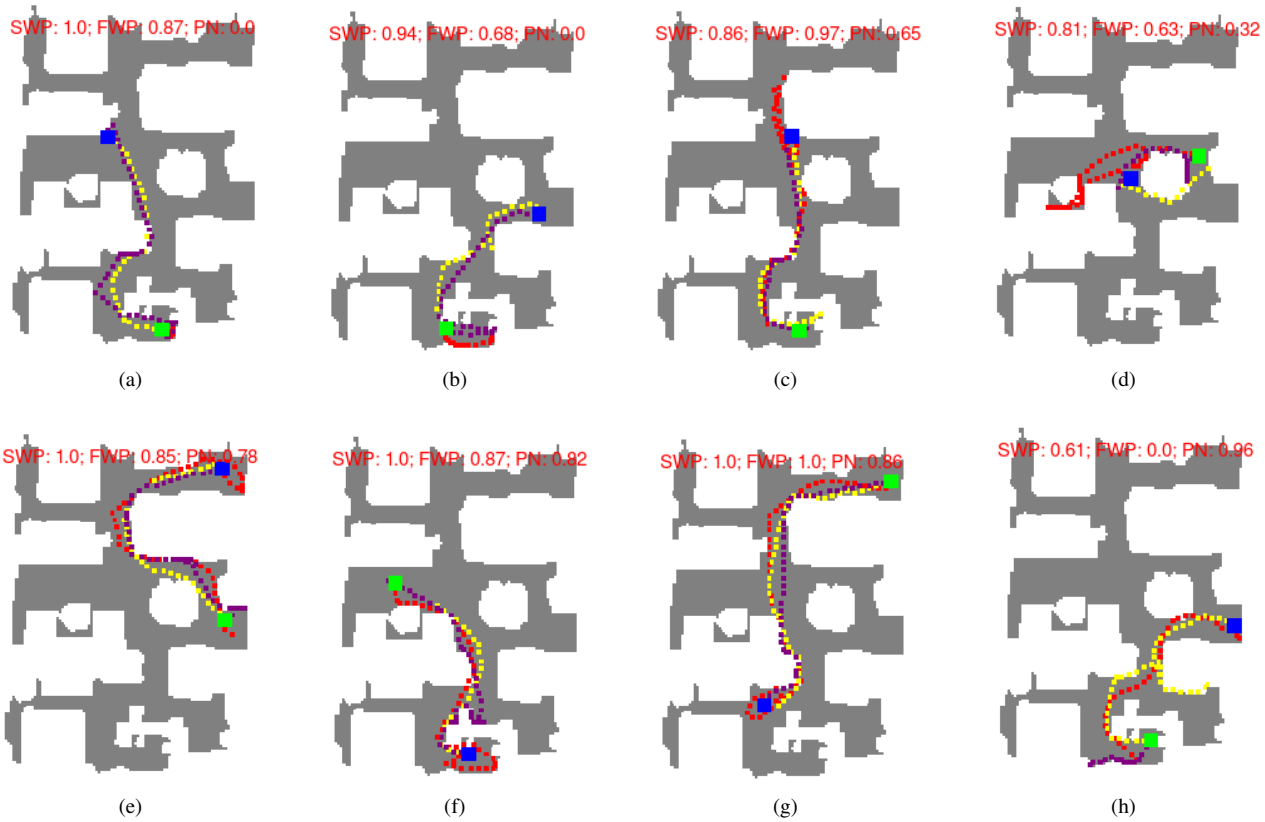


Fig. 6: Test time paths traced by the PointNav, SWP-10 and FWP agents for different episodes. The start and target positions are represented by the green and blue squares respectively. The paths traced are shown in red for PointNav, yellow for SWP and purple for the FWP agents. The SPL values for the three agents for the respective episode are also shown in each sub-figure. We recommend that this figure be viewed in digital format and zoomed in for better clarity.

observation is that episodes b and h have start locations somewhat close to each other, as well as target locations, but the FWP agent (shown in purple) is able to successfully navigate episode-b but not episode-h. This, however, is quite rare an occurrence from our observation of the other test episodes as well (not presented here). Thus, agents can fail for even a slight increase in the level of difficulty at times.

### *D. Transfer Learning (TL) Experiments*

One of the test of robustness of a learned policy on one task is its ability to generalize to another, but related, task. To this end, we undertake TL of VAE representations and DRL policies from *Quantico* to the *Pleasant* scene in the Gibson dataset [10]. Both scenes correspond to indoor homes with living rooms, bedrooms, etc., and have similar, albeit not same, furniture/appliances. Similar tests were also undertaken by [12] for evaluating TL for robot navigation. The VAE is fine-tuned for *Pleasant* using the pre-trained VAE weights from *Quantico*. The SWP-10 trained for *Quantico* is used as starting policy network weights and fine-tuned on *Pleasant* following a PointNav approach (i.e., without any A* curriculum), and this agent after transfer learning (TL) on the new scene is referred to as TL-SWP. After re-training for $\sim$ 60k episodes on the new scene, on 500 test episodes, TL-SWP has a mean SPL of 0.87 and a mean success rate of 0.95. Thus, with fewer episodes of re-training on the new scene, TL achieves reasonable performance. In future work, we expect to conduct a more systematic TL investigation on Habitat scenes similar to [12], where the number of scenes used in the first training was shown to have an impact on the transfer learning performance.

## VI. CONCLUSIONS

We look at the task of point-to-point navigation (PointNav) in the Habitat environment [1]. An agent gets photo-realistic RGBD images from this environment and learns an optimal navigation policy using Deep Reinforcement Learning (DRL). We introduce two ways of improving, in terms of speed and final metrics, the DRL training process: (1) We separate the perception and control tasks by pre-training a VAE to learn an embedding for the RGBD data. (2) We use A*, a traditional robotic path planning algorithm like training wheels on a bike that eventually come off, and train the DRL algorithm in an incremental curriculum. This involves increasing the path length required to successfully complete the PointNav task. We experiment with two curricula: one that increases the path length in a continuous fashion, and

another that does this discretely. At test time, we do not need the help of A* any more.

Our agents learn faster compared to the PointNav baseline as well as achieve higher final SPL scores and success rates. The perception embedding is trained once for an environment, and can then be used for a variety of agent policies. The training of a number of sub-policies allows these policies to be used independently for shorter-distance navigation, or to be used to build up more sophisticated long range navigation policies. The ability to scan a real-world environment and set up its photo-realistic virtual twin for training direct perception-to-navigation policies is expected to accelerate the deployment of mobile robots around the homes and factories of our future. Our work is a step improvement in this direction.

In future work, we plan on using conventional local path planning approaches to augment the reward function for DRL and try to imbibe the network with a neural map [26].

## REFERENCES


[1] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A platform for embodied ai research," *Arxiv: https://arxiv.org/abs/1904.01201*, 2019.

[2] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[3] S. Grossberg, F. Guenther, D. Bullock, and D. Greve, "Neural representations for sensory-motor control, ii: Learning a head-centered visuomotor representation of 3-d target position," *Neural Networks*, vol. 6, no. 1, pp. 43–67, 1993.

[4] T. Lesort, N. Diaz-Rodriguez, J.-F. Goudou, and D. Filliat, "State representation learning for control: An overview," *Neural Networks*, vol. 108, pp. 379–392, 2018.

[5] D. Ha and J. Schmidhuber, "World models," *Arxiv: https://arxiv.org/abs/1803.10122*, 2018.

[6] D. Gordon, A. Kadian, D. Parikh, J. Hoffman, and D. Batra, "Splitnet: Sim2sim and task2task transfer for embodied visual navigation," *Arxiv: https://arxiv.org/abs/1905.07512*, 2019.

[7] O. Nachum, S. Gu, H. Lee, and S. Levine, "Near-optimal representation learning for hierarchical reinforcement learning," *Arxiv: https://arxiv.org/abs/1810.01257*, 2019.

[8] M. Tschannen, O. Bachem, and M. Lucic, "Recent advances in autoencoder-based representation learning," *Arxiv: https://arxiv.org/abs/1812.05069*, 2018.

[9] A. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine, "Visual reinforcement learning with imagined goals," *Arxiv: https://arxiv.org/abs/1807.04742*, 2018.

[10] F. Xia, A. Zamir, H. Zhi-Yang, A. Sax, J. Malik, and S. Savarese, "Gibson env: Real-world perception for embodied agents," *CVPR, Arxiv: https://arxiv.org/abs/1808.10654*, 2018.

[11] D. Watkins-Valls, J. Xu, N. Waytowich, and P. Allen, "Learning your way without map or compass: Panoramic target driven visual navigation," *Arxiv: https://arxiv.org/abs/1909.09295*, 2019.

[12] E. Kolve, R. Mottaghi, W. Han, E. Vanderbilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, "Ai2-thor: An interactive 3d environment for visual ai," *Arxiv: https://arxiv.org/abs/1712.05474*, 2017.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *Arxiv: https://arxiv.org/abs/1312.5602*, 2013.

[14] E. wijmans, A. Kadian, A. Morcos, s. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra, "Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames," *ICLR: https://openreview.net/forum?id=H1gX8C4YPr*, 2020.

[15] M.-Y. Liu, T. Breuel, and J. Kautz, "Unsupervised image-to-image translation networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 700–708.

[16] P. Chakravarthy, P. Narayanan, and T. Roussel, "Gen-slam: Generative modeling for monocular simultaneous localization and mapping," *Arxiv: https://arxiv.org/abs/1902.02086*, 2019.

[17] A. Chang, A. Dai, T. funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3d: Learning from rgb-d data in indoor environments," *International Conference on 3D Vision (3DV)*, 2017.

[18] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," *Arxiv: https://arxiv.org/abs/1609.05143*, 2016.

[19] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, and A. R. Zamir, "On evaluation of embodied navigation agents," *Arxiv: https://arxiv.org/abs/1807.06757*, 2018.

[20] A. F. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia, "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," *Arxiv: https://arxiv.org/abs/1710.03937*, 2018.

[21] A. Levy, G. Konidaris, R. Platt, and K. Saenko, "Learning multi-level hierarchies with hindsight," *Arxiv: https://arxiv.org/pdf/1712.00948.pdf*, 2019.

[22] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Arxiv: https://arxiv.org/abs/1412.6980*, 2014.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *Arxiv: https://arxiv.org/abs/1707.06347*, 2017.

[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9(8), pp. 1735–1780, 1997.

[25] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *Arxiv: https://arxiv.org/abs/1506.02438*, 2016.

[26] D. S. Chaplot, S. Gupta, D. Gandhi, A. Gupta, and R. Salakhutdinov, "Learning to explore using active neural mapping," *ICLR*, 2020.