APPROACHES FOR EFFICIENT AUTONOMOUS EXPLORATION USING DEEP REINFORCEMENT LEARNING

A Thesis

Submitted to the Faculty

of

Purdue University

by

Thomas W. Molnar

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

May 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Eugenio Culurciello, Co-Chair

School of Electrical and Computer Engineering

Dr. Zhongming Liu, Co-Chair

School of Electrical and Computer Engineering

Dr. Avinash Kak

School of Electrical and Computer Engineering

Approved by:

Dr. Dimitrios Peroulis

Head of the Electrical and Computer Engineering Graduate Program

ACKNOWLEDGMENTS

I would like to thank my family for their unconditional love and support over the years, none of this would have been possible without them.

Thank you to my graduate committee professors Dr. Eugenio Culurciello, Dr. Zhongming Liu and Dr. Avinash Kak for their advice and guidance throughout my graduate program. I am grateful for Eugenio welcoming me into his lab and teaching me while under his advisorship. Thank you as well to my lab-mates Juan Carvajal and Lukasz Burzawa who assisted me throughout the course of my program. Their input and guidance has been invaluable, and I will always cherish our friendships.

I am also thankful to Mr. David Hankins and Dr. Eric Dietz for their mentorship and handling of the Purdue Military Research Initiative. Without their efforts and the initiative, I would have not been in this position to study at Purdue and pursue my degree. Lastly, thank you to Purdue University as a whole for providing me with this opportunity to complete my graduate degree.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

ABBREVIATIONS

AI	Artificial Intelligence
ACM	Augmented Curiosity Module
D-ACM	Depth-Augmented Curiosity Module
OF-ACM	Optical Flow-Augmented Curiosity Module
C-ACM	Depth + Optical Flow Combined-Augmented Curiosity Module
A2C	Advantage Actor Critic
A3C	Asynchronous Advantage Actor Critic
ANN	Artificial Neural Network
CapsNets	Capsule Networks
Caps-EM	Capsules Exploration Module
CNN	Convolutional Neural Network
ICM	Intrinsic Curiosity Module
MWH	My Way Home
MWH-G	My Way Home Giant
MWH-M	My Way Home Mirrored
RGB-D	Red, Green, Blue and Depth
RL	Reinforcment Learning
Deep RL	Deep Reinforcement Learning
SLAM	Simulatenous Localization and Mapping
UAV	Unmanned Aerial Vehicle

ABSTRACT

Molnar, Thomas W. MS, Purdue University, May 2020. Approaches for Efficient Autonomous Exploration using Deep Reinforcement Learning. Major Professor: Eugenio Culurciello.

For autonomous exploration of complex and unknown environments, existing Deep Reinforcement Learning (Deep RL) approaches struggle to generalize from computer simulations to real world instances. Deep RL methods typically exhibit low sample efficiency, requiring a large amount of data to develop an optimal policy function for governing an agent's behavior. RL agents expect well-shaped and frequent rewards to receive feedback for updating policies. Yet in real world instances, rewards and feedback tend to be infrequent and sparse.

For sparse reward environments, an intrinsic reward generator can be utilized to facilitate progression towards an optimal policy function. The proposed Augmented Curiosity Modules (ACMs) extend the Intrinsic Curiosity Module (ICM) by Pathak et al. These modules utilize depth image and optical flow predictions with intrinsic rewards to improve sample efficiency. Additionally, the proposed Capsules Exploration Module (Caps-EM) pairs a Capsule Network, rather than a Convolutional Neural Network, architecture with an A2C algorithm. This provides a more compact architecture without need for intrinsic rewards, which the ICM and ACMs require. Tested using ViZDoom for experimentation in visually rich and sparse feature scenarios, both the Depth-Augmented Curiosity Module (D-ACM) and Caps-EM improve autonomous exploration performance and sample efficiency over the ICM. The Caps-EM is superior, using 44% and 83% fewer trainable network parameters than the ICM and D-ACM, respectively. On average across all "My Way Home" scenarios, the

Caps-EM converges to a policy function with 1141% and 437% time improvements over the ICM and D-ACM, respectively.

1. INTRODUCTION

Unmanned Aerial Vehicles (UAVs) have grown tremendously in recent years with their capabilities and range of applications. An accessible technology and equipable with a range of sensors, UAVs have been utilized to help with tasks such as wildfire monitoring, target tracking and search and rescue [1–3]. Despite efforts to develop autonomous UAV navigation and exploration abilities, current approaches have difficulties when faced with complex real-world environments and lacking a priori knowledge of an environment. Modern UAVs have been paired with Global Positioning Software (GPS) for autonomous outdoor exploration, however this method is not practical for indoor applications or in GPS-denied outdoor environments [4]. Moreover, a number of approaches rely on prior awareness of an environment, information which may not be available or attainable in all real world instances [5, 6]. Some proposed solutions for autonomous navigation have leveraged laser range finders (LIDARs), Red, Green, Blue and Depth (RGB-D) sensors and stereo vision to implement real-time Simultaneous Localization and Mapping (SLAM) [7–9]. SLAM involves creating a 3D representation of an environment and using the model to localize one's self within the world. However, this approach demands intensive computational operations and limits performance in real world scenarios.

The objective of this research work is to improve upon the capabilities of current Deep Reinforcement Learning (Deep RL) methodologies for autonomous exploration. The improvements proposed in this research are intended to generalize well in real world applications with robotic and UAV systems and expand autonomous agents' abilities. This work proposes two methods utilizing Artificial Neural Networks (ANNs) and capable of autonomously controlling an agent and exploring without a priori knowledge of novel environments. Furthermore, the approaches display high sample efficiency to maintain a low computational overhead cost, generalize per-

formance well across different environments and lastly utilize standard and readily available sensor equipment, such as RGB-D cameras. ViZDoom, a Doom-based AI research platform, is used to evaluate architecture designs in custom-built environments, thus facilitating rapid design prototyping and performance evaluation [10]. The first method is premised on utilizing an internal reward generator in combination with depth and optical flow information. These modules are referred to as the Augmented Curiosity Modules (ACMs). There are three types of these modules, the Depth-Augmented Curiosity Module (D-ACM), Optical Flow-Augmented Curiosity Module (OF-ACM) and Depth and Optical Flow Combined-Augmented Curiosity Module (C-ACM). The second proposed method demonstrates a novel incorporation of Capsule Networks (CapsNets) with an Advantage Actor Critic (A2C) algorithm. This module, called the Capsules Exploration Module (Caps-EM), does not use intrinsic reward signals. The D-ACM and Caps-EM both present a significant improvement upon other state-of-the-art Deep RL methods.

1.1 Artificial Neural Networks

ANNs are a set of algorithms that roughly mimic the human brain and are used to recognize patterns. The human nervous system consists of cells, or neurons, which are connected by axons and dendrites [11]. Synapses are the areas where axons and dendrites connect. The strength of synapses vary according to external stimuli and play a key role in the learning process. In a similar vein, ANNs consist of multiple neurons, or nodes, that are connected by numerical values, or weights, representing the strength between neurons. An ANN learns by updating the values of these weights through a training process in order to provide a correct output. Fig 1.1 illustrates comparable designs between biological and ANNs.

Functionally, ANNs accept an input and provide an output. Various ANN approaches utilize different network architectures, or connection patterns. For example, this research work explores feed-forward networks, wherein unidirectional connections

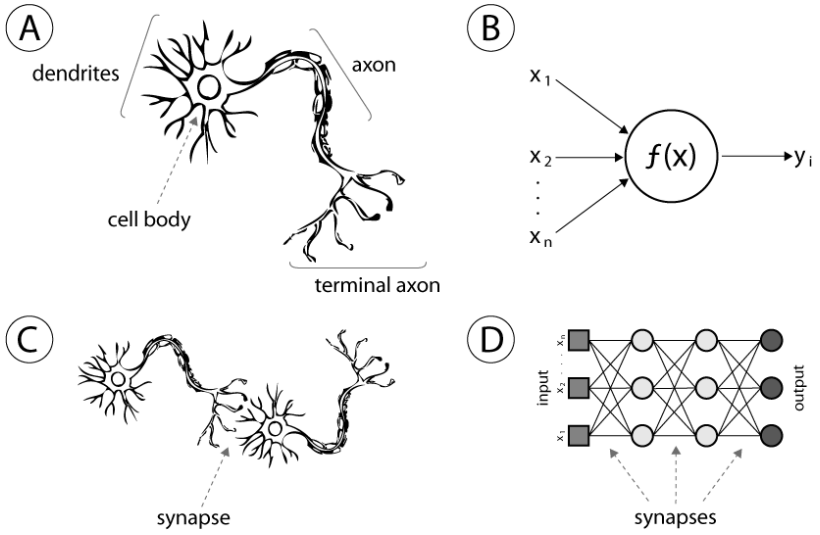


Fig. 1.1. Illustration of Biological Neural Network and Artificial Neural Network [12]. Subfigure A represents a human neuron, while B shows a corresponding artificial neuron. C and D illustrate comparable biological and artificial synapses, also referred to as weights for ANNs.

between network layers allow for data to travel from the input to output direction [12]. Alternatively, there are also feedback, or recurrent, networks that incorporate feedback loops and network connections flowing either forward or backward rather than solely unidirectional. Fig 1.2 shows an example network architecture of a feed-forward ANN with multiple network layers. In the input layer, the ANN receives information, such as an image, number or audio sample. In the hidden layer or across multiple hidden layers of a ANN, a myriad of mathematical operations may be performed on the input data, typically done to identify patterns in the data. The input data is converted to numerical values to express these patterns through vectors [11]. Lastly with the output layer, the ANN provides the output of the operations performed by the hidden layer(s) on the input.

In order to learn, an ANN must be told whether or not its actions and output are correct through feedback. The feedback process is called back-propagation. In this procedure, the network's output for a given input is compared to the desired, or

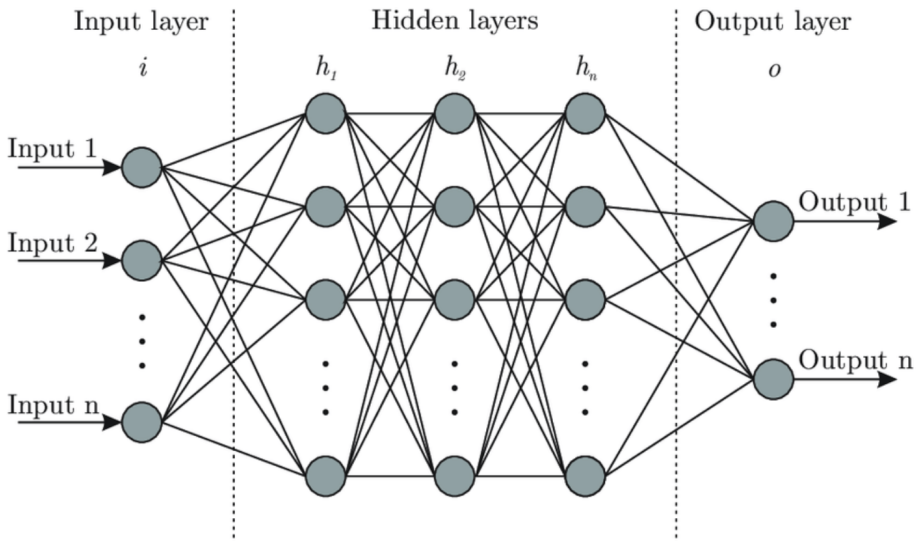


Fig. 1.2. Representation of an Artificial Neural Network [13]. Illustrating a feed-forward multilayer ANN, this figure shows a generic architecture composed of an input layer, n hidden layers and an output layer. Circles represent individual neurons, and lines connecting neurons are connective weights.

known, output for the same input. The difference between the two outputs is used to update the weights of the neuron connections. Implementing small changes over time, repeatedly conducting back-propagation allows a network to minimize the error between its own output and the desired one in order to eventually provide the correct output and behavior.

The main learning paradigms used with ANNs are Supervised Learning, Unsupervised Learning and Reinforcement Learning. With a Supervised Learning approach, a network learns using a labeled dataset. In this dataset, training examples are paired with the correct output, or ground truth, that the network should produce. The ANN may then be explicitly told whether its output is correct based on the known ground truth output. Conversely in Unsupervised Learning, labeled data is not used and a specific output for an associated input is not provided. Therefore the network instead extrapolates useful information and features from inputs rather than trying to

return a certain output. Lastly with Reinforcement Learning, an ANN strives to take actions to complete a task or to accrue the largest possible reward in a scenario. As a network interacts or controls an agent within an environment by taking actions, the environment may provide positive or negative rewards. These rewards subsequently reinforce certain actions and facilitate the network learning various behaviors.

1.2 Reinforcement Learning

Reinforcement Learning (RL) is partly based in psychology, wherein neurological function motivates reward driven behavior through dopamine release [14, 15]. RL is a prevalent machine learning approach premised on goal-based learning of behavior. Modeled around a Markov Decision Process (MDP), a traditional RL scheme usually consists of the following components:

State: Current condition of an agent

Actions: Discrete set of actions available to an agent

Reward: Feedback from an agent's environment

Policy: Mapping of an agent's state to actions

Value: Expected reward factor attainable using a given action in a particular state

Environment: World that an agent engages with [16].

Within RL, an agent receives an input state. With the input, the agent's governing policy function maps the input state to an action. The environment receives an agent's action and returns the next state of the agent in the world, as well as a reward signal depending on the quality of the agent's taken action. Based on the feedback, the agent's policy function, or strategy, is updated in such a way as to maximize the number and amount of rewards that the agent receives and to achieve an objective. The policy associated with the ideal behavior that an agent uses to maximize rewards or achieve an objective is regarded as the optimal policy function. In Deep RL specifically, ANNs are used to approximate policy functions by mapping input states to actions.

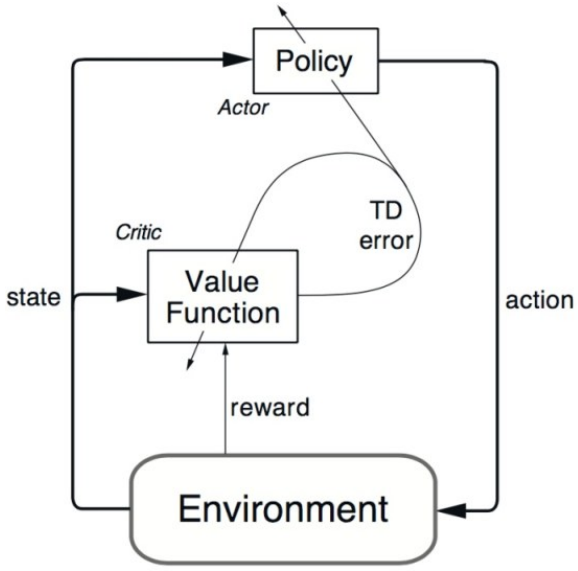


Fig. 1.3. Representation of Actor Critic Algorithm [17]. An actor residing in a given state uses its current policy function to take an action in a given environment space. The critic judges the action by estimating the value function based on the new state and rewards provided by the environment. Based on the outcome of the agent's action, the actor updates its policy function as suggested by the critic to best optimize its behavior.

1.2.1 Advantage Actor Critic Algorithm

The Advantage Actor Critic (A2C) algorithm is a RL-based algorithm consisting of an actor and a critic. The critic measures the quality of an actor's actions, while the actor controls an agent's behavior. As the agent takes actions, the critic observes these actions and provides feedback. From this feedback, the actor updates its policy function to improve performance. Fig. 1.3 illustrates the flow of how an actor interacts with an environment through an Actor Critic (AC) algorithm.

Actor-critic methods output a policy function and a value function. Given input state s_t and an action a, the policy $\pi_\theta(a|s_t)$, parameterized by network parameters θ, outputs probabilities for each available action. In the environments used in this research, the agent has a discrete action space with four possible movements: move

forward, move left, move right and no action. Sampling an action, a_t, from the policy, results in transitioning to next state s_{t+1} and receiving reward r_{t+1}. The critic, expressed as the value function $\hat{q}(s_t, a_t, w)$ with parameters w, returns an estimate of the expected final total reward obtainable by the agent from the given state. The value function $V_v(s_t)$, with network parameters v, also represents a critic to the policy function and returns the average value of a given state. The critic estimating the value function can also be based on the action value, referred to the Q value, as used in Q learning.

A2C methods offer a variant for the value estimate to reduce the problem of high variability, as shown in Equation 1.1, where the advantage function, with γ the discount factor, accounts for future rewards losing value. This function estimates how a specific action is better than other actions at a given state. The gradient of the loss function is shown in Equation 1.2.

$$A\left(s_t, a_t\right) = r_{t+1} + \gamma V_v\left(s_{t+1}\right) - V_v\left(s_t\right) \tag{1.1}$$

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta\left(a_t|s_t\right) A\left(s_t, a_t\right) \tag{1.2}$$

The set of network parameters θ and v are updated in the gradient's direction. The advantage function indicates how a specific action is better relative to the average action taken in a given state. This conveys the advantage of taking one action over another to receive more rewards. The advantage value is considered during the gradient update, or learning process, for the actor policy.

The actor and critic exist as separate models that are trained and optimized individually. The policy update for the network parameters, or weights θ, uses the q value of the critic as shown in Equation 1.3.

$$\Delta\theta = \alpha\nabla_\theta(log\pi_\theta(s_t, a_t))\hat{q}_w(s_t, a_t) \tag{1.3}$$

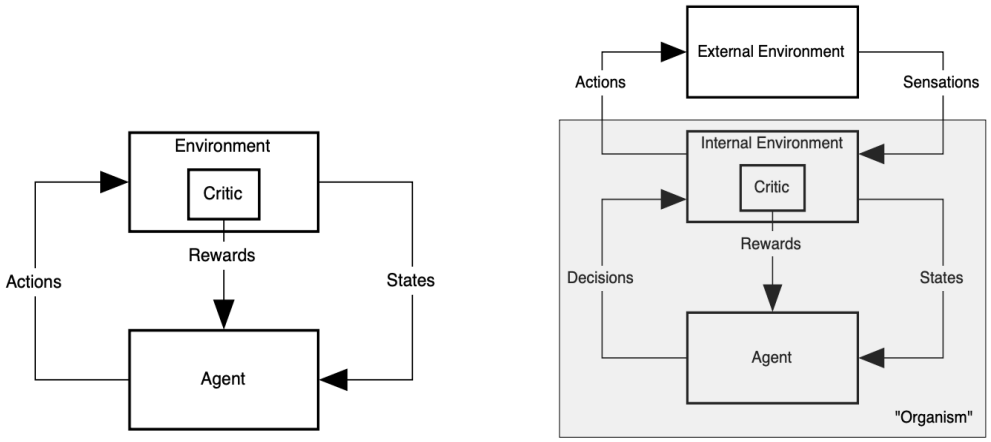
The critic updates its value parameters using the actor's output action a_{t+1} for the next state s_{t+1} in Equation 1.4. The hyperparameter β controls the strength of entropy regularization.

$$\Delta w = \beta(r(s_t, a_t) + \gamma \hat{q}_w(s_{t+1}, a_{t+1}) - \hat{q}_w(s_t, a_t))\nabla_w \hat{q}_w(s_t, a_t) \tag{1.4}$$

The Asynchronous Advantage Actor Critic (A3C) algorithm operates similarly but implements multiple agents that interact with different copies of an environment in parallel [18]. A global network hosts the shared parameters. At each time step, the agents have copies of the shared network. If an independent agent reaches t_{max} time steps or a terminal state, that network copy's gradient is computed from its own interaction. Independently of the other agents, the global network's parameters are then asynchronously updated by the independent agent's experience. As agents independently update global parameters, agents could potentially operate with different policy versions such that the aggregated parameter update would not be optimal [18]. In this research work, an A2C implementation is used that applies synchronous updates of global parameters [19]. The global update waits until all agents have finished their current interaction. Agents then begin new experiences with the same network parameters, enabling faster network training. This also allows for efficient use of Graphics Processing Units (GPUs).

1.2.2 Intrinsic Rewards

Traditionally, RL algorithms expect dense and well-shaped reward functions to continuously update their policy while interacting within an environment. However in general navigation and exploration tasks, positive feedback is received only when a target is reached. Exploring novel environments for a specific target poses the challenge of how to adequately provide positive external rewards to an artificial agent. In real world scenarios, unlike in traditional simulation environments, reward factors that an agent can interact with tend to not be well-shaped or common. This phenomenon, referred to as reward sparsity, limits a RL-based ANN by restricting its



(a) Standard Reinforcement Learning Interaction

(b) Reinforcement Learning Interaction with Internal Factor

Fig. 1.4. Agent Interaction with Environment in Reinforcement Learning [21]. (a) External critic in the environment provides a reward to an agent. (b) Decoupled external and internal environment components, in which the reward is provided by an internal critic.

ability to learn meaningful behavior and develop an optimal policy function. While "shaping" dense rewards for each specific environment is a possibility, this is not a scalable approach [20]. A viable alternative is to generate rewards "intrinsic" to the agent which complement external rewards. Intrinsic rewards help incentivize an agent to explore new states and provide feedback for an agent to better predict the outcomes of its actions [21]. When receiving sensory input, an agent retains an internal interpretation of the input, whereby an internal critic offers feedback to the agent, as illustrated with Fig 1.4. Under this premise, an internal critic offers an intrinsic reward.

This internal reward approach may be effectively implemented to help an agent acquire knowledge applicable to a range of problems, particularly efficient exploration in this instance. As a result, internal reward factors can help satisfy the desire for implementing a highly sample efficient and flexible model that generalizes across a variety of different environments. This internal component can instill a sense of

curiosity in an autonomous agent, therein incentivizing exploration of unknown parts of an environment, even given a lack of explicit external rewards.

As illustrating by [22], intrinsic rewards can be integrated with a Deep RL paradigm to improve performance. Using the reward signal as a scaled combination of both extrinsic rewards that an agent encounters and reward factors intrinsic to the agent, behaviors can more readily be encouraged and reinforced. [22] train an A3C agent using normal external rewards and intrinsic reward to perform navigation for instance.

1.3 Capsule Networks

[23] propose using a Capsule Network (CapNet) architecture to address shortcomings of Convolutional Neural Networks (CNNs). The max pooling operation used with CNNs reduces the spatial size of data flowing through a network and thus loses information. This leads to the problem that "[i]nternal data representation of a convolutional neural network does not take into account important spatial hierarchies between simple and complex objects" [23]. A CapNet resolves this by encoding the probability of detection of a feature as the length of their output vector. The state of the detected feature is encoded as the direction where that vector points. If detected features move around an image, then the probability, or vector length, remains constant while the vector orientation changes.

The idea of a capsule resembles the design of an artificial neuron but extends it to the vector form to enable more powerful representational capabilities. A capsule may be then understood as a small group of neurons [24]. As [23] note, "[t]he activities of the neurons within an active capsule represent the various properties of a particular entity that is present in the image. These properties can include . . . pose (position, size, orientation), deformation, velocity and texture." The use of capsules strives to bridge the discrepancy between ANNs that use feature detectors to output scalar values and more complicated highly-tuned computer vision methods, such as SIFT [25], that provide a vector encapsulating the pose of a feature [24].

Additionally, some advantages of CapNets over CNNs include viewpoint invariance, utilization of fewer trainable network parameters and superior generalization to new and different viewpoints. Viewpoint invariance is ensured by the use of pose matrices that recognize features irrespective of the viewing perspective. Due to the nature of capsules grouping together individual neurons, connections between layers need fewer parameters. Lastly with generalization to new viewpoints, the use of pose matrices can encode viewing characteristics, such as rotations, as linear transformations.

In an architecture with multiple layers of capsules, an active capsule receives vector inputs from capsules in a lower level. There are four operations performed on the input: matrix multiplication of input vectors, scalar weighting of input vectors, sum of weighted input vectors and lastly a vector-to-vector nonlinearity. These operations are illustrated in Fig. 1.5.

An approach called dynamic routing is the iterative method used to send lower-level capsule outputs to higher level capsules with similar outputs. Dynamic routing, a type of "routing-by-agreement," enables neurons in a layer to accept the "most active feature detector in a local pool" in a lower capsule layer [23]. In turn the dynamic routing method aids with segmenting overlapping objects. As noted by [23] as well, "[f]or low level capsules, location information is "place-coded" by which capsule is active. As we ascend the hierarchy, more and more of the positional information is "rate-coded" in the real-valued components of the output vector of a capsule. This shift from place-coding to rate-coding combined with the fact that higher-level capsules represent more complex entities with more degrees of freedom suggests that the dimensionality of capsules should increase as we ascend the hierarchy."

Dynamic routing addresses how to calculate a network forward pass with capsules. The method determines the vector c_i, which is all the routing weights for a given lower level capsule i. This is done for all lower level capsules. After this, the routing algorithm looks at each higher level capsule, such as capsule j, to check each input and update weights in the formula. A lower level capsule tries to map its output to the higher level capsule whose output is most similar. A dot product gauges this

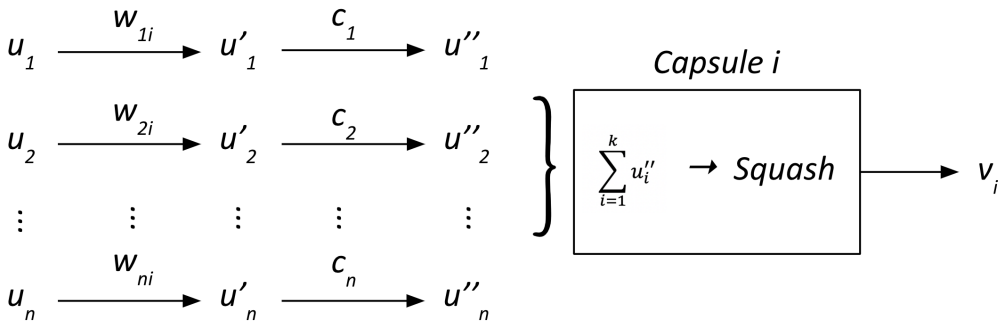


Fig. 1.5. Illustration of Capsule Operations. A capsule receives input vectors u_1 through u_n. Vector lengths encode probabilities that lower-level capsules detected an object, while the vectors' directions encode the state of detected objects. An affine transformation with weight matrices W_{1i} through W_{ni} is applied to each vector. The weight matrices encode spatial and other relationships between lower level features and higher ones. After multiplication, the vectors u'_1 through u'_n represent the predicted position of higher level features. These vectors are multiplied by scalar weights c_1 to c_n, derived using the routing algorithm, to determine which higher level capsule a lower level capsule's output maps to. The k weighted input vectors u''_1 through u''_n that map to Capsule i are then summed to form one vector. The Squash nonlinear activation function takes the vector, forces it to length of max one, while not changing its direction, and then outputs vector v_i [23].

similarity between a capsule input and output. Upper level capsules consist of a weighted sum of lower level capsules. The algorithm repeats the process of matching lower level capsule outputs to the appropriate higher level capsule r times, where r is the number of routing iterations.

1.4 Outline

This section presents this document's structure. Chapter 2 "Related Work" provides a survey of the current state of research work relevant to this research. Chapter 3 "Methodology" reviews the procedures used in the experiments to evaluate and

compare module performance. The Main Body consists of content from published articles covering the progression of this research work. Based on the research article "Augmented Curiosity: Depth and Optical Flow Prediction for Efficient Exploration," Chapter 4 discusses the Augmented Curiosity Modules architectures and presents experiment results and conclusions. Chapter 5 focuses on the research work from the article "Applied Capsule Networks for Reinforcement Learning-based Exploration" and presents the Capsules Exploration Module architecture and module experiment findings. Lastly, Chapter 6 encapsulates concluding remarks and discussion of this research and future work.

2. RELATED WORK

To expand autonomous capabilities with navigation and exploration, several approaches have been taken in the field of deep learning. [26–28] incorporate SLAM through use of ANNs to enable real-time use and effective feature detection across challenging 3D environments. However, their approaches still struggle in real world cases with higher complexity and variability. Other proposed means include incorporate environment mapping and leveraging artificial memory structures for improving autonomous exploration [29–31]. While these methods build upon existing long short-term memory (LSTM) baselines, they can suffer from difficulty with simultaneously accessing memory and planning actions as well as handling high dimensional inputs. Furthermore, these approaches do not scale efficiently beyond 2D environments to 3D simulations or generalize to more realistic and complex situations. Additionally, they tend to exhibit low sample efficiency, requiring well-shaped rewards and a large amount of interactions to converge to optimal policy functions that govern an agent's behavior. While recent approaches for UAV autonomous navigation and item delivery leverage Global Positioning Software (GPS) or Global Navigation Satellite System (GLONASS) to provide localization and guidance information for UAV flight outdoors, leveraging these techniques would not be practical for indoor scenarios or in GPS-denied environments [32–34].

Intrinsic motivation signals have been used with Deep RL to address sparsity of well-shaped and frequent rewards through using predictive error as a reward signal [21, 22, 35–37]. However as discussed by [38], prediction of future states in high dimensional space presents a challenging task that factors into performance of intrinsically motivated learning. Additionally as demonstrated by the "noisy TV" problem, stochastic dynamics present an obstacle to intrinsic motivation methods [38, 39]. An agent will tend to pursue transitions in environments with high entropy that do not

facilitate achieving the desired goal, subsequently impairing learning of an effective policy. [40] propose using imitation learning to resolve the issue of sparse rewards for RL and to better generalize performance to complex environments, however their method requires using expert knowledge for pre-training and is untested in dynamic environments. This research work presents modules capable of effectively predicting future states in complex environments by utilizing improved network embeddings and addressing the problem of stochastic dynamics in an environment. Moreover, the proposed methods incorporate high sample efficiency and function well in complex environments.

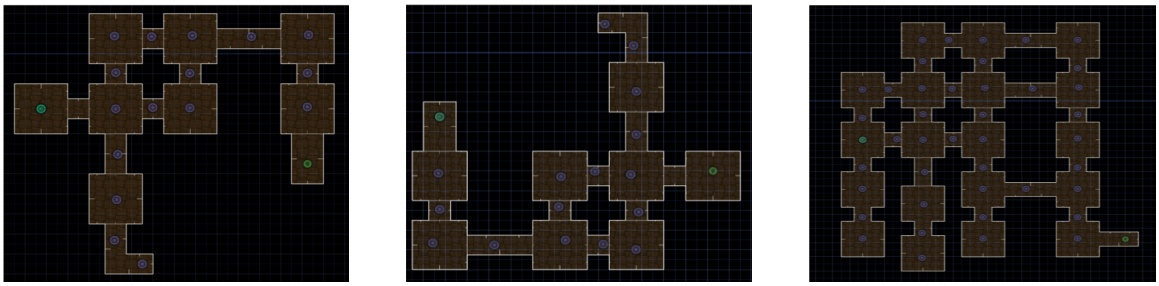
CapsNets have been recently proposed as an alternative to traditional CNNs and appear advantageous for use in Deep RL and exploration despite limited analysis in research [41, 42]. Given that CapsNets are a recent development, published research on their applications is limited. [43] integrates CapsNets with Deep-Q Learning algorithms to evaluate performance across several different environments, including one with a task of exploring a maze. However, discussion of the architecture used by the author is limited, and the results show that CapsNets underperform traditional CNNs. Other work by [44] applies CapsNets to recurrent networks, and [44–46] successfully use CapsNets for image and object classification tasks. CapsNets have also been used for problems with autonomous driving in [47] and are effective with predicting depth for SLAM implementations as in [48]. [49] demonstrate how CapsNets may result in reduced neural network model training time and offer a lower number of training parameters relative to similar CNN architectures. [50] additionally highlight how capsules present more explainable internal operations than CNNs for better understanding of deep learning models. The Caps-EM offers novel work on utilizing CapsNets in a Deep RL approach, pairing CapsNets with an A2C algorithm, and with respect to autonomous exploration of environments.

3. METHODOLOGY

This chapter discusses the experiment procedures for assessing each of the proposed modules as well as the various evaluation scenarios used to compare the implementations. This portion also reviews how curriculum training is used as part of the evaluation process and to judge how modules' learned policy functions perform in different scenarios.

3.1 Experiment Environments

Environments in ViZDoom, a Doom-based AI research platform were set-up to evaluate each modules' abilities to explore and search for objectives in environments with sparse external rewards [51]. In these scenarios, the agent's goal is to locate a piece of armor. The simulations restart after the agent reaches its goal and receives a terminal reward of +1, or after exceeding 2100 time steps without finding the target. The base scenario used for evaluations are "MyWayHome-v0" (MWH), a part of the OpenAIGym [52]. MWH is composed of 8 different rooms, each room having its own distinct wall texture. To add variability and robustness to the testing process, two new scenarios were created. The first, "My Way Home Mirrored" (MWH-M), is a re-oriented version of MWH. Besides providing an additional testbed for the various networks, MWH-M tests how well networks' learned knowledge applies between scenarios with curriculum training. To evaluate the modules' limitations in complex scenarios, the scenario "My Way Home Giant" (MWH-G), with 19 rooms, was also constructed. These scenarios are shown in Fig. 3.1.



(a) My Way Home (b) My Way Home Mirror (c) My Way Home Giant

Fig. 3.1. ViZDoom Scenarios. The target is the green circle at the rightmost part of each image. In the dense case, an agent may start at the leftmost green circle or any purple circle. For the sparse case, an agent spawns at the location indicated by the green circle situated at the left of each scenario.

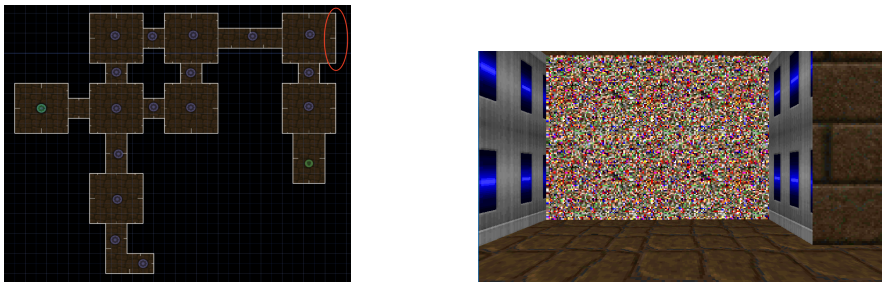
For each scenario, variability is incorporated in two more ways through variability in reward sparsity and variability in visual texture features [53]. For reward sparsity, a scenario has either a sparse or dense setup to offer different degrees of complexity with exploration. In both cases, the target location remains constant, while the agent's start location may vary. With the dense setting, the agent starts randomly at 1 of 17 uniformly distributed locations to begin exploring from in MWH or MWH-M. There are 42 possible start locations in MWH-G. Some start locations are far away from the target, while others are close to the target. These diverse positions allow the agent to reach the target even by random actions. Conversely with sparse, the start location is far from the target and does not vary. This requires the agent to take several directed actions to reach its goal.

For visual features in scenarios, experiments incorporate two variants of the texture of the maze walls. While the layout of the environment remains the same, the texture, or pattern, of the walls is modified to either be uniform or vary across every room in the environment. The variants are referred to as either uniform texture scenarios or varied texture scenarios, respectively, as shown in Fig. 3.2. In uniform scenarios, a model must have a high-level and abstract understanding of the envi-

(a) Uniform Texture (b) Varied Texture

Fig. 3.2. Variability in Visual Texture Features. (a) Shows a frame from the agent's point of view in a uniform texture scenario. (b) Agent's view of a varied texture scenario.

ronment to adequately explore and locate the target without information from rich visual features in the environment. Each scenario then has 4 variations. For example, the MWH scenario has the following setups: MWH Dense Texture, MWH Dense Uniform, MWH Sparse Texture and MWH Sparse Uniform.



(a) "Noisy TV" Wall Location (b) "Noisy TV" Wall

Fig. 3.3. "Noisy TV" Wall. (a) Shows the location of the "noisy TV" wall in the MWH scenario. (b) Agent's view of the "noisy TV" wall.

In order to study the effect of stochastic dynamics on the modules' performance, the "noisy TV" problem is also simulated in an evaluation environment. Using a custom-made "noisy TV" wall in ViZDoom, as shown in Fig. 3.3b, the wall models a source of high entropy. The wall displays one of fifteen unique static noise images

and cycles through the images at a rate of 0.1 game ticks per image. While this wall animation of simulated noise is not purely stochastic in nature, it suffices to produce a source of perceived randomness from the perspective of an agent. This wall is located in one location in the MWH Dense, Texture scenario, as highlighted in Fig. 3.3a. This particular location is used, because the agent will view this wall prior to reaching the goal objective. In experiments, this "noisy TV" problem is evaluated using the "noisy TV" wall in the MWH Dense Texture scenario.

3.2 Curriculum Training

To evaluate how each of the proposed approaches transfer and generalize knowledge between different scenarios, curriculum training, or curriculum learning, is used with the MWH-M and MWH-G environments as discussed in Section 3.1 [54]. For this procedure, a module is trained until having converged to an optimal policy in each of the MWH Dense Varied Texture, Dense Uniform Texture, Sparse Varied Texture and Sparse Uniform Texture scenarios. The learned network parameter values associated with these respective scenarios are then pre-loaded into the same module prior to beginning training again in the respective MWH-M and MWH-G Dense Varied Texture, Dense Uniform Texture, Sparse Varied Texture and Sparse Uniform Texture scenarios. In this way, the module begins training in the MWH-M and MWH-G environments with the prior knowledge learned from the MWH scenarios. The various modules are also allowed to train in MWH-M and MWH-G as well without use of curriculum training, and the results of the two different approaches are compared. Applying this method illustrates how the ANN training process network can be made more efficient and enable quicker convergence to an optimal policy function than without curriculum training.

4. AUGMENTED CURIOSITY MODULES (ACMS)

4.1 Overview

This chapter discusses the proposed Augmented Curiosity Modules including the rationale behind their design as well as an analysis of their performance in testing scenarios. The ACM are based on the Intrinsic Curiosity Module (ICM) by [53], who propose an intrinsic reward generator that supplements external rewards found in an environment. The ICM receives three inputs, an agent's current state, its next state and the action taken by the agent to move from the current state to the next state. In the inverse model, the ICM uses the current state feature embedding and the next state feature embedding to predict the agent's sampled action. The embeddings are outputs from shared weight neural networks which transform the original input state space into a feature space. Given that the labels for training are the actions sampled from the policy function at each step, this process falls under the self-supervision paradigm. As the only incentive for the inverse model is to predict the action leading to the next state, the training process ensures that the feature space only encodes information influencing the specific actions.

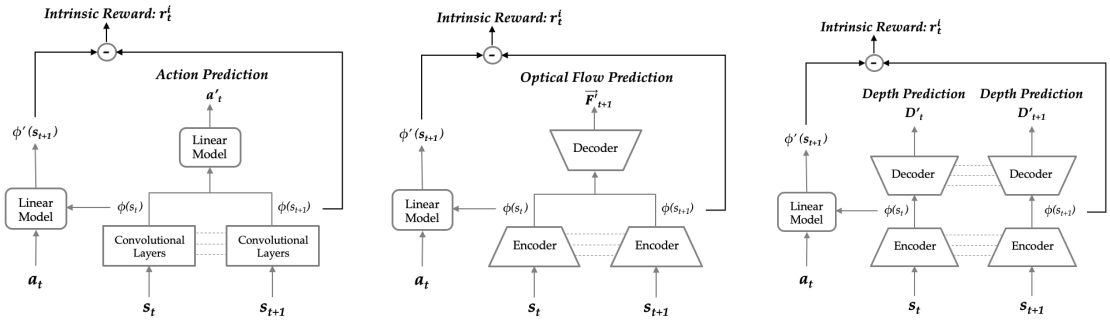
Provided the current state embedding and an agent's sampled action as inputs, the ICM trains a forward dynamics model to predict a feature space embedding of the next state. An intrinsic reward is generated from the error between the forward model's prediction of the next state embedding and the actual next state ground truth embedding. The intrinsic reward factor for example then grows large whenever the agent explores previously unvisited areas and makes inaccurate predictions. In turn, an agent experiences a rewarding sense of "curiosity" by visiting new and unfamiliar areas. An A3C algorithm strives to maximize the sum of an extrinsic reward provided by the environment and the intrinsic reward signal provided by the ICM [18].

With this proposed research approach to reward sparsity, the notion of intrinsic reward signals are leveraged. Additionally, the ACMs, named the Optical Flow-Augmented Curiosity Module (OF-ACM), Depth-Augmented Curiosity Module (D-ACM) and Depth and Optical Flow Combined-Augmented Curiosity Module (C-ACM) utilize optical flow data, depth data and a combination of the two, respectively, to develop superior embeddings for exploration. Optical flow describes the motion of elements in a visual scene caused by the relative motion between an observer and the given scene. Optical flow offers knowledge on how pixels change position in subsequent frames. In static scenarios, optical flow gives information on how the agent's position changes. Depth images also provide a better sense of relationships between objects than standard RGB images [55]. By using optical flow and depth information rather than only pixels from RGB images, the OF-ACM and D-ACM incorporate localization information and higher level representations of a given scene. Whereas the ICM inverse model predicts the action relating two consecutive states, the ACMs have modified inverse models predicting either a depth image or the optical flow between two states. In the C-ACM, two different curiosity-based predictive modules are initialized to separately generate depth and optical flow predictions.

This chapter demonstrates how the combined use of intrinsic rewards systems and supplementary input data results in a capable exploration system, even in scenarios with sparse external rewards. Through utilizing readily available information in depth and optical flow in addition to standard RGB image data, a more efficient approach is achieved.

4.2 Augmented Curiosity Modules Architectures

The ICM generates a feature representation of the input states by predicting the action that connects two subsequent states. The ACM approaches extend this framework to support prediction of depth, optical flow and a combination of the two



(a) ICM (b) OF-ACM (c) D-ACM

Fig. 4.1. (a) The ICM encodes 42x42 RGB frames from the agent's point of view at states s_t and s_{t+1} into ϕ_t and ϕ_{t+1} by a Convolutional Layer block. Embeddings are concatenated and passed to linear layers block to predict action a'_t. Through a linear block, ϕ_t and label a'_t are used to predict ϕ'_{t+1}. The difference between ϕ'_{t+1} and ϕ_{t+1} is used as intrinsic reward r_t^i [22]. (b) OF-ACM leverages latent space in an encoder-decoder as ϕ_t and ϕ_{t+1} embeddings. (c) D-ACM predicts a depth image from each input frame. Convolutional Layer blocks consist of four convolutional layers, each with 32 filters and kernel size 3x3. Each layer is followed by batch normalization and an Exponential Linear Unit (ELU) as an activation function. Linear Model blocks are defined as two fully connected layers with an ELU activation between them. Encoder and Decoder blocks consist of four layers of 32 filters with 3x3 kernels followed by equal numbers of filters and kernel size deconvolutions. Dashed lines represent shared weights between networks.

when creating feature embedding representations. These network architectures are described in Fig. 4.1.

The original ICM assumes two key arguments with regards to the curiosity reward. First, predictions of a future state in the forward model using a feature space instead of pixel space is more advantageous than previous approaches. When the dimension of predictions is lower, the predictions are simpler. More importantly, changes in pixels themselves might not be the real objective, as these changes may or may not be influenced by an agent's actions. Secondly, to model only changes in an environment that are direct outcomes of an agent's actions, the inverse model prediction must

provide a proper embedding in the feature space. The ICM's embeddings of the current and next state, ϕ_t and ϕ_{t+1}, are obtained from predicting an action when given the successive states, s_t and s_{t+1}, as inputs. This encodes information from the input state to the embeddings that is related to an agent's actions in the environment.

As a result, the quality of the curiosity reward depends on the embeddings. Predicting only actions a'_t with the inverse model does provide an adequate embedding for a curious agent. However by using additional information in the prediction, the embeddings can better summarize changes in pixels and interpret textural and structural aspects of an environment.

The A2C component for the ACMs consists of four layers of a 3x3 convolution kernels with stride 2 and padding 1 following by a batch normalization operation and ReLU activation function. The output of the fourth layer is passed through an LSTM then through two separate linear layers that output the policy function or value function. Similarly as the ACMs, the input is a 42x42 RGB frame of the agent's point of view for a given state.

4.2.1 Optical Flow-Augmented Curiosity Module (OF-ACM)

The first proposed variant of the ACMs uses the inverse model to predict optical flow instead of the action between frame inputs. This method uses the OpenCV library's implementation of dense optical flow with Gunnar Farneback's algorithm [56]. The output $\overrightarrow{F}_{t+1}$ consists of a 2D matrix measuring the displacement of each pixel in a frame when compared to its predecessor. OpenCV's implementation provides a real-time optical flow output that is used as the ground-truth for training the modified inverse model's prediction. The network architecture changes considerably given that this approach reconstructs a 2D pixel displacement matrix from image inputs. The inverse model uses an autoencoder architecture. States s_t and s_{t+1} are input frames and are transformed through convolutional layers into features in a latent space. After concatenating these features, the result is passed through deconvolution layers to

generate the desired output matrix. During training, the autoencoder's latent space learns to encode the key information relevant to reconstructing the optical flow prediction $\overrightarrow{F'}_{t+1}$ from input images. This is due to how network loss only depends on creating an accurate optical flow reconstruction. This latent space serves as the new feature space for embeddings ϕ_t and ϕ_{t+1}. The action a_t and embedding ϕ_t are inputs to the forward model, which outputs the predicted next state's embedding ϕ'_{t+1}.

While the ICM's action prediction process accounts for pixel information, the OF-ACM enables embeddings to contain information on the displacement of tracked pixels. Even if pixels are redundant in predictions, the constraint of predicting dense sets of pixels provides embeddings a better awareness of the changes in the perceived environment correlated to the agent's movement.

4.2.2 Depth-Augmented Curiosity Module (D-ACM)

The D-ACM extends the principles from the OF-ACM but rather with depth images. The inverse model uses an autoencoder to obtain embeddings from the latent space features. The autoencoder predicts the depth images D'_t and D'_{t+1} corresponding to the RGB inputs. The ViZDoom platform provides ground-truth depth images that are used as labels. These depth images present pixels in gray scale, and the pixel intensity is varied dependent on how near or far an object is from the agent. While this approach does not necessarily encode the influence of an agent's actions in embeddings, it provides embeddings with structural information of the environment. From previous work on localization and navigation tasks by [57, 58], 3D information appears to provide a better representation of environments than 2D texture information. These depth-based embeddings contain important information regarding how agents' movement affects its perceived 3D environment. Therefore, they offer advantageous representations and more efficiently inform the agent when venturing to a new area.

4.2.3 Depth + Optical Flow Combined-Augmented Curiosity Module (C-ACM)

With the C-ACM approach, the D-ACM and OF-ACM, as shown in Fig. 4.1, are setup and operate independently to each contribute towards a single intrinsic reward factor. Each module still receives the same input, an RGB frame of the agent's view. In this approach however, the D-ACM and OF-ACM prediction error outputs are both considered. The C-ACM intrinsic reward signal is an equally weighted value of the intrinsic reward from each the D-ACM and OF-ACM. The singular intrinsic reward factor is then used in the same manner as previously discussed to supplement the overall reward signal. The motivation behind this approach is to leverage the strengths of both the D-ACM and OF-ACM. Each module handles and forms predictions of different information, depth and optical flow data respectively, of a given scene and context. As a result, each module maintains separate embeddings of the environments for formulating different predictions of the next scene. Ultimately by pooling together each prediction error, the C-ACM could provide a better intrinsic reward signal for improving autonomous exploration.

4.3 Results

Table 4.1 below illustrates differences in the size and performance of each module. The percent difference columns compare the proposed ACM variants to the ICM, which serves as the baseline for performance. The ICM consists of fewer trainable parameters than the other variants, however the discrepancy is marginal. Also noted is the comparison in module performance with respect to timing, where the ICM completes a single training step fastest. The D-ACM and OF-ACM are comparable with respect to timing and parameters. These standardized timing parameters were obtained from testing each module type individually in MWH Varied Texture while utilizing twenty parallel workers on the GeForce GTX 1080 GPU with 8114 MiB of

memory. With this information, the test results are standardized in terms of relative time.

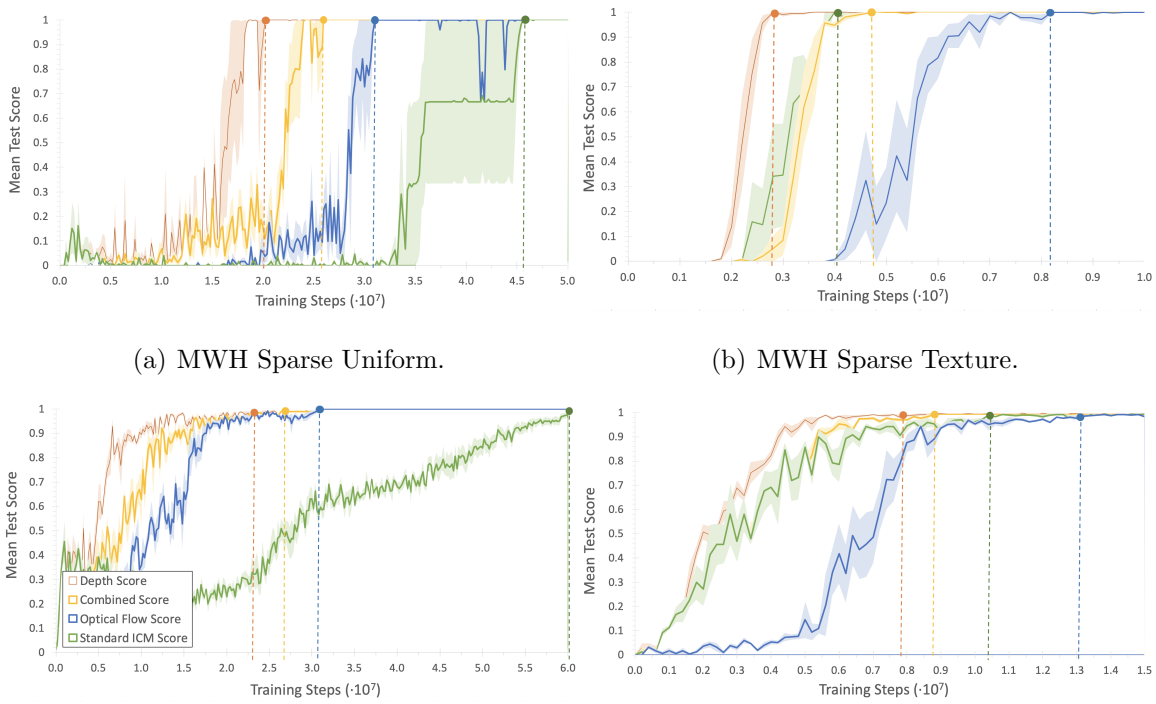
Table 4.1.
Module Size and Timing Comparisons.

Module Type	# of Trainable Parameters	% Difference	Time to Complete 1 Step (ms)	% Difference
ICM	915,945	Baseline	1.13	Baseline
D-ACM	944,170	+3%	1.36	+18.5%
OF-ACM	953,675	+4.1%	1.63	+36.2%
C-ACM	1,308,560	+35.3%	2.46	+74.1%

Table 4.2.
My Way Home Scenario Time Performance Results. Time required to converge to an optimal policy function is in seconds.

Module Type	Scenario			
	Dense, Varied	Dense, Uniform	Sparse, Varied	Sparse, Uniform
ICM (s)	1.24E+04	6.78E+04	4.52E+03	5.14E+04
D-ACM (s)	1.02E+04	3.06E+04	3.74E+03	2.72E+04
% Difference	-21.9%	-121.6%	-20.9%	-89.0%
OF-ACM (s)	2.14E+04	4.89E+04	1.32E+04	5.05E+04
% Difference	+41.8%	-38.7%	+65.8%	-1.8%
C-ACM (s)	2.10E+04	6.64E+04	1.13E+04	6.27E+04
% Difference	+40.9%	-2.1%	+60.1%	+18.0%

Fig. 4.2(a-d) shows the performance of the ICM relative to the predictive modules in the MWH dense and sparse scenarios. The results of Fig. 4.2 are summarized in Table 4.2. The ICM performance lags behind that of the D-ACM but is comparable to



(a) MWH Sparse Uniform.

(b) MWH Sparse Texture.

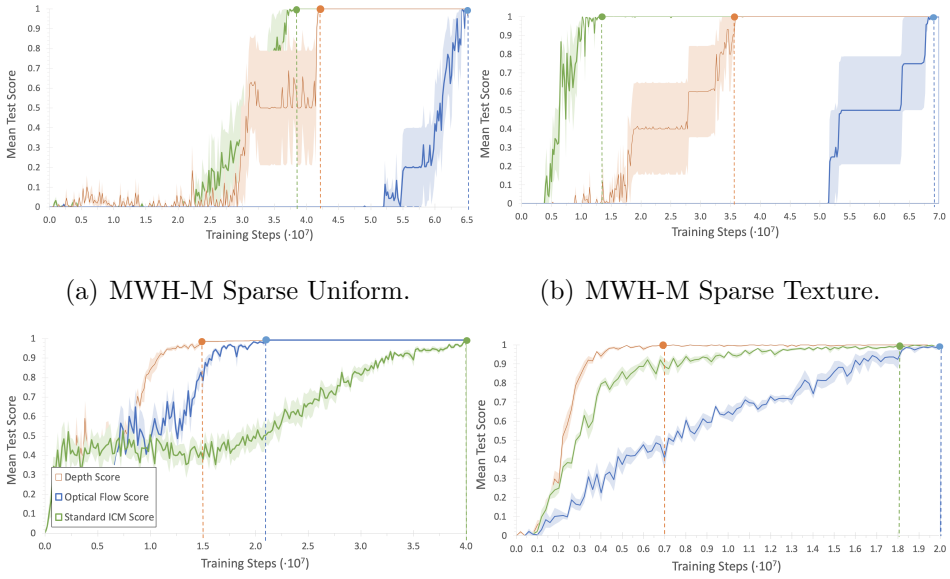
(c) MWH Dense Uniform.

(d) MWH Dense Texture.

Fig. 4.2. My Way Home Results. Across evaluation scenarios, a minimum of five instances of each module variant are averaged for the score trend line. The shaded area around each trend line indicates the one standard error range. The ovals roughly indicate where a module has completely converged to an optimal policy function.

the OF-ACM. The uniformity in maze texture proves to be significantly challenging to the different modules, as they converge to optimal policies faster in the varied texture scenarios. These evaluations demonstrate the superior performance of the D-ACM over the others. In initial tests, the C-ACM fails to outperform the D-ACM, typically matching the OF-ACM or ICM performance. Thus, the C-ACM is omitted in the subsequent evaluations.

Figs. 4.3(a-d) and 4.4(a-d) show results from testing in the MWH-M scenarios, with and without use of curriculum training. Results for these plots are summarized below in Table 4.3. From the results, one is able to gauge how well the ICM, D-ACM and OF-ACM are able to apply previously learned knowledge.



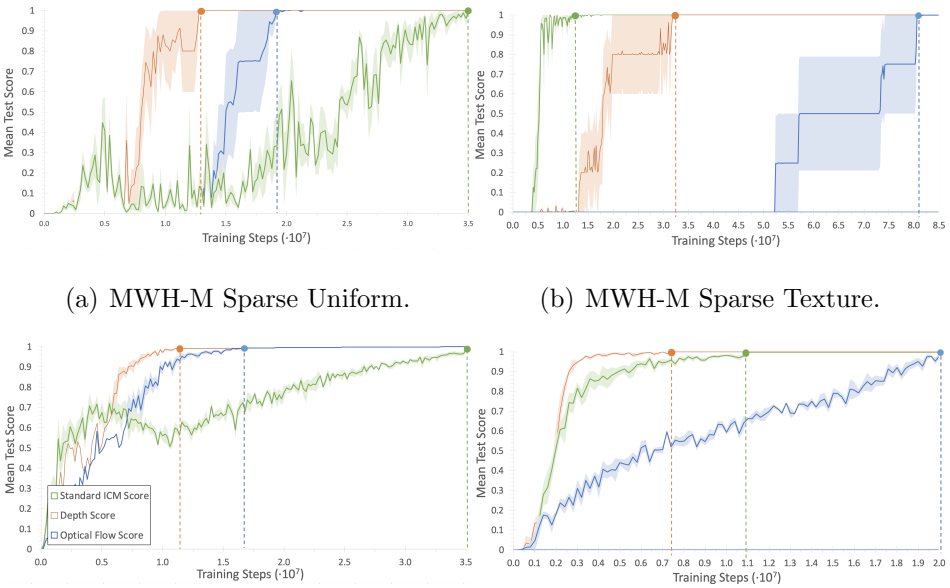
(a) MWH-M Sparse Uniform.

(b) MWH-M Sparse Texture.

(c) MWH-M Dense Uniform.

(d) MWH-M Dense Texture.

Fig. 4.3. My Way Home Mirrored Results.



(a) MWH-M Sparse Uniform.

(b) MWH-M Sparse Texture.

(c) MWH-M Dense Uniform.

(d) MWH-M Dense Texture.

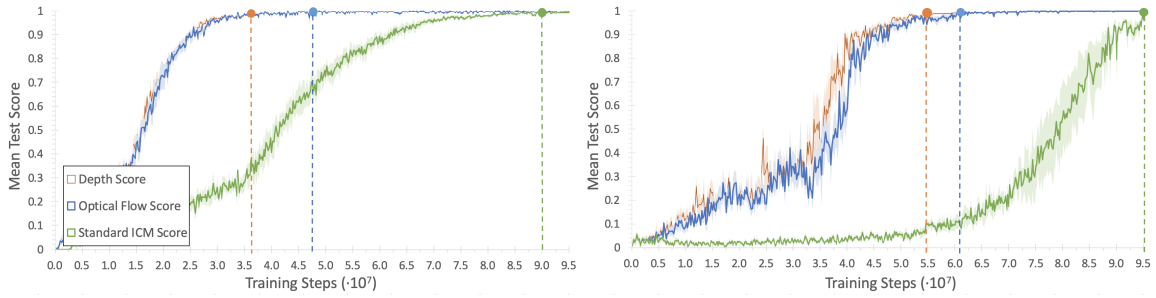
Fig. 4.4. My Way Home Mirrored with Curriculum Training Results.

As Table 4.3 depicts, each of the modules is able to leverage curriculum training to obtain time improvements with converging to an optimal policy. The OF-ACM variant benefited the most from using curriculum training, followed by the D-ACM and lastly ICM. The ICM and D-ACM prove fairly comparable overall in the scenarios. The ICM performs slightly better in sparse scenarios, whereas the D-ACM moreso in the dense cases.

Table 4.3.
My Way Home Mirrored Scenario Time Performance Results, with and without Curriculum (Curr.) Training.

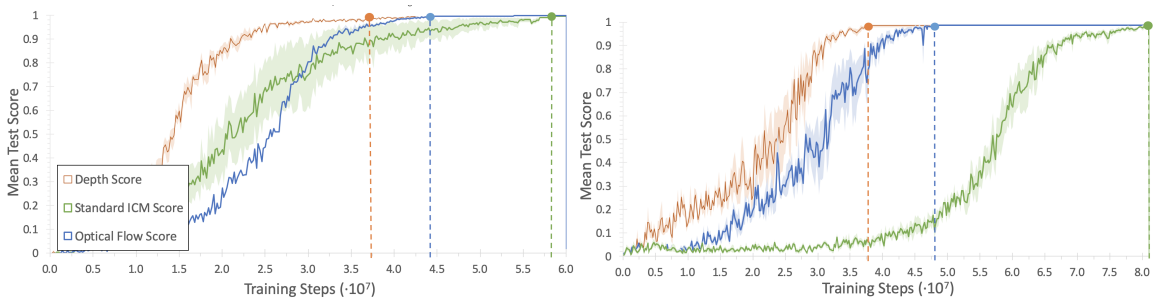
Module Type	Scenario			
	Dense, Varied	Dense, Uniform	Sparse, Varied	Sparse, Uniform
ICM (s) w/ Curr. Training	1.24E+04	3.96E+04	1.41E+03	3.96E+04
ICM (s) w/o Curr. Training	2.03E+04	4.52E+04	1.70E+03	4.29E+04
% Difference	+38.9%	+12.4%	+17.1%	+7.7%
D-ACM (s) w/ Curr. Training	1.02E+04	1.50E+04	4.22E+03	1.70E+04
D-ACM (s) w/o Curr. Training	8.84E+03	2.04E+04	4.83E+03	5.78E+04
% Difference	-15.4%	+26.5%	+12.6%	+70.6%
OF-ACM (s) w/ Curr. Training	3.26E+04	2.69E+04	1.32E+05	3.26E+04
OF-ACM (s) w/o Curr. Training	3.23E+04	3.42E+04	1.11E+05	1.06E+05
% Difference	-0.9%	+21.3%	-18.9%	+69.2%

Following evaluations in the MWH-M scenarios, the modules were tested in the MWH-G to determine their ability to perform in larger and more complex environments. Shown below in Fig. 4.5(a-b) and Fig. 4.6(a-b) are the results for performance in MWH-G, without and with use of curriculum training respectively.



(a) MWH-G Dense Texture. (b) MWH-G Dense Uniform.

Fig. 4.5. My Way Home Giant Results.



(a) MWH-G Dense Texture. (b) MWH-G Dense Uniform.

Fig. 4.6. My Way Home Giant with Curriculum Training Results.

The results of MWH-G are summarized in Table 4.4. In this scenario, all modules failed to achieve any form of an ideal policy when operating in the sparse scenario variants. Although the instances were allowed to run for 20E+7 training steps, they never reached the target in the training phase, regardless of the use of curriculum training. The D-ACM significantly outperformed the ICM and OF-ACM in the giant scenario as well, while the ICM benefits the most from use of curriculum training in this high complexity scenario. The modules did successfully manage to develop an optimal policy in the dense scenario variants, even given uniform textures.

Table 4.4.
My Way Home Giant Scenario Time Performance Results, with and without Curriculum (Curr.) Training.

Module Type	Scenario	
	Dense, Varied	Dense, Uniform
ICM (s) w/ Curr. Training	6.55E+04	9.15E+04
ICM (s) w/o Curr. Training	1.02E+05	1.07E+05
% Difference	+35.8%	+14.5%
D-ACM (s) w/ Curr. Training	5.10E+04	5.17E+04
D-ACM (s) w/o Curr. Training	5.92E+04	7.55E+04
% Difference	+13.9%	+31.5%
OF-ACM (s) w/ Curr. Training	7.12E+04	7.82E+04
OF-ACM (s) w/o Curr. Training	7.82E+04	1.02E+05
% Difference	+9.0%	+23.3%

4.4 Conclusions

Given the results, the D-ACM improves upon the ICM with an average 63.3% time improvement in MWH scenarios, 40.3% in MWH-M and 57.1% in MWH-G. This approach develops more generalized network embeddings across different scenarios and converges faster to an optimal policy function in environments with sparse external rewards. OF-ACM seems promising in larger scenarios, as shown in MWH-G. The D-ACM and OF-ACM perform similar to the ICM in MWH and MWH-M but drastically outperform the ICM in the more complex MWH-G. This highlights how depth and optical flow information enable better environment representations for exploration. However, when dealing with simpler scenarios, the representations do not offer a significantly discernible advantage. The C-ACM did not appear ad-

vantageous in any of the scenarios, primarily due to its relatively larger number of network parameters and subsequently longer training process.

This work illustrates how by leveraging additional information, rather than solely pixels from RGB images, a Deep RL agent can improve performance and scalability across scenarios to better handle the issue of reward sparsity. Additionally, performance may be improved by refining the combination of depth and optical flow data, such as with a combined loss function. While efforts on combining OF-ACM and D-ACM has not produced a module more effective than depth alone, this subject still presents an area of interest. Lastly while module performance given added stochastic dynamics is evaluated in the following chapter, future work could potentially further focus on evaluations in dynamic scenarios with moving elements independent of an agent.

5. CAPSULES EXPLORATION MODULE (CAPS-EM)

5.1 Overview

This chapter demonstrates how CapsNets perform well for approximating policy functions when paired with an A2C algorithm to improve autonomous agent navigation performance in sparse reward environments. Across a variety of test scenarios, the proposed Caps-EM uses a relatively small network size to improve upon the ICM and D-ACM capabilities, which are presented as performance baselines. Critically relevant is the fact that Caps-EM does not incorporate the use of intrinsic rewards, which the ICM and D-ACM approaches both use to converge to adequate policy functions. This research work highlights how strictly using external reward factors, Caps-EM achieves a more encompassing comprehension of image inputs and abstract world representation to achieve more meaningful action in given scenarios. Traditional CNNs fail to replicate these same advantages of CapsNets. While the Caps-EM struggles in certain test environments modeling extremely sparse external rewards, the module generalizes well across various scenarios with use of curriculum training and shows the capabilities of CapsNets in instances of realistic scenarios. Applied within a Deep RL framework, CapsNets advance autonomous system capabilities for navigation and exploration in challenging environments that can potentially be applied to robotics and UAVs for example.

5.2 Capsules Exploration Module Architecture

CapsNets initially appeared advantageous for the task of exploration even prior to experimentation. They can discern both the probability of detection of a feature, stored in an output vector's magnitude, and the state of the detected feature, stored

in a vector's direction [23]. Conversely, traditional CNNs are only able to handle the probability of detection of a feature. This difference proves vital as CapsNets can then maintain spatial relationships of observed items in environment. This distinction hypothetically enables creation of more sophisticated network embeddings of the environment space. This work demonstrates experimentation with combining CapsNets and A2C components as not previously explored in other published literature. Importantly, Caps-EM does not use intrinsic rewards like the approaches discussed in Fig. 4.1. The architecture implementation of the Caps-EM is illustrated in Fig. 5.1. It is important to note as well that strictly using an A2C network design exclusively with CNNs and no use of intrinsic reward signals cannot explore effectively. Such an approach failed to learn effectively and converge to a policy function in any of the evaluation scenarios unless paired extensively with imitation or curriculum learning.

As done by [23], the network proposed in Fig. 5.1 uses a convolutional layer before the capsule layers to detect basic features in the 2D RGB image inputs. The subsequent capsule layer then uses the detected basic features to produce combinations of the features. In addition to using RGB images as inputs to the network, other inputs were experimented with to analyze the impact on performance. Depth images and concatenated depth and RGB images were also used with CapsNet. As illustrated by the D-ACM, depth images can convey superior structural information of an environment than RGB images alone. Additionally then by concatenating together RGB and depth images of the same scene to generate a tensor input with 4 channels, these experiments can discern any significant advantages or disadvantages to the varied inputs.

From experimentation with various CapsNets architecture designs, using more than one convolution before the capsule layers masks the benefits of the capsule layers by lowering the data resolution and degrading performance. Additionally, using a larger network architecture with more trainable network parameters was not found to increase the module's performance in converging to an optimal policy function or with being generalizable. In fact, using a larger network architecture degrades over-

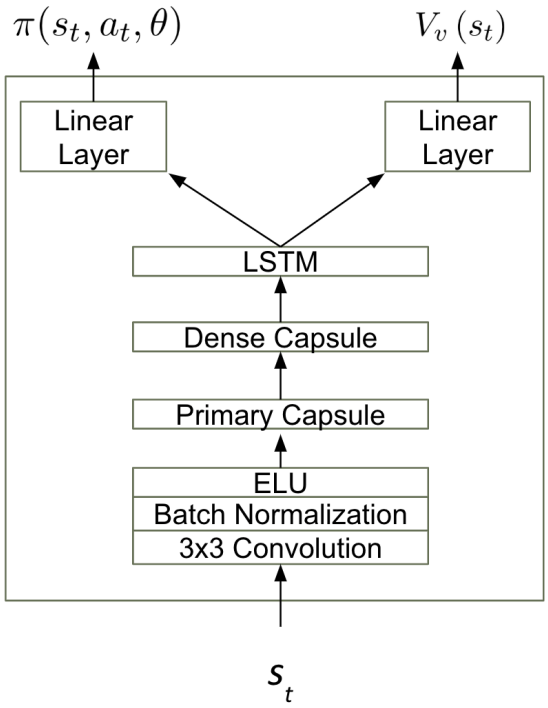


Fig. 5.1. Caps-EM A2C Network Architecture. In the Caps-EM, the input s_t, a 42x42 RGB image, first passes through a series of a 3x3 convolution, batch normalization function and ELU. ELU offers a faster learning rate than ReLU [59]. The remaining layer blocks consist of Capsule Network layers. The first lower level Primary Capsule layer consists of a 9x9 convolution, with stride of two and padding of zero, followed by a Squash activation function. This layer has 32 input and output channels with capsule dimension of eight. The outputs are dynamically routed to the second higher level Dense Capsule layer consisting of 196 input capsules of dimension eight and four output capsules of dimension 16. Three routing iterations are used in the routing algorithm. Outputs of the Dense Capsule layer are passed to an LSTM and linear layers to provide the policy function $\pi(s_t, a_t, \theta)$ and state value function $V_v(s_t)$.

all performance due to the network needing longer to train to converge to a policy. Three routing iterations are used between capsule layers as recommended by [23] to help prevent overfitting with training data and optimize the loss faster than using one routing iteration. However in experiments, the capsule layers still displayed a

tendency for overfitting between the training and testing phases. In these instances, the early stopping method was used to handle overfitting once the network successful achieved an adequate policy function [60]. Neither did incorporating dropout significantly improve the problem of network overfitting [61]. Dropout with p values of 0.25 as well as 0.5 were applied to various layers of the Caps-EM module, with the main effect only being a slowed training rate. The architecture described for Caps-EM was found to be one that balanced the desire for a generalizable network across all evaluation scenarios with a minimal number of network parameters.

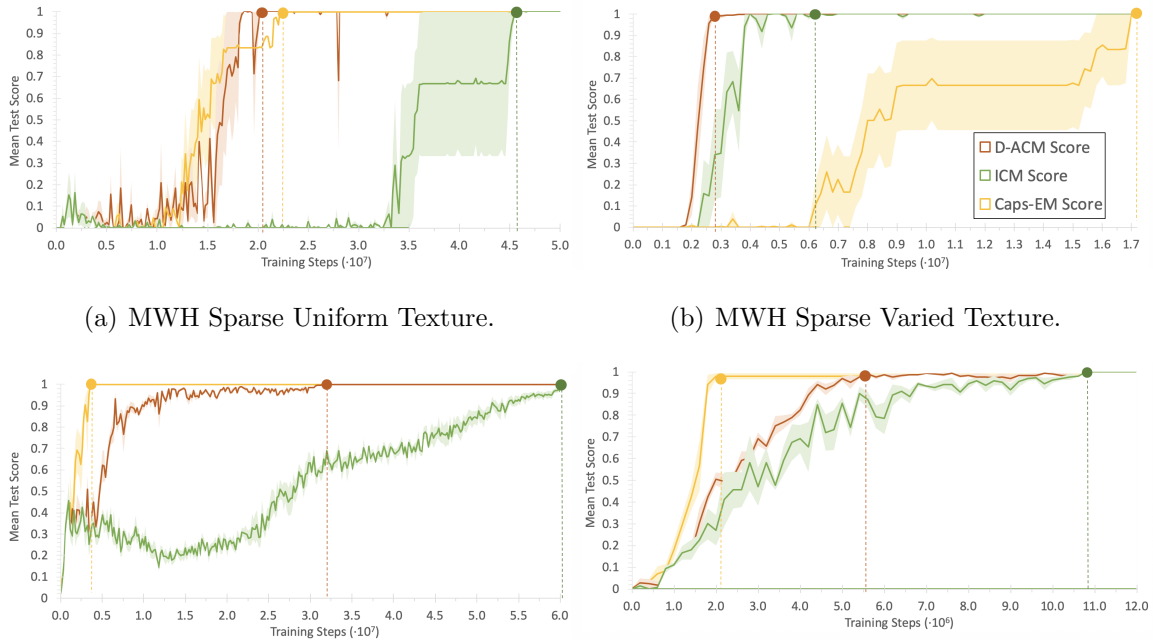
5.3 Results

For analyzing the Caps-EM performance, the ICM and the D-ACM are used as baselines for comparison. Table 5.1 compares these three approaches, where the percent difference rows indicate a module's improved or degraded metrics relative to the ICM. The number of trainable parameters of each module are used as a comparison metric to account for the differences in module size and scaling. The size, or number of trainable parameters, of each module has a direct impact on the efficiency and required time to complete the neural network model training process. The Caps-EM architecture has substantially fewer trainable network parameters, in turn completing a single training step more efficiently as well. The times to complete one training step as shown are standardized values obtained from running each module variation with one worker on a GeForce GTX 1080 GPU with 8114 MiB memory on the MWH Dense Varied Texture scenario. Result tables displayed further on showing timing analysis are based on these standardized times to present an equivalent metric of comparison. Plots are presented with the mean testing score of a module relative to the number of training steps taken.

As shown in Fig. 5.2 and Table 5.2, the Caps-EM performs exceptionally well in the dense setup scenarios. The Caps-EM completes the MWH Sparse Uniform scenario in roughly the same number of training steps as D-ACM. However, the

Table 5.1.
Module Size and Timing Comparisons.

Module Type	# of Trainable Parameters	% Difference	Time to Complete 1 Step (ms)	% Difference
ICM	915,945	Baseline	13.6	Baseline
D-ACM	944,170	+3%	17.6	+30%
Caps-EM	515,301	-44%	6.21	-54%



(a) MWH Sparse Uniform Texture.

(b) MWH Sparse Varied Texture.

(c) MWH Dense Uniform Texture.

(d) MWH Dense Varied Texture.

Fig. 5.2. My Way Home Results. A minimum of five instances of each module variant are averaged for the score trend line. The shaded area around trend lines indicates the one standard error range. Ovals approximate where a module has converged to a policy function.

Caps-EM performance is superior when considering the actual time to converge to an optimal policy and how Caps-EM does not use intrinsic rewards. Conversely in

Table 5.2.
My Way Home Scenario Time Performance Results. Time required to converge to an optimal policy function is in seconds.

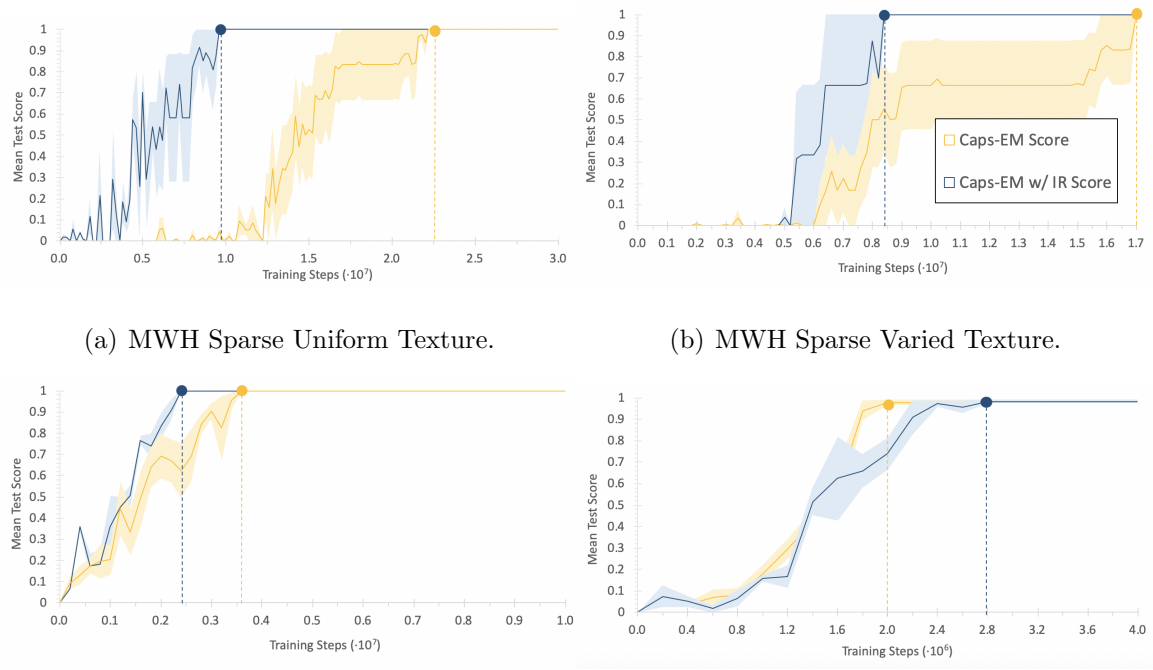
Module Type	Scenario			
	Dense, Varied	Dense, Uniform	Sparse, Varied	Sparse, Uniform
ICM (s)	1.46E+05	8.15E+05	5.44E+04	6.18E+05
D-ACM (s)	9.69E+04	2.64E+05	4.85E+04	3.52E+05
% Difference	-51%	-209%	-12%	-75%
Caps-EM (s)	1.24E+04	2.48E+04	1.07E+05	1.37E+05
% Difference	-1076%	-3182%	+49%	-353%

the Sparse Varied Texture scenario, the Caps-EM performs worse than both the ICM and D-ACM.

In order to assess how integrated intrinsic rewards with Caps-EM affects the module's performance with exploration, Fig. 5.3 shows a direct comparison of Caps-EM with and without intrinsic rewards in the same scenarios. The Caps-EM with 515,301 trainable parameters is 42% smaller than Caps-EM with intrinsic rewards, referred to as Caps-EM (IR), which has 733,705 trainable parameters. Caps-EM (IR) incorporates the approach discussed in Fig. 4.1a to generate an intrinsic reward based on the accuracy of next state predictions. Caps-EM requires 6.21E-03 seconds to complete one training step, whereas Caps-EM (IR) takes 2.09E-2 seconds and is 236% slower. Table 5.3 shows a comparison of performance with respect to time across the MWH scenarios and illustrates that Caps-EM (IR) exhibits poorer performance in each setup. In this analysis, intrinsic reward do not necessarily improve performance in the Sparse Varied Texture scenario.

Table 5.4 illustrates the use of different inputs to the Caps-EM. While the standard variant of Caps-EM receives 2D RGB inputs, a modified module, denoted as Caps-EM (RGB + Depth), receives an input of a concatenated 2D RGB and depth image

of the same scene. With this table, one can see how this varied input improves Caps-EM performance. While the Caps-EM with only the RGB input managed 1076%, 3182% and 353% percent improvement across the MWH Dense Varied, Dense Uniform and Sparse Uniform scenarios, respectively, the Caps-EM with RGB and depth inputs has an improvement of 1453%, 5967% and 400% difference across the same scenarios. In the case of the sparse varied scenario also, performance is improved as the Caps-EM with RGB and depth inputs lowered from a +49% difference to +24%. Experiments strictly using only depth images as inputs to Caps-EM did not display model convergence to a policy function in either of the sparse scenarios. However as shown, the combination of depth and RGB inputs ultimately improves module performance across all test scenarios and reduces module overfitting in texture scenarios.



(a) MWH Sparse Uniform Texture.

(b) MWH Sparse Varied Texture.

(c) MWH Dense Uniform Texture.

(d) MWH-M Dense Varied Texture.

Fig. 5.3. My Way Home Scenario Caps-EM Results, with and without Intrinsic Rewards (IR).

Table 5.3.
My Way Home Scenario Caps-EM Time Performance Results, with and without Intrinsic Rewards (IR).

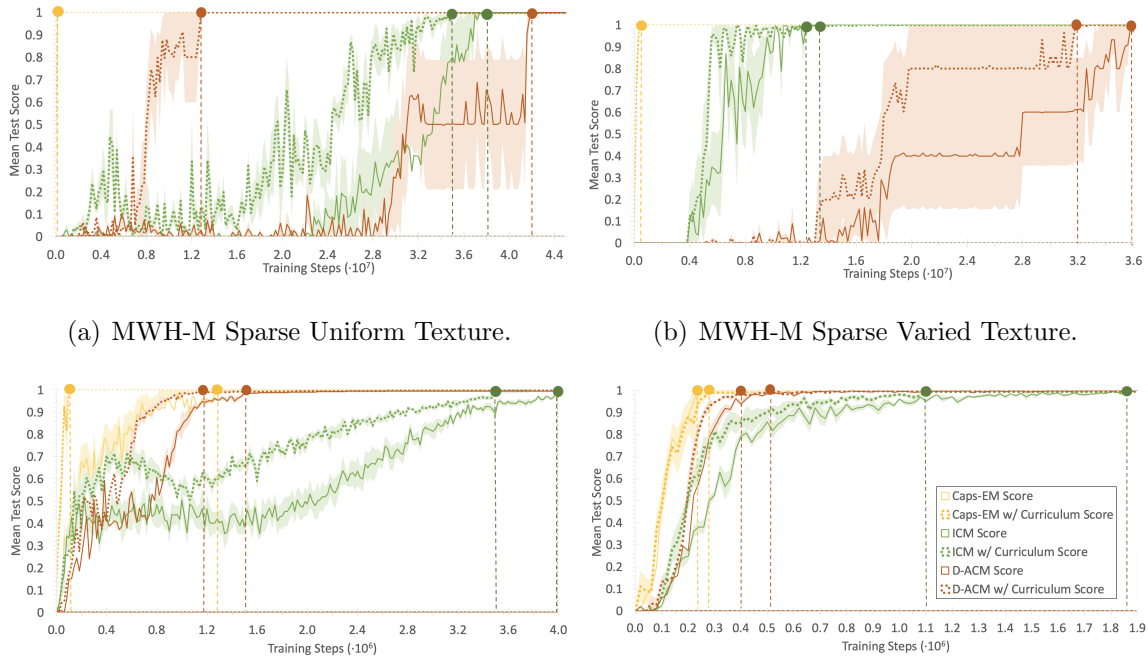
Module Type	Scenario			
	Dense, Varied	Dense, Uniform	Sparse, Varied	Sparse, Uniform
Caps-EM (s)	1.24E+04	2.48E+04	1.07E+05	1.37E+05
Caps-EM (IR) (s)	5.74E+04	5.22E+04	1.78E+05	2.09E+05
% Difference	+78%	+52%	+40%	+35%

Table 5.4.
My Way Home Scenario Caps-EM Time Performance Results, with RGB and RGB + Depth Inputs.

Module Type	Scenario			
	Dense, Varied	Dense, Uniform	Sparse, Varied	Sparse, Uniform
ICM (s)	1.46E+05	8.15E+05	5.44E+04	6.18E+05
Caps-EM (RGB) (s)	1.24E+04	2.48E+04	1.07E+05	1.37E+05
% Difference	-1076%	-3182%	+49%	-353%
Caps-EM (RGB + Depth) (s)	9.41E+03	1.34E+04	7.12E+04	1.24E+05
% Difference	-1453%	-5967%	+24%	-400%

Fig. 5.4 and Table 5.5 show how well each module applies learned network parameter weights from the MWH scenarios to the MWH-M scenarios. The expectation is that the knowledge should generalize well and enable the modules to converge to a successful policy faster than without using curriculum training. The Caps-EM fails to converge to a policy function in the MWH-M Sparse Varied Texture and Sparse Uniform Texture scenarios within 1.0E+8 training steps when not using curriculum training. This may due to the how extremely sparse these respective scenarios are in

design, combined with how the Caps-EM lacks an intrinsic reward signal to motivate exploration. Experiments showed that Caps-EM (IR) was able to converge to a policy in roughly 2.5E+7 training steps in MWH-M Sparse Uniform Texture and in 3.0E+7 steps in MWH-M Sparse Varied Texture with no curriculum training. Moreover, using Caps-EM (RGB + Depth) without curriculum training did not converge to an optimal policy function in the Sparse, Uniform or Sparse, Varied scenarios and did not improve performance over standard Caps-EM. However when using curriculum training in these same scenarios, the Caps-EM without intrinsic rewards performs exceptionally well.



(a) MWH-M Sparse Uniform Texture.

(b) MWH-M Sparse Varied Texture.

(c) MWH-M Dense Uniform Texture.

(d) MWH-M Dense Varied Texture.

Fig. 5.4. My Way Home Mirrored with Curriculum Training Results.

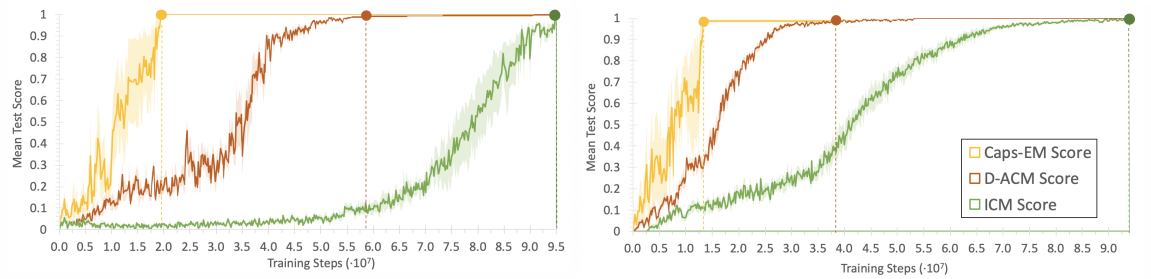
Figure 5.5 and Table 5.6 demonstrate each module's performance in MWH-G. None of the modules converge to a successful policy function in the MWH-G Sparse Uniform or Sparse Varied scenario variants, regardless of the use of curriculum training. The Caps-EM for example reached in excess of 12.6E+7 training steps without

learning a useful policy function. MWH-G results illustrate how the Caps-EM performs well in dense and high complexity scenario variants and significantly outperforms the ICM and D-ACM. Applying Caps-EM (RGB + Depth) to MWH-G Sparse Uniform and Sparse Varied did not result in improved performance to achieve an effective policy function in the scenarios.

Table 5.5.
My Way Home Mirrored Scenario Time Performance Results, with and without Curriculum (Curr.) Training.

Module Type	Scenario			
	Dense, Varied	Dense, Uniform	Sparse, Varied	Sparse, Uniform
ICM (s) w/ Curr. Training	1.49E+05	4.76E+05	1.70E+05	4.76E+05
ICM (s) w/o Curr. Training	2.45E+05	5.44E+05	2.04E+05	5.10E+05
% Difference	+39%	+13%	+17%	+7%
D-ACM (s) w/ Curr. Training	7.93E+04	1.94E+05	5.46E+05	2.20E+05
D-ACM (s) w/o Curr. Training	1.15E+05	2.64E+05	6.26E+05	7.49E+05
% Difference	+31%	+27%	+13%	+71%
Caps-EM (s) w/ Curr. Training	1.71E+03	6.21E+03	2.48E+03	1.24E+03
Caps-EM (s) w/o Curr. Traning	1.86E+03	8.07E+03	–	–
% Difference	+8%	+23%	–	–

Fig 5.6 presents the performance of each module in the My Way Home "Noisy TV" scenario, while Table 5.7 compares module performance in the MWH "Noisy TV" scenario and the MWH Dense, Texture scenario. A significant difference in module performance between the two scenarios is that the ICM, D-ACM and Caps-EM demonstrate policy overfitting in the training phase in the "noisy TV" scenario. The same behavior is not present with these modules in MWH Dense, Texture where the modules apply knowledge from training to perform well in the testing phase. While the ICM and D-ACM appear to converge quicker to an adequate policy function



(a) MWH-G Dense Uniform Texture.

(b) MWH-G Dense Varied Texture.

Fig. 5.5. My Way Home Giant Results.

Table 5.6.
My Way Home Giant Scenario Time Performance Results.

Module Type	Scenario	
	Dense, Varied	Dense, Uniform
ICM (s)	1.26E+06	1.29E+06
D-ACM (s)	6.70E+05	9.16E+05
% Difference	-88%	-41%
Caps-EM (s)	8.07E+04	1.18E+05
% Difference	-1457%	-994%

than in MWH Dense, Texture, early stopping is used with training in the "noisy TV" scenario for the ICM, D-ACM and Caps-EM. Additionally, all modules except the Caps-EM (RGB + Depth) do not converge to a policy function that successfully completes the scenario in 100% of test simulations.

The best performance that the ICM, D-ACM and Caps-EM achieve are with policy functions that are successful 98%, 95% and 84% of simulations, respectively. Given this, the ICM, D-ACM and Caps-EM do not converge to an optimal policy function that achieve a mean test score of 1 within testing, whereas the Caps-EM (RGB + Depth) does. Of all modules, the Caps-EM most significantly underperforms relative

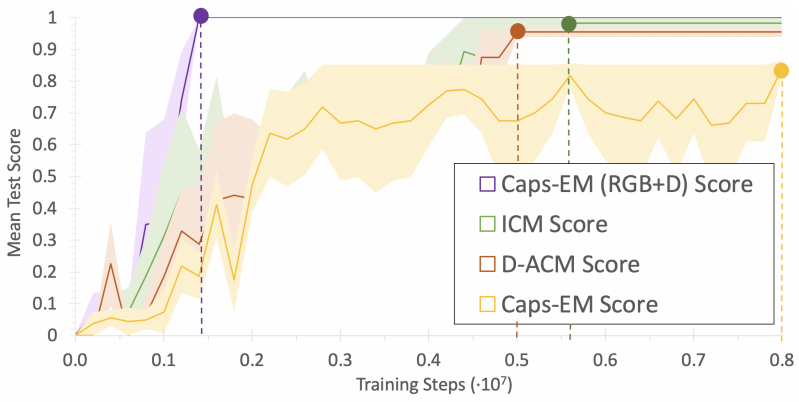


Fig. 5.6. My Way Home "Noisy TV" Results.

Table 5.7.
My Way Home "Noisy TV" Time Performance Results.

Module Type	Scenario		
	"Noisy TV"	Dense, Varied	% Difference
ICM (s)	7.61E+04	1.46E+05	-48%
D-ACM (s)	8.81E+04	9.69E+04	-9%
Caps-EM (s)	4.97E+04	1.24E+04	+301%
Caps-EM (RGB + Depth) (s)	9.32E+03	9.41E+03	-1%

to its performance in MWH Dense, Texture with being unable to converge to an adequate policy scoring a mean test score above 90%. While not truly a stochastic source of noise, the "noisy TV" wall still introduces a source of entropy to the scenario and significantly affects the modules' performances. However the Caps-EM (RGB + Depth) proves to be resilient to this type of stochastic variability and distraction while navigating through the scenarios.

5.4 Conclusions

The Caps-EM architecture leverages the A2C algorithm to perform well with autonomous navigation and exploration in sparse reward environments. More compact and efficient with 44% and 83% fewer parameters than the ICM and D-ACM, respectively, the Caps-EM on average outperforms both the ICM and D-ACM across the MWH, MWH-M and MWH-G scenarios. The Caps-EM converges to a policy function in MWH, on average across all four scenario variants, 437% and 1141% quicker than the D-ACM and ICM, respectively, without the use of intrinsic rewards. Similarly in MWH-M scenarios when using curriculum training, the Caps-EM has a 10,726% and 13,317% time improvement on average over the D-ACM and ICM, respectively. Lastly with MWH-G variants, the Caps-EM has a 703% and 1226% time improvement on average over the D-ACM and ICM, respectively.

Experiments with varied inputs demonstrate how a combination of RGB and depth images enable improved performance across all MWH scenarios. Pairing CapsNets with the concatenated input results in improved performance in each MWH scenario. Relative to Caps-EM with only RGB inputs, Caps-EM (RGB + Depth) achieves a 377% improvement in policy convergence in the MWH Dense Varied scenario, 2785% improvement in Dense Uniform, 25% improvement in Sparse Varied, and 47% improvement in Sparse Uniform. Furthermore, this variation of the Caps-EM proves resilient to stochastic noise as demonstrated with the MWH "Noisy TV" scenario, in part by avoiding the use of intrinsic rewards. Conversely, the ICM and D-ACM suffer from overfitting and decreased performance in this scenario as hypothesized and shown in prior research on intrinsically motivated agents.

While the Caps-EM struggles to converge effectively in certain sparse scenarios, such as with MWH-M Sparse Uniform Texture and Sparse Varied Texture, the module readily applies learned knowledge using curriculum training to generalize well across these scenarios. The intrinsic reward factor used by the D-ACM and ICM likely enables these modules to better handle these specific sparse reward scenarios.

However simply combining intrinsic rewards with the Caps-EM does not significantly improve performance in these scenarios as shown. The ICM and D-ACM do exhibit larger architectures with more network parameters and lower relative performance in other scenarios though. Caps-EM offers a more lightweight yet still capable design overall.

The results additionally confirm the hypothesis of Caps-EM's ability to maintain better spatial relationships and hierarchies than CNNs for improved performance on average. This finding is evident in dense scenarios where the Caps-EM maintains relationships between the Varied Texture rooms well. Future work could explore how to improve the Caps-EM performance in the extremely sparse environments, such as MWH-M Sparse Uniform Texture, to address this weakness. Experimentation with Caps-EM variants that did incorporate intrinsic reward factors did improve effectiveness in these edge cases to a degree. However, this module variant does not appear to be a viable solution as its performance on average in other scenarios is significantly worse. Another area of interest is how modules would perform with added dynamics and moving objects beyond only the stochastic noise of only the "noisy TV" wall.

6. CONCLUSIONS AND FUTURE WORK

6.1 Future Work

A key factor to achieving proficient autonomous exploration is ensuring that an agent has an adequate understanding of its surrounding environment. This research explores how different sets of data, such as RGB images, depth images and optical flow data, may be leveraged to obtain improved network embeddings for quicker convergence to an optimal policy function. Considering which modalities may provide richer information of an environment as an agent moves, 3D structure and mapping appear to be advantageous to achieve improved network embeddings. Point clouds have been increasingly used in computer vision, robotics and autonomous driving for the advantages of 3D data for resolving 2D problems. 3D information can convey superior geometric, shape and scale data [62, 63]. 3D data is usually represented as a point cloud which "preserves the original geometric information in 3D space without discretization" [64]. For example while a depth image only provides information from a particular point of view, a 3D point cloud can capture every point in a scene in 3D.

Software for AI research, such as Airsim [65], allows for utilization of 3D point cloud information through the projection of 2D image points to the 3D world plane using only RGB and depth images. Future work could explore pairing 3D point cloud data with either an intrinsic reward-based network or capsule network model to analyze the performance relative to the findings shown in this research and other state-of-the-art work. While 3D point clouds are inherently unstructured and without order, standard CNNs cannot not be directly applied to point clouds. Thus approaches such as PointNet and PointNet++ could be utilized to learn local features from 3D data [66, 67]. One potential method could be to incorporate self-supervised 3D prediction of a scene using only RGB and depth images as inputs. Using CNN

encoders to generate embeddings of the input scenes, a PointNet-based decoder may be used to output the corresponding point cloud label of the same scene to use for training the predictive model.

6.2 Conclusions

This research work highlights how autonomous exploration capabilities may be improved through the incorporation of depth image prediction in junction with intrinsic reward factors. In practical applications, this methodology is accessible and does not require hardware or equipment beyond a standard RGB-D camera to obtain the ground truth depth images. Furthermore as shown, the D-ACM is a relatively compact module, not having a significantly larger number of trainable parameters than the ICM. This allows for the D-ACM policy function to be trained fairly quickly and deployed using minimal hardware. This process is also shown to be expedited through the incorporation of curriculum training.

Through novel experiments with the application of CapsNets, the proposed Caps-EM module presents a further compact module than the D-ACM. Similarly the Caps-EM does not require complex hardware systems, merely a RGB-D camera as well. In line with other state of the art work applying CapsNets, the experiments here demonstrate how incorporating CapsNets enabled use of fewer trainable parameters than the other comparable CNN-based approaches and subsequently resulted in more efficient module training time. With the minor modification of providing a combined RGB and depth input to the Caps-EM, the Caps-EM (RGB + Depth) notably improves Caps-EM performance across several scenarios. This module furthermore addresses shortcomings with navigating through scenarios with high entropy like MWH "Noisy TV." With the significant improvements in autonomous exploration performance shown with Caps-EM, this research highlights how CapsNets maintain superior representations and relationships of an environment and objects than other approaches, such as the D-ACM and ICM.

As previously mentioned, future work with the ACMs and Caps-EM could explore performance given added dynamic and moving elements in an environment independent of the agent. Additionally while having performed successfully in ViZDoom, these modules could be deployed in more advanced simulators, such as Airsim, to further evaluate abilities with handling and navigating through higher complexity environments. While the modules controlled an agent with a limited action space in ViZDoom, performance may be impacted when having to operate an agent with a larger action space, such as that associated with a UAV. Ultimately the methods proposed with this research explore novel avenues within deep reinforcement learning and present ideas progressing autonomous systems' capabilities.

REFERENCES

REFERENCES

[1] H. X. Pham, H. M. La, D. Feil-Seifer, and M. Deans, "A distributed control framework for a team of unmanned aerial vehicles for dynamic wildfire tracking," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 6648–6653.

[2] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka, "Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue," *IEEE Robotics Automation Magazine*, vol. 19, no. 3, pp. 46–56, Sep. 2012.

[3] A. C. Woods and H. M. La, "Dynamic target tracking and obstacle avoidance using a drone," in *Advances in Visual Computing*, G. Bebis, R. Boyle, B. Parvin, D. Koracin, I. Pavlidis, R. Feris, T. McGraw, M. Elendt, R. Kopper, E. Ragan, Z. Ye, and G. Weber, Eds. Cham: Springer International Publishing, 2015, pp. 857–866.

[4] G. Nützi, S. Weiss, D. Scaramuzza, and R. Siegwart, "Fusion of imu and vision for absolute scale estimation in monocular slam," *Journal of Intelligent & Robotic Systems*, vol. 61, no. 1, pp. 287–299, Jan 2011. [Online]. Available: https://doi.org/10.1007/s10846-010-9490-z

[5] H. M. La, "Multi-robot swarm for cooperative scalar field mapping," 2016.

[6] H. M. La, W. Sheng, and J. Chen, "Cooperative and active sensing in mobile sensor networks for scalar field mapping," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 1, pp. 1–12, Jan 2015.

[7] S. Ramasamy, R. Sabatini, A. Gardi, and J. Liu, "Lidar obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid," *Aerospace Science and Technology*, vol. 55, pp. 344 – 358, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1270963816301900

[8] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, *Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera*. Cham: Springer International Publishing, 2017, pp. 235–252.

[9] K. McGuire, G. de Croon, C. De Wagter, K. Tuyls, and H. Kappen, "Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1070–1076, April 2017.

[10] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, "Vizdoom: A doom-based AI research platform for visual reinforcement learning," *CoRR*, vol. abs/1605.02097, 2016.

[11] C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing, 2018. [Online]. Available: https://books.google.com/books?id=achqDwAAQBAJ

[12] V. G. Maltarollo, K. M. Honorio, and A. B. F. da Silva, "Applications of artificial neural networks in chemical problems," in *Artificial Neural Networks*, K. Suzuki, Ed. Rijeka: IntechOpen, 2013, ch. 10. [Online]. Available: https://doi.org/10.5772/51275

[13] F. Bre, J. Gimenez, and V. Fachinotti, "Prediction of wind pressure coefficients on building surfaces using artificial neural networks," *Energy and Buildings*, vol. 158, 11 2017.

[14] D. Lee, H. Seo, and M. W. Jung, "Neural basis of reinforcement learning and decision making," *Annual Review of Neuroscience*, vol. 35, no. 1, pp. 287–308, 2012, pMID: 22462543. [Online]. Available: https://doi.org/10.1146/annurev-neuro-062111-150512

[15] J. X. Wang, Z. Kurth-Nelson, D. Kumaran, D. Tirumala, H. Soyer, J. Z. Leibo, D. Hassabis, and M. Botvinick, "Prefrontal cortex as a meta-reinforcement learning system," *bioRxiv*, 2018. [Online]. Available: https://www.biorxiv.org/content/early/2018/04/13/295964

[16] R. Sutton and A. Barto, *Reinforcement learning: An introduction*. Cambridge Univ Press, 1998, vol. 116.

[17] A. Sarkar, "A brandom-ian view of reinforcement learning towards strong-ai," *CoRR*, vol. abs/1803.02912, 2018. [Online]. Available: http://arxiv.org/abs/1803.02912

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016.

[19] Y. Wu, E. Mansimov, S. Liao, R. B. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," *CoRR*, vol. abs/1708.05144, 2017.

[20] M. Grześ, "Reward shaping in episodic reinforcement learning," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '17. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 565–573.

[21] A. Barto and O. Imek, "Intrinsic motivation for reinforcement learning systems," *Intrinsically Motivated Learning in Natural and Artificial Systems*, vol. 4, 01 2005.

[22] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," *CoRR*, vol. abs/1705.05363, 2017. [Online]. Available: http://arxiv.org/abs/1705.05363

[23] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *CoRR*, vol. abs/1710.09829, 2017. [Online]. Available: http://arxiv.org/abs/1710.09829

[24] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *Proceedings of the 21th International Conference on Artificial Neural Networks - Volume Part I*, ser. ICANN'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 44–51.

[25] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, pp. 1150–1157 vol.2.

[26] S. Bhatti, A. Desmaison, O. Miksik, N. Nardelli, N. Siddharth, and P. H. S. Torr, "Playing doom with slam-augmented deep reinforcement learning," *CoRR*, vol. abs/1612.00380, 2016. [Online]. Available: http://arxiv.org/abs/1612.00380

[27] R. Kang, J. Shi, X. Li, Y. Liu, and X. Liu, "Df-slam: A deep-learning enhanced visual slam system based on deep local features," *ArXiv*, vol. abs/1901.07223, 2019.

[28] S. Wen, Y. Zhao, X. Yuan, Z. Wang, D. Zhang, and L. Manfredi, "Path planning for active slam based on deep reinforcement learning under unknown environments," *Intelligent Service Robotics*, 01 2020.

[29] A. Khan, C. Zhang, N. Atanasov, K. Karydis, V. Kumar, and D. D. Lee, "Memory augmented control networks," *CoRR*, vol. abs/1709.05706, 2017. [Online]. Available: http://arxiv.org/abs/1709.05706

[30] E. Parisotto and R. Salakhutdinov, "Neural map: Structured memory for deep reinforcement learning," *CoRR*, vol. abs/1702.08360, 2017. [Online]. Available: http://arxiv.org/abs/1702.08360

[31] J. Zhang, L. Tai, J. Boedecker, W. Burgard, and M. Liu, "Neural SLAM," *CoRR*, vol. abs/1706.09520, 2017. [Online]. Available: http://arxiv.org/abs/1706.09520

[32] C. Wang, J. Wang, X. Zhang, and X. Zhang, "Autonomous navigation of uav in large-scale unknown complex environment with deep reinforcement learning," in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2017, pp. 858–862.

[33] A. Patrik, G. Utama, A. Gunawan, A. Chowanda, J. Suroso, R. Shofiyanti, and W. Budiharto, "Gnss-based navigation systems of autonomous drone for delivering items," *Journal of Big Data*, vol. 6, 12 2019.

[34] M. Kan, S. Okamoto, and J. H. Lee, "Development of drone capable of autonomous flight using gps."

[35] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel, "Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks," *CoRR*, vol. abs/1605.09674, 2016. [Online]. Available: http://arxiv.org/abs/1605.09674

[36] S. Mohamed and D. J. Rezende, "Variational information maximisation for intrinsically motivated reinforcement learning," 2015.

[37] H. Kim, J. Kim, Y. Jeong, S. Levine, and H. O. Song, "EMI: exploration with mutual information maximizing state and action embeddings," *CoRR*, vol. abs/1810.01176, 2018. [Online]. Available: http://arxiv.org/abs/1810.01176

[38] Y. Burda, H. Edwards, D. Pathak, A. J. Storkey, T. Darrell, and A. A. Efros, "Large-scale study of curiosity-driven learning," *CoRR*, vol. abs/1808.04355, 2018. [Online]. Available: http://arxiv.org/abs/1808.04355

[39] Y. Burda, H. Edwards, A. J. Storkey, and O. Klimov, "Exploration by random network distillation," *CoRR*, vol. abs/1810.12894, 2018. [Online]. Available: http://arxiv.org/abs/1810.12894

[40] M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto, "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4423–4430, 2018.

[41] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *CoRR*, vol. abs/1710.09829, 2017. [Online]. Available: http://arxiv.org/abs/1710.09829

[42] E. Xi, S. Bing, and Y. Jin, "Capsule Network Performance on Complex Data," *arXiv e-prints*, p. arXiv:1712.03480, Dec 2017.

[43] P. Andersen, "Deep reinforcement learning using capsules in advanced game environments," *CoRR*, vol. abs/1801.09597, 2018. [Online]. Available: http://arxiv.org/abs/1801.09597

[44] L. Annabi and M. G. Ortiz, "State representation learning with recurrent capsule networks," *CoRR*, vol. abs/1812.11202, 2018. [Online]. Available: http://arxiv.org/abs/1812.11202

[45] F. Deng, S. Pu, X. Chen, Y. Shi, T. Yuan, and S. Pu, "Hyperspectral image classification with capsule network using limited training samples," *Sensors*, vol. 18, no. 9, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/9/3153

[46] M. Pöpperl, R. Gulagundi, S. K. Yogamani, and S. Milz, "Capsule neural network based height classification using low-cost automotive ultrasonic sensors," *CoRR*, vol. abs/1902.09839, 2019. [Online]. Available: http://arxiv.org/abs/1902.09839

[47] A. D. Kumar, "Novel deep learning model for traffic sign detection using capsule networks," *CoRR*, vol. abs/1805.04424, 2018. [Online]. Available: http://arxiv.org/abs/1805.04424

[48] S. Prakash and G. Gu, "Simultaneous localization and mapping with depth prediction using capsule networks for uavs," *CoRR*, vol. abs/1808.05336, 2018. [Online]. Available: http://arxiv.org/abs/1808.05336

[49] N. H. Phong and B. Ribeiro, "Advanced capsule networks via context awareness," *CoRR*, vol. abs/1903.07497, 2019. [Online]. Available: http://arxiv.org/abs/1903.07497

[50] A. Shahroudnejad, A. Mohammadi, and K. N. Plataniotis, "Improved explainability of capsule networks: Relevance path by agreement," *CoRR*, vol. abs/1802.10204, 2018. [Online]. Available: http://arxiv.org/abs/1802.10204

[51] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, "Vizdoom: A doom-based AI research platform for visual reinforcement learning," *CoRR*, vol. abs/1605.02097, 2016.

[52] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016.

[53] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *ICML*, 2017.

[54] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: ACM, 2009, pp. 41–48.

[55] L. Tai and M. Liu, "Towards cognitive exploration through deep reinforcement learning for mobile robots," *CoRR*, vol. abs/1610.01733, 2016.

[56] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Image Analysis*, J. Bigun and T. Gustavsson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 363–370.

[57] Y. He and S. Chen, "Advances in sensing and processing methods for three-dimensional robot vision," *International Journal of Advanced Robotic Systems*, vol. 15, no. 2, p. 1729881418760623, 2018.

[58] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, no. 4, pp. 333–349, Oct 1997.

[59] D. Pedamonti, "Comparison of non-linear activation functions for deep neural networks on MNIST classification task," *CoRR*, vol. abs/1804.02763, 2018. [Online]. Available: http://arxiv.org/abs/1804.02763

[60] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *CoRR*, vol. abs/1206.5533, 2012. [Online]. Available: http://arxiv.org/abs/1206.5533

[61] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[62] Y. Guo, F. Sohel, M. Bennamoun, M. Lu, and J. Wan, "Rotational projection statistics for 3d local surface description and object recognition," *International Journal of Computer Vision*, vol. 105, no. 1, p. 63–86, Apr 2013. [Online]. Available: http://dx.doi.org/10.1007/s11263-013-0627-y

[63] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, and J. Wan, "3d object recognition in cluttered scenes with local surface features: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2270–2287, Nov 2014.

[64] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," 2019.

[65] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065

[66] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *CoRR*, vol. abs/1612.00593, 2016. [Online]. Available: http://arxiv.org/abs/1612.00593

[67] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *CoRR*, vol. abs/1706.02413, 2017. [Online]. Available: http://arxiv.org/abs/1706.02413