Channel Islands

CALIFORNIA STATE UNIVERSITY

An Ensemble Approach to Robotic Exploration With Deep Reinforcement Learning

A Thesis Presented to

The Faculty of the Computer Science Department

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

by

Student Name:
Gabriel Orellana

Advisor:
Jason Isaacs

May 2021

APPROVED FOR MS IN COMPUTER SCIENCE

[illegible] May 26, 2021

Advisor: Jason Isaacs Date

[illegible] May, 26, 2021

Brian Thoms Date

[illegible] May 26, 2021

Bahareh Abbasi Date

APPROVED FOR THE UNIVERSITY

[illegible]
Jill Leafstedt (May 28, 2021 12:34 PDT) 5/28/2021

Interim Dean Dr.Jill Leafstedt Date

An Ensemble Approach to Robotic Exploration With Deep Reinforcement Learning

Gabriel Orellana

May 21, 2021

Abstract

Artificial intelligence is one of the fastest growing areas of development today. Its power is being applied to a growing quantity of areas, by an increasing number of companies. Deep reinforcement learning (DRL) has played a large role in the advances made in these areas. One of the areas where DRL is being applied is that of autonomous robotics. From self-driving cars to robotic assistants, DRL has been shown to be effective for training autonomous actors to explore new areas. However, the majority of approaches use a single model for the exploration process. This can lead to reduced performance in environments in which the weaknesses of the chosen model are amplified. This paper introduces an ensemble approach to the application of DRL in autonomous robotic exploration, in an attempt to improve performance. Instead of a single model, I utilize three independently trained actor-critic models working in tandem, in an attempt to mitigate the weaknesses an individual model can have. Working with a 2D simulation environment, the individual component models of the ensemble, consisting of stable baselines implementations of A2C, PPO2, and ACKTR, were trained in a variety of simulated environments. The three ensemble implementations are then evaluated against these individual model components in a series of test environments. The results show that ensemble methods can produce better results, depending on implementation, and serve to reduce the downsides of individual model performances without sacrificing overall performance.

Contents

List of Figures

1 Introduction

1.1 Motivation

The rise of Artificial Intelligence (AI) and machine learning has been swift and impactful. The technology has grown rapidly and expanded its use to a multitude of diverse areas which affect the everyday life of people worldwide. These impacts are hard to avoid and have drawn my interest in a variety of their implementations.

In gaming, AI was initially unable to compare with human. It has now been developed to the point where a machine can not only defeat the top chess masters, but also defeat the top human players at the game of GO [1], which was originally considered so complex that some believed computers would never reach the point of mastery [2].

In healthcare, AI has begun to make its impact felt, with its ability to review vast quantities of medical data rapidly and provide analysis and insights to medical professionals. This enables increased accuracy in diagnoses and increased effectiveness in treatments, not to mention the impacts in reduction of costs. Additionally AI has shown effectiveness in predicting health issues and possible adverse reactions to treatments and medication [3]. AI has been used in Dermatology, Radiology, Screenings, Psychiatry, and Disease Diagnosis, among other fields, with uses such as classification programs which can identify cancerous tumors in patients.

AI has also made its impacts felt in the finance industry, where it has been used in a variety of different ways. Companies now use machine learning to evaluate credit applications, underwrite loans, manage risk, analyze market trends, identify patterns, forecast changes, and in some cases, trade stocks [4]. The financial impacts of these implementations are shifting the way markets and business operate. AI assistants are now available for consumers to consult when making personal finance decisions. These utilize machine learning to provide advice, as well as natural language process to provide real time automated customer service support. AI can be used to help people save money by identifying spending habits and suggesting changes to improve savings rates. It has also been used to improve cybersecurity, and prevent fraud.

One of the largest areas in which AI has had a large impact is in the area of robotics. Robotics spans many of the aforementioned categories and has widespread usage. In healthcare, AI enabled robots are being used to sanitize, deliver supplies, provide care and in some cases, even perform surgery[5]. AI robots are now being used in agriculture to help with planting, maintenance, and the harvesting of crops. AI classifiers can determine when to water, how to identify and eliminate weeds, apply pesticides to kill insect infestations, and pick crops all based on machine learning [6]. High profile uses of AI robotics such as advances towards fully self-driving automobiles are some of the most talked about developments. Tesla and other manufacturers are moving rapidly towards cars which require no human interaction in order to operate not only successfully, but with a higher safety than human operated vehicles.

Although there have been large advances in these areas, there is still plenty of room for improvement. Efforts are underway to improve robotic operation, particularly in unknown environments. The DARPA SubT Challenge [7] is one of these efforts. Its goal is to promote the development of new techniques which can quickly navigate through complex, underground environments such as the one shown in Figure 1, mapping the areas and identifying artifacts in them.

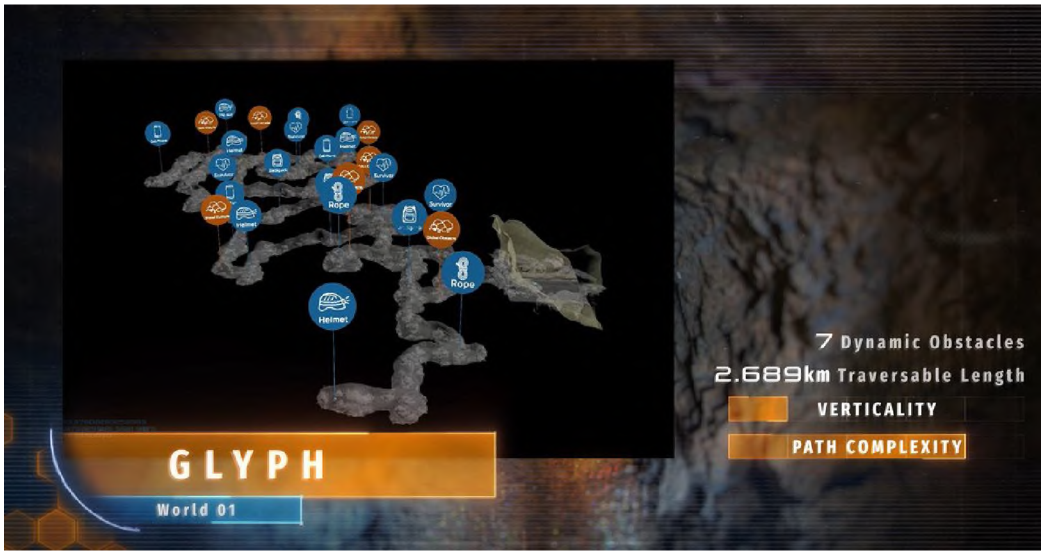

Figure 1: SubT Environment Example [7]

The resulting advancements would ideally be applied to many areas such as search and rescue, and disaster recovery and relief in urban and underground settings. The use of robotics is key as many of these environments may be inaccessible or inhospitable for human relief workers due to environmental hazards, but would not prohibit robotic operation. One of the key elements in this is the mapping of these unknown environments. Deep Reinforcement Learning has been shown to be effective [8] when applied to this domain and is thus the focus of this paper.

One of the approaches to Deep Reinforcement Learning which has been shown to be particularly effective is an ensemble approach. Ensemble methods in general have been shown to yield improved results in comparison to single models. This can be attributed to the built-in biases which can be found in individual models due to the way in which they operate. The use of multiple models reduces the impact of the individual biases.

1.2 Contribution

The goal of this paper is to show whether the application of an ensemble approach to the use of Deep Reinforcement Learning models in robotic unknown area exploration produces better results than the use of a single DRL model.

The results of this research show that ensemble methods can improve performance when applied to robotic exploration of unknown areas over some models, and can help to mitigate the negative impact of individual model weaknesses. I show that the ensemble methods, depending on implementation, can outperform the majority of their component models, and in some environments, outperform all the component models.

1.3 Outline

In order to show this I will first go through a brief overview of the backgrounds for the different fields of study related to the endeavour to provide the reader the context of the discussion, the terms that will be used, and a cursory understanding of the techniques which will be applied. Having done this, I will then provide an explanation of the methodology used. This includes an overview of each of the DRL models used, an introduction to the training and evaluation environment, an explanation of the training process, and a layout of the evaluation method utilized. The results of the study will then be presented along with a discussion of their implications, and a presentation of the limitations, and possible future work.

2 Background

To properly set the stage for this topic, it is worthwhile to examine the various fields of study relating to the topic at hand so that the reader may be familiarized with these foundations and better understand the process used in this research. Those areas bearing primary relevance are Machine Learning, Reinforcement Learning (RL), Deep Learning, and, Deep Reinforcement Learning. In addition to this, an examination of the use of ensemble methods in reinforcement learning is also provided.

2.1 Machine Learning

L.G. Valiant, who wrote one of the early seminal papers on the topic, described Machine Learning with the following [9]: "a program for performing a task has been acquired by learning if it has been acquired by any means other than explicit programming." Machine learning deals with the creation of programs which have the ability to learn and improve their capabilities as their experience/training grows. Machine learning provides the capability to analyze vast quantities of data. Murphy defines machine learning as [10] "a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or perform other kinds of decision making under uncertainty." Tracing its origins back to 1959, when an IBM engineer named Arthur Samuel coined the term [11], Machine Learning has grown in leaps and bounds over the last 70 years. Some of the highest profile jumps made via machine learning were with regards to teaching machines to play games, and several of these lead to the development of key algorithms. In the 1950s Samuel developed a program which could play checkers [12] and the program he created eventually developed into the now famous min-max algorithm. Checkers was picked due to the simplicity of the rules which Samuel noted [12], "permits greater emphasis to be placed on learning techniques."

Within Machine Learning, there are three subcategories, differentiated in the manner in which the learning is achieved. These are Supervised, Unsupervised, and Reinforcement Learning. Supervised (or predictive) learning models assume there is a supervisor who divides the training materials into classes and then the program is trained by evaluating the features presented and their correlation to the provided classes. As laid out by Murphy [10], “the goal is to learn a mapping from inputs x to outputs y, given a labeled set of input-output pairs $\mathcal{D} = (x_i, y_i)_{i=1}^N$ where $\mathcal{D}$ is called the training set, and N is the number of training examples [10].

In unsupervised learning, the program is not provided the classes, but rather it attempts to define the classes and relevant features by finding those features which have a higher data variance and provide separability [13]. In this form of learning, using the only thing provided is the inputs, $\mathcal{D} = (x_i)_{i=1}^N$, and it is up to the model to extract information deemed "useful" from the data [10].

2.2 Reinforcement Learning

Reinforcement learning is distinguished from the two aforementioned machine learning methods in that it emphasizes learning by the program from “direct interaction with its environment, without relying on exemplary supervision or complete models of the environment [14].” Reinforcement learning, in its essence, is the presentation of a situation in which the program is given an environment in which it employs trial and error, in order to solve the given problem. A state with possible actions is presented, and the program is given a reward or penalty for whatever action it takes. This reward policy, as it is referred to, is developed by the creator of the program, and is the limit of the instructions given to the program. The program is given the policy and instructions to maximize the final reward. Based off the state input and the current state of the agent, an action is taken, typically randomly selected from the available actions. As a result of this action, the environment and state of the agent are updated, and a reward value is given. The program then explores the environment and by trial-and-error method and attempts to learn the best method for maximizing reward [15]. Additionally, in some cases the performance of the agent may be measured against the optimal performance. The difference between the two is referred to as regret. This measure gives perspective on the long-term effects of individual actions and allows the incorporation of actions which may initially have negative effects, but over the long term maximize value. Reinforcement learning differs from supervised and unsupervised learning in that it has a trade-off between exploration and exploitation [16]. It is essential for the reinforcement learning agent to properly balance these two in order to obtain optimal results. Sutton describes this well:

> To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to exploit what it has already experienced in order to obtain reward, but it also has to explore in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favor those that appear to be best [16].

Finding this balance can be extremely difficult and poses one of the major obstacles to success in this kind of machine learning. To add to the problem, in stochastic environments, the agent must try each action a multitude of times in order to properly determine the associated reward. This difficulty is encountered in most reinforcement learning endeavours, the author's included.

2.2.1 Reinforcement Learning Components

There are four primary components of reinforcement learning, the policy, the reward, the value function, and the environment. In order to better understand each of these, it is worthwhile to examine the Markov Decision Process first.

2.2.2 Markov Decision Process

The Markov Property, named after Russian mathematician Andrey Markov, is used in reference to a stochastic process which has a memory-less property to it. A process which is stochastic has the Markov property if, given the present state, the future states of the process do not depend on the past, but only on the present state. The probability of the succeeding state S_{t+1} depends only on the current state S_t and not on prior states $S_1, S_2, ..., S_{t-1}$ [17]. The Markov property is important in reinforcement learning because in order for actions and values in reinforcement learning to be effective and informative, the individual state representations must themselves be informative. Bringing in the concepts of reward, action, and discount, a Markov Decision Process (MDP) can be defined as follows [17]:

- $\mathcal{S}$ is a finite set of *states.*
- $\mathcal{A}$ is a finite set of *actions.*
- $\mathcal{P}$ is the *state transition probability matrix,*
  $\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$ where s is a state and a is an action
- $\gamma \in [0, 1]$ is called the *discount factor.*
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a *reward function*

Since each learning episode in reinforcement learning is able to be depicted as a sequence of states, actions, and rewards, and each state depends only upon the prior states and actions taken, and the actor is ignorant of which state comes next, we can say that the process satisfies the Markov Property. Thus we can describe the process used in reinforcement learning as a Markov Decision Process.

2.2.3 Policy

Informally, the policy in reinforcement learning is the set of rules which the agent follows to go from the state data to an action choice [16]. More formally, the mathematical definition of a reinforcement learning policy is defined in terms of the Markov Decision Process. With the above definition of the Markov Decision Process as a tuple of the form $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, \mathcal{R})$, we can define a policy as follows[16]:

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$$

The aim of reinforcement learning is to discover the best policy, that is, the policy which maximizes the agents reward. This policy can be anything from a search process, to a lookup table or process requiring complex calculations [16].

2.2.4 Reward

The reward in reinforcement learning is a value produced by the environment as a result of the action of an agent. As previously stated, the goal of the agent is to maximize the reward, and the goal as a whole is to develop a policy which results in a maximal goal. Rewards in reinforcement learning are much like the rewards used in behavioural development. For example, when training a dog to perform a trick such as “roll over", when the dog performs the correct action (rolling over) you give the dog a treat (reward). If the dog performs an incorrect action, the reward can be either nothing, or something like a verbal admonishment. Whatever the reward is, the purpose is to indicate to the agent (dog in this case) that the correct (or incorrect) action was taken. The reward provides information used to modify the policy. If the action results in a poor reward, the policy should (often) be updated to prefer an action which produces a better reward for the same state, and vice versa. Ideally, the training continues with the dog learning that it receives maximal reward when it performs the correct action immediately, until the task has been mastered i.e. the agent has been trained and has developed the optimal policy.

2.2.5 Value Function

As opposed to the reward, the value function is a determinant of long-term success. The value function is a predictor of the future rewards possible given the current state [16]. As Sutton states [16], “rewards determine the immediate, intrinsic desirability of environmental states, values indicate the long-term desirability of states after taking into account the states that are likely to follow and the rewards available in those states." This is analogous to delayed gratification in human behavior. Eating the entire cake might generate an immediate reward of the enjoyment the act of eating provides, however, knowledge of the long term ramifications of this action (the value function) tells us that there will be negative effects of this that outweigh this short term reward (weight gain, upset stomach, etc.).

$$Q(s_t,a_t) \leftarrow (1-\alpha)\cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Figure 2: Bellman's Equation [8]

As shown in Figure 2, we can write the value function mathematically, where a is the action, and s is the state, as $(Q(s,a))$. The value function is also a function of the environment, like the reward. As Surmann notes, "it appropriately rewards each decision and future states - that is the transition - to evaluate the V(s) and Q(s,a) according to the Bellman Equation [8]."

In a Markov Decision Process with n states, there are a corresponding number of linear equations for the n unknown value functions. Using dynamic programming, these equations can be solved to generate the optimal value function, that is, the value function which dictates the best obtainable performance in the MDP [17].

2.2.6 Environment

The Environment is the final component of reinforcement learning. When referring to the environment, we typically mean a model of the environment in which the agent is ultimately expected to operate. It may not reflect the actual environment in all its details, but rather, it provides learning cues which allow the agent to learn how to behave in the actual environment. A good analogy for this would be a motorcycle training course.

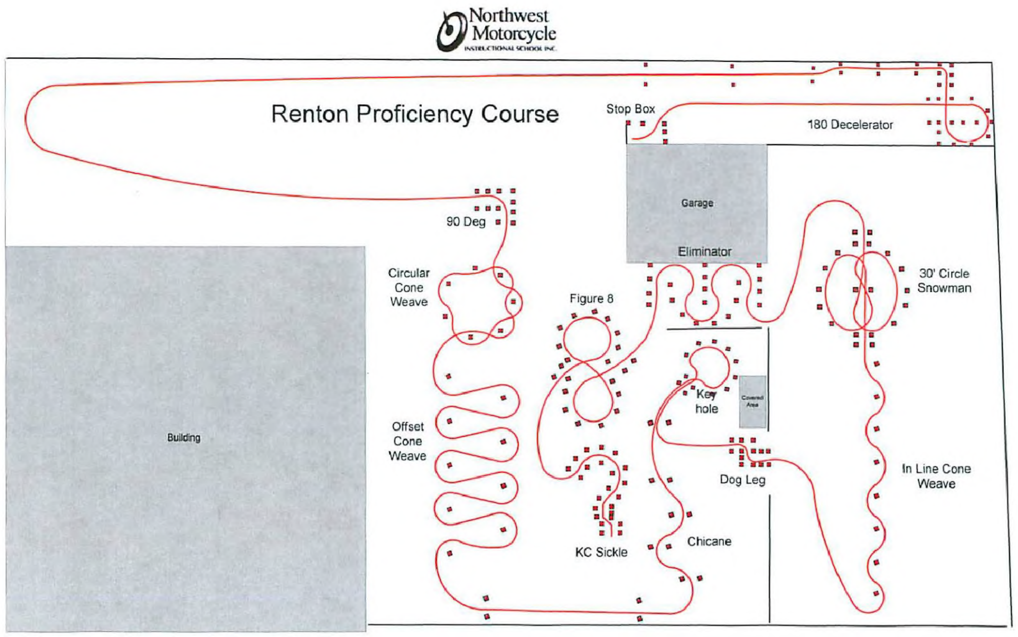

Figure 3: Motorcycle Training Course Layout [18]

The course does not reflect the actual environment a motorcycle rider would likely encounter in the real world. Few or no roads follow a complex pattern like that in Figure 3. However, this model of the environment enables the rider to develop the skills they would need to handle the array of diverse obstacles and events which they may possibly encounter throughout their riding lifetime. So to with reinforcement learning environments, the environment should present to the agent those things which will allow it to make the necessary inferences about the effects of its actions and the resulting states [16]. The MDP is the mathematical framework which describes an environment. It provides the current state of the agent, including all the inputs this entails, as well as the reward value that results from the agent's actions.

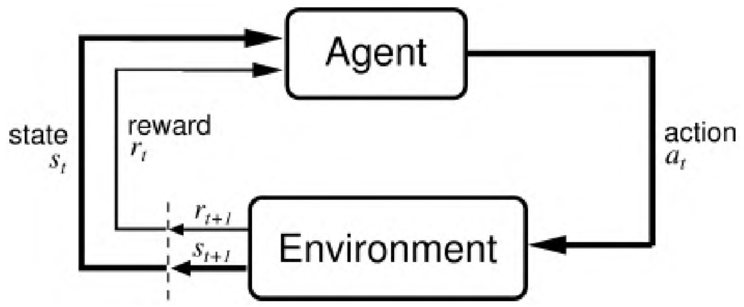

Figure 4: Agent-Environment Interaction [19]

Figure 4 shows the interaction between the Agent and the Environment. In order for reinforcement learning to be effective, the training environment must model the actual environment in such a way that the agent can transition to the actual environment for validation and perform well.

2.3 Deep Learning

As Machine Learning developed, more complex problems were targeted. The initial implementations of Machine Learning algorithms were too inefficient to feasibly tackle these. The method used by Samuel, for example, required mapping values to all possible positions of pieces on the checkerboard. However, for games such as chess or GO, the number of positions is exponentially higher than in checkers. In general, conventional techniques lacked the ability to process raw data [20]. As time progressed, the advances in computational technologies provided new avenues for tackling these more advanced problems. This is where Deep Learning (DL) techniques began being developed. Deep learning is a form of machine learning which utilizes multiple layers to extract higher level feature information [21]. It has been shown to be particularly effective when implemented with artificial neural networks. The learning process used with DL are primarily supervised, while effective unsupervised deep learning is still a goal of the community. Deep learning has increased in effectiveness as processing power has grown, allowing the utilization of increasing numbers of layers and feature input [22]. Deep learning addresses the problem of how to deal with processing enormous quantities of data when using learning algorithms.

Deep Learning traces its origins back to 1943, when Walter Pitts and Warren McCullock created a computer model of the neural networks of the human brain. They achieved this by mimicking the thought process of humans using a combination of mathematics and algorithms they dubbed "threshold logic."[9] Following this development, in the 1960 Henry J. Kelley developed what is considered the basics of a continuous Back Propagation Model which was subsequently simplified by Stuart Dreyfus in 1962. The significance of this discovery and its utility would not be realized until 1985 [9]. In 1970, Kunihiko Fukushima was working on designing a set of neural networks with multiple pooling and convolutional layers. In 1979, he produced an artificial neural network, named Neocognitron. This utilized a hierarchical, multilayered design which enabled the network to learn to identify visual patterns [9]. This was one of the first deep learning models.

2.3.1 Back Propagation

Rumelhart et al. in their 1986 paper, brought attention to the effective use of back-propagation in deep learning [23]. Back-propagation, with regards to deep learning, deals with the transmission of information, informing the neural network as to whether it has predicted accurately [24]. In deep learning, the neural network has a set of parameters, processes input data, and then makes a guess about that data using these parameters. This guess is then evaluated with a loss function. With back-propagation, the error determined by the loss function is propagated, in reverse, through the network, allowing it to alter the parameters and adjust based on the error [24]. This is akin to the process used by mortar teams. They make their initial aim based off a best guess, just as the neural network makes its initial guess. Once the shell lands, they see how far off their shot was, and adjust their aim accordingly. This parallels the back-propagation where the network adjusts its parameters. The team then fires again and refines until the target is hit, just as the network adjusts until the error is minimized or eliminated. The use of back-propagation was key in the rise of deep learning in its effectiveness and widespread use.

Recently DL has been used to process the vast quantities of data being produced everyday. Companies like Amazon, Google, Microsoft, and Facebook in particular have been using DL to process the data they harvest from users each day [25]. This data is processed by DL algorithms and the results are analyzed to extract data about users, trends, and economics [26]. As more and more data is produced, the DL usages will increase as well.

2.4 Deep Reinforcement Learning

Deep Reinforcement Learning combines the approaches of both deep learning and reinforcement learning. This combination of Deep Learning and Reinforcement Learning has become increasingly popular [27][28]. Its effectiveness has been demonstrated in multiple areas. One of the domains in which the use of Deep Reinforcement Learning has been utilized is in the area of robotics. Within robotics, the application of DRL with regards to exploration of areas by autonomous robot has shown promise. Digor et al. demonstrated effective strategies for exploration of areas using 3D laser scanning[29]. Niroui et al. showed the effectiveness of deep reinforcement trained rescue robots in unknown, cluttered environments [30]. Uslu et al. showed that frontier exploration via deep reinforcement learning is effective[31].

2.5 Ensemble Approach

One potential method for the improvement of performance is to utilize an ensemble of models to collaboratively generate actions. This approach is used mainly in classification models but can be applied to many areas. Ensemble methods such as random forests have been shown to produce superior results than single models [32]. Additionally, the use of models which differ tend to produce better results when used in an ensemble [33]. Hansen and Salamon (1990) showed that in order for these ensembles to be more accurate, the diversity and accuracy of the individual components is crucial [34]. Yang et al. presented a use of the ensemble approach in their paper on deep reinforcement learning for stock trading [35]. In this paper, they use a trio of actor-critic models (A2C, PPO, and DDPG) in an attempt to improve the rate of return in stock trading. This approach proves to outperform both the individual algorithms separately, as well as two separate baselines, when measured against the risk-adjusted return [35].

3 Objective

With the promise shown by ensemble methods, a question arises. Can ensemble methods be effective when applied to robotic exploration? The goal of this paper is to answer this question. I do so by taking a trio of reinforcement learning models, and training them in a simulated robotic exploration environment. These models take in the simulated sensor information as inputs, and output actions from a set of defined actions available. Along with training the models, I develop three different ensemble method implementations. These ensemble methods take in the actions proposed by these three trained models and choose an action to take within this simulated environment based of different factors. I then take the trained models, and evaluate the ensemble methods' effectiveness by measuring the ensembles' effectiveness on separate simulated maps. The performance of each of the three ensemble implementations is measured against each other, as well as the three trained component models. The goal of this being, to determine whether the ensemble methods provide an advantage over the individual models when performing robotic exploration. The contribution of this work is in providing evidence that ensemble methods can produce improved performance over individual models when applied to robotic exploration. They can help mitigate the weaknesses which individual model may be prone to in certain environments. This knowledge serves as a starting point for a more in depth exploration of ensemble applications in robotic exploration.

4 Implementation

In this section, I will go over the methods used in the research process, including the environment used for the training and evaluation, the robot being simulated, the various technological components used, and the Reinforcement Learning model types implemented.

4.1 Environment

For the training and testing environment for this experiment it was not feasible to conduct training and testing in a real world environment and thus I opted to work with a simulation. As far as requirements for the simulation, it was necessary that the computational requirements were relatively low, due to the limitations in the hardware available. A simulation environment which was lightweight and would allow for running high numbers of iterations in relatively short periods of time was ideal. In the end, the 2D simulation environment created by the Roblearn team [8] for their work on autonomous mobile robot navigation in an indoor environment was selected.

4.1.1 Robot

The robot used in the simulation is the Turtlebot 2 based version used by the Roblearn team. The sensors simulated are a Hokuyo UST-20LX and Intel Realsense D435. The Hokuyo UST-20LX is a compact 2D laser scanner with the following specifications [8]:

- range: 0.06-20m
- angle: 270°
- resolution: 0.25°
- precision: +/- 40 mm
- frame rate: 40 Hz

The robot is capable of recognizing sharp turns up to 135° due to its large opening angle of 270, which also allows it to perceive nearly all of its immediate surroundings.
The second sensor, the Intel Realsense D435 is an RGB-D sensor which has the following specifications [8]:

- RGB-D Sensor
- Vertical opening angle: 60°
- Horizontal opening angle: 90°
- Field of View: 86 x 57 x 94° (+/- 3°)
- Resolution: 1280 x 720
- Frame Rate: 90 fps
- Distance: 0.2 - 10 meters

Due to it's horizontal opening angle, only one-third of the field of view of the laser scanner is covered. In contrast to the laser scanner, the RGB-D sensor's vertical opening angle allows it to perform a 3D scan of the surroundings [8]. For the simulation, the number of available actions for the robot were reduced to seven, with each action being a combination of an angular and linear velocity. This constitutes the action space. The available actions are the following linear and angular value pairs:

- angular = 1.25, linear = 0.3
- angular = 1.0, linear = 0.4
- angular = 0.5, linear = 0.5
- angular = 0.0, linear = 0.6
- angular = -0.5, linear = 0.5
- angular = -1.0, linear = 0.4
- angular = -1.25, linear = 0.3

In order to generate the possible observations for the robot, the 1081 values of each of the prior four laser scans are combined into a vector, along with the compass value orientation of the robot toward the goal. For orientation values, a vector with a size of 128 is mapped from the 0 to 360° orientation value of the robot. For this vector, all values are 0 except for the direction to the goal, which is given a value of 1. The laser input is given by distance values which are normalized between [0,1] with a maximum distance of 20 meters and provided in polar coordinates [8]. The robot has an update frequency of 2Hz.

4.1.2 Roblearn Simulation2D

The simulation environment itself provides several tangible benefits and is ideal as it is simple to implement and provides flexibility for environment design and expansion. The simulation is written in C++ and can handle eight floating-point operations simultaneously, allowing for the modeling of the laser beam intersections [8]. The environment is depicted by lines and circles which represent obstacles. The robot is shown as a circular shape with the laser scan coverage area emitted from it as the origin point. The orientation of the robot is indicated by the notch in the circle depicting the robot. The laser scan coverage area is depicted by green area and complies with the specifications of the UST-20LX on the plot which is rendered using gnuplot as seen in Figure 5. The gaps in laser coverage shown are due to obstacles blocking the sensor field of view. To simulate measurement error, Gaussian noise has been added to each laser beam. The goal point for the robot is indicated by a purple circle [8].

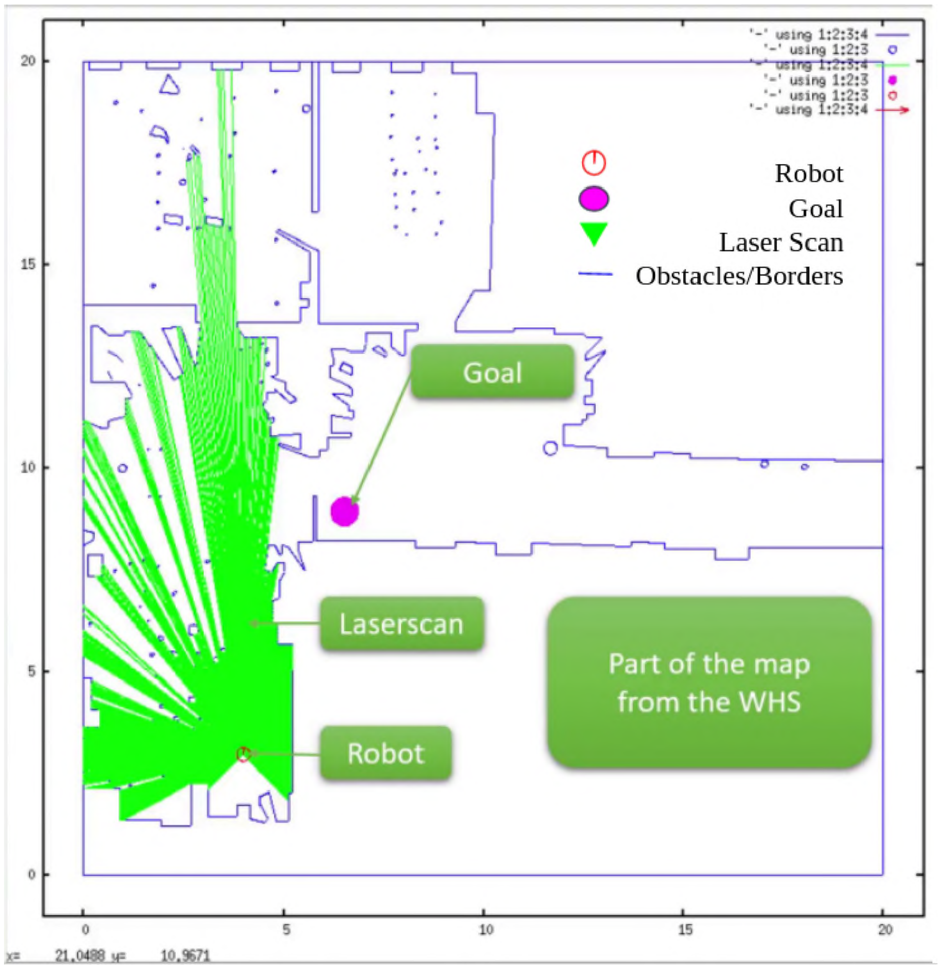

Figure 5: An example map in the Roblearn environment. The Robot is shown in Red, Laser Beams in Green, and the Target Point in Magenta [8].

The input vector for the neural network is a fused laser scan which is the point wise minimum of the 2D laser scan and the converted scan from the RGB-D sensor. This keeps the vector small, but it still contains 3D information from the environment [8].

Environmental maps can be generated from actual data in real world scenarios, generated from the inkscape editor by hand, or directly generated with Python. Maps can be set to run with different configurations for the start location of the robot, the goal location, and their relationship to each other. The node selection is set by the Mode passed to the environment after it is created. The modes we used for our training and evaluation were: Pair All, All Random, and All Combination. Pair all uses sets of paired robot and goal locations. This mode is needed for some maps where the obstacles are designed for such pairings. All Combination picks the next start and goal locations from a list of each. All random picks randomly from these lists of start and goal locations. Roblearn provides guidelines for which modes to pair with the various maps included in their GitHub repository. Another option is to specify whether the environment should reset when the robot reaches the goal point, or generate a new goal. This also has guidelines specified in the Roblearn GitHub repository (https://github.com/RoblabWh/RobLearn).

4.1.3 Gym

In order to simplify the interaction between the environment and model, I decided to make use of some of the tools provided by OpenAI in an effort to reduce redundancy of development and utilize trusted, proven implementations. OpenAI is a non-profit AI research laboratory which has a stated goal of aiding in the promotion and development of AI so that humanity as a whole can benefit [36]. OpenAI provides several useful tools for AI, and particularly reinforcement learning, research. Out of the tools provided by OpenAI, I utilized their implementation of Gym, as well as implementations of some of their actor-critic models. Gym is a toolkit which is designed for the development and comparison in reinforcement learning.

One of the major impediments in reinforcement learning, as identified by OpenAI, was the need for better benchmarks, and a lack of standardization of environments. OpenAI created Gym to address these issues [37]. It has the benefits of being able to be used with a multitude of agents, as well as compatibility with computation libraries such as TensorFlow and Theano [37]. The Gym library consists of a collection of different environments for common reinforcement learning problems. The environments have a shared interface which allows the writing of general algorithms for navigation [37]. This means that, by implementing your own environment with the provided interface, custom environments can take advantage of the universality the gym interface provides and have a multitude of models interact with them interchangeably.

Gym environments implement the step, reset, render, and close methods, as well as other optional methods. In order to get observations within the Gym environments, the step method is provided. This returns the following: observation, reward, whether or not the end state has been reached, and diagnostic information. The reset method resets the environment and returns an initial observation. Render call the rendering component of the environment for visualization while the close method is self-evident [37].

4.1.4 Gym Environment

In order to use the simulation environment with the stable baselines models, the environment first had to be modified to match the OpenAI Gym framework within which the models are designed to operate. In order to implement this shared interface it was necessary to modify the existing environment created by the Roblearn team to inherit from the GymEnvironment class and implement the Gym interface. This entailed implementing the step, reset, render, and close methods, as well as specifying the action and observation spaces for the environment. The action space is defined as a Discrete Gym space with a size of seven. Discrete spaces are defined in Gym as: "A list of possible actions, where each timestep only one of the actions can be used [38]." This correlates to the seven available actions for the robot as it can only take one of these action at each step. The observation space is a Gym Box shape with a low value of zero, a high value of one and a shape of the combination of the 1081 laser scan values and the 128 orientation values. A Box shape is defined in Gym as: "A N-dimensional box that contains every point in the action space [38]."

4.1.5 Reward

The goal for the robot is to reach the goal in the environment and the reward for actions of the agent is based upon this goal. The reward function used is that defined by the Roblearn team [8] and as they describe is tied to the environment and is a reflection of how well the agent's actions have translated toward an achievement of the goal. The initial reward is 0. The reward for the agent reaching the goal is 20. The penalty for a collision is -20. The intermediate reward, that for each step which does not result in a collision or reaching the goal, is based upon two components. These are:

1. A small reward is given if the distance from the robot to the goal is shorter than the last distance, a small positive reward is given, otherwise it is given a small negative reward [8].

2. The robot is given a small positive reward if it is oriented more towards the goal than in the prior state, and a correspondingly small negative reward if it is oriented further from the direction of the goal [8].

Additionally, if the maximum allowed steps is reached, the agent is subject to the same penalty as a collision of -20. If an agent reaches the terminal state, or the maximum allowed steps has been reached, the agent is awarded the cumulative reward for each step including the final step reward/penalty. As Surmann et al. describe [8]: "The extended reward function leads to robust and fast learning results. While learning, the agents often reach circling states but are able to recover."

4.2 Models

Another of the initial decisions that needed to be made was what implementation to use for the Actor-Critic models. I decided to make use of a key tool provided by OpenAI, their implementation of reinforcement learning algorithms, particularly actor-critic algorithms. These algorithms are proven and chosen because they are on the cutting edge in terms of reliability and sample efficiency among policy-learning algorithms [39]. There are third parties which provide implementations of these OpenAI algorithms, and I elected to use the stable baselines implementations for my study.

4.2.1 Stable Baselines

Stable Baselines is a set of implementations of Reinforcement Learning algorithms. These implementations are improvements upon the OpenAI Baselines which feature code cleanup as well as major structural refactoring. These include [38]:

- Unified algorithm structure
- Unified code style
- Documentation of classes and methods

- Additional code coverage and test
- Support for and implementation of additional algorithms

The Stable Baseline implementations were ideal as they would allow the use of the multiple models required for an ensemble implementation with a standardize interface. Additionally, they provide improved implementations which could be plugged into Gym environments with ease. Out of the many available Stable Baselines algorithm implementations, I selected the following.

4.2.2 Actor-Critic Algorithms

Actor-Critic algorithms consist of two networks (actor and critic) working in tandem to find a solution to a given problem. The actor portion is responsible for choosing actions at each step in the process, which the critic network gauges the quality or the Q-value of a given input state [40].

4.2.3 Value-Based vs Policy-Based Methods

The two main types of learning algorithms utilized in Reinforcement Learning are Value-Based and Policy-Based. Google DeepMind is a prominent implementation of a value-based Q-learning method [1]. Surman et al. make use of the policy-based GA3C model for their aforementioned robotic exploration reinforcement learning implementation [8]. In value-based algorithms, actions are selected based on the predicted value of the input state or action. The higher the value, the better the action. These algorithms attempt to learn the state or state-action values and act by choosing the best possible action for the given state. Q Learning utilizes this algorithm type.

Policy based algorithms map input states to output actions based upon a policy they learn directly [40]. Policies in reinforcement learning pertain to the strategy agents employ when attempting to reach their goal. As opposed to value-based methods, agents using policy-based methods refer to their policy when choosing actions at each step.

There are advantages and disadvantages to the use of each of these methods. Because they have quicker convergence, policy-based methods are typically better for stochastic and continuous environments as opposed to Value-based methods which tend to be more steady and sample efficient [41].

4.2.4 Model Selection

For my purposes, policy-based methods were the ideal choice as my goal was for the robot to learn an optimal policy which it could use for autonomous exploration. With this type of method chosen, the next step was to decide which specific models to use as part of the ensemble. Out of the many available Stable Baselines models, I selected the following three:

4.2.5 Advantage Actor-Critic

Advantage Actor-Critic (A2C) is a deterministic, synchronous variant of the Asynchronous Advantage Actor Critic (AC3) introduced by Minh et al. in their paper on asynchronous deep reinforcement learning methods [42]. A3C computes estimators of returns and advantage functions via fixed-length segments of experience in an architecture that shares layers between the policy and value function, and as the name implies, utilizes asynchronous updates. A2C was developed as a synchronous alternative to A3C by researchers [43]. These researchers determined that by waiting for each actor to complete its segment of experience and then performing an update, taking an average over all the actors, they were able to remove the asynchronous portion of the model. This has an advantage of more effectively utilizing GPUs. A2C combines the two reinforcement learning algorithm types, policy based and value based in order to gain the benefits of both while limiting the shortcomings. A2C and A3C are some of the most popular DRL algorithms. For this paper, the stable baselines implementation of the OpenAI A2C model was selected for its ease of implementation and abundance of support.

4.2.6 Proximal Policy Optimization 2

For the second model in my ensemble I selected the stable baselines implementation of the OpenAI PPO2 model. Proximal Policy Optimization (PPO) algorithms were introduced by Schulman et al [44] in their 2017 paper. They proposed them as “a new family of policy gradient methods for reinforcement learning [44]." As opposed to standard policy gradient methods which perform a single gradient update once per sample, PPO uses epochs of mini batch updates, performed multiple times. In simpler terms, PPO collects experiences interacting with the environment in small batches, then uses each batch to update its decision making policy. After the policy has been updated, the old batch of experiences is discarded and a new one is collected using the updated policy. Because of this policy update method, PPO has less variance in training, though this comes at the cost of bias. Another benefit of this is that PPO tends to ensure that the agent does not devolve into taking senseless actions [45]. PPO has some of the benefits of Trust Region Policy Optimization (TRPO), with the added benefit of being simpler to implement, more general, and have better sample complexity. According to Schulman et al. [44] TRPO "is similar to natural policy gradient methods and is effective for optimizing large nonlinear policies such as neural networks." The PPO2 model I use is OpenAI's implementation of the PPO made for GPU. As opposed to PPO1 which used MPI, it makes use of vectorized environments and uses these for multiprocessing [38].

4.2.7 Actor Critic using Kronecker-factored Trust Region

For my final ensemble model, I selected Actor Critic using Kronecker-factored Trust Region (ACKTR). ACKTR was introduced by Wu et al. in their paper titled *Scalable Trust-Region Method for Deep Reinforcement learning using Kronecker-factored approximation* [46]. In the paper they outline their actor critic model which takes three distinct techniques and combines them. These techniques are: actor-critic methods, trust region policy optimization for more consistent improvement, and distributed Kronecker factorization to improve sample efficiency and scalability [47]. TRPO can suffer from scalability issues, and because of this, it can be impractical for large deep networks. This is due to the need to use large batches for approximations which is computationally expensive. It has been shown to perform poorly on tasks requiring Recurrent Neural Networks (RNN) or Convolutional Neural Networks (CNN) [48]. ACKTR improves upon this by applying the Kronecker-factored approximation to both the actor and critic, as well as keeping a running average of the curvature information. As a result, ACKTR scale more effectively with larger deep network models and solves tasks which TRPO cannot handle. One of the drawbacks of ACKTR is that it is not as straightforward for shared parameter networks such as RNN or others [48]. For this paper, the stable baselines implementation of the OpenAI ACKTR model was utilized.

These three models were selected for several reasons. The first reason was that the models used needed to be able to operate in my environment. Since the Simulation2D environment is a Box type, this narrowed down the number of models. Out of the eligible models, A2C, PPO2, and ACKTR stood out as they were recommended by Stable Baselines for discrete action spaces, with multiprocessing [38].

4.3 Training

For the training process, there were several decisions that needed to be made for how to conduct the training process. Some of the key decision were:

- Map Selection
- Number of Parallel Environments
- Map Training Order
- Training Episodes
- Training Timesteps

4.3.1 Tools

During the training process, several tools were utilized which provided metrics and guided the selection of the model and training parameters. I made use of the Stable Baselines callback functions such as the EvalCallback. This callback is an EventCallback that is triggered when a new best model is found during the training process. During training, it conducts an evaluation of the model at the interval specified by the user. It outputs the score for the model during the evaluation, and if the model has recorded a new high score, it saves the model to the directory specified by the user. This proved invaluable during the training process as it allowed me to automatically save the model during the training, and avoid having an over-fitted model at the end of the process. Additionally, I utilized the stable baselines integration offered with TensorBoard. TensorBoard is a visualization toolkit provided by TensorFlow. It allows you to visualize a variety of different metrics related to the training process for reinforcement learning models, including key histogram presentations of loss and accuracy measures [49]. These functions can be set up such that they are called periodically during the training process.

4.3.2 Vectorized Training Environments

One way to improve training with a single agent is to make use of vectorized training environments. With vectorized environments, multiple independent environments are stacked into a single environment. This allows for the training of a single agent on multiple environments per step, as opposed to a single environment. The actions, observations, rewards, and end state flags are all vectors with the same number of dimensions as the number of environments [38]. Vectorized training environments have the potential to improve the training process, especially for policy learning, due to the increased variety of experienced states. For my training, I elected to use a vectorized environment of 12 environments. This was the maximum number of environments my system could handle before showing a large performance drop-off.

4.3.3 Map Selection

When choosing which maps to use for the training process, there were several factors to consider. The Roblearn team included six maps in their repository. The first step was to divide these into a training and test set. I elected to go with an even split of three training and three evaluation maps. For a balanced training set, a selection of maps with varying difficulty levels seemed prudent. Fortunately, the Roblearn team identified difficulty levels with regards to the included maps.

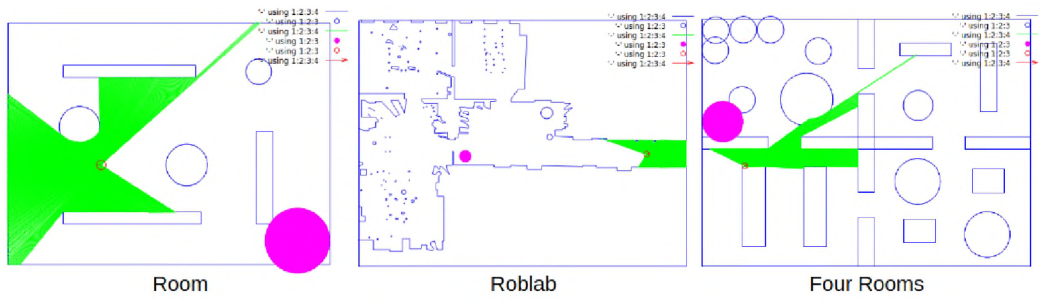

Figure 6: Training Maps

<u>Training Map Set (Figure 6)</u>

- Room - Difficult

- Roblearn - Medium
- Four Rooms - Difficult

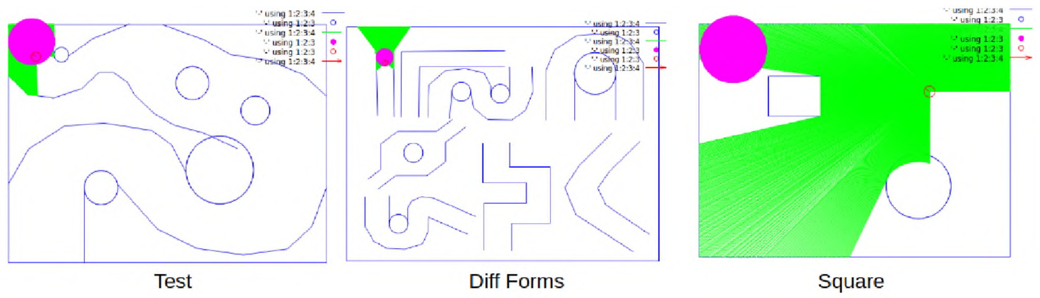

Figure 7: Evaluation Maps

<u>Evaluation Map Set (Figure 7)</u>

- Test - Medium
- Diff Forms - Difficult
- Square - Easy

4.3.4 Training Order

With the training and evaluation sets determined, the training order was another problem to consider. Initially, I performed training on single maps. Using a vectorized environment consisting of 12 of the same map, the model was trained until proficient on that map and then sequentially trained on the next map in the same fashion. This method proved sub-optimal, as the model tended to develop behaviors that were biased toward the original map and were slow to overcome these during further training on the following maps. I determined that it was more effective to use a vectorized environment consisting of four individual environments for each map, for a total of twelve environments within the vectorized environment. This allowed the models to learn a variety of behaviours adapted to more varying obstacles. A capture of the models being trained can be seen in Figure 8.

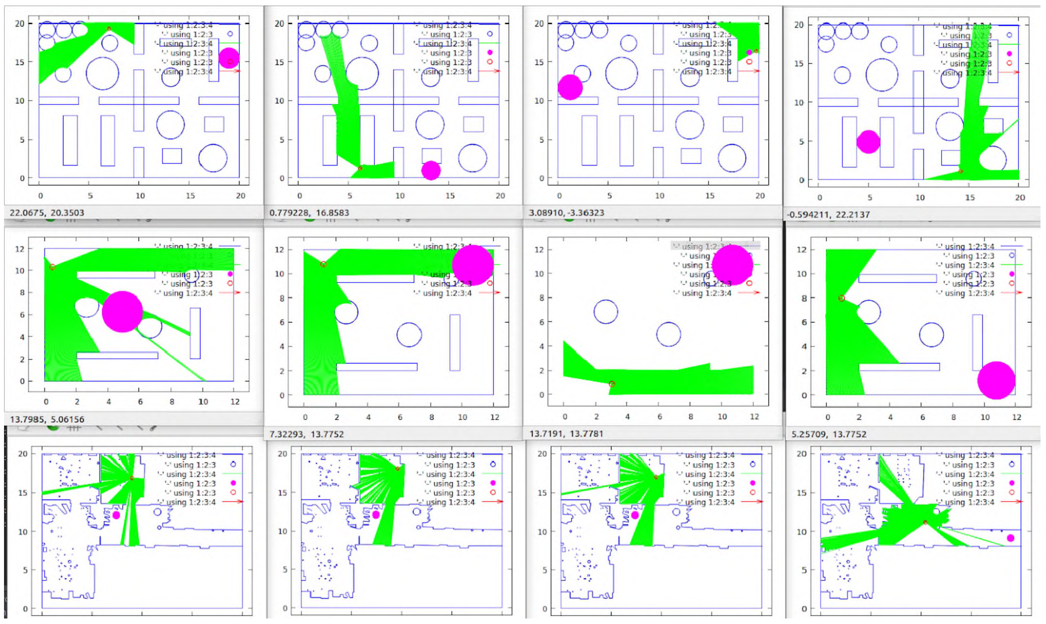

Figure 8: Visualization of Training on Twelve Maps Simultaneously

4.3.5 Episodes and Timesteps

When determining the number of episodes to run the training for each model, I used the TensorBoard metrics to identify points when the episode reward values, loss, and advantage were no longer improving. Through a repeated trial and error process where training order was changed and parameters were modified, I was able to determine the episode counts where improvement in the models began to drop off. This point was different for each of the models and thus I elected to use a different number of episodes for each. For A2C I ran the training for 100,000 episodes each, on the twelve environments in the vectorized environment, for a total of 1,200,000 episodes. For PPO2, I ran the training for 200,000 episodes on each, for a total of 2,400,000 episodes total. For ACKTR, I ran for a total of 1,000,000 episodes. The difference in episode counts is a result of the sample efficiency of the models. ACKTR is the most sample efficient of the three, followed by A2C. PPO2 was the most sample inefficient and required the most training episodes to train effectively. For the optimal number of timesteps to run per episode, I chose to follow the Roblearn teams lead and use 1000 as the maximum allowable number. This was to ensure if the robot reached a loop state this would not lead to excessively long episodes with no progress was being made.

I thus trained the model for the number of episodes specified above, on the *Room*, *Roblab*, and *Four Rooms* maps simultaneously, for an episode total of 3,000,000 episodes for ACKTR, 3,600,000 episodes for A2C, and 9,600,000 episodes for PPO2.

4.4 Policy

For each of the three models in the ensemble method, I used the stable baselines Multi-layer Perceptron (Mlp) Policy model. Multi-layer Perceptrons are one of the most useful types of neural networks. The stable baselines Mlp Policy implements actor critic using two layers of size 64 [38].

4.4.1 Model Parameters

When determining the optimal model parameters, I began with the stable baselines default values as the starting point, then made modifications to these in order to try to improve the training results. This was largely a trial and error process where one of the parameters would be modified, then the training re-run, and the training results consulted to determine if improvements had been made. TensorBoard was key here as it provided visualization for the various training results and allowed me to determine parameter modification effects.

4.4.2 Learning Rate

Learning rate played the most important role in improving the models' behaviour. Learning rate, also refered to as alpha, determines the rate at which a model values new experiences versus old ones. It plays a key role because with too high a learning rate, the model will set its policy early on, become overly-optimized, and not update it over time. If the learning rate is too low, the model will accept too much from new experiences and may never converge to a useful policy. Fine tuning the learning rate for each model took a significant amount of time and is likely still an area where improvement could be made. During training, I found a learning rate which decreased over time was the most effective form.

4.4.3 Entropy Coefficient

Another key parameter is entropy. Entropy controls the random nature of the model's decisions. Higher entropy is better in the beginning as we would like the model to explore more and try more things. Ideally, entropy should decrease over time, as we want the model to stabilize and develop an intelligent policy. A balance in initial entropy, as well as the rate of decrease is important to ensure the model does not get too set based on early choices, and still tries new actions as time goes on. I made modification to the entropy coefficient for the models in order to optimize the entropy values over the course of training for each of the models.

4.4.4 Final Values

For the A2C model, I used a gamma discount factor of 0.99, a value coefficient for the loss calculation of 0.25, an entropy coefficient for the loss calculation of 0.01, a maximum value for gradient clipping of 0.5, an alpha value (RMSProp decay) of 0.99, an epsilon value of 1e-5, and a learning rate value of 0.003. I kept the A2C model at a constant learning rate as this produced the best results. I used five as the number of steps in each environment to run before updating. These values were the result of a trial and error process outlined in the A2C training section 4.5.1.

For the PPO2 model, I used an alpha value of 0.99, an entropy coefficient value of 0.005 for the loss calculation, and used 1000 as the number of training mini batches per update and 4 for the number of epochs when optimizing the surrogate. I used a learning rate of 2.5e-4, a max gradient norm of 0.5, a lambda factor for trade-off of bias vs variance for Generalized Advantage Estimator of 0.95, and a clip range of 0.2. I again limited the steps to 1000.

Finally, for the ACKTR model I used an gamma discount factor of 0.99, an entropy coefficient for the loss calculation of 0.01, a value coefficient for the loss calculation of 0.25, a weight for the Fisher loss on the value function of 1.0, a learning rate of .25, 0.5 as the clipping value for the maximum gradient, and a gradient clipping for Kullback-Leibler of 0.001. I used a linear scheduler for the learning rate updater. I limited the steps to 1000 for this as well. Training time for ACKTR took approximately 8 hours.

4.5 Training Results

When evaluating training, I looked at the TensorBoard metrics for the training session. In particular, I was interested in Episode Reward, Advantage, and Loss.

Episode reward was the primary metric used overall when determining improvement in the model during training. At the end of training, the goal was a model which scored close to the optimal score for the map, in the allotted number of timesteps. This varied depending on whether training on a map continued after a goal was reached or ended. Regardless, if no improvement in episode reward was shown over long periods of time, this was a good indicator that the training was no longer producing improved results. In some cases, lower rewards could occur for some time, followed by improvements. This made it difficult to know when to effectively end training, as ending before this secondary rise in improvement leads to lower model performance, but this was difficult to anticipate.

Advantage was another metric used when evaluating training effectiveness. Advantage is a measure of the value of a certain action, given the specific state at the time. Mathematically, it is defined as $A(s,a) = E[r(s,a) - r(s)]$ where r is reward and r(s,a) is the expected reward given state s and action a, and r(s) is the expected reward prior to an action being selected [50]. The advantage should grow overall with the training, though there may be dips during the process.

Loss, being a measure of the difference between the chosen action and the optimal, we would like to decrease over time. However, there are periods where increased loss are acceptable, as they lead to future improved results. This was the case in my training, as I observed loss increasing initially, leveling of and decreasing at times, but then resuming the increase until hitting a plateau. This is in part due to the increased complexity as the model performs better and is able to achieve greater rewards.

The discount factor was the final metric I used to determine effectiveness. This determines how much the agent values current versus future rewards. Again there needs to be a balance here, otherwise the model will choose actions which lead to a short term reward and never explore options that may result in short term penalties, but pay off in the long term. In my case, too low of a discount factor could lead a model to never move away from the goal, since that results in a small negative reward. This would lead to it only traveling directly towards the goal which is not optimal in a large number of situations.

4.5.1 A2C Training

A2C training initially proved frustrating. The model was prone to devolving into spinning in circles, especially earlier on in the training process. In order to combat this, adjustment of the learning rate was needed. Patience was also required as the model could recover from these spinning episodes if properly configured, and given enough time. In order to achieve this, I trained the model for 10 million episodes to ensure that it had maximized its learning. Another difficulty encountered was configuring the number of steps to execute before performing an update. Initially I set this to 300, in line with what the RobLearn team used for their A3C model. However, this proved to reduce the model's ability to adapt, and I ended up reducing the number of steps down to five. This improved performance greatly. Overall training for the model in the final configuration took approximately 6.5 hours. The final result was the most effective model of the three. The tensorboard results for the training can be seen in Figure 9. The entrop loss values and discounted rewards show the progress of the training, with the entropy loss decreasing over time paired with the discounted rewards increasing.

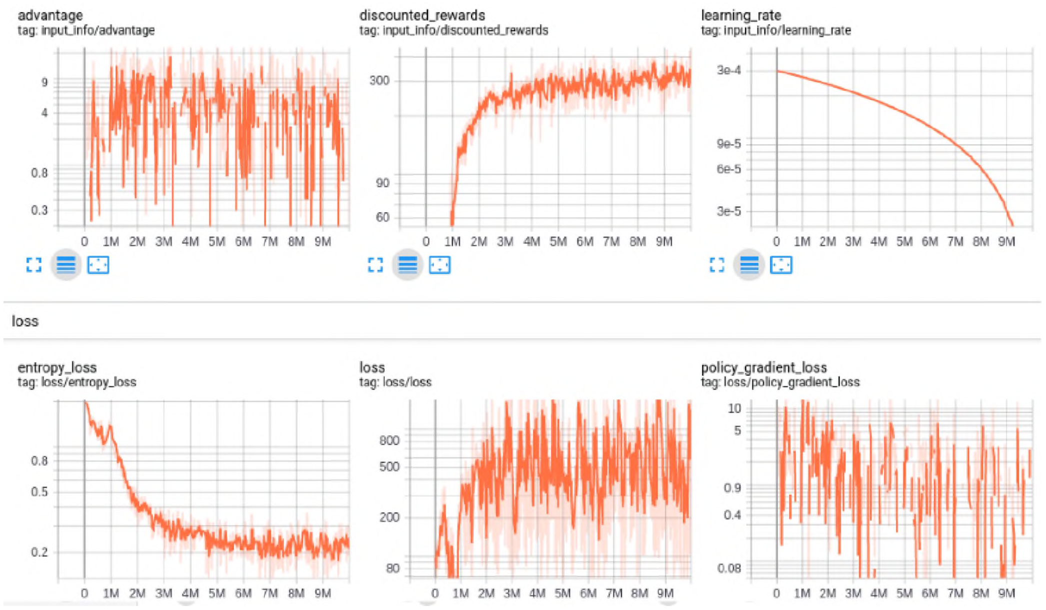

Figure 9: A2C Training Results

4.5.2 PPO2 Training

PPO2 proved the most difficult to effectively train. Like the A2C model, it was prone to falling into continual circles. However, it did not recover as well and this led to long training sessions with no improvement. I was unable to reach a point with the model where it had mastered the *Four Rooms* map. I was able to train it to the point where it successfully performed in the other two maps and I used a training length of ten million episodes for this as well, taking over ten hours to complete. The tensorboard results for the training can be seen in Figure 10. The difficulties training this model are evinced by the spikes in the entropy loss and loss values which were difficult to stabilize, as well as the spikes in discounted reward and value function loss.

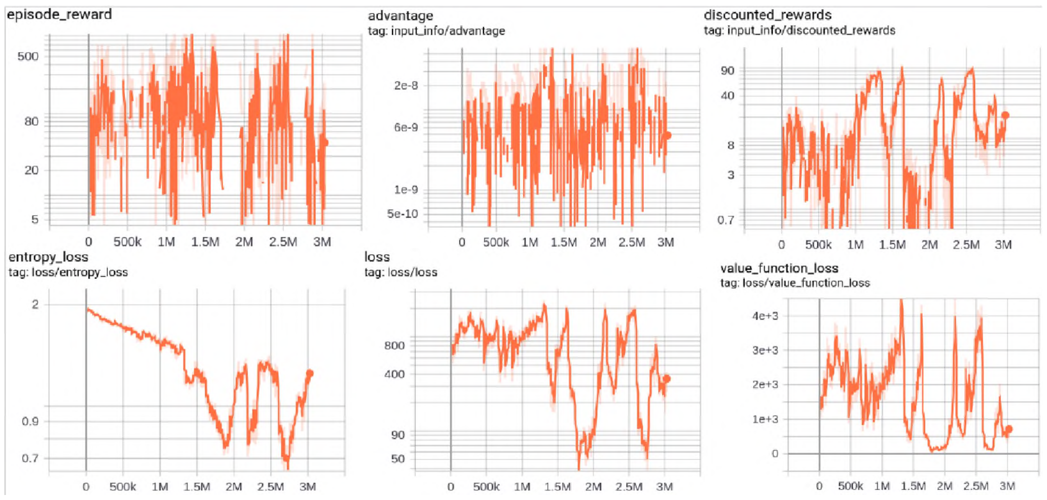

Figure 10: PPO2 Training Results Showing Large Spikes in Advantage and Loss

4.5.3 ACKTR Training

ACKTR Training took the longest of the three models with some interesting quirks. When running the training, rewards would plateau, and even decrease in some cases, for long periods (one million plus episodes). Due to this, initial training was not as effective. Eventually I discovered that extended training would overcome these plateaus. The final training was performed over the course of seven million episodes. I selected this as the cutoff due to episode reward leveling off. The advantage and discounted rewards leveled off as well, along with the learning rate dropping to zero. Total training time for the model was 380 minutes. The tensorboard results for the training can be seen in Figure 11. The steady progression of the episode reward, as well as the increase and leveling of the advantage values show the progress of the training to a maximized point. Additionally, the steady decrease of the entropy loss is another positive indication of effective training.

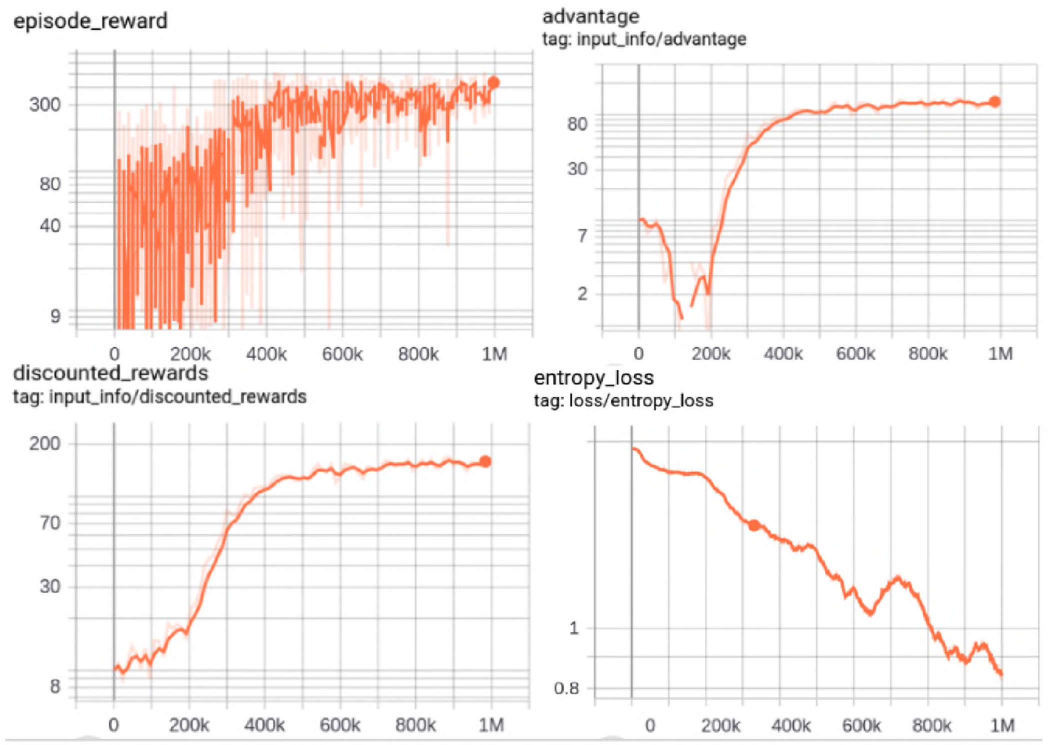

Figure 11: ACKTR Training Results

4.6 Ensemble Method

The basic operation of the ensemble method used is illustrated in Figure 12.

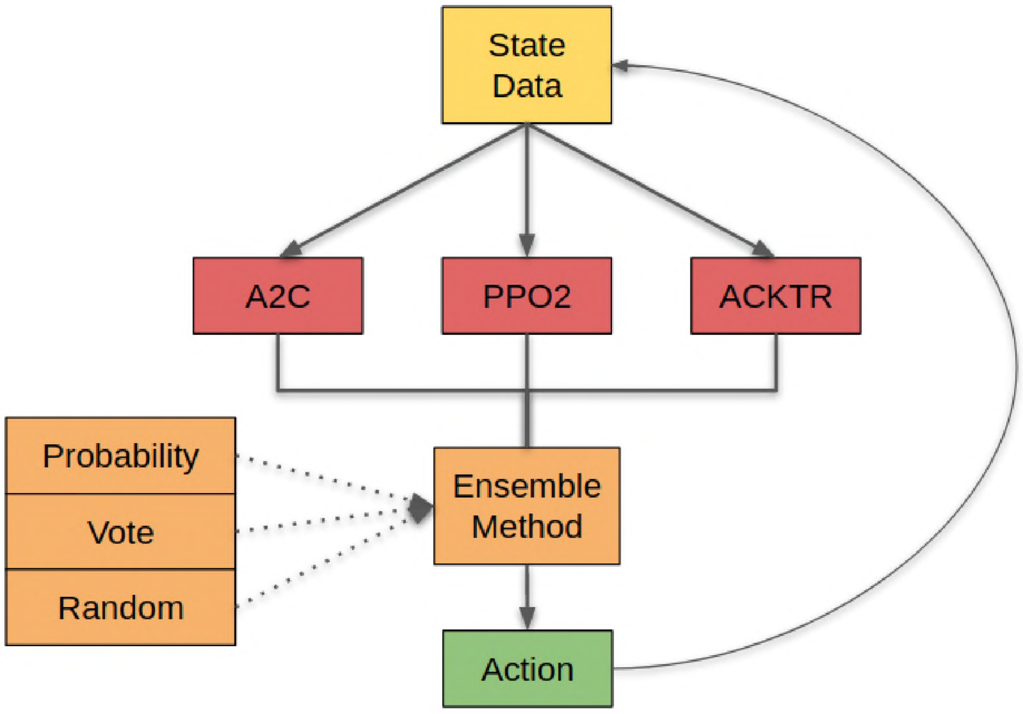

Figure 12: Ensemble Method

At each step, the models are each presented with the current state. Each model then generates its recommended action. From here, the ensemble method logic selects one of the actions and uses it as the action to take. The environment is updated using this action and the process repeats.

As far as the ensemble logic is concerned, there were three different methods used.

4.6.1 Random Method

The first method used is a random selection method. In this method, a random action is picked from the actions generated by each of the three models. The random selection is done using the Python Random library Sample method:

```
def ensemble_strategy_random(action, action2, action3):
    actions = (action, action2, action3)
    chosen_action = rand.sample(actions, 1)

    return chosen_action
```

This random selection serves as a baseline for the ensemble methods to see if chance performs better than the following, more selective methods.

4.6.2 Vote Method

The second variation is a voting method. In this, the action selections of each model are compared. If any two or three models agree on an action to take, that action is selected. Otherwise, a random action is selected from the three again using the Python Random Sample method:

```
def ensemble_strategy_vote(action, action2, action3):
    if action2 == action3:
        chosen_action = action2
    elif action == action2:
        chosen_action = action
    elif action == action3:
        chosen_action = action
    else:  # pick random action since no majority
        actions = (action, action2, action3)
        chosen_action = rand.sample(actions, 1)

    return chosen_action
```

For this method, I attempted to leverage the experience of multiple models. The theory being, if two models agree on an action, there is a greater likelihood that the action is a better one.

4.6.3 Probability Method

In the third variation, each model outputs a probability vector listing the probability for each of the seven action choices possible. These probability vectors are summed, and the action with the highest combined probability is selected. This method was chosen to benefit from the combined knowledge of each of the models while ideally mitigating a single model's preference:

```
def ensemble_strategy_probability(probVec1, probVec2, probVec3):
    combined_probabilities = probVec1 + probVec2 + probVec3

    chosen_action = np.argmax(combined_probabilities[0])

    return chosen_action
```

The thought behind this method is leverage the probability vectors in order to determine the action with the overall highest probability. This action should have the greatest likelihood of being the best choice.

4.7 Evaluation

In order to evaluate the ensemble approach I elected to run each of the three ensemble methods on each of the three evaluation maps. I also ran each of the three individual models on the three evaluation maps for comparison. For all of these, I ran them on each map for 100 episodes, keeping the timestep limit at 1000 steps to avoid lengthy loops. The same mode is used for determining robot start location and goal location on each map for each evaluation run. The start and goal for the *Test* map is set and does not change. For the *Diff Forms* and *Square* maps, we used the pair combination mode which uses a set order. This ensures that the same sequences are used for each evaluation to remove the possibility of variation due to different start and goal locations. The start and end locations for the *Diff Forms* map changed after each episode and were on a set rotating schedule. For the *Square* map, each episode continues until the robot has a collision, or the max number of timesteps is reached. Due to this, the better a model performs, the more goals it will encounter in each episode. However, the order of start and goal points is set for each episode to ensure overall parity. For each episode I track the score and add it to the total score for the model, at the end I take the sum of the model's scores and generate the average for that map to use for map performance comparisons. I take the overall average of each model's performance over all three maps as well. This average is used for overall performance comparison.

5 Results

The performance of the models was measured by their total score for the one hundred episodes on each evaluation map, as well as their total score average across the three evaluation maps. The results of the evaluation are presented in Table 1 showed that the ensemble method performed better on average than two of the three individual models.

Table 1: Evaluation Results

	A2C	PPO2	ACKTR	Random	Vote	Probability
Diff Forms	404.13	241.68	309.45	363.57	376.15	359.25
Test	607.82	239.47	334.83	436.13	493.81	324.05
Square	628.91	77.73	588.35	533.30	600.58	635.26

Results from 100 episodes on three maps. Performance was measured by the sum of each model's total score for each map.

On the *Diff Forms* map, all three ensemble methods outperformed ACKTR and PPO2 while falling short of A2C, with the Vote method scoring the highest. On the *Test* map, the Vote and Random ensemble methods outperformed ACKTR and PPO2, while the A2C model narrowly outperformed all models. The Probability ensemble fell just shy of the ACKTR but outperformed PPO2. On the *Square* map, PPO2 had a large fall off in performance, scoring well below all other models. The Probability ensemble outperformed all models for this map, while A2C performed the second best. The Vote ensemble outperformed the other two individual models, while the Random ensemble only outperformed PPO2.

One item of note in these results is that the poor performance of the PPO2 model on the *Square* map did not lead to a corresponding degradation in the Ensemble models' performances. One of the hoped benefits of an ensemble is the minimization of the weakness of the individual component models and it would appear to have worked in this instance.

Table 2: Overall Averages

	A2C	PPO2	ACKTR	Random	Vote	Probability
Avg. Score	546.95	186.29	410.88	444.33	490.18	439.52

Average score for each model across the three maps

The total average for each model are presented in Table 2. This shows that the A2C ensemble method had the highest overall average score, outperforming all three ensemble models, along with the two other individual models. The Vote ensemble method had the highest average for the ensembles, outperforming ACKTR and PPO2 as well. Its average was lower than that of A2C by 10.38%. The Random ensemble was outperformed by A2C on average by 18.76%, while the Probability ensemble was outperformed by A2C on average by 24.44%. Vote, Random, and Probability outperformed ACKTR on average by 19.30%, 8.14%, and 6.97% respectively, and outperformed PPO2 on average by 163.12%, 138.51%, and 135.93% respectively.

The Vote method outperforming the Random method was anticipated, though one might have expected the probability method to produce better results. Since the random method randomly picks a choice out of the three presented actions, there was not an expectation that this might perform better than the individual models, but likely under perform compared to the other two ensembles. Because the voting method uses an intelligent approach where it selects an action if two or models agree, there is a greater likelihood that the action it selects is the optimal one. The probability ensemble we expected to perform better, since they are combining the probabilities generated by each model and there is a greater likelihood that the chosen action has a high probability of being optimal. However, it could have suffered as a result of possible bias on the part of the PPO2 model. Some refinement in the implementation of the probability method could lead to increased performance, as it is taking the probabilities of all the possible actions, not just the recommended actions of each model.

6 Conclusion and Future Work

6.1 Conclusion

In this study, the goal was to determine whether an ensemble method approach to autonomous robot exploration would prove more effective than the use of a single model. The results show that although the ensemble approach does not always outperform every individual model, on average it outperformed the majority of the component models. This would indicate that an ensemble approach would be beneficial for use, especially in unknown environments which may contain obstacles that would put an individual model at a disadvantage. The benefit of the ensemble method versus an individual model is that it has the ability to mitigate the individual model's weakness and provide more balance. This leads to better average performance. I was able to demonstrate that the vote and random ensemble methods used here are able to outperform the individual component models most of the time.

The results also show that the implementation of the ensemble plays a large role in performance as well. This is evinced by the differences in the three ensemble implementations performance. The Probability ensemble performed the poorest out of the three, though it did outperform the three component models in two of the three maps and all three models in one of the maps. Overall, its average was well below that of the other two ensemble methods. The Vote model performed the best, indicating that using a consensus of action choices provides more optimal results.

Another takeaway from the results is that model training, while key to the effective performance of the ensemble, is not a deal-breaker in terms of effectiveness. If one or more of the ensemble members is not as effectively trained, it will not necessarily render the ensemble ineffective. This was shown when the PPO2 model performed poorly on the *Square* map, but the ensembles all still outperformed the ACKTR model. However, the poor performance of the PPO2 model may have been the reason the ensembles' results were below the A2C model's results by a larger factor for this map. Effective training was by far the largest challenge experienced during the process. Refining the separate training processes was both time consuming and difficult. This could be considered a drawback when comparing the use of an ensemble versus a single model. It would be much simpler and faster to train a single model to proficiency, though the results may not be as good.

6.2 Future Work

One of the downsides of working only in a simulated environment is that things do not always translate to the actual performance of a real robot. The next step would be to implement the method on an actual robot instead of just in simulations and perform evaluations of its performance to determine the real impact of the ensemble. This was not feasible due to a number of factors during the time this was conducted but would definitely be the preferred next step in the process. Other future possibilities to improve upon the work here follow.

6.2.1 Ensemble Improvements

There is likely room for improvement for the implementations of the ensemble methods. One possible improvement would be to modify the voting method to use best performing model's action in event of tie instead of making a random selection from the three choices. To do this one could track the running average reward for each model's chosen actions and then choose the action of the model with the highest average in the event of ties. Implementation of this would have to be done carefully as it could favor a single model unduly, especially early in the exploration process. The drawback of this being an undue bias in model choice if one model has been selected more initially. Also, immediate rewards are not necessary reflective of long term success which could be another drawback of choosing a model with better early scoring.

Another way to improve the ensemble would be to try it with different reinforcement learning models. There are several options for this. One way to do this would be to remove the poorest performing model and replace it with another model trained on the same environments. Another would be to train the models on separate environments to see if they balance out. A further option would be to increase the number of models within the ensemble, either adding new model types, increasing the number of each model type, or some combination thereof.

6.2.2 Training Improvements

Training models effectively is key to the process. The following are some possible training improvements. One future area of study would be to measure the impact of poorly trained component models on the overall performance of the ensemble. Substituting controlled models into the ensemble and measuring the score differences could help determine how big of an impact this has, and may help to devise ways to mitigate the effects.

6.2.3 Hyperparameter Optimization

One of the most important parts of Reinforcement Learning is the proper configuration of the models. Even small changes in the hyperparameters used for training can have a large impact in the success or failure of the model to develop a maximal policy for the environment. There are some tools available which provide hyperparameter optimization. I discovered an integration of one of these tools, Optuna, but its integration with Stable Baselines did not exist at the time I was conducting my study. Recently, in Stable Baselines 3, they have introduced integration with this tool and it could prove invaluable in the improvement of individual and ensemble model performance.

6.2.4 Parallelization

Another possible improvement to the ensemble approach would be to make use of parallel agents when training the models. One of the major advantages of the Roblearn group's approach was the ability to utilize parallel agents with GA3C. They used eight trainers, on thirty-two environments. One of the drawbacks of the Stable Baselines implementations is a lack of built in parallelization support. Going forward, building a library which incorporates the Stable Baselines models and allows for parallel agents in training would likely provide a significant gain in individual model, and ensemble performance.

6.2.5 Map Improvements

One further way to improve model training would be to create additional maps which provide training artifacts that would improve the models' ability to make inferences about their environments and the obstacles therein. Maps which encourage the model to explore corners and not focus solely on the center portions of maps would be one possibility. This could help reduce the issues observed with the models where they struggled on maps such as the square, room, and four rooms.

6.2.6 Applications

In the future, one potential improvement would be to modify the ensemble training. In order to tackle more real world problems such as those targeted in the SubT Challenge, an expansion of the focus of the model training to focus on mapping as much of the unexplored area as possible. A combination of the current approach presented here, along with this focus on exploration, would likely result in a robot which would be better suited at the combination of exploration of unknown environments along with artifact identification. One such application is that proposed by Herdering et al. [51] in their paper which proposes using DRL to maximize coverage while searching and avoiding unknown obstacles. This would be key for search and rescue operations and other similar scenarios.

References

[1] *AlphaGo: The story so far.* [Online]. Available: `https://deepmind.com/research/case-studies/alphago-the-story-so-far` (visited on 04/24/2021).

[2] *Go, Jack Good.* [Online]. Available: `http://www.chilton-computing.org.uk/acl/literature/reports/p019.htm` (visited on 04/24/2021).

[3] K.-H. Yu, A. L. Beam, and I. S. Kohane, "Artificial intelligence in healthcare," *Nature Biomedical Engineering*, vol. 2, no. 10, pp. 719–731, Oct. 2018, ISSN: 2157-846X. DOI: `10.1038/s41551-018-0305-z`. [Online]. Available: `https://www.nature.com/articles/s41551-018-0305-z`.

[4] *Quantitative Trading - An Introduction For Investors*, Apr. 2018. [Online]. Available: `https://speedtrader.com/introdution-quantitative-trading/` (visited on 04/24/2021).

[5] S. Panesar, Y. Cagle, D. Chander, J. Morey, J. Fernandez-Miranda, and M. Kliot, "Artificial Intelligence and the Future of Surgical Robotics," *Annals of Surgery*, vol. 270, no. 2, pp. 223–226, Aug. 2019, ISSN: 0003-4932. DOI: `10.1097/SLA.0000000000003262`. [Online]. Available: `https://journals.lww.com/annalsofsurgery/fulltext/2019/08000/artificial_intelligence_and_the_future_of_surgical.7.aspx`.

[6] V. S. Bisen, *How AI Can Help In Agriculture: Five Applications and Use Cases*, Mar. 2021. [Online]. Available: `https://medium.com/vsinghbisen/how-ai-can-help-in-agriculture-five-applications-and-use-cases-f09c3dc326c9` (visited on 04/24/2021).

[7] *DARPA Subterranean (SubT) Challenge.* [Online]. Available: `https://www.darpa.mil/program/darpa-subterranean-challenge` (visited on 04/24/2021).

[8] H. Surmann, C. Jestel, R. Marchel, F. Musberg, H. Elhadj, and M. Ardani, "Deep Reinforcement learning for real autonomous mobile robot navigation in indoor environments," *arXiv:2005.13857 [cs]*, May 2020. [Online]. Available: `http://arxiv.org/abs/2005.13857` (visited on 02/18/2021).

[9] K. D. Foote, *A Brief History of Deep Learning*, Feb. 2017. [Online]. Available: https://www.dataversity.net/brief-history-deep-learning/ (visited on 02/14/2021).

[10] K. P. Murphy, *Machine Learning: A Probabilistic Perspective.* MIT Press, Sep. 2012, ISBN: 9780262304320.

[11] K. D. Foote, *A Brief History of Machine Learning*, Mar. 2019. [Online]. Available: https://www.dataversity.net/a-brief-history-of-machine-learning/ (visited on 02/12/2021).

[12] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, Jul. 1959, ISSN: 0018-8646. DOI: 10.1147/rd.33.0210.

[13] J. Dy and C. Brodley, "Feature Selection for Unsupervised Learning," *Journal of Machine Learning Research*, vol. 5, pp. 845–889, Aug. 2004.

[14] M. Lee, *1.5 Summary.* [Online]. Available: http://incompleteideas.net/book/first/ebook/node11.html (visited on 02/15/2021).

[15] B. Osinski and K. Budek, *What is reinforcement learning? the complete guide*, Jul. 2018. [Online]. Available: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/ (visited on 02/28/2021).

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning, second edition: An Introduction.* MIT Press, Nov. 2018, ISBN: 9780262352703.

[17] X. Han, *A Mathematical Introduction to Reinforcement Learning*, 2018. [Online]. Available: /paper/A-Mathematical-Introduction-to-Reinforcement-Han/19fddc8990f1bfee8f64ffe5de5b9c4d41c68ba7 (visited on 04/25/2021).

[18] *Motorcycle Cone Course Layout (Page 1) - Line.17QQ.com.* [Online]. Available: https://line.17qq.com/articles/cuweuwqqx.html (visited on 04/25/2021).

[19] M. Lee, *3.1 The Agent-Environment Interface.* [Online]. Available: http://incompleteideas.net/book/first/ebook/node28.html (visited on 04/25/2021).

[20] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, ISSN: 1476-4687. DOI: 10.1038/nature14539. [Online]. Available: https://www.nature.com/articles/nature14539 (visited on 04/25/2021).

[21] L. Deng and D. Yu, "Deep Learning: Methods and Applications," *Foundations and Trends in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, Jun. 2014, ISSN: 1932-8346. DOI: 10.1561/2000000039. [Online]. Available: https://doi.org/10.1561/2000000039 (visited on 04/07/2021).

[22] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *arXiv:cs/9605103*, Apr. 1996. [Online]. Available: http://arxiv.org/abs/cs/9605103 (visited on 04/07/2021).

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986, ISSN: 1476-4687. DOI: 10.1038/323533a0. [Online]. Available: https://www.nature.com/articles/323533a0.

[24] *A Beginner's Guide to Backpropagation in Neural Networks*. [Online]. Available: http://wiki.pathmind.com/backpropagation (visited on 04/26/2021).

[25] J. Gauci, E. Conti, Y. Liang, K. Virochsiri, Y. He, Z. Kaden, V. Narayanan, X. Ye, Z. Chen, and S. Fujimoto, "Horizon: Facebook's Open Source Applied Reinforcement Learning Platform," *arXiv:1811.00260 [cs, stat]*, Sep. 2019, arXiv: 1811.00260. [Online]. Available: http://arxiv.org/abs/1811.00260 (visited on 05/13/2021).

[26] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *Journal of Big Data*, vol. 2, no. 1, p. 1, Feb. 2015, ISSN: 2196-1115. DOI: 10.1186/s40537-014-0007-7. [Online]. Available: https://doi.org/10.1186/s40537-014-0007-7 (visited on 05/13/2021).

[27] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, Nov. 2017, ISSN: 1558-0792. DOI: 10.1109/MSP.2017.2743240.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv:1312.5602 [cs]*, Dec. 2013. [Online]. Available: `http://arxiv.org/abs/1312.5602` (visited on 04/07/2021).

[29] E. Digor, A. Birk, and A. Nüchter, "Exploration Strategies for a Robot with a Continously Rotating 3D Scanner," Lecture Notes in Computer Science, N. Ando, S. Balakirsky, T. Hemker, M. Reggiani, and O. von Stryk, Eds., pp. 374–386, 2010. DOI: `10.1007/978-3-642-17319-6_35`.

[30] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 610–617, Apr. 2019, ISSN: 2377-3766. DOI: `10.1109/LRA.2019.2891991`.

[31] E. Uslu, F. Çakmak, M. Balcılar, M. F. Amasyalı, and S. Yavuz, "Frontier-based autonomous exploration algorithm implementation," in *2015 23nd Signal Processing and Communications Applications Conference (SIU)*, May 2015, pp. 1313–1316. DOI: `10.1109/SIU.2015.7130081`.

[32] B. Ghimire, J. Rogan, and J. Miller, "Contextual land-cover classification: Incorporating spatial dependence in land-cover classification models using random forests and the Getis statistic," *Remote Sensing Letters*, vol. 1, no. 1, pp. 45–54, Mar. 2010, ISSN: 2150-704X. DOI: `10.1080/01431160903252327`. [Online]. Available: `https://doi.org/10.1080/01431160903252327`.

[33] T. G. Dietterich, "Ensemble Methods in Machine Learning," Lecture Notes in Computer Science, pp. 1–15, 2000. DOI: `10.1007/3-540-45014-9_1`.

[34] L. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, Oct. 1990, ISSN: 1939-3539. DOI: `10.1109/34.58871`.

[35] H. Yang, X.-Y. Liu, S. Zhong, and A. Walid, "Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy," Social Science Research Network, Rochester, NY, SSRN Scholarly Paper ID 3690996, Sep. 2020. [Online]. Available: `https://papers.ssrn.com/abstract=3690996` (visited on 02/24/2021).

[36] *OpenAI Charter*. [Online]. Available: https://openai.com/charter/ (visited on 04/30/2021).

[37] OpenAI, *Gym: A toolkit for developing and comparing reinforcement learning algorithms*. [Online]. Available: https://gym.openai.com (visited on 04/30/2021).

[38] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, *Stable baselines*, https://github.com/hill-a/stable-baselines, 2018.

[39] *Algorithms — Spinning Up documentation*. [Online]. Available: https://spinningup.openai.com/en/latest/user/algorithms.html (visited on 05/01/2021).

[40] M. Wang, *Advantage Actor Critic Tutorial: minA2C*, Jan. 2021. [Online]. Available: https://towardsdatascience.com/advantage-actor-critic-tutorial-mina2c-7a3249962fc8 (visited on 04/17/2021).

[41] S. Karagiannakos, *The idea behind Actor-Critics and how A2C and A3C improve them*, Nov. 2018. [Online]. Available: https://theaisummer.com/Actor_critics/ (visited on 04/17/2021).

[42] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *International Conference on Machine Learning*, PMLR, Jun. 2016, pp. 1928–1937. [Online]. Available: http://proceedings.mlr.press/v48/mniha16.html (visited on 02/18/2021).

[43] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *arXiv:1611.05763 [cs, stat]*, Jan. 2017. [Online]. Available: http://arxiv.org/abs/1611.05763 (visited on 04/16/2021).

[44] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv:1707.06347 [cs]*, Aug. 2017. [Online]. Available: http://arxiv.org/abs/1707.06347 (visited on 02/24/2021).

[45] C. Trivedi, *Proximal Policy Optimization Tutorial (Part 1: Actor-Critic Method)*, Jun. 2020. [Online]. Available: https://towardsdatascience.com/proximal-policy-optimization-tutorial-part-1-actor-critic-method-d53f9afffbf6 (visited on 04/17/2021).

[46] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation," *arXiv:1708.05144 [cs]*, Aug. 2017. [Online]. Available: `http://arxiv.org/abs/1708.05144` (visited on 04/15/2021).

[47] *Openai baselines: ACKTR and A2C*, Aug. 2017. [Online]. Available: `https://openai.com/blog/baselines-acktr-a2c/` (visited on 04/16/2021).

[48] J. Hui, *RL — Actor-Critic using Kronecker-Factored Trust Region (ACKTR) Explained*, Sep. 2018. [Online]. Available: `https://jonathan-hui.medium.com/rl-actor-critic-using-kronecker-factored-trust-region-acktr-explained-670777ec65ce` (visited on 04/18/2021).

[49] *TensorBoard.* [Online]. Available: `https://www.tensorflow.org/tensorboard` (visited on 05/08/2021).

[50] S. Zychlinski, *The Complete Reinforcement Learning Dictionary*, Nov. 2019. [Online]. Available: `https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e` (visited on 05/09/2021).

[51] A. Herdering, H. Quintero, S. Centeno, M. Beylik, and J. Isaacs, "Deep Reinforcement Learning for Autonomous Search," in *Proceedings of the 2021 Computer Science Conference for CSU Undergraduates*, Mar. 2021.