AN ABSTRACT OF THE THESIS OF

Saurabh Satish Desai for the degree of Master of Science in Computer Science presented on December 9, 2021.

Title: Auxiliary Tasks for Efficient Learning of Point-Goal Navigation

Abstract approved: _____
Stefan M. Lee

Top-performing approaches to embodied AI tasks like point-goal navigation often rely on training agents via reinforcement learning over tens of millions (or even billions) of experiential steps – learning neural agents that map directly from visual observations to actions. In this work, we question whether these extreme training durations are necessary or if they are simply due to the difficulty of learning visual representations purely from task reward. We examine the task of point-goal navigation in photorealistic environments and introduce three auxiliary tasks that encourage learned representations to capture key elements of the task – local scene geometry, transition dynamics of the environment, and progress towards the goal. Importantly, these can be evaluated independent of task performance and provide strong supervision for representation learning. Our auxiliary tasks are simple to implement and rely on supervision already present in simulators commonly used for point-goal navigation. Applying our auxiliary losses to agents from prior works, we observe a $>4\times$ improvement in sample efficiency – in 17 million steps, our augmented agents outperforms state-of-the-art agents trained for 72 million steps.

Auxiliary Tasks for Efficient Learning of Point-Goal Navigation

by

Saurabh Satish Desai

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented December 9, 2021
Commencement June 2022

Master of Science thesis of Saurabh Satish Desai presented on December 9, 2021.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Saurabh Satish Desai, Author

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Stefan Lee for being patient with me during the course of this work. I thank him for his guidance throughout the last two years. I would also like to thank Dr. Alan Fern, Dr. Prasad Tadepalli and Dr.Cory Simon for agreeing to be on my committee. I had the pleasure to interact with some very talented people at OSU and am grateful for their inputs/assistance.

I would like to thank my parents and my brother for their constant support and motivation. Finally, I would like to express my gratitude towards my grandparents for their blessings.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

Figure Page

LIST OF TABLES

Table Page

LIST OF ALGORITHMS

Algorithm Page

Chapter 1: Introduction

Recently, there has been a growing academic interest in embodied AI agents functioning in simulated, photorealistic environments. These agents have been applied to tasks such as navigating to point goals [1][2][5][9][13], following instructions [6][17] and answering questions based on visual observations [26].

In this work, we focus our attention to the task of point-goal navigation as specified in Savva et. al., [2]. In this task, an agent is spawned at a random location in an unseen indoor environment and is expected to navigate to a goal location specified in relative Euclidean coordinates. The agent cannot always travel in a straight line to the point-goal as there might be walls and other obstacles such as tables, sofas etc. as in any indoor setting. The agent has no access to a map of the environment but has GPS+Compass that provides pose information at each time step. It must navigate to the point-goal using visual observations and pose information alone.

At each training step, the agent receives a reward from the environment for the action it takes for progressing towards goal. The agents trained in such a setting must learn a policy to map visual observations to actions. This is most commonly framed as a large-scale reinforcement learning problem that requires millions or even billions of experiential steps. The agent is assessed by its ability to reach the point-goal using the most optimal path. The state-of-the-art approach to point-goal navigation, DD-PPO [27], achieves nearly perfect performance after 2.5 billion simulation steps (approximately 80 years of real-world experience) using a neural architecture. It takes about 6 months to train such an agent on one standard GPU.

We hypothesize that one reason for this slow learning can be attributed to learning internal state representations solely from task rewards. There are two interdependent problems involved while training a competent agent. The first being able to represent observations in useful way and second being able to use this representation to make

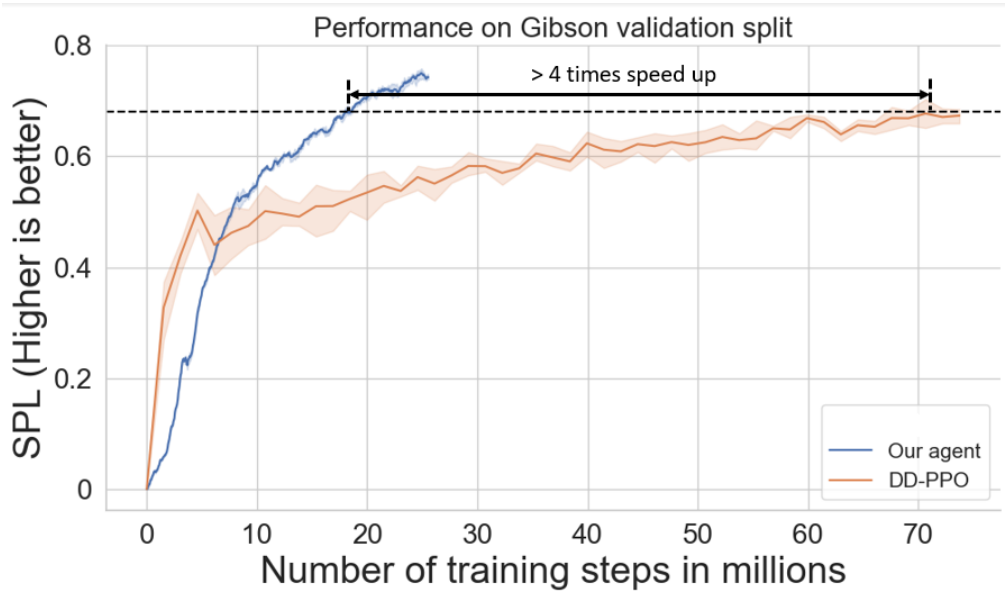


Figure 1.1: Learning to navigate from visual inputs is challenging. Agents must learn useful visual representations from task-feedback alone. In this work, we introduce auxiliary tasks that significantly improve the sample efficiency of training point-goal navigation agents. In the validation performance plot above, our agents achieve an SPL of 0.68 in 17 million steps which takes DD-PPO almost 72 million – a $\times 4$ improvement in sample efficiency.

correct decisions. Without a strong task-relevant representation of observations, it is difficult to take correct actions. Without positive reinforcement from the environment for correct actions, it is difficult to learn a strong, task-relevant representation of observations. With just task rewards, it takes a substantial amount of experience to resolve this dependency and learn a good agent. If we can accelerate the process of representation learning, we might be able to shortcut the dependency and train the agent in fewer steps. In our work, we try to see if this can be made possible by training the agents along with auxiliary supervision.

Simulated environments such as Habitat Savva et al. [21] are filled with task-related signals that could be used for auxiliary supervision. In this work, we choose to use following three auxiliary tasks :

- **Depth Prediction** – predicting coarse estimates of pixelwise depth given the visual observation. Depth is very useful for navigation as it helps determine free space.
- **Inverse Dynamics** – predicting action between two consecutive steps. It is good for an agent to know the consequence of its own action.
- **Remaining Path Length Prediction** - predicting the geodesic distance to goal from current location. Having a sense of actual distance that needs to be traversed to get to the goal is useful.

Each of these auxiliary tasks are implemented as small auxiliary networks. These networks transform some form of intermediate agent state representation to auxiliary task predictions which are then used to compute the task loss using auxiliary signal from environment.

In this work, we demonstrate that the addition of each of these auxiliary tasks enhances the sample efficiency of point-goal training. We use agents from Savva et. al., [22] as our baseline. We observe a $\times 5$ improvement in sample efficiency – task-augmented agents after 15 million experiential steps perform as well as or better than the baseline agents trained for 75 million steps. We compare our task-augmented agents with the state-of-the-art agents trained using DD-PPO [27] algorithm, which uses a more sophisticated neural architecture and a decentralized distributed training regime. Our best task-equipped agent is $\times 4$ more sample efficient than the DD-PPO as shown in Fig. 1.1.

Chapter 2: Related work

2.1 Auxiliary Tasks in Reinforcement Learning

Poor sample efficiency is a common problem for reinforcement learning – especially for model-free agents like those commonly used for pixel-to-action tasks like point-goal navigation. As such, prior work has explored applying auxiliary tasks to boost training efficiency and improve performance. In [14], Jaderberg et al. introduced a suite of unsupervised auxiliary tasks for an RL agent in Atari games and simple maze environments. These unsupervised objectives do not leverage additional signals provided by the simulator – in contrast, our auxiliary losses include predicting additional observation modalities (RGB→Depth) and aspects of the task itself (remaining path length).

Recent work has also examined auxiliary tasks to promote 'curiosity', improving the exploration abilities of agents. Pathak et al. [20] does this by providing intrinsic reward to agents for taking actions which lead to predictable changes in the observation encoding. [20] learns to encode these observations via an inverse dynamics task like ours; however, their inverse dynamics model is separate from the agent and is just used to compute intrinsic 'curiosity' rewards. As such, they do not use this as an auxiliary loss for the agent's observation representation. There are also a number of works focusing on forward-prediction as a means to learn representation encoders [12, 11, 19]. A common challenge in these frameworks is how to determine the quality of future predictions, a task made even more complex in the perceptually rich environments we consider here.

Most relevant to our auxiliary tasks is the work of Mirowski et al. [18] who studied two auxiliary tasks for navigation in simple 3D maze environments – depth prediction and loop closure detection. As we are not operating in maze-like environments, we do not consider the loop closure tasks. We adapt [18]'s depth prediction task to our setting – significantly altering the architecture to fit the needs of our more visually complex setting. We compare against the original structure as a baseline and significantly outperform it.

2.2 Point-Goal Navigation

The point-goal navigation task is a fundamental embodied AI problem on which other complex tasks can be built. As such, it has received significant attention as interest in embodied tasks has grown [21, 23, 8, 27, 2, 1]. A variety of approaches have been proposed including those with strong inductive biases from simultaneous-localization-and-mapping research [8] to methods that draw purely from deep reinforcement learning [21, 27]. [27] demonstrates near-perfect performance on this task by training a neural architecture for over 2.5 billion steps of experience – amounting to over 80 years of continuously exploring simulated environments. We focus instead on improving sample efficiency for these agents. Similar in goal, [23] studies the use of pretrained mid-level representations as input signals for point-goal navigators – demonstrating improvements in sample efficiency. In contrast, we do not assume access to relevant pretrained encoders and instead seek to train a model from scratch efficiently.

Chapter 3: Point Goal Navigation

In this chapter, we will first review the preliminaries of point-goal navigation. Our work follows the definition of point-goal navigation from Savva et. al., [22]. Point-goal navigation is a task where an agent is spawned at a random location in an unseen photorealistic environment and is expected to navigate to a point-goal. The point-goal is specified in Euclidean coordinates with respect to the agent' starting position. We assume that the agent gets its pose information at each time step (i.e. it has GPS+Compass) and hence can determine the relative goal coordinates at each position. Also, the agent do not have a map of the environment. The agent has visual sensors (i.e. RGB or Depth camera) to observe its surroundings. Thus, the input to the agent at each time step t is visual observation I_t and goal coordinates g_t. The agent can turn or move forward. The agent can also call `stop` to end the episode. Concretely, the actions that can be taken are - {left 15°, right 15°, forward 0.25m, stop}. The objective of this task is to learn the best policy π which can map the observed input to a distribution over actions a_t so that the agent reaches the goal position efficiently. In our case, we use neural architecture to learn this policy using PPO algorithm [25]. PPO is a popular policy gradient approach to solving reinforcement learning problem. We also use Generalized Advantage Estimation, Schulman et al. [24], with PPO.

3.1 Visual observation

The agents have a color vision sensor located at a height of 1.5m from the center of the agent's base and oriented to face 'forward'. This sensor provides RGB frames at a resolution of 256^2 pixels and with a field of view of 90 degrees. In addition, an idealized depth sensor is available, in the same position and orientation as the color vision sensor. The depth sensor gives the pixel-wise distance of the objects in front of the agent. The field of view and resolution of the depth sensor match those of the color vision sensor. We designate agents that make use of the color sensor by RGB, agents that make use of the depth sensor by Depth, and agents that make use of both by RGBD. Agents that

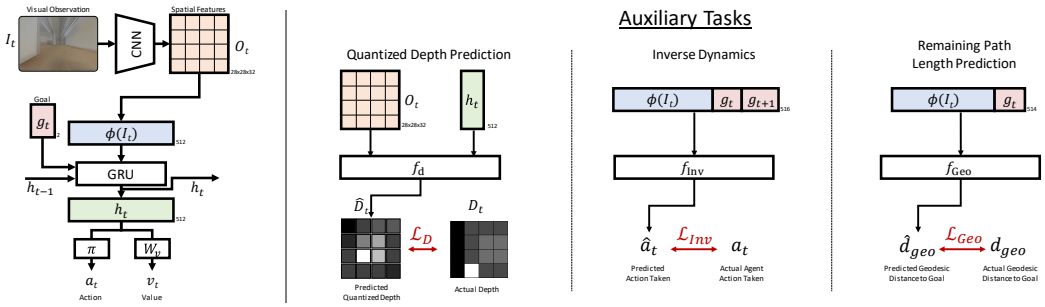


Figure 3.1: Our agent architecture (left) includes a simple convolutional observation encoder and a GRU-based recurrent policy. We consider three auxiliary tasks (right) – depth prediction, inverse dynamics, and remaining path length prediction – based on representations from the policy. Gradients for auxiliary task losses are propagated to corresponding agent modules, providing additional supervision to learned representation.

use neither sensor are denoted as Blind. The performance of baseline agents for different input modalities are shown in Fig. 3.2. The metric SPL (discussed later in Sec. 2.3.4) is a standard metric that measures the ability of an agent to reach the point-goal using the most optimal path. The depth input seems to be the clear winner over the RGB agent. Surprisingly, the RGB agent only narrowly outperforms the blind agents. This suggests that the RGB information may not be adequately utilized and there might be some scope of improvement for RGB agents.

3.2 Point-Goal Agent architecture

We use the neural agent architecture from Savva et. al., [22] for our baseline policy. The model consists of a visual encoder followed by a recurrent state encoder. The visual encoder $\phi(\cdot)$ is a simple CNN which consumes the visual observation I_t and produces an embedding $\phi(I_t)$. Further, this embedding is concatenated with relative goal coordinates g_t and passed to the recurrent state encoder $f(\cdot)$ (implemented as a Gated Recurrent Unit (GRU) [10]) to produce an encoded state representation h_t. This representation is

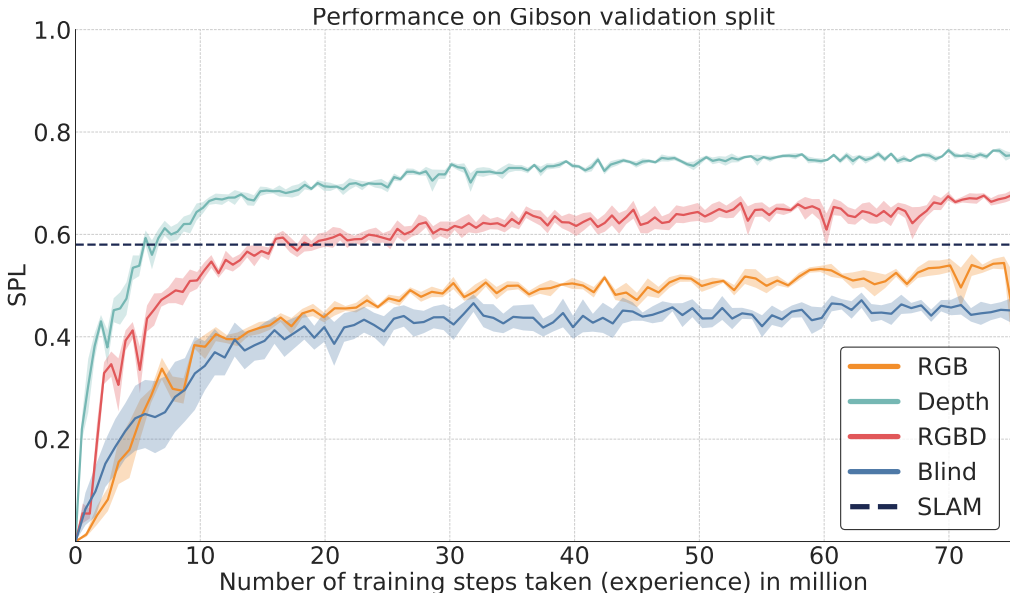


Figure 3.2: RGB agents perform significantly worse than their depth-equipped counterparts; performing similarly to blind agents. This plot is reproduced from Savva et. al., [21] for convenient reference.

used to derive policy distribution a_t over actions as well as a value estimate v_t.

$$h_t = f(\phi(I_t), g_t, h_{t-1}) \tag{3.1}$$

$$a_t \sim \pi(h_t) = \text{Categorical}(W_a h_t) \tag{3.2}$$

$$v_t = W_v h_t \tag{3.3}$$

Here g_t is calculated using the current position P_t and original Euclidean goal coordinates. In compact layer notation, the SimpleCNN network architecture is {Conv 8×8, ReLU, Conv 4×4, ReLU, Conv 3×3, ReLU, Linear, ReLU}. Agents are trained using PPO [25] algorithm with Generalized Advantage Estimation (GAE) [24].

Chapter 4: Auxiliary Tasks for Point Goal Navigation

In this chapter, we discuss our auxiliary tasks for point-goal navigation. As discussed previously, one of the reasons behind slow paced learning of embodied-AI agents might be the fact that we are trying to solve a 'chicken-and-egg' problem. We want good, task-relevant representation of sensory inputs to frame a competent policy and we want positive reinforcement from environment to build a good representation. If we could just alleviate one of these problems, we could easily accelerate the learning. Here is where we think auxiliary tasks could be of use. We try to design auxiliary tasks to embed additional useful information into the representation. Concretely, we use supervision from signals available in the simulation environment to shape representation. This is possible because simulated environments are filled with plethora of signals such as geodesic path to the goal, semantic scene segmentation, etc. We will understand each auxiliary task in detail - their motivation and implementation. Refer Fig. 3.1 for a detailed schematic of agent and auxiliary task architectures.

We would like our agents to have a very good understanding of the local scene geometry to avoid obstacles and navigate through free space. We would also want our agent to understand how the actions taken affect its position with respect to the goal. At each time step, the agent should be able to predict how far the goal might be. We design our auxiliary tasks around inducing this information in the representation.

4.1 Quantized Depth Prediction

From Fig. 3.2, it is clear that depth is more useful as an input modality for point-goal navigation than RGB. In fact, the depth agents achieve nearly twice the performance of the RGB agents. This seems a bit extreme if seen from the point of view of human perception. Human beings rely on binocular (from both eyes) cues and monocular (from a single eye) cues for depth perception. Monocular cues include motion parallax, change in relative size of objects, etc. You and I could shut one of our eye and still walk around

without bumping into objects around us. This means human beings are able to estimate depth from a monocular sensory input but our point-goal agents are unable to do so. In order to resolve this, we develop a quantized depth prediction task similar to one in Mirowski et. al., [18]. We predict a low-resolution representation of current depth image using the intermediate visual encoding and navigation history. So for RGB agents, this boils down to monocular depth estimation problem. We also try this auxiliary task on depth agents and report results.

4.1.1 Depth Quantization

The original depth map has dimensions 256×256 with depth being a floating point value in range 0 and 1 i.e $\in (0, 1)^{256 \times 256}$. We quantize each of these depth values into 8 bins (classes) - $\{0, 0.05, 0.175, 0.3, 0.425, 0.55, 0.675, 0.8, 1\}$ where $0 - 0.05$ is class 0, 0.05-0.175 is class 1 and so on. The smaller depths are emphasized to better differentiate between close-by and nearly-colliding objects. In order to obtain central field of view of agents, we center-crop depth map to 128×128. Earlier, we had experimented with full sized depth map and did not find any significant drop in agent performance on account of cropping. This makes sense intuitively as predicting the depth of floor and ceiling might not be very useful from navigation perspective. We need to only care about the general structure of scene geometry in order to avoid obstacles and detect free space and hence, we avoid pixelwise predictions. Rather, we aggregate non-overlapping, neighbouring patches of pixels to obtain a depth target map. This low resolution, quantized representation captures the general structure and is easier to learn while still providing significant benefit. We carry out an average pooling with kernel size 32 of the cropped map followed by quantization to produce the output target map $D_t \in [0, \ldots, 7]^{4 \times 4}$. With this, we turn our auxiliary depth prediction into a 8-way classification problem for each of 4×4 spatial locations. As depth agents receive depth map as input, this auxiliary task takes form of a down-sampled autoencoding problem.

4.1.2 Depth Auxiliary Network

Since we desire our agent to infer the depth using monocular cues such as motion, we would want agent's current visual observation and memory state to encode local scene

geometry. Let $O_t \in \mathbb{R}^{32\times28\times28}$ be the final spatial feature representation of $\phi(I_t)$. Given O_t and the hidden state $h_t \in \mathbb{R}^{512}$, we add an auxiliary depth prediction network f_d such that:

$$\hat{D}_t = f_d([O_t + h_t]) \tag{4.1}$$

where $+$ denotes a spatially-tiled concatenation along the channel dimension – i.e. $[O_t + h_t] \in \mathbb{R}^{544\times28\times28}$. f_d is a series of convolutional layers followed by a fully-connected layer to predict depth-class distributions for each spatial cell as shown in Fig. 4.1. Here **CONV** block represents series of operations - {Conv2d 3×3, GroupNorm, ReLU} in order. We have used GroupNorm consistently with group size of 32 following Wu et. al., [28] in order to exploit the correlation among layer channels. GAP is global average pooling layer. The layer FC is a fully connected layer which maps vector of size 1024 to that of size 128. This is reshaped to 8×16 to match ground-truth label encoding. We experimented with the setting where we avoid use of fully connected layer to retain the spatial information and our auxiliary network produces a matrix $\hat{D}_t \in \mathbb{R}^{8\times4\times4}$. But this surprisingly produced poor results compared to the architecture described above. The final auxiliary loss $\mathcal{L}_D$ is

$$L_D = -\frac{1}{16 \times N}\sum_{i=1}^{N}\sum_{k=1}^{16}\sum_{j=1}^{8} I(j == y_{ik}) log(p_{ijk}) \tag{4.2}$$

where N is the batch size, I is indicator function which is 1 for ground truth class y_{ik} for i th batch and k th spatial cell and 0 otherwise. p_{ijk} is softmax probability.

4.2 Inverse Dynamics

The previous auxiliary task was designed to embed depth perception in agent representation. We would also like to induce an understanding of how an agent' actions affect its state with respect to the environment. This understanding will help the agent with short-term navigation decisions such as the agent will know that if it moves forward towards an obstacle, it will not make a progress towards its goal. Following prior work on self-supervised representation learning from video [3, 15], this knowledge can be induced using an inverse dynamics formulation – predicting the action responsible for an observed state change in the environment.

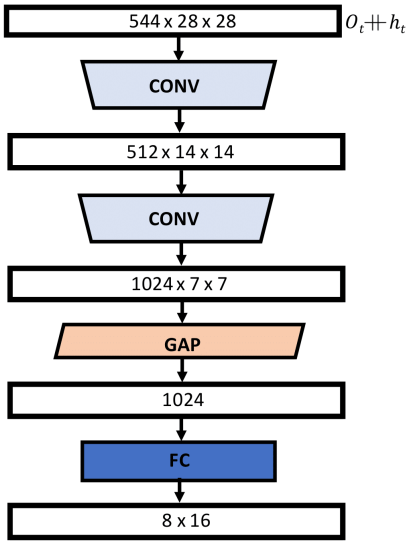


Figure 4.1: A schematic of auxiliary depth prediction network. It consumes tiled block $(O_t \# h_t)$ to predict distribution over pixel classes.

4.2.1 Inverse Dynamics Auxiliary Network

In inverse dynamics task, we train a neural network to predict action responsible for change in agent' state. This state change can be represented by the consecutive goal locations i.e. g_t and g_{t+1}. Concretely, the auxiliary network predicts the action a_t using the visual encoding $\phi(I_t)$ concatenated with the goal coordinates g_t and g_{t+1}. Note that here we deliberately avoid using the memory state h_t as consecutive state sequences are highly correlated. In fact, the action a_t is decoded from memory state h_t. This would make our auxiliary task a rather trivial one, undermining the need to learn a better visual encoding. The auxiliary network f_{Inv} acts as follows :

$$\hat{a}_t = f_{\text{Inv}}(\phi(I_t), g_t, g_{t+1}) \tag{4.3}$$

We implement f_{Inv} as a simple feed-forward network that predicts a distribution over the actions taken given the concatenation of the inputs. For training, we take (a_t, I_t, g_t, g_{t+1}) tuples directly from agent policy rollouts. The auxiliary loss $\mathcal{L}_{\text{Inv}}$ is just the cross-

entropy loss of the predicted and actual actions as follows :

$$L_{Inv} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{4} I(j == y_i) log(p_{ij}) \tag{4.4}$$

where N is the batch size, I is indicator function which is 1 for ground truth class y_i and 0 otherwise. p_{ij} is softmax probability.

4.3 Remaining Path Length Prediction

So far we have tried to encode an understanding of the immediate state of the environment such as the local scene geometry and transition dynamics. But we need to remember that an important aspect of point-goal navigation is to choose the most optimal (shortest) path to reach the point-goal. In order to encourage such behavior, we already have a time penalty of -0.01 added to the reward received at each time step. To augment this even further, we try to induce the longer range navigability into the representation. For example, the agent must be able to avoid dead-ends in favor of well-connected pathways (e.g. hallway) when the Euclidean distance to the goal is large and vice-versa. To encode this in representation, we develop an auxiliary task to predict the remaining *geodesic* distance to the end goal i.e. the distance an ideal agent would traverse to reach the end goal.

4.3.1 Remaining Path Length Auxiliary Network

The auxiliary network is designed to compute an estimate of geodesic distance to point-goal using the current visual observation I_t and goal coordinates g_t at time t. Use of agent's historical state can encourage counting strategies whereby the agent might just try to see how long it has been navigating rather than using sensory inputs to trivially solve the prediction task. The auxiliary network f_{Geo} works as following:

$$\hat{d}_{\text{geo}} = f_{\text{Geo}}(\phi(I_t), g_t) \tag{4.5}$$

We implement f_{Geo} as a simple feed-forward neural network and define the auxiliary loss $\mathcal{L}_{\text{Geo}}$ as an L2 regression loss on this geodesic distance prediction as follows :

$$L_{\text{Geo}} = \frac{1}{N}\sum_{i=1}^{N}(\hat{d}^i_{\text{geo}} - d^i_{\text{geo}})^2 \tag{4.6}$$

where N is the batch size and $\hat{d}^i_{\text{geo}}$ is the predicted geodesic distance and the d^i_{geo} is the optimal ground truth distance.

In this chapter, we were able to understand the point-goal agent architecture as well as the architecture of our three auxiliary tasks. These auxiliary networks use some intermediate representation from the original point-goal network and are trained in conjunction with policy loss as :

$$L_{\text{Total}} = L_{\text{Policy}} + \alpha \times L_{\text{Auxiliary}} \tag{4.7}$$

where α is weightage for auxiliary loss. Further, we examine the experimental settings used for training and evaluating these agents.

Chapter 5: Experimental Setting

5.1 PointNav in Habitat

We use Habitat Simulator to carry out our experiments. The PointGoal navigation setting is similar to Habitat challenge 2019 [22]. An agent starts at a random location in an unseen environment and must navigate to the goal given in relative Euclidean coordinates. Visual observation (RGB or Depth) and pose information from GPS+Compass are the only two ways the agent can make sense of its surroundings. We follow the training and validation splits on the Gibson dataset [29] in accordance with Savva et. al., [22].

5.2 Gibson dataset

We need to simulate realistic environments in order to train and evaluate the navigation performance of embodied agents. Gibson dataset [29] consists of spaces of building floors scanned using a 3D scanner. The entire dataset consists of 572 buildings composed of 1447 floors. We use the standard scenarios provided by Anderson et. al., [5] which are a selective subsets of original Gibson space. A scenario consists of starting position of agent and its goal location. It is ensured that the locations are physically reachable for our agent i.e. we make sure that there is 1m of open space around the locations and that they are not under of over other objects. The minimum distance between initial and target locations is 1 m, and the maximum depends on the size and navigation complexity of each space. The initial and target points are picked from the same floor to remove the need to climb stairways as the feasibility/complexity of that becomes agent-specific. We use $2\times$ the agent's body width i.e. 0.4m as the distance threshold τ for successful navigation when action *stop* is called.

5.3 Training & Testing

An agent learning an optimal policy for point-goal navigation is treated as a model-free reinforcement learning problem. The auxiliary networks are trained in a standard super-

vised learning fashion in conjunction with this RL task. At each step, the environment provides dense reward r_t for an action based on the change in geodesic distance to goal – i.e.

$$r_t = D(P_t, g) - D(P_{t+1}, g) - \lambda \tag{5.1}$$

Here λ is slack penalty which acts as cost for staying alive. This penalty encourages the agent to traverse shorter paths and end episodes sooner. The episode will end when the agent takes the action *stop*. Then if the agent ends up being within 0.2 meters radius of the point-goal, the episode is considered to be successful and agent receives an additional +10 reward. The agent is trained using Proximal Policy Optimization (PPO) algorithm [25] with Generalized Advantage Estimation (GAE) [24].

During testing, we use standard validation split of Gibson dataset, consisting of indoor environments not seen during training. We track the agent performance using the metrics described below.

5.4 Metrics

We report standard metrics for point-goal navigation [4] –

- **Success.** Percentage of episodes in which the agent calls `stop` within 0.2 meters of the goal location.
- **Success weighted by inverse normalized Path Length (SPL).** SPL considers both success and path efficiency – weighting a binary success indicator by the normalized ratio of the agent's path length and the shortest path.

$$\frac{1}{N}\sum_{i=1}^{N} S_i \frac{l_i}{max(p_i, l_i)} \tag{5.2}$$

where N is the total number of test episodes, l_i is the shortest path distance from the agent's starting position to the goal in episode i, p_i is the length of the path actually taken by the agent in this episode. Let S_i be a binary indicator of success in episode i. Both are reported as percentages. We report means and 95% confidence intervals over multiple trials.

5.5 Implementation details

The mini-batch size of observations used is 4 and the agent trains for 4 epochs on each batch. The batch is generated using 8 rollout workers with rollout length of 128 steps. Adam optimizer with a learning rate of 2.5×10^{-4} is used. It takes approximately 21 hours for RGB agents and 15 hours for depth agents to train on two Tesla V100 GPUs for 25 million steps of experience. We have used the public repository from Savva et. al., [22] and added our auxiliary tasks on top of the existing code. We set auxiliary-task loss weights (α in Eq. 2.10) to 1.0, 0.2 and 1.0 for quantized depth prediction, inverse dynamics and remaining path length prediction tasks respectively.

Chapter 6: Results

We augment the baseline agents with our auxiliary tasks and train for three random initializations to take into account any variations in performance over multiple runs. We also combine these auxiliary tasks to see how well they work in presence of each other. In all, we train a total of 30 agents (3×{`RGB`, `Depth`}×{`Baseline`, `+DepthPred`, `+InvDyn`, `+RemainPath`, `+All`}) for 25 million experiential steps. The performance of the agents on Gibson validation set is reported in Tab. 6.1 during the course of training at 5, 10, 15, 20 and 25 million steps. It is important to note that the auxiliary tasks are turned off during validation and are only used for training. For simplicity, hereon we will use the notation Metric@XXM which is short for the metric value after certain million time steps of training e.g. SPL@20M for SPL at 20 million steps. We also show the entire training curves in Fig. 6.2 and Fig. 6.1.

6.1 Auxiliary tasks improve sample efficiency

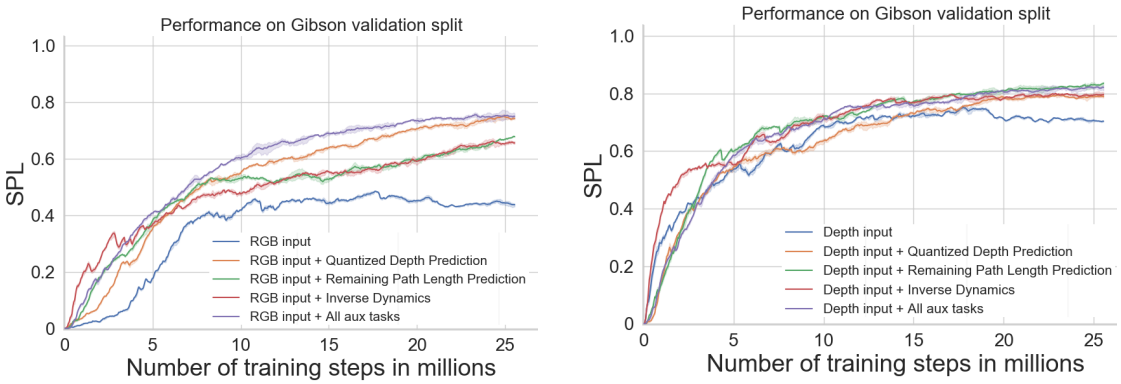
Our task-equipped agents, in general, seem to be more sample efficient than the baseline agents from Savva et. al., [22]. This is more evident for RGB agents than for Depth agents. After 5M steps, our RGB agents achieve ∼37 SPL on average which is more than double of the baseline at 17.8 SPL (`rows 2-4 vs 1` in Tab. 6.1). By 10 million steps, these agents achieve SPL in range 47.6 - 55.4. Such high values are not reached by the baseline agents even by the end of training i.e. 25 million steps in our experiments (`row 1`). By 15 million steps, our best RGB agent (`row 2`) achieves 63.8 SPL, outperforming the ∼57 SPL at 75 million reported in Savva et. al., [22] for the baseline agent (see Fig. 3.2). Similarly, our best depth agent achieves 77.2 SPL@15M (`row 9`) compared to the ∼75.4 SPL@75M reported in Savva et. al., [21]. This represents a 5x (75M/15M) improvement in sample efficiency over the baseline agents.

	Row		SPL Percentage (↑)					Success Percentage (↑)				
			@5M	@10M	@15M	@20M	@25M	@5M	@10M	@15M	@20M	@25M
RGB	1	Baseline Agent	17.8 ±0.7	41.8 ±0.7	44.2 ±1.2	45.1 ±1.0	44.3 ±1.8	27.0 ±1.2	65.1 ±1.1	69.9 ±1.5	74.6 ±1.1	78.0 ±1.9
	2	+ Depth Pred.	35.6 ±0.6	55.4 ±0.0	63.8 ±1.0	70.6 ±0.7	74.7 ±1.7	56.5 ±1.5	74.8 ±0.3	81.3 ±0.5	84.6 ±0.7	**86.8** ±1.6
	3	+ Inv. Dyn.	37.0 ±0.9	47.6 ±1.5	55.3 ±0.8	59.1 ±2.8	65.5 ±0.8	53.0 ±2.1	66.7 ±1.4	74.9 ±1.0	78.0 ±2.1	81.0 ±1.0
	4	+ Remain. Path.	38.5 ±2.1	53.2 ±2.3	52.7 ±1.4	59.7 ±1.0	66.9 ±0.6	55.5 ±2.4	76.4 ±2.3	78.9 ±2.3	83.9 ±0.8	85.6 ±1.3
	5	+ All	**40.8** ±0.7	**60.5** ±2.5	**68.8** ±1.6	**73.4** ±2.4	**75.1** ±3.0	**61.1** ±0.8	**77.04** ±2.4	**81.7** ±1.0	**85.5** ±2.5	85.6 ±3.5
Depth	6	Baseline Agent	53.7 ±2.0	68.4 ±1.3	72.1 ±0.9	70.9 ±0.6	70.5 ±0.9	68.7 ±1.8	82.1 ±1.2	89.1 ±0.3	90.9 ±1.3	92.1 ±1.0
	7	+ Depth Pred.	54.6 ±1.7	63.6 ±1.6	72.8 ±0.4	78.6 ±0.8	78.8 ±1.0	71.6 ±1.4	80.7 ±1.7	87.0 ±0.7	91.0 ±0.3	90.4 ±0.7
	8	+ Inv. Dyn.	55.2 ±1.7	71.2 ±2.4	77.0 ±0.9	78.2 ±0.8	79.4 ±1.9	73.0 ±1.9	84.3 ±1.9	89.0 ±0.7	91.1 ±1.2	92.4 ±1.4
	9	+ Remain. Path.	**60.0** ±1.1	**72.0** ±0.6	**77.2** ±1.2	**80.8** ±1.1	**83.0** ±2.2	**75.0** ±1.7	**86.9** ±1.1	**91.0** ±0.2	**92.2** ±1.5	**92.6** ±1.0
	10	+ All	58.37 ±1.4	71.46 ±0.7	75.65 ±1.6	80.48 ±1.9	81.9 ±1.0	71.54 ±1.4	85.05 ±1.3	88.08 ±1.6	91.23 ±2.1	91.54 ±1.1

Table 6.1: Navigation metrics (SPL and Success percentages) on Gibson validation set as a function of training time (5, 10, 15, 20 and 25 million steps). We report means over three runs and 95% confidence intervals. We find our auxiliary task-equipped agents consistently outperform their corresponding baseline agents by a significant margin. Our auxiliary tasks have a stronger effect on the RGB agents.

6.2 Auxiliary tasks affect Depth and RGB agents differently

The efficacy of the auxiliary tasks changes with respect to the input type of the agent. Depth prediction can be considered as the best auxiliary task for RGB agents. RGB agents with depth prediction outperform other task-equipped RGB agents by a significant margin (∼9 SPL). This helps to corroborate our hypothesis that depth provides important scene geometry information for navigation that vanilla RGB agents have trouble extracting. For depth agents, we could say that inverse dynamics and remaining path length prediction are equally effective, with remaining path length being only marginally outperforming inverse dynamics task (`row 7` and `8`). As one would expect, depth prediction is least useful of all auxiliary tasks for depth agents as depth is already provided as input to the agent. Though, it is worth emphasizing that the depth agent does still benefit from the depth prediction task (`row 6`), improving by 8 SPL@25M. The augmented agents achieve similar success percentages as the baseline, with improvements in navigation efficiency being a strong driver for improvements in the SPL metric. This might also due to the fact that success scores for depth agents are already quite close to perfect and thus, difficult to improve.



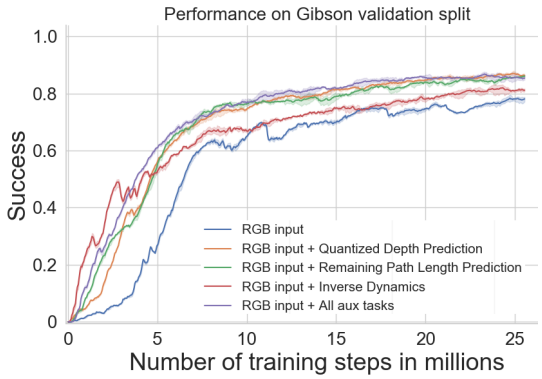
(a) Validation SPL over training for RGB agent

(b) Validation SPL over training for Depth agent

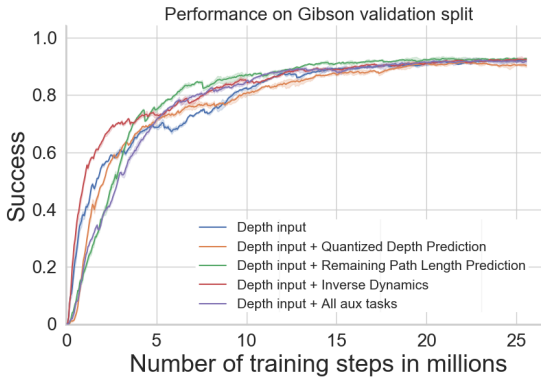
Figure 6.1: SPL on Gibson validation set for RGB (left) and Depth (right) agents as a function of number of training steps. We compare baseline agents with our auxiliary task-equipped agents. For RGB input, our agents consistently outperform the baseline from the start of training. Our depth agents have a slower start but outpace the baseline model after 20 million steps.

6.3 RGB with depth prediction outperforms RGB-D

As seen previously, the RGB agent benefits tremendously from the use of auxiliary depth prediction supervision. In fact, it benefits the RGB agents the most when compared to the other two auxiliary tasks. An interesting alternative to this would be to just provide depth as part of input along with the RGB. In our work, we do not explicitly experiment with such a RGB-D agent, but we report results from Savva et. al., [22]. We compare the performance of our RGB agent equipped with depth prediction task with the RGB-D agent from Savva et. al., [22] (see Fig. 3.2). It should be noted that both these agents are trained under similar training conditions and then evaluated for same Gibson split. Suprisingly, we found our task-equipped agent is considerably more sample efficient than the RGB-D agent, (74.6 SPL@25M vs. 70 SPL@75M). This suggests that the role of auxiliary tasks might not be to just insert new information but to actually shape the representation learning during training. Also, it is worth noting that Depth agents outperform RGB-D agents. This would mean that RGB is, in fact, harming the agent performance. This seems unintuitive as the additional input information, even when



(a) Validation Success over training for RGB agent



(b) Validation Success over training for Depth agent

Figure 6.2: Success on Gibson validation set for RGB (left) and Depth (right) agents as a function of number of training steps. We compare baseline agents with our auxiliary task-equipped agents. For RGB input, our agents consistently outperform the baseline from the start of training but not necessarily with a very large margin. For depth input, the success curves seem to eventually converge to same value for all the agents.

not very useful, should not hurt performance. We hypothesize that that agents find it difficult to segregate and parse the depth channel information when concatenated with RGB.

6.4 Combining multiple auxiliary tasks

Previously, while working with individual auxiliary tasks, we experimented with the associated auxiliary loss weights and chose them carefully so that the auxiliary loss does not dominate the policy loss. In this section, we simultaneously train multiple auxiliary tasks in conjunction with policy. Hence, we need to recalibrate the auxiliary loss weights in order to maintain balance between the sum of auxiliary losses and the policy loss. We first observe the performance when we have all the three auxiliary tasks active (refer rows 5 and 10 in Tab. 6.1). For RGB input, this combination works well and the models outperform agents equipped with individual auxiliary tasks. Depth agents do not show any remarkable improvements over our best agent i.e. one equipped

				SPL Percentage (↑)					Success Percentage (↑)				
Row	Depth Pred.	Inv. Dyn.	Path Pred.	@5M	@10M	@15M	@20M	@25M	@5M	@10M	@15M	@20M	@25M
1		✓	✓	**42.9**	53.3	58.4	62.7	63.6	**63.0**	76.5	80.6	84.3	80.3
2	✓	✓		42.1	59.0	67.2	71.8	72.9	61.5	73.9	79.7	83.4	83.3
3	✓		✓	37.0	51.0	58.4	64.4	68.9	55.6	69.7	74.9	80.8	81.5
4	✓	✓	✓	40.8	**60.5**	**68.8**	**73.4**	**75.1**	61.1	**77.04**	**81.7**	**85.5**	**85.6**

Table 6.2: Navigation metrics (SPL and Success percentages) on Gibson validation set for the RGB agent as a function of training time (5, 10, 15, 20 and 25 million). We present all combinations of auxiliary tasks. Underlined entries denote results for combinations that outperform the independent performance of each individual constituent loss. We find the combination of task often lead to further improvements.

with remaining path length prediction. But the performance does not seem to degrade either as the SPL and success scores are within variances of our best agents for depth input. This is another instance where auxiliary tasks seem to have different effect for RGB and depth agents. Further we would like to examine the relationship between various auxiliary tasks. We present all possible combinations in Tab. 6.2 for the RGB agent. We underline any result for a combination that outperforms each individual constituent loss. We find that Inv. Dyn. + Path Pred. (`row 1`) and Depth Pred. + Inv. Dyn. (`row 2`) contain complementary information to an extent. On the other hand, Depth Pred. + Path Pred. (`row 3`) do not show any improvement over constituent losses. This suggests depth prediction and remaining path prediction compete when acting on common representation. Despite this, combining all three yields further gains across most time points. Currently, the loss coefficients are kept static.

6.5 Comparison to state of the art

So far we compared performance of task-equipped agents with the Habitat Savva et. al., [21] agents that form our baseline architectures. However, state-of-the-art in point-goal navigation Wijmans et. al., [27] achieves nearly perfect SPL ($\geq$99) in Gibson validation – significantly higher than our models and those of Savva et. al., [21]. It is however important to consider how this impressive result is achieved. In Wijmans et. al., [27], agents architectures have significantly greater complexity [1] and are trained on an ex-

[1] Using a SE-ResNeXt50 [30] network for encoding observations

tended dataset (Gibson and Matterport3D [7]) for 2.5 billion steps of experience in a distributed RL framework. This represents an increase in model capacity, training set size, and a training time (100x). Our focus in this work is on training efficiency and our results question whether these extreme training protocols are necessary. Taking the first 75 million steps of the DDPPO agent's training regime, we find our RGB agents achieve the same SPL in only 17 million steps (a >4x improvement) (see supplementary - Sec. 4). An interesting question is at what point our agents would reach optimal performance compared to the 2.5 billion steps required in Wijmans et. al., [27]. Unfortunately, we cannot scale to these extremes. Our existing experiments required 40 days of GPU time – scaling by over 100× is simply infeasible given our infrastructure.

6.6 Comparison with prior auxiliary tasks

We examine how existing work on auxiliary tasks in visually-simple settings transfers to photo-realistic environments. Specifically, we examine two auxiliary tasks from prior work, depth prediction [18] and reward regression [14]. We limit our discussion to RGB agents to be consistent with these works.

6.6.1 Depth prediction

We contrast the performance of the depth prediction architecture proposed in Mirowski et. al., [18] with our own. Developed in simple game-like environments, this task requires agents to predict dense pixel-wise depth from non-spatial features (i.e. $\phi(I_t)$) – a difficult task in complex, realistic environments. We find the architecture for depth prediction employed in Mirowski et. al., [18] causes the learning to be delayed and achieves lower performance when compared with our results (Tab. 6.3 `row 2 vs.` Tab. 6.1 `row 2`). By 5 million steps, agents augmented with Mirowski et. al., [18] achieve 1.5 SPL on average - which is nearly 1/17 of the SPL achieved by the baseline agent (Tab. 6.3 `row 1` vs `row 2`). Over the period of training, we see an improvement in performance of task-equipped agents and it surpasses the baseline at 15 million steps. By 25 million steps, we see their depth-augmented agents outperform the baseline but is still below our results by ∼10 SPL. Interestingly, this approach fails to improve the success metric significantly whereas our approach results in an improvement of ∼8% success.

	Row		SPL Percentage (↑)					Success Percentage (↑)				
			@5M	@10M	@15M	@20M	@25M	@5M	@10M	@15M	@20M	@25M
RGB	1	Baseline Agent	17.8 ±0.7	41.8 ±0.7	44.2 ±1.2	45.1 ±1.0	44.3 ±1.8	27.0 ±1.2	65.1 ±1.1	69.9 ±1.5	74.6 ±1.1	78.0 ±1.9
	2	+ Dense Depth Pred. [18]	1.5 ±0.4	13.7 ±1.0	51.3 ±0.8	61.0 ±3.7	64.8 ±2.8	2.14 ±0.5	17.8 ±1.4	68.6 ±0.2	77.4 ±3.6	79.4 ±2.3
	3	+ Reward Regression [14]	27.2 ±0.3	37.3 ±1.0	42.0 ±1.6	41.6 ±1.1	51.1 ±1.7	41.1 ±0.6	57.4 ±1.3	66.6 ±1.9	70.8 ±0.9	80.3 ±2.0
	4	Ours (Best from Tab. 6.1)	**35.6** ±0.6	**55.4** ±0.0	**63.8** ±1.0	**70.6** ±0.7	**74.7** ±1.7	**56.5** ±1.5	**74.8** ±0.3	**81.3** ±0.5	**84.6** ±0.7	**86.8** ±1.6

Table 6.3: Navigation metrics (SPL and Success percentages) on Gibson validation set as a function of training time (5, 10, 15, 20 and 25 million steps). We report means over three runs and 95% confidence intervals. We compare the baseline agent with the agents equipped with existing auxiliary tasks for navigation [18, 14]. Agents with [18] are slow to learn whereas those with reward regression only reach 51.1 SPL@25M. One can note that our best task-equipped RGB agent (`row 4`) outperforms all the agents.

6.6.2 Reward regression

We apply reward regression task proposed in Jaderberg et. al., [14] to our problem – predicting the immediate reward using the encoded observation $\phi(I_t)$ and relative goal vector g_t. We find reward regression to be less effective than our approaches in improving either SPL or success metric for the navigator. Initially by 5 million steps, the reward-regression agents show some promise, leading over the baseline agents by $\sim$ 10 SPL (compare Tab. 6.3 `rows 1` and `3`). But this lead does not extend further into training. By 25 million steps, agents equipped with reward regression achieve 51.1 SPL which is lower than that achieved by any of our individual auxiliary task-equipped agents (see supplementary Sec. 5 for learning curves).

Chapter 7: Analysis

7.1 Effect on Collision Rate

We try to see how auxiliary tasks help agents avoid collisions with objects in their surroundings. In Tab. 7.1, we compare the average collisions per episode for various RGB agent architectures across 250 test episodes. We note that task-equipped agents collide less frequently than the baseline agents. The baseline agent, on account of its poor navigation capability in initial stages of training, terminates episodes prematurely and result in low collision count. This count rise significantly as the agent improves at reaching its target goal. We hypothesize that the quantized depth prediction loss will act as a proxy for free space - enabling the agent to have a strong sense of when a forward action might result in a collision. We confirm this hypothesis in our experiments where the depth prediction task result in lowest collisions. Even when comparing at similar success rates (@25M for Baseline vs. @10M for Depth Pred.), the Depth Pred. augmented agent reduces collisions by nearly half.

We observe that collisions are not strictly bad for navigation. As shown in Kadian et. al., [16], an agent can in fact use the wall to slide against to increase SPL. Augmented agents achieve lower collision rates with more efficient paths by avoiding any inefficient or unnecessary collisions.

7.2 Decoding Position from Agent Representation

A point-goal agent receives dynamically-changing relative goal coordinate as input at each time step t. But they do not receive any information of their own position in the environment. Here, we try to examine if agent representations encode this information and use it to form policy. For this, we take agents trained for 25 million steps and freeze their weights. For each observation, we get the hidden policy state h_t. We use this as input to learn a simple fully-connected neural network that predicts the agent's

Row		Average collisions per episode (↓)				
		@5M	@10M	@15M	@20M	@25M
1	Baseline Agent	**31.63**	54.97	54.90	62.40	67.01
2	+ Inv. Dyn.	37.03	39.66	42.69	49.45	49.54
3	+ Depth Pred.	44.17	**37.97**	**37.25**	**34.57**	**34.89**
4	+ Remain. Path.	46.75	44.45	56.19	42.58	43.93

Table 7.1: Average collisions per episode for RGB agents in 250 test episodes. We find auxiliary-augmented agents collide significantly less, with the depth prediction auxiliary task inducing the fewest.

position coordinates in three dimensions. We train this network on 10240 positions from the training set and then measure the proportion of predictions which fall within 4m, 2m and 1m of true position for 2400 instances on validation set. The results are shown in Tab. 7.2. We can observe that with remaining path prediction task we achieve the best performance. These results suggest that the remaining path prediction task induces significantly more information about agent position. We also observe that since the baseline agent and inverse dynamics agent do not encode long range information they perform poorly.

7.3 Remaining Path Length Identifies Openness

The task of remaining path length prediction predicts how far is the goal from the agent in terms of geodesic distance using the current observation and goal in relative coordinates. This distance can be as low as the Euclidean distance between current and goal positions - in an open area, the agent has to just move in straight line to the goal. In case the goal is quite far or visual observation is of a cluttered space or even a dead end in a room, this same distance can be much greater than Euclidean distance. Imagine the goal being on other side of the wall in front of you. Fig. 7.1 shows a pair of qualitative examples for the remaining path length task. In both, the Euclidean distance to goal is around 6.5 meters. However, the “dead end” example on left predicts a significantly higher remaining path

	% of predictions within range (↑)		
	< 4m	< 2m	< 1m
Baseline agent	57.2	51.0	48.4
+ Depth Pred.	71.3	66.9	62.5
+ Inv. Dyn	59.1	52.3	50.6
+ Remain. Path.	**80.1**	**76.6**	**74.5**

Table 7.2: Percent of samples for which the decoder predictions fall within a radius of 4m, 2m and 1m. The decoder is most accurately able to extract the localization information using the encoder trained with remaining path length prediction task.

length (10.2m) whereas the open area on the right keeps its estimate low (7.6m). We found this trend to be consistent across several randomly picked qualitative examples.

7.4 Pre-training vs. Auxiliary Task Learning

Each auxiliary task induces certain sort of knowledge into the encoder of the agent. We would like to investigate if we can induce it via pre-training and will that be sufficient to achieve similar or even better performance. Specifically, we collect agent trajectories from our best performing agent in the training environments to create a static dataset of observations and auxiliary task targets. For depth prediction task, we additionally record the hidden policy state. The models are trained until they reach similar loss levels as those observed during auxiliary learning. Then these encoder weights are used as warm-start for full RL training. From Tab. Tab. 7.3, we see that pretraining causes mild improvements in most cases. But these improvements never come close to that offered by auxiliary training paradigm. So we might hypothesize that it is not just the information which is being fed but how this information is learnt that is of significance.

Figure 7.1: Despite of both views in above example having similar Euclidean (L2) distance to the goal, they have different true geodesic distances(Geo). The prediction for geodesic distance (Pred) differs based on the "opennness" of the current frame.

	SPL (↑)				
	@5M	@10M	@15M	@20M	@25M
Baseline	17.8	41.8	44.2	45.1	44.3
Depth Pred. (Pre.)	23.9	43.9	47.2	46.8	46.2
Depth Pred. (Aux.)	**35.6**	**55.4**	**63.8**	**70.6**	**74.7**
Inv. Dyn. (Pre.)	25.0	40.9	45.2	44.7	46.2
Inv. Dyn. (Aux.)	**37.0**	**47.6**	**55.3**	**59.1**	**65.5**
Remain. Path (Pre.)	27.9	36.3	44.6	48.6	47.1
Remain. Path (Aux.)	**38.5**	**53.2**	**52.7**	**59.7**	**66.9**

Table 7.3: We compare the agents equipped with auxiliary tasks (Aux.) (Tab. 6.1) with the agents using a pretrained (Pre.) encoder trained independently for these tasks. The task-equipped agents consistently outperform the agents with pretrained encoders.

Chapter 8: Conclusion

In this work, we examined the effect of auxiliary task training towards improving the sample efficiency of point-goal navigation. We observed significant reduction ($5\times$) in training times over baseline agents from prior work. Specifically, we introduced three auxiliary tasks - Depth Prediction, Inverse Dynamics and Remaining Path Length Prediction. We achieved varying degree of success with these tasks. Each task induced distinct information which helped agent navigate.

Bibliography

[1] Gibson challenge @ CVPR 2020. http://svl.stanford.edu/gibson2/challenge.html.

[2] Habitat Challenge 2019 @ Habitat Embodied Agents Workshop. CVPR 2019. https://aihabitat.org/challenge/2019/.

[3] Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to see by moving. *ICCV*, 2017.

[4] Peter Anderson, Devendra Singh Chaplot Angel Chang, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir R. Zamir. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

[5] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On Evaluation of Embodied Navigation Agents. *arXiv preprint arXiv:1807.06757*, 2018.

[6] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683, 2018.

[7] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niebner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. In *2017 International Conference on 3D Vision (3DV)*, pages 667–676. IEEE, 2017.

[8] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. *ICLR*, 2020.

[9] Devendra Singh Chaplot, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Modular visual navigation using active neural mapping.

[10] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP*, 2014.

[11] Karol Gregor, Danilo Jimenez Rezende, Frederic Besse, Yan Wu, Hamza Merzic, and Aaron van den Oord. Shaping belief states with generative environment models for rl. *NIPS*, 2019.

[12] Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Bernardo A. Pires, and Rémi Munos. Neural predictive belief representations. *arXiv preprint arXiv:1811.06407*, 2018.

[13] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2017.

[14] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[15] Dinesh Jayaraman and Kristen Grauman. Learning image representations tied to ego-motion. *ICCV*, 2015.

[16] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra. Are we making real progress in simulated environments? measuring the sim2real gap in embodied visual navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[17] Jacob Krantz, Erik Wijmans, Arjun Majumdar, Dhruv Batra, and Stefan Lee. Beyond the nav-graph: Vision-and-language navigation in continuous environments. 2020.

[18] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

[19] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[20] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *ICML*, 2017.

[21] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research. *ICCV*, 2019.

[22] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. 2019.

[23] Alexander Sax, Bradley Emi, Amir R. Zamir, Leonidas J. Guibas, Silvio Savarese, and Jitendra Malik. Mid-level visual representations improve generalization and sample efficiency for learning visuomotor policies. 2018.

[24] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *ICLR*, 2016.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[26] Erik Wijmans, Samyak Datta, Oleksandr Maksymets, Abhishek Das, Georgia Gkioxari, Stefan Lee, Irfan Essa, Devi Parikh, and Dhruv Batra. Embodied question answering in photorealistic environments with point cloud perception. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6659–6668, 2019.

[27] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. DD-PPO: Learning near-perfect pointgoal navigators from 2.5 billion frames. 2020.

[28] Yuxin Wu and Kaiming He. Group normalization. *ECCV*, 2018.

[29] Fei Xia, Amir R. Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. CVPR(2018). Gibson dataset license agreement available at https://storage.googleapis.com/gibson$_m$*aterial/Agreement*

[30] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.