Zurich University
of Applied Sciences

School of
Engineering
InIT Institute of Applied
Information Technology

MSE Master's Thesis

Entropy-Aware Active Vision through Voxelized Octree Exploration of 3D Scenes

Autor	Juan F. Ribera Laszkowski
Betreuung	Prof. Dr. Thilo Stadelmann
Nebenbetreuung	Dr. Giovanni Toffetti Carughi
Industriepartner	Sutter Landtechnik GmbH
Datum	14.02.2022

DECLARATION CONCERNING THE INDEPENDENT WRITING OF A MASTER THESIS AT THE SCHOOL OF ENGINEERING

By submitting this Master's thesis, the student assures that he/she has written the thesis independently and without outside help. (In the case of group work, the performance of the other group members does not count as outside help).

The undersigned student declares that all cited sources (including Internet pages) in the text or appendix are correctly accounted for, i.e. that the master's thesis does not contain any plagiarism, i.e. no parts that have been taken over in part or in full from another's text or work under pretence of one's own authorship or without citation of the source.

In the event of misconduct of any kind, Sections 39 and 40 (Dishonesty and Procedure in the Event of Dishonesty) of the ZHAW Examination Regulations and the provisions of the Disciplinary Measures of the University Regulations shall come into force.

Date, Signature **Juan Ribera**

ABSTRACT

The large and still increasing popularity of deep learning, along with the growing availability of 3D labeled datasets has been setting the stage to the continuous development of new algorithms in scene understanding and autonomous systems. Where traditional approaches to navigation depend on task subdivisions and map awareness, newer approaches take on the problem of navigation from the perspective of model-free solutions with partial observability, given that challenging real life scenarios, such as rescue missions and dynamic scenes, are unknown environments.

In this thesis, we propose an embodied agent to tackle the problem of object exploration given partial observability in an unknown environment by exploiting octrees for the efficient navigation of 3D scenes. Furthermore, given the necessity to incorporate the temporal dimension in visual object recognition tasks, we propose extrinsic rewards for the scanning of voxelized objects with a reinforcement learning agent in order to cover various trajectories around an object of interest, therefore reducing the uncertainty of such objects and their characteristics. We also take into account the level of entropy in a simulated Unity environment to adjust the behavior of our agent on-the-fly, improving on the exploratory performance of current methods in active vision. Our results outperform our Unity implementations of previous classical and geometric approaches and improve upon current state-of-the-art exploration methods that are motivated by coverage maximization and semantic curiosity. We achieve better exploratory performance by at least a factor of two in the scanning of objects and cover 8% more of the environment in comparison to our baselines.

Furthermore, by using octrees, voxels and panoramic vision, our method is able to adapt to new environments without changes in its behavior or fine-tuning, bridging the gap between synthetic data and real data for the exploration of 3D environments, where the data distribution plays a key role given the commonly seen noise, outliers and missing data in RGB-D sensors. Finally, given the increasing amount of algorithms, newer and extensible benchmarks are needed for testing more sophisticated challenges that are representative of the real world. Motivated by recent works in the Unity 3D engine and the ML-Agents plugin, we demonstrate the applicability our approach in three 3D environments, test its performance in two environments inspired by the DARPA Subterranean Challenge and aim to further motivate the creation of new benchmarks, custom-made testing environments, exploration methods, and the reproducibility and extensibility of research results.

Our approach aims to serve as a baseline for computer vision methods that incorporate the temporal dimension for increased certainty about objects, in tasks such as synthetic data generation, rescue missions, autonomous driving, exploration and navigation, point-to-goal tasks, etc. The code base for our method can be found at https://github.com/Ademord/ma-unity. All Unity 3D Assets are protected by copyright.

PREFACE

I would like to give special thanks to my supervisors Thilo Stadelmann and Giovanni Toffeti for their continuous support during this Thesis work, for their great feedback and constructive discussions. I am very grateful to have been given the opportunity to explore and contribute to the ongoing research topics of reinforcement learning, computer vision and cloud robotics.

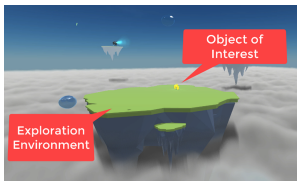
I would also like to thank Rickard Eriksonn, Devendra Singh Chaplot and Aakarshan Chauhan, as well as my friends and family for their constant support and contributions during the development of this thesis.

Contents

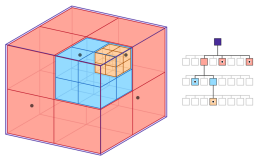
1

INTRODUCTION

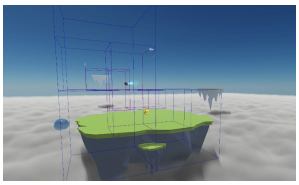
(a) 3D Explorer Drone [1].



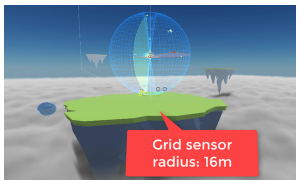
(b) Training Island.



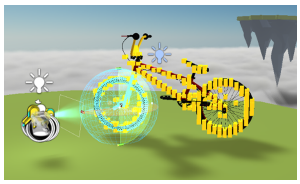
(c) Octree data structure [2].



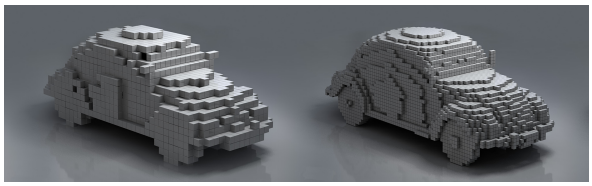
(d) Octree island subdivision.



(e) Grid sensor for vision.



(f) Object of interest and voxel scanner.



(g) Voxelization used to motivate exploration of objects [3].

1.1. MOTIVATION

At the NeurIPS 2017, Pieter Abbeel [4] presented a video of robots cleaning a living room, bringing a bottle of beer, and performing tasks usually seen in science fiction movies. Later it was showed that the robot was being controlled by a human with a remote controller: his presentation hinted towards the fact that the robots that we use today are physically capable of performing such actions, so building state of the art robotics is not a hardware problem anymore but a software one. As of 2021, tasks like teaching a robot how to pick a bottle of beer can still be a very challenging task. Nevertheless, current algorithms have come a long way since Pieter's talk and much has been done to improve robots' understanding of their environment and to study what set of tools are required to achieve not just perception in robots but intelligent agents [5, 6]. Works like DeepMind's recent publication [7] hint towards a promising future where reward maximization methods are a promising path towards general artificial intelligence. Where current AI models outperform humans at specific tasks and alleviate repetitive workloads, general AI would potentially outperform humans at nearly every cognitive task. It is therefore imperative to continue research in disciplines such as reinforcement learning as it is one of the most promising field to achieve intelligent robotic behavior [8, 4, 5, 7].

Accordingly, a current project at ZHAW linked to this thesis work aims to construct a milking robot that leverages machine learning methods to outperform traditional milking robot technologies, in cooperation with the industrial partner Sutter Landtechnik GmbH (SLG). Most of today's milking robots use a 2D laser scanner to estimate the position of the cow, the udder and the cow's teats, and several measurements are necessary. This is time consuming and if the cow moves during the measurement, the position estimation must be reinitialized. ZHAW therefore proposes innovative changes to the milking robot's architecture, using

electric drives and a more compact kinematic structure. Moreover, given that inexpensive 3D cameras have been introduced to the market over the past few years, it is now possible to leverage high resolution 3D point clouds to dynamically estimate the position of the cow teats while also reducing the costs of manufacture.

This thesis aims to contribute to this project but also to the fields of practical reinforcement learning, computer vision research and practical machine learning. To this end, we abstract the scenario of a milking robot as an unknown environment with objects of interest, a reinforcement learning agent's task is to explore such an environment and the objects in it. In other words, the aim of this masters thesis is to select and implement a computer vision enabled pipeline that leverages active vision to reduce the uncertainty over time in a new environment, one of which can be a milking robot's environment. In this sense, uncertainty is defined as the amount of unscanned voxels in the scene, the unexplored locations and the measured temporal semantic entropy given by an object detector.

The proposed solution is inspired by the human brain's physiology and approximates the discovery process a human would follow in an unknown environment. It also leverages the concept of temporal semantic entropy proposed by Chaplot et al. [9] to define this dimension of uncertainty. Concretely, the task at hand is exploration for coverage maximization, for which we propose a novel approach that leverages extrinsic rewards from voxel scans constructed from ubiquitous RGB-D information. From the perspective of architecture design, it proposes a pipeline that can be further extended to a multitude of use cases and even other semantic models, in an attempt to contribute to the research on semantic-curiosity-motivated exploration.

Finally, while there are increasingly more 3D datasets out there for benchmarks on computer vision tasks, reinforcement learning algorithms have seen themselves limited to black box environments and popular games such as Atari 2600, Quake III, Doom, Minecraft and Gazebo simulations. However, as deep reinforcement learning algorithms become more sophisticated, new environments and benchmarks must emerge since existing environments and benchmarks based on them become less informative [10]. This work contributes to existing agent testing environments with three 3D Unity-based scenarios, which are easy to extend to further use cases. They also differentiate themselves from traditional benchmarks by being developed with improved realism, which aims to close the gap between synthetic and realistic data. These environments are used to further evaluate the exploration task at hand and the feasibility of knowledge transfer across environments. All of this leaves the door open for future research in simulated applied vision systems and for in-robot implementations of the exploration policies studied.

1.2. PURPOSE AND RESEARCH QUESTIONS

In this work, the problem of uncertainty in an environment presented by the milking robot is looked from a more general perspective. We therefore look at exploration policies that can also contribute to a plethora of other use cases. More concretely, we propose a reinforcement learning based solution to explore a 3D scene and the objects in it. To this end, we use extrinsic motivation while also leveraging the uncertainty in the environment to adapt such behavior. Finally, a set of research scenarios are proposed to show the versatility of our exploration policies and contribute to existing benchmark environments. Regarding this task, this work tackles the following research questions:

- How can an embodied agent increase the overall certainty about an object's characteristics, i.e., how can trajectories around objects of interest be covered to reduce the uncertainty about such objects?
- How can these objects be found in large and unknown environments by the same agent?

1.3. APPROACH AND METHODOLOGY

This work focuses on leveraging reinforcement learning techniques to manipulate an agent (camera) motivated by abstractions of the perceived environment. These abstractions are meant to simplify the complexity present in the environment and take the form of translations of camera input into voxels. Voxels, accordingly, represent the direct goal for a scanning agent to obtain view from an object from multiple angles. The problem that is further studied within this master's thesis is two-fold: 1) uniform exploration of large open spaces and 2) thorough scanning of objects from multiple angles, as a mean to reduce the uncertainty in a 3D scene over time. We define, therefore, uncertainty through the semantic entropy observed at a given point in time, which is inspired by Chaplot et al. [9].

The strategy taken for this project work is the following:

A state of the art analysis was done to propose a set of exploration policies using reinforcement learning that answer the research questions. First, this analysis includes the evaluation of related efforts in scene understanding, active vision, and RL-based exploration. Second, it includes implementations of exploration policies and the respective optimization of the learning environment and the learning agent and its algorithm to solve the task optimally. It is also important to note that this process has an iterative nature, where the validation of the models' results triggers changes for optimizations in the learning environment and the agent, in order to achieve the best possible results. The used data is of synthetic origin, and was modeled, assembled and simulated in the Unity game engine. Finally, the results were critically evaluated and compared to a set of baselines. The generalization capabilities, reliability and limits of the results are then discussed and further improvements on this field considered.

1.4. CONTRIBUTIONS

Our contributions can be summarized into the following points:

- We build upon knowledge acquired through a previous publication, namely *"Teat Pose Estimation via RGBD Segmentation for Automated Milking"* [11] and the Bachelor's thesis Active Scene Understanding from Image Sequences for Next-Generation Computer Vision [12], to propose a novel voxel-based exploration method, which builds upon model-free reinforcement learning and accounts for the abstract concept of "uncertainty" in an environment, defined through the amount of different object classes present in such an environment. This definition is known as "semantic entropy, or as "the inconsistencies in the temporal class density", as inspired by the concepts presented by Chaplot et al. [9].
- We demonstrate that our method outperforms our Unity implementations of classical and geometric exploration for scanning strategies, and further improves upon exploration methods motivated by coverage maximization [13] using octrees and adapts its exploration pace through semantic entropy.
- We show that Proximal Policy Optimization (PPO), with minimal hyperparameter tuning and any domain-specific algorithmic changes or architectures, achieves final performances and converges faster than the Soft Actor-Critic algorithm [14].
- By using voxels and grid sensors for extensive exploration, our method abstracts the complexity in the 3D world through voxels and therefore is capable of visual-agnostic exploration and transfer learning tasks.
- We evaluate our approach through relevant statistics for exploration and for thorough scanning of objects, where a set of baselines were defined and two DARPA-inspired evaluation environments [15] were implemented.
- We further demonstrate that our method excels in new 3D scenarios that represent real life situations where exploration time is critical to save not just time and costs but, more importantly, lives and further losses.
- We present the capabilities of Unity-based environments for the development of new benchmarks and its compatibility with the OpenAI gym platform [16] and trainers from Stable Baselines 3 [17].
- Finally, we demonstrate that an agent equipped with panoramic vision outperforms agents with ordinary monocular vision in the given exploration task, inspired by Rill and Faragó [18] and Wojek et al. [19].

1.5. SCOPE AND LIMITATION

Due to the time constraints, some limitations were laid out to ensure the work was finished within schedule.

- The approach used focuses on the manipulation of synthetic data, namely, 3D voxels in the Unity Engine, while the manipulation of point clouds to generate these voxels is left out. This allows the masters thesis to focus on the research on exploration policies for scanning of objects than on computer vision methods for point cloud manipulation.

- This work focuses on the results observed in computer vision and machine learning approaches limited to models compatible with the Unity Engine (ONNX implementations). Other techniques, which for example, use lasers for 3D scene understanding, are not taken into account.
- Methods mentioned in 2.1 sample data from diverse exploration trajectories to then train and compare a semantic detector's performance. They then indirectly evaluate the performance of each exploration policy. This work does not take into account supervised nor unsupervised training of semantic vision networks (segmentation, recognition) given the time constraint.
- This work focuses on scene understanding and camera manipulation techniques using reinforcement learning in the game engine Unity 3D.

1.6. TARGET GROUP

One one hand, this work is of special interest for researchers in the field of computer vision, reinforcement learning, and Unity using ML-Agents. This is due to the fact that 3D understanding and environment discovery is still a rapidly growing field. On the other hand, as demonstrated in the practical application of the proposed method in Chapter 5, a broad audience in the industry is the public interest for this work, such as security and rescue offices, warehouses, factories, where dedicated the scanning of objects for quality assurance or cues in a traffic accident, fires or other unexplored scenes is of special priority. This can also be extended to other modules such as data collection pipelines, semantic modules, airport robots, hospital robots, etc.

1.7. OUTLINE

The following Chapter 2 covers the Foundations for this work, which starts with the related efforts in the fields that this work touches upon. Then it presents the related concepts from octrees, scene understanding, 3D environments and reinforcement learning. Accordingly, concepts related to computer vision, such as history, 3D object recognition, point cloud segmentation, etc., are pushed to the Appendix, to concentrate in the immediately necessary content. Chapter 3 describes the methodology, the nature of the data and the reinforcement learning agent's architecture. Chapter 4 presents the achieved results, and chapter 5 examines the validity and reliability of the presented results. Finally, chapter 6 provides a conclusion and proposes further steps on research for this topic.

2

FOUNDATIONS

This chapter introduces the related efforts and theory required to understand the proposed solution to the research questions. Section 2.1 describes the related works to this thesis in fields such as robotic navigation and exploration, 3D vision, active vision, etc. Afterwards, Section 2.2 introduces the concept of voxels as 3D data structures that reduce complexity in a scene, while Section 2.3 covers the efficiency provided through octrees in the manipulation of 3D volumes. The limitations of current 3D scene understanding techniques are presented in Section 2.4 and used for motivation in Section 2.5, which goes over the promise and exploitation of 3D environments and synthetic data using game engines and simulation environments. Finally, section 2.6 introduces the basics of reinforcement learning, Q-learning, how function approximators allow agents to learn more sophisticated environments and wraps up the theoretical background with proximal policy optimization.

2.1. RELATED WORK

We study the problem of how to maximize simulated robotic exploration to eventually find objects and scan them thoroughly. This refers to techniques on exploration and coverage in embodied contexts. Embodied contexts are those that take into account an agent's physical and cognitive abilities. This is slightly related to our previous work on milking robots, where their limitations serve as out initial motivation. Accordingly, it is tightly related to 3D vision (what is currently possible to understand from images), active SLAM in robotics (what are navigation policies and perception methods) and intrinsic motivation (picking what sections of an environment to explore). Related efforts are surveyed below.

Milking Robots. Given that current laser technology used in cow milking robots limits the performance of the systems, a considerable technological growth is possible by leveraging the latest technological advancements. As stated by Pal et al. [20], laser technology is not capable of differentiating between a cow's teat and a leg, therefore manipulating the suction cups in the wrong direction. Along these lines, Pal et al. contribute by proposing a fast and reliable solution to the problem of 3D pose recognition of cow teats using TOF [21], RGB-D [22] and Thermal Imaging [23]. However, even though their pipeline provides a new accurate way for estimating the teats' poses, it lacks any intelligence with respect to knowing what is a cow teat (semantics). Rastogi and Ryuh [24] take a similar stand against the limitations of laser assisted edge detection technologies, where current solutions cannot differentiate between a healthy and a diseased teat. To improve on this, they propose two functional yet limited alternatives to the task: a Haar-cascade classifier and a YOLO classifier for cow teats, approaches which work on real time but lack reliable accuracy. The Haar cascade classifier fails to detect any teats in the occluded tests, and YOLO fails to detect a fourth teat, even with a prediction threshold of 0.5. In more general terms, O'Mahony et al. [25] review 3D computer vision systems and techniques for precision dairy farming. More specifically, by looking at Time of Flight and streoscopic vision systems and conclude that robust systems which adapt to weather conditions, herd characteristics, farmyard layout, etc., (generally unknown scenarios) are required. Hence they foresee that the future state of the art technologies use Geometric Deep Learning for processing data in non-Eucledian domain, such as graphs and manifolds. On this note, Cao et al. [26] provide a comprehensive review of deep learning methods in the graph and manifold domain, including history, background, applications and

benchmark datasets as reference for research in geometric deep learning. First, this thesis' work aims to provide a solution to the milking robot uninformativeness problem, by allowing exploration of the environment around the cow's udder, so that hidden teats are not ignored. Second, it aims to also contribute to other fields such as surveillance and discovery, such as car accidents in police investigations, inspection drones from fire departments, etc.

3D Vision. Given the technological lacklustre in current solutions implemented in milking robots, it is imperative to look into current state of the art of 3D vision, in order to bridge the gap between research and applications. Several works have been grabbing inspiration from biological vision and the brain's physiology by separating the "what" from the "where" [27, 28]. Others include the concept of information over time to their proposals to not only study the behavior of familiarity given seen objects in the brain, but also how the temporal dimension can improve action recognition and allow accurate object classification [29, 30, 31, 32]. Accordingly, the interpretation of the information present in our environments, whether through RGBD data, sparse point clouds, etc., specially in the presence of heavy occlusion, and the generation of semantic features from it is deeply studied in many works, including for practical setups such as industrial inventories [33, 34, 35, 36].

In the field of robotics, Lin et al. [37] provide a segmentation-based architecture that leverages the concept of primitive shapes to reduce object shapes to known grasps based on these simpler shapes. Inspired by two-stream hypothesis of visual reasoning, Jang et al. [38] present a semantic grasping framework that learns object detection, classification and grasp-planning in an end to end fashion. Their ventral stream recognizes the objects class and the dorsal stream interprets the geometric relationships necessary to execute successful grasps. Similarly, Cheng, Agarwal, and Fragkiadaki [39] propose an architecture using active vision for manipulating objects under occlusions. More specifically, they use artificial agents to learn gripper and camera control policies using reinforcement learning in the presence of occlusions. They propose hand-eye controllers that learn how to move the camera to keep the object within the field of view. In lines with curriculum learning research, they show evidence that agents for both policies and object detection provide better performance when initially trained in an environment without occlusions. Another work worth mentioning is the approach proposed by Levine et al. [40], which learns successful eye-hand coordination for robotic grasping on novel objects using CNNs and doesn't require camera calibration. They use a continous feedback loop to correct mistakes and suggest to use reinforcement learning as future work to learn a wider variety of grasp strategies.

These works provide useful insight into camera intrinsics and extrinsics, robotic movement mechanics and limitations of each approach. However, pre-processing and manipulation of high-level RGBD data continue to be memory expensive and computationally complex. This thesis' work proposes an efficient voxel-based approach that reduces the complexity in the environments and allows exploration and understanding of 3D environments, which to the best of our knowledge hasn't been exploited before for tasks aimed at exploration of environments and objects.

Navigation in Classical Robots. As described extensively by Chen, Gupta, and Gupta [13], classical robotic tasks attempt to build a map of their environments and then apply a path planning algorithm to reach some destination. Work in this direction has been extensively studied [41, 42, 43]. The research mentioned, however, follows mostly a passive SLAM approach, whereas an active SLAM approach to discover an environment is less studied. Chen summarizes key efforts in active SLAM, contributing to Cadena's [44] extensive literature: active SLAM has been formulated as a partially observable Markov decision process (POMDP) [45] or as choosing actions that reduce uncertainty in the environment's mapping [46]. These methods rely on sensors for their measurements and are highly affected by noise and view the exploration problem as a geometry problem, ignoring the semantics the environment could provide (e.g. doors). Chen, Gupta, and Gupta [13] contribute to this research by proposing a learning-based approach, investigating different policy architectures, reward functions and training paradigms, outperforming classical geometry-based approaches and generic learning-based exploration techniques.

Exploration for Navigation. Orthogonal works [47, 48] to Chen, Gupta, and Gupta [13]'s learn policies on how to avoid collision and prefer open space than maximizing environment coverage without leveraging from their learned maps, even mimicking SLAM techniques [49]. Similarly, many works study reinforcement-learning-based exploration [50, 51, 52, 53, 54, 55], proposing intrinsic reward functions that prefer novel states. This work proposes a variety of exploration agent variants using extrinsic rewards and analyzes the influence of intrinsic curiosity [52]. Similar to Chen's work, this thesis studies how learning-based techniques

can be exploited in real world deployments.

Other related efforts use [56] reward function based on pixel reconstruction to learn how to explore and how to solve tasks. Bai et al. [57] study Gaussian Process regression in an information-theoretic exploration method tested on simplistic environments. Kollar and Roy [58] learn a trajectory that maximizes the accuracy of the SLAM-derived map when compared to the assumed ground-truth map. In contrast, this thesis' learning policy mandates in real-time the action the agent should take when balancing two interests: using a 3D map to prefer open undiscovered spaces and exploring voxels in salient objects.

Active Vision. The topic of active vision has been actively studied in the fields of services medicine industry and agriculture, and it involves a robot which analyzes its environment and its own state to collect multiple views to acquire more information about the environment itself [59]. An alternative term used by Chaplot is *active perception* [60], which refers to "the problem of actively moving the sensors around at test time to improve performance on the task by gaining more information about the environment", which has been applied in a variety of applications such as object detection [61], amodal object detection [62], scene completion [56], and localization [63, 64].

In other words, visual tasks like object recognition and reconstruction, scene exploration, target tracking, etc., can benefit substantially from the proposal of active vision approaches for "more information given more views". In theory, the promise of more information to reduce the uncertainty about our environment and the characteristics of the objects in it could provide great value, but in practice this is not so easily implemented [59].

The most important factors that discourage these model-based approaches are the complexity of real-world models (which are biased in some way) [65], imprecision of motion mechanics, unreliability of visual perception, general uncertainty about the environment and the changing task requirements [66, 67]. For more information on active vision, please refer to Zeng et al. [59]'s survey.

In this work, we focus on the following: general uncertainty about the environment, unreliability of visual perception, a biased model of reality that limits the dynamics present in such environment and the changing task requirements. Firstly, we tackle the problem of general uncertainty in an environment through octree-based exploration. Secondly, we avoid unreliable visual detectors by reducing the uncertainty about objects in an environment through perception voxelized structures. Third, motivated by Chaplot's extensive work [68, 69, 70, 71, 9], we achieve these agent behaviors through a model-free approach using a self-supervised policy and does not rely on end-task reward [14, 72], in contrast to other approaches [63, 56, 62]. Fourth, our baseline behavior policy for the exploration of environments and objects is applicable to a multitude of tasks and further scenarios, from rescue missions or danger assessment to product quality assurance. Finally, we attack the hidden disease of reproducible research by exploiting Unity 3D simulated environments for modeling, training, testing, benchmarking and further use case evaluations.

Intrinsic Rewards. This thesis' work is tightly related to exploration using reinforcement learning [73, 74, 50, 75, 8]. The mentioned works set up the problem as a Markov Decision Process which attempts to learn high reward paths, by motivating intrinsic rewards which direct the agents toward previously unseen [76] or unknown spaces [52] in the environment. This thesis' work not only motivates the agent to visit unknown spaces but also rewards it for scanning voxels in the environment. This takes a different direction from the related work since it does not utilise semantics to define objects of interest but voxels as a less complex form of the inherent structure of objects. Moreover, in contrast to the work by [9], the focus lies in exhaustive exploration as a baseline trajectory discovery, where the improvement of semantic models has a secondary, tangential purpose.

Visual Navigation and Exploration. According to Chaplot et al. [9], visual navigation works can be separated into two categories depending on whether the location of the goal is known or not. Navigation problems where the destination point is given, include the point-goal task [77, 78], or the vision-language [79] task where the destination is given in natural language. These tasks do not require extensive exploration since the destination is given a priori as coordinates (explicit destination) or as a path (implicit destination).

In contrast, navigation problems where the location of the goal is not given, include a variety of methods [9]. They include: navigating to a fixed set of objects [68, 80, 77, 81, 82, 83], navigating to an object specified by language [69, 84] or by an image [71, 85], and navigating to a set of objects in order to answer a question [86, 87]. These methods must be able to explore the environment extensively to find the goals. However, some

methods do not really solve the exploration problem if the agent is spawned close to the destination goal. They instead focus instead on other challenges; for example, models for First Person Shooter (FPS) video games [68, 80, 81, 83] are able to train reactive agents to avoid obstacles and enemies, collective sporadic rewards scattered across the environment. Accordingly, other models focus on recognizing the goal (visual perception) [71, 85], understanding the goal through language [69, 84] or understanding the visual properties which make up the goal [86, 87]. Reinforcement-learning is successfully applied in these challenges but exhaustive exploration, especially in large environments, remains to be studied with more depth.

One of the most relevant related the efforts for this thesis is the work done by Parrot [88] on their newest flying drone *ANAFI Ai* in 2021. The latest version of the ANAFI Ai is capable of autonomous flight and obstacle avoidance, with an improved dual vision system for added stabilization and increasedaccuracy of the perceived depth maps. They achieve navigation and obstacle avoidance by mapping depth maps into a occupancy grid, therefore simplifying environment around the drone to navigate. Parrot also enables the possibility for developers to write custom flight missions. This thesis' work goes in a similar direction, where we voxelize the agent's perceived environment. However, our method aims to maximize the explored area [70, 13, 89] using the octree nodes as a part of the reward signal and further exploits the benefit of voxels and semantic entropy to model a visual agnostic behavior to reduce the uncertainty about objects. Furthermore, our method proves its applicability to a multitude of real world scenarios, using realistic 3D scenarios in Unity as proof of concept.

2.2. FROM PIXELS TO VOXELS

Images are the ubiquitous source of information for visual tasks. They are pictures that are stored in electronic form, more specifically, matrices where each cell is called a pixel. Each cell in the matrix has a value: for black and white images this value falls in the range from 0 to 255 and in color images the cell is extended to hold three values instead of one, making it capable of storing the color a representation based on a given color schemes. There are many color schemes but red, green, blue RGB is a widely used and can be combined to represent any color. In other words, RGB images are pictures which are stored electronically as three-dimensional matrices following a given RGB color scheme.

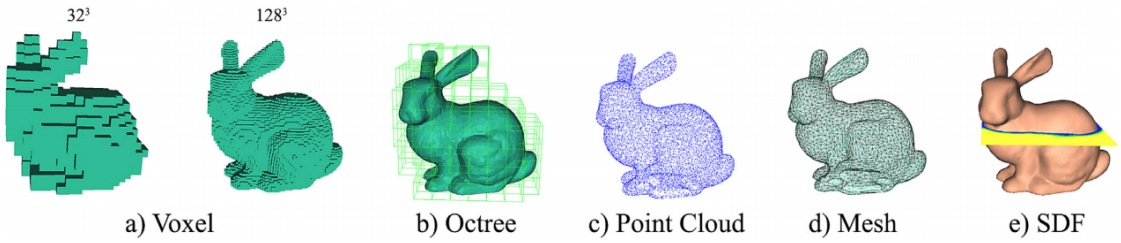


Figure 2.1: The Stanford Bunny in different 3D representations [90] .

Moreover, RGB-D images contribute to this representation by storing a fourth channel, i.e. a depth map, alongside the 2D color information (RGB). Depth is the distance from the camera to the surface of objects in an image and it can be measured, for example, using time-of-flight sensors. These sensors measure the time in the signal from when the sensor was emitted to when the signal returned to the sensor, after being reflected by an object's surface. The most common types of signals used in these sensors are sound and light. Infrared light is usually preferred since it guarantees less noise and allows distinction from ambient light [21]. In general, these devices have a limited depth of field, which limits the information captured from surfaces that are too close or too far away from the camera. Research on 3D information recovery from a scene has been extensively studied [90] for it is important for a wide range of applications, such as autonomous driving, industrial imaging, etc. Similarly, the amount of RGB-D datasets available online is massive in comparison to 3D datasets of labeled point clouds or meshes [91]. RGB-D image representations are therefore known as "2.5D" data [92], since they are not actual 3D representations of objects and spaces. Accordingly, Representations of 3D data can be categorized into Euclidean and non-Euclidean (geometric) representations. Eucledian representations include voxels and octrees, and non-Eucledian representations include point clouds and meshes. Fig 2.1 displays some of the possible representations of 3D data.

Figure 2.2: Voxelized representation of Minas Tirith in Minecraft [93].
A multitude of voxels can also represent high-resolution data, at a high computational memory cost (image below).

This work focuses mainly on endeavors that work with voxel-based representations of 3D data. Similar to how 2D data can be represented in a 2D matrix, 3D data can be represented as a regular grid in the three-dimensional space [92], where each cell or region contains information about the objects that are inside of it, if it is occupied or not, etc. These regions in a 3D grid are called voxels. Figure 2.2 visualizes a video game called Minecraft, where the low resolution can be seen as voxels, even though the rendering engine used is not voxel-based. The main disadvantage of using voxels in high-resolution data is the amount of unnecessary memory storage required. However, one can leverage sparse voxel octrees (SVO) to down-sampling 3D scenes and navigate a simplified version of the environments. Accordingly, this thesis work leverages SVOs and low-resolution voxels (occupancy grids) to explore 3D simplified environments and and scan objects. The motivation for this is two-fold: first, as described by [94], SVOs provide a huge reduction of memory requirements for most scenes, since empty space is not explicitly stored. Second, voxels describe the 3D structure of objects in a low-resolution fashion, and they can be used as extrinsic rewards to motivate a learning agent to explore thorough trajectories around objects. These trajectories then serve the purpose of exploring objects from multiple angles. For further discussion on different representations of 3D data, one can refer to these outstanding surveys [92, 95].

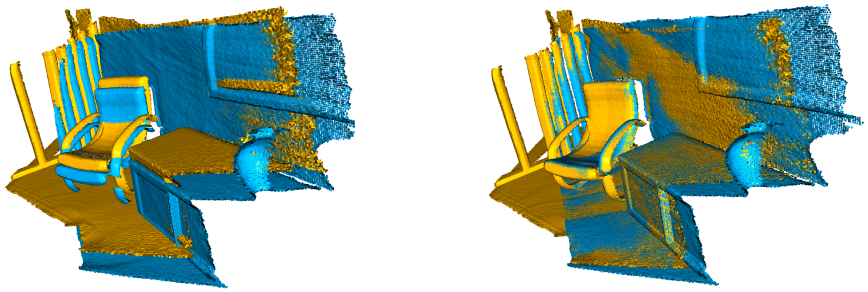


Figure 2.3: Two input point clouds (left). Result after running the ICP Algorithm until convergence (right) [96].

Given an RGBD image, the process of generating a voxel grid from a triangle mesh is called voxelization [97]. A voxel contains per default a "0" in the space it encloses. A "1" is assigned if the voxel is intersected with a triangle. However, one must obtain a point cloud beforehand from the RGBD data sent by a camera. Many works [98, 99] occupy themselves with point clouds from RGBD data, point cloud registration and the problem of outlier filtering when matching the RGB channel and the depth channel. Outlier filtering is required given that the alignment between the two channels can suffer from varying lighting conditions, reflective characteristics and particular sensor attributes. The further alignment in point cloud registration must also take into account a global world model when merging two point clouds together. Figure 2.3 displays the ICP algorithm [96] for matching two point clouds together. Ultimately, a point cloud can then be voxelized or down-sampled to voxels for further processing [100].

2.3. OCTREES

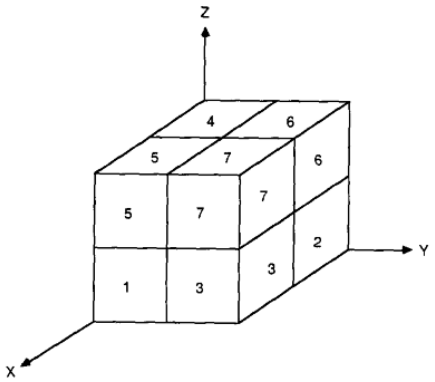


Figure 2.4: Labels of the octants of a cube (octant 0 is in the back and is not shown) [101].

The octree encoding uses a hierarchical tree data structure, where analog to quadtrees, each node can have up to eight children [102]. Octrees are used to model 3D spaces, since they allow the representation of a cubical region as shown in Figure 2.4. Whereas the root node represents the entire space, the rest of the octree is populated by the recursive subdivision of the 3D space into eight octants as shown in Figure 2.5. Once the maximum resolution has been reached, nodes represent the leaves of the octree. Leaves in a node that have the same attributes can be reduced to a node by a process called condensation. Consequently, leaf nodes can be "point nodes" or "empty nodes", and a node that contains leaves is called a "region node".

Information represented in leaves can be boolean or non-boolean if one decides to store more information in the tree such as temperature, etc. Condensation, however, is not a trivial process in non-boolean leaves. Finally, the implementation of octrees can be pointer-based or array-based. The pointer-based approach references child nodes as they are occupied by an object and are memory efficient. Array-based octrees define a priori all possible children in the hierarchy of the octree, making addressing of any node possible at the cost of memory.

The resolution of the 3D space represented through an octree can be given as 2^n, and it is the octree's length space. The resulting volume enclosed by the octree consists of then $2^n \times 2^n \times 2^n$ octants.

It is worth clarifying that the difference between a voxel grid and an octree is that a voxel grid is a fixed-resolution representation of 3D data, whereas the octree is multiresolution. When using in a rendering engine, octree rendering methods can load areas with higher resolution as one navigates the environment while keeping the other areas unloaded and preserving memory efficiency. A voxelgrid, in contrast, would always keep the same fixed resolution as one zooms in and out a render. Figure 2.5 aims to visualize the differences between an octree and a voxelgrid. For information on the complex manipulation of octrees and implementations of the search and insertion operations, please refer to OpenGenus IQ [104] and Geeks for Geeks [105].

Applications of Octrees. Octree's main application is using it as an acceleration structure given that memory and inference times increase cubically as the size of the 3D data increases. First, they excell as sparse data structures where empty regions consume little to no memory [106]. Second, in situations where meshes

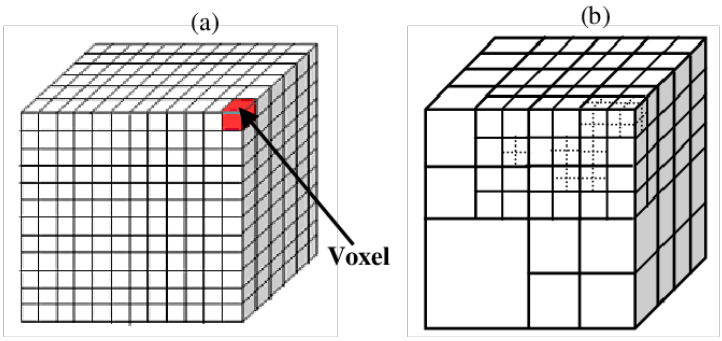


Figure 2.5: (a) Voxel grid (b) Octree representation [103].

are too big for the available hardware, they allow efficient scaling through loading of 3D models at lower resolutions. Third, their efficient $O(log_2 n)$ search queries when verifying the visibility of a cube make them an efficient data structure for high-quality ray casting of static scenes [107].

Concretely, as proven by Elseberg, Borrmann, and Nüchter [108], "octrees are well suited for storing large point cloud data, as it is a lossless compression, which reduces the size of a point cloud by a factor of roughly two and comes with a fast indexing". It is important to repeat and highlight that *lossless compression* through octrees can be accelerated by a factor of two. To further support our claim of using octrees for exploration with a numeric example: one of the three datasets used by Elseberg, Borrmann, and Nüchter [108], the *Bunker Valentin* point cloud, has a file size of 665.7 MB in .txt format. Using octrees with lossless compression the file size goes down to 172.1 MB. Furthermore, using a leaf size node of 5.45 cm, the file size goes down to 14.93 MB.

The efficient representation of high-resolution 3D models using octrees in comparison to polygon-based methods has been further studied in a multitude of other works such as fast collision detection in models of over 25M polygons [109], solar radiation [110], voxelized full-waveform airborne LIDAR data [111], volume rendering methods [112], adaptive resolution SLAM [113], etc. In terms of limitations, octree-based techniques are not straight-forward since octrees are difficult to implement and manipulate. Moreover, operations on octrees in dynamic scenes where nodes need to be continuously recalculated are not efficient. This thesis limits itself to the usage of sparse boolean octrees as a mapping structure for a 3D scene where newly discovered nodes provide an intrinsic reward to the agent. Hence, the limitations of octrees are disregarded.

2.4. SCENE UNDERSTANDING

In their 2020 Hype Cycle for Artificial Intelligence, Gartner projects smart robots to be at their peak of expectations in about 2 to 5 years. This projection goes hand-in-hand with another projection in the same hype cycle. Gartner projects that computer vision will be on the slope of enlightenment around the same period. This is where the experts predict computer vision will enable applications to exploit another input stream: vision. The same way that computers and embedded systems are able to interpret audio and text input from sensors, keyboards, microphones, etc., systems will have access to vision to perform a variety of tasks. As Singh et al. [114] describes, "the ability for robots and computers to see and understand the environment is becoming a burgeoning field, needed in autonomous vehicles, augmented reality, drones, facial recognition, and robotic helpers". Since the rise of the CNN [115] deep learning based methods for image classification have reached state of the art performance for 2D detection.

The visible growth over the past few years of 3D sensors in our day-to-day surroundings (LIDAR, depth cameras, radar) comes hand-in-hand with the requirement for efficient and accurate machine-learning-powered scene understanding methods that can process the data coming from these sensors. Suitable example applications are self-driving cars and robots where they need a sense of embodiment to interpret their environments and navigate around them or augmented reality experiences. Consequently, even though research in computer vision has been recently pushing the state-of-the-art techniques in 3D object detection, much remains to be done to improve the workflow and research of techniques using 3D data. Libraries like Tensorflow3D [116], Pytorch3D [117], Torch-Points3D [118], Pytorch

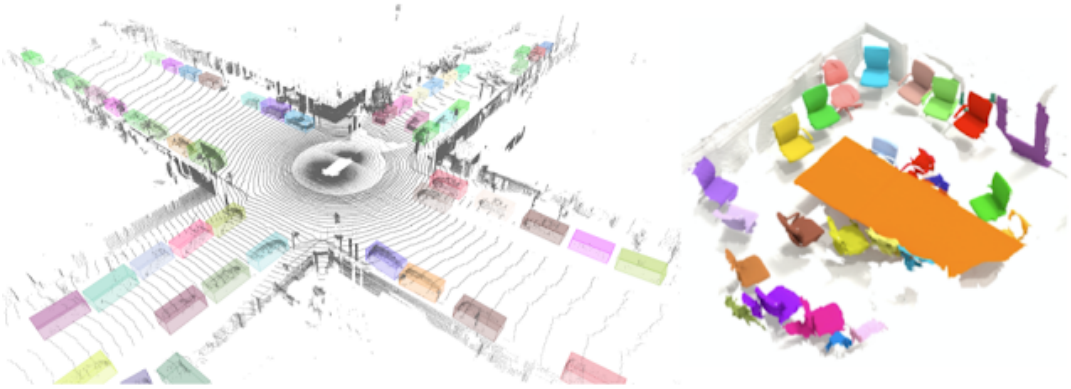


Figure 2.6: 3D object detection (left) and 3D instance segmentation (right) using Tensorflow 3D as found in [116].

Geometric [119], etc., aim to make deep learning models and dataset processing tools available to developers and researchers. Some of the main research areas include: 3D semantic segmentation, 3D object shape prediction, point cloud registration and point cloud densification.

Given the inherent sparse nature of 3D data in our surroundings, denoted as open space, appropriate data structures and algorithms for efficient memory and inference operations are required. Accordingly, sparse convolutional models with pooling operations are seen in the core backbones of most state-of-the-art methods used in outdoor autonomous driving and indoor benchmarks, such as Waymo, NuScenes and ScanNet [116]. Tensorflow, for example, uses the 3D submanifold sparse U-net architecture to extract both coarse and fine features from voxels. This network consists of three components: an encoder, a bottleneck and a decoder. A set of prediction heads can then be added according to the task at hand. Implementations involving sparse convolutions combined with CUDA techniques show improvements of up to 20x with respect to previous implementations [116].

Applications of 3D Scene Understanding. Scene understanding techniques have been seen in a multitude of tasks, such as: rendering textured meshes or point clouds, camera position estimation, fitting meshes with textures, pose estimation, point cloud completion, and point cloud registration [116, 117]. A great effort has gone particularly into:

- Semantic segmentation, where models predict per-voxel semantic scores using one output head, which are then mapped and used to label 3D points [116, 120, 34, 35].
- Instance segmentation, where models group voxels that belong to the same object together using a per-voxel instance embedding vector and then predict a semantic score per-voxel. When the input consists of a point cloud instead of an image, methods using sparse 3D convolutions are preferred [116, 121].
- Object detection, where models uses box prediction and classification losses to predict per-voxel semantic scores, size, rotation matrices, and center, which are then reduced to accurate box proposals during inference time [116, 122].

Limitations of 3D Scene Understanding. As documented by Surendran and Hemanth [123], the major drawbacks of current deep learning techniques include computational complexity and execution speed particular to each technique. Nevertheless, attaining high accuracy is one of the core challenges, especially in terms of distinguishing similar categories (selectivity) and in terms of robustness to rotations, scaling, translations and illumination changes (invariance). One of the main causes for these symptoms in scene understanding research is the limited amount of publicly available 3D data sets [124]. Research [120] highlights that the lack of rich 3D datasets is a major challenge in scene understanding research. For example, the Oakland dataset consists of less than 2 million labelled points, the NYU benchmark contains only indoor scenes and other data sets created using a 3D Velodyne LIDAR provide low point density meshes than those using a static scanner. Another cause consists of the lack of accessibility for developers to hardware [114] for 3D workflows.

These two delaying factors are fortunately slowly changing: RGB-D data sets continue to be released to support the development of new algorithms. For example, the work by Hackel et al. [120] from ETH aids to

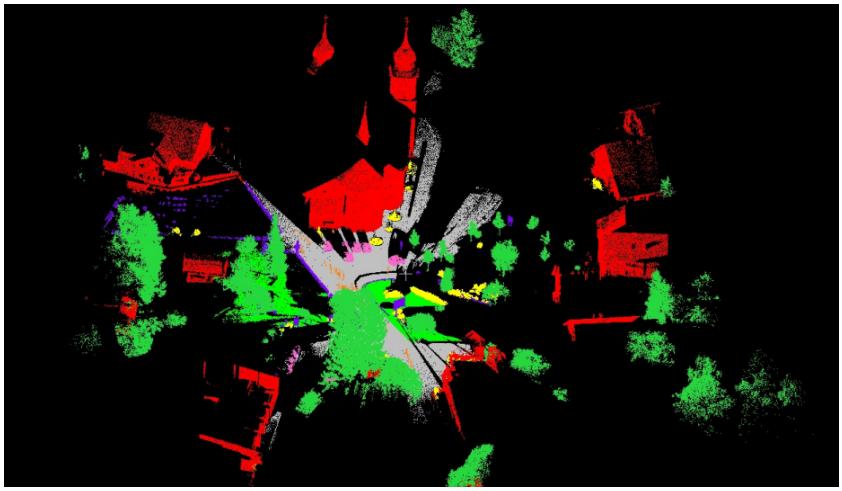


Figure 2.7: Class labels example of Semantic3D [120].

the limited datasets problem by contributing with Semantic3D, the largest known labelled 3D point cloud data set of natural scenes, containing over 4 billion points. Another example is the MediaPipe data set released by Google in November of 2020 [125]. Moreover, accessibility to low-cost 3D sensors continues to increase: the newest smartphones, for example, are equipped with time-of-flight cameras, not only for photography but also extend their capabilities for AR applications and gaming [126]. Even though it is still not possible for any RGB-D data set to be as big as the ImageNet data set (~ 5 million images), techniques continue to be developed to exploit geometric features and to utilize tools and concepts from 2D methods for 3D visual interpretation.

This thesis is motivated by the limitations present in the scene understanding field, and describes the promise of synthetic data in the following sections, which is then later exploited in the approach described in Chapter 3. Similarly, given the limited amount of time and the extensive library of 3D methods, this work is constrained to 2D object detectors to define the concept of temporal uncertainty in a 3D scene, explained in the same chapter. For more information on 2D object detectors, please refer to Appendix A.3.1.

2.5. 3D ENVIRONMENTS

Figure 2.8: 3D hyper-realistic simulated desert using Unreal Engine 5 (early access) [127].

Current software allows the creative development of simulated 3D environments, which can be executed and visualized on a variety of platforms, such as smartphones, gaming consoles, etc. It is remarkable how the applications of simulated hyperrealistic environments continue to grow every year in architectural renderings, advertisements of specialized computer software, video games, movies, video calls [128], etc.

The advent of 3D computer graphics, GPUs, rendering engines and techniques, and game engines continue to allow developers to come up with more sophisticated world simulations. Life-like simulations are therefore possible through 3D environment modeling [129], which involves steps such as:

1. **Detailed concept development** of blueprints and insight of state-of-the-art art techniques
2. **High-definition sketching** based on the predefined concepts
3. **Environment assets creation** of the objects that will be in the scene
4. **Hyper-focused texturing** of the given objects, where natural textures must be applied to allow realism under different perspectives and lighting conditions.
5. **High/Low-Poly modeling** depending on the purpose of the project and hardware constraints for rendering (real-time factor).
6. **3D rendering and optimization** of parameters and light settings.

The following sections give a brief overview of what game engines are and how they enable the creation of synthetic data for machine learning models.

2.5.1. GAME ENGINES AND SIMULATION ENVIRONMENTS

Game Engines, like integrated development environments (IDEs), are software frameworks that allow the development of video games and 3D environments by providing an abstraction layer over a multitude of functionalities. They consist of a main game program, a rendering engine, an audio engine, a physics engine, and an artificial intelligence module, which take care of the following:

- **Input** from a multitude of devices such as keyboards, mice, screens, etc. This input can be obtained through polling or event-based mechanisms (i.e., obtaining the position of a cursor is polling-based and detecting a click from a mouse is event-based).
- **Graphics** which are generated by converting the geometric and color information of a scene from the virtual space of the application into a picture. This requires the usage of 3D assets, which are usually created using 3D modeling software such as Blender, Autodesk 3ds Max, Maya, etc.
- **Physics**, where the laws of gravity, friction, and collision for the movement of a virtual object are simulated.
- **Artificial Intelligence** to give the characters in the environment a personality, such as animals, etc.
- **Sound** to represent sound effects, dialogue, music, etc.
- **Networking** to abstract TCP/UDP and API integrations in the development of multiplayer games.

Physics simulators are an important part of robotics research, as they provide an environment for researchers to test and evaluate algorithms and/or architectures without the constraints and real-world complexity of the physical environment, including the physical degradation of robots. When simulating robots, one of the first questions we need to ask is what level of complexity we are willing to tackle, and where the simulations are going to be used. Moreover, simulations can run faster than real time, are parallelisable, and allow the environment to be reset or modified without physical effort. Finally, simulations are able to produce different sensory inputs which can be extremely valuable for robotics research.

Collins et al. [130] define a robotics simulator as an end-user software application that contains a realistic physics engine, collision and friction detection, GUI, supports 3D assets, allows a scripting mechanism, and can model robotic joints and actuators. Relevant game engines and simulation environments that can be used in the development of machine learning or robotic applications include: Unity 3D [131], Unreal Engine [127], MuJoCo [132], Gazebo Simulator [133] and CARLA [134]. Collins et al. [130] provide an in-depth review of physics simulators for robotics applications with extensive comparative tables that set one simulator apart from other for each use case. Factors taken into account in the evaluation of the simulators include: fidelity of rigid or soft body contact dynamics, locomotion over irregular terrain, support for torque and vision sensors, noise simulation in sensors to approximate real world policies, domain randomization to diversify training data, texture randomization, randomization of object mass, inertia, and friction

coefficients, support for multiple physics engines, support for parallel simulation, support for headless mode and rapid dynamic solvers, support for CPU and GPU optimizations.

(a) (b)

Figure 2.9: Comparison of the capabilities between two simulators of creating realistic 3D environments: (a) CARLA [134], (b) Unreal Engine [135].

Across the simulators evaluated in the review, where some support more features than others, Unity and UE not only are capable of performing all functionalities described but are the only ones that provide hyper-realistic renderings. Similarly, the authors missed to consider these two engines themselves as general simulation platforms and only focus on physic-specific simulators. In other words, simulators built on top of Unity or UE such as Airsim or CARLA were evaluated. Juliani et al. [10] recognize that simulators are not all equally capable of providing meaningful challenges to learning systems, where a multitude of factors need to be considered to create a worthwhile benchmark for research. Moreover, they stress that modern game engines are not only powerful tools capable of simulating realistic worlds with sophisticated physics and complex interactions, but also are precisely engineered to be intuitive, easy to use, interactive, and provide cross-platform compatibility. It is therefore also our belief, in line with this publication, that game engines are more appropriate for the development and testing in the foreseeable future of AI research. Finally, we would like to draw attention that with the recent release of Unity Engine 5, as shown in Figure 2.9, the reality gap between synthetic data and realistic data is not far from being closed. Therefore, Unity and Unreal Engine are of special interest to this masters thesis. These game engines are preferred given their simple interface, mature building blocks, multiplatform capabilities, optimized physics engine, 3D rendering, and powerful environment modelling tools. They also allow integration with Python or C++ scripts, respectively. Finally, both frameworks provide an asset store and an active community of developers who actively contribute through forums and release free or inexpensive plugins, 3D assets, shaders, etc.

For further analysis on landscape simulators, environments and platforms, refer to [130, 10].

2.5.2. SYNTHETIC DATA

Since the quality of a data set distribution can have a negative influence on the quality of an algorithm's prediction and generalization ability, it is important to analyze the quality of the dataset. Accordingly, it is claimed that data quality represents 80 percent of the work on AI [136], making it important to exploit techniques that allow us to create quality datasets. In general, a data set with an similar number of examples per class can be considered to be a data set with good distributional quality. A technique that enables an alternative method in the creation of quality data sets is synthetic data fabrication.

Synthetic data is generated through simulations, algorithms, or computers, making the results independent of the experimenter. A big misconception is that synthetic data can create a skewed image of scientific reality. This concern does not take into account, however, that data synthesis may have the potential to make scientific efforts more efficient. Data synthesis makes the research process more objective by providing a common ground for comparison and allowing experiments to be repeated, such as benchmarks. Additionally, synthetic data can improve the efficiency of experiments because the creation of the data does not have to be done repeatedly. Finally, synthetic data can also prevent researchers from falling into the trap of data dredging, which is when the researcher tests multiple hypotheses using a single dataset. In a June 2021 report on synthetic data[137], Gartner predicted by 2030 most of the data used in AI will be artificially

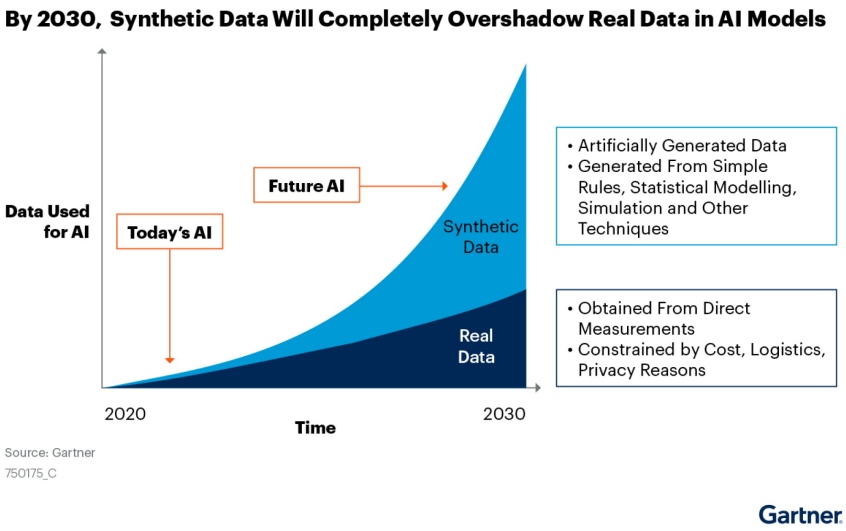


Figure 2.10: Synthetic data will become the main form of data used in AI [137].

generated by rules, statistical models, simulations, or other techniques. This goes in conjunction and is supported by the breath-taking improvements in Unreal Engine 5's rendering pipeline [127] to develop hyper-realistic 3D worlds, and the innovative advancements shown by Adobe in the past months (2020-2021) through their 3D Substance product line. Adobe 3D Substance includes thousands of models, textures, lighting systems, and uses artificial intelligence to reduce the technical complexity in 3D design and modelling [138]. Synthetic data generation is not only a cost-saving technique but it can also sometimes even be better than real-world data, since it includes rare but critical corner cases in the distribution quality of a dataset.

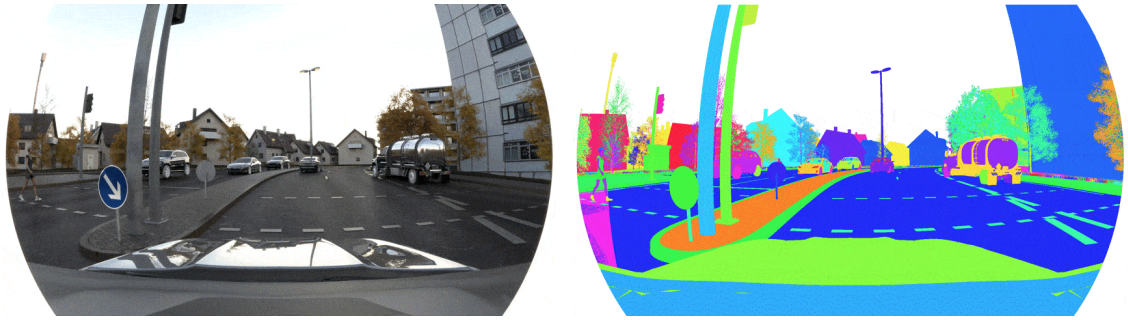


Figure 2.11: Developers can alter and randomize objects, colors, lighting, materials and poses in realistic 3D environments to quickly generate synthetic data with perfect labels [136].

There are many techniques to generate synthetic data, such as variational autoencoder-decoders algorithms, GANs and simulations [136]. The factors mentioned in section 2.5.1 that make a good physics simulator such as domain randomization, texture randomization, noise simulation, etc., allow today's game engines to create sophisticated and realistic synthetic datasets, where algorithms and agents can learn more general patterns. Figure 2.11 visualizes how developers and researchers are capable of manipulating 3D environments to generate realistic datasets with extensive domain randomization. It is therefore of special interest in this masters thesis to use the Unity 3D game engine to analyze exploration policies in simulated and realistic environments.

2.6. REINFORCEMENT LEARNING

Reinforcement learning is an umbrella term used to describe a class of algorithms for learning from experience. The basic idea is that a computer is given experiences in the real world (in a computer game, for example), and then has to learn how to act in that environment. The computer learns over time to increase its performance through trial and error and by watching what it does and making changes to its behavior

policy. The performance is measured by the maximization of a reward function. Reinforcement learning is at the core of many of the most successful modern applications of artificial intelligence, such as AlphaGo, a computer that can play the board game Go better than any human being.

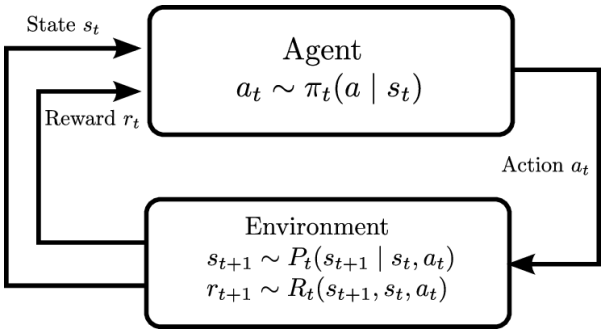


Figure 2.12: Agent-environment relationship in an MDP according to Sutton, Barto, et al. [139]: at each time step t, the agent observes the environment's current state s_t and a reward signal r_t. The agent then selects an action a_t given s_t and following policy π_t. This action changes the environment state in the next time step to s_{t+1} and yields reward r_{t+1}.

Despite the massive amount of recent research into the topic, reinforcement learning is still an extremely challenging field. This is due to the amount of information required to solve complex real world problems using reinforcement learning. For example, in the situation where one must create a simulation of a robot, one must take into account the factors and physics involved in the real world.

Reinforcement learning is different from supervised learning in that the computer is given no labelled training data in the reinforcement learning case. Instead, it learns by trial and error and by watching its own actions. The idea is that, by observing what it does and seeing what results it gets, the computer will automatically learn what actions it should take to get better results in the future [75].

Common concepts involved in a reinforcement learning problem are described below.

Agent. The agent is the entity that actually does the learning. This is the actor that makes the decision for which action to take next based on the state observed from the environment [75]. For this action, the agent can receive a reward from the environment, from which the agent updates his value function or his policy. The agent then chooses a new action based on a new state and the interaction loop repeats. An complete interaction occurs at time step t, which is part of a sequence of time steps $t = 0, 1, ..., T$. The agent can be a single entity such as a robot or a group of entities that work together to solve the problem. The agent-environment interaction loop can be visualized in Figure 2.12.

Action. The action refers to the action that is going to be taken. This could be one of the moves in a chess game, or it could be a move in a board game such as Go [75].

Learning from Experience. The term "experience" as used in the context of reinforcement learning does not refer to a specific object or event. Rather, it refers to a sequence of events that have happened to the agent. However, the experiences that are fed into the reinforcement learning algorithm are derived directly from the environment [75].

Environment. The environment is the world in which the agent learns and collects observations of the state it finds itself in. It is also the world in which the agent must decide which actions to take. Accordingly, the environment provides rewards to the agent for each of these actions. Similarly, when the agent is, for example, damaged, the reward function decrements the reward that the agent receives for that action. The environment also provides the agent with feedback when it takes an action. It provides the agent with new visual observations, positions, velocity, health of enemy targets, etc. These observations are all factors that contribute to the agent's perception of the environment. Environments, in general, can vary and be either a game of chess or a real-world application where one robot is being controlled by a human operator [75].

A reinforcement learning environment can also be formally referred to as a Markov Decision Process (MDP) in other contexts, where each state is a possible outcome of the environment and each action is a possible change to the environment. An MDP is a formalism that can describe any decision-making process that can be described by a transition matrix. Moreover, for a reinforcement learning algorithm used in RL, an MDP

may be thought of as a discrete-time Markov chain, where time is broken into discrete slots of a specified duration, and state is broken into discrete states. This formalism allows a generalised analysis of the behaviour of an agent in terms of the utility or benefit of the agent in any state and any action taken in any state. It is described by a 5-tuple (S, A, R, P, μ), where:

- S is a set of states the environment can be in,
- A is set of actions the agent can select,
- $R(s, a, s')$ is the reward function that maps state-action-next-state tuples to real valued rewards,
- $P(s'|s, a)$ is the probability of transitioning to state s' given previous state s and the agent choosing action a while in s,
- $\mu(s)$ is the probability of starting in state s.

If an environment is fully observable, the agent has full knowledge of the state it is in. In a partially observable environment, the appropriate formalism is a Partially Observable Markov Decision Process (POMDP), where the agent must rely on its internal knowledge of its surroundings to make decisions. This is a generalization of the MDP to cases where the environment is partially observable, and it includes a set of observations O and a conditional probability distribution, $P(o|s)$, for what observation is seen in which state [75].

Reward Signal and Return. A reward signal describes how well the agent did in a given situation. A reward signal can be thought of as a kind of scoreboard. It will assign a numerical score to each of the actions available to the agent in a given state. These scores are then combined to calculate a global reward which is to be maximized [75]. In other words, the reward signal R_t is the reward the agent obtains at time step t. The cumulative reward G_t from this signal is the agent's objective. A simple return is defined as:

$$G_t = R_{t+1} + R_{t+2} + ... + R_{t+T} = \sum_{k=0}^{T-t-1} R_{t+1+k} \tag{2.1}$$

where T is the final time step. A return can also include a discount rate γ that discourages future rewards, as shown in 2.2. A discount rate is used, for example, if the MDP is a continuous problem that never ends or to include uncertainty of future predictions [75].

$$G_{t,discounted} = \sum_{k=0}^{T-t-1} \gamma^k * R_{t+1+k}. \tag{2.2}$$

Additionally, it is worth mentioning that a **sparse** reward signal refers to the lack of instantaneous rewards or penalties given to the agent per transition regardless of distance to the goal. In other words, it rewards the agent in states that are close to a goal. In contrast, a **dense** reward function rewards (or penalizes) the agent at most of the transitions it takes. This feedback can be counterproductive since the agent could have too many distractions at each timestep. This concept will prove useful in Chapter 5.

Value Function. Whereas the reward signal assigns a reward to the actions that are available to the agent at a given state, the value function specifies what is good in the long run. The value function calculates a value for a state, as the expected cumulative reward from the rewards in the future states, starting from that state. [75] In other words, the state-value function, $V_\tau(s)$ is the expected future reward starting from state s and following policy π. It can be written as

$$V_\pi(s) = \mathbb{E}_\pi \{r_t \mid s_t = s\} = \mathbb{E}_\pi \left\{ \left(\sum_{k=0}^{T-t-1} \gamma^k r_{t+1+k} \right) \mid s_t = s \right\} \tag{2.3}$$

where γ is the discount rate, s_t is the state at time step t, r_t is the reward at time step t, and $\mathbb{E}$ is the expected value [140].

Figure 2.13 shows a grid-world environment where the agent must move from one cell to another to reach the goal. Given that there is a pit in which the agent can fall in. If the agent falls it is penalized through a

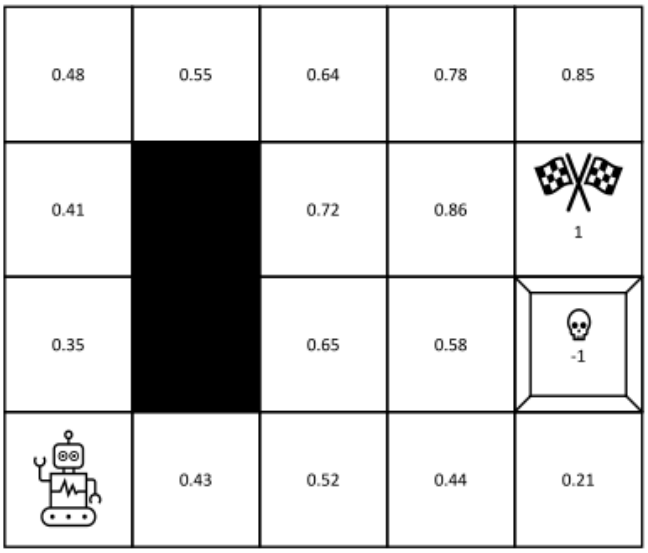


Figure 2.13: A grid-world environmentm. Each cell represents a state, and it is associated with a corresponding value, which is the value of the state-value function $V(s)$. The agent gets a reward when it reaches the goal with two flags, and given a penalty when it falls down the pit with a skull. Taken from [140].

negative reward. Hence, the state-value function for the cells around the pit is lower to represent that these are unwanted states.

Policy. The policy π is the rule by which the agent decides which actions to take. In other words, a policy is a mapping from states to actions, and it defines how the agent will behave at any given state [75]. The policy, just like the reward function, is learned based on what the agent experiences, i.e., what states provide higher reward than others. A policy can be designed as a look-up table or through function approximation. When the policy is stochastic $\pi(a|s)$ denotes the probability to take action a given state s, and the optimal policy $\pi*$ is the policy that always prefers the action corresponding to the highest expected reward [75].

Episode. An episode—also called a *trajectory* or *rollout*—is a single event (i.e., the sequence of steps of an agent) which is composed of several time steps. An episode is defined by a sequence of states and a set of actions in an MDP [75]. For example, a video game will have many states (positions of objects, etc.), and many actions (movements of the character). At the end of an episode, the environment is reset and a new episode begins with a fresh set of observations and actions.

Exploration and Explotation. Reinforcement learning problems try to balance exploration of new states and explotation by following the actions to states that provide the best rewards. Finding an optimal policy requires the agent to explore new unknown states which could allow the discovery of a better state-value than the one currently known. An example of a policy that prefers exploration is the ϵ-greedy policy. This policy assigns ϵ chance to choose a random action and $1-\epsilon$ to choose an action based on the known policy [75].

2.6.1. Q-LEARNING

Q-learning one of the most well known reinforcement learning algorithms. It is a model-free learning algorithm and therefore it does not require any assumptions to be made about the environment or about the reward function. At the start of each episode, the agent picks an action (also called a "policy") from some policy distribution. The agent then receives a reward which is calculated based on the policy that was chosen. The agent then uses the reward to update the policy distribution, which is often called the Q-function. The Q-function is an action-value function $Q_\pi(s,a)$, which, in addition to the state-value function (reference), it also considers the action space [75]. It can be written as:

Q-learning learns an optimal ϵ-greedy policy. The agent has an ϵ chance of choosing a random action and $1-\epsilon$ of choosing an action based on the optimal policy. The optimal action is the one that maximizes the action-value function $\max_a Q_\pi(s,a)$. Q-learning works by starting from some arbitrary $Q_\pi(s,a)$, and iteratively updates itself by taking into account the immediate rewards it receives. This is performed by taking an action and then updating its $Q_\pi(s,a)$ in the following manner:

$$Q_\pi(S_t,A_t) \leftarrow Q_\pi(S_t,A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q_\pi(S_{t+1},a) - Q_\pi(S_t,A_t)\right] \tag{2.4}$$

where λ is the learning rate, Q_π is the action-value function, and π_t is the agent's policy at step t. In the case of Q-learning, this is the optimal policy π^*. The learning rate determines how strongly the old values of Q_π should be modified by the incoming update. This process is done until the episode ends. The environment then resets and the learning continues from a starting position [75]. The Q-learning algorithm can be seen in Algorithm 1 below:

Algorithm 1: Q-learning

initialize $Q_\pi(s,a)$ with random values;
for *each episode* **do**
  reset environment;
  while ***not*** *done* **do**
    choose action A based on state S using policy π
    perform action A and observe R,S';
    $Q_\pi(S,A) \leftarrow Q_\pi(S,A) + \alpha\left[R + \gamma \max_a Q_\pi\left(S',a\right) - Q_\pi(S,A)\right]$
    $S \leftarrow S'$
    if *episode end* **then**
      done;
    end
  end
end

After n episodes, Q-learning should be able to learn a policy that allows the agent to solve the task successfully. There are several variants of Q-learning that differ from the method described above. A popular variant of Q-learning is SARSA ("Synchronous", "Asynchronous", "Reinforce"), which can be used for learning both discrete (e.g. chess, Go) and continuous (e.g. Atari games, robotics) problems [75]. Additionally, it is one of the algorithms available in the open source deep reinforcement learning OpenAI Gym [16].

2.6.2. FUNCTION APPROXIMATION

Tables are a feasible solution for small state space searches for state-value and action-value functions. However, when the state space is large, such as in robotics applications, tables become impractical. Function approximation methods such as Gaussian processes, neural networks, and deep neural networks, make it easier to approximate large state-value and action-value functions [75].

NEURAL NETWORKS.

Neural networks are particularly good for modeling nonlinear relationships as they are essentially universal approximators. They use a large number of interconnected neurons (perceptrons) to approximate the function $y = f*(x,\theta)$, which defines a mapping from input x to output y, by learning the parameters θ [140].

A neuron consists of an input (or data) connection, and an output (or hidden) connection, as visualized in the model in Figure 2.14. Inputs x are multiplied by a set of training weights θ and summed. An activation function h is then applied to the sum. The output $h(x,\theta)$ of a neuron is a nonlinear function of the input, and as a result a large number of neurons is needed to approximate a complex function. The weights are then trained by backpropagation in order to minimize the cost of the function approximation (i.e. the training set error). A common activation function, ReLU is defined as follows:

$$h(x,\theta) = \begin{cases} 0 & \text{if } z < 0, \\ z & \text{if } z \geq 0, \end{cases} \tag{2.5}$$

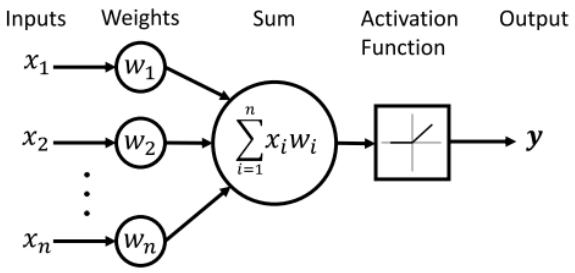


Figure 2.14: Neuron. A single neuron consists of an input (or data) connection and an output (or hidden) connection. The input x is multiplied by the training weights θ and summed. An activation function is then applied to the sum to obtain a nonlinear function $h(x,\theta)$ of the inputs.

where z is the summed output of the neuron. By tuning the weights of a neural network, non-linear functions can be approximated. Accordingly, a neural network could be leveraged to approximate the value function or policy of an agent by feeding state s as input to the network [140]. A common and efficient method to tune the weights of a neural network is batch gradient descent. Gradient descent can be defined as

$$\theta_{i,j} \leftarrow \theta_{i,j} - \alpha \frac{\partial L}{\partial \theta_{i,j}} \tag{2.6}$$

where α is the learning rate, $\theta_{i,j}$ are the neural network's weights and L is the loss function. A loss function is a measure of how well the neural network approximates the desired function. Common loss functions for value functions include squared error and mean squared error, where common loss functions for policy functions is cross-entropy. In the squared error $L = (y - \hat{y})^2$, $\hat{y}$ is the predicted output from the network and y is the true value (ground truth). For more information on backpropagation and gradient descent, see [75].

Neural networks have been shown to be a very powerful approach to nonlinear function approximation. For example, they are able to approximate the square root, logarithm, exponential, etc. However, this is not the case for complex functions, such as $f(x,\theta) = 1/1 + x \cdot \sin(\theta)$. While neural networks are an important tool in reinforcement learning, they are only good for low-dimensional functions and cannot be directly applied to high-dimensional or discrete state spaces. This has led to the emergence of methods using deep neural networks [75].

DEEP NEURAL NETWORKS

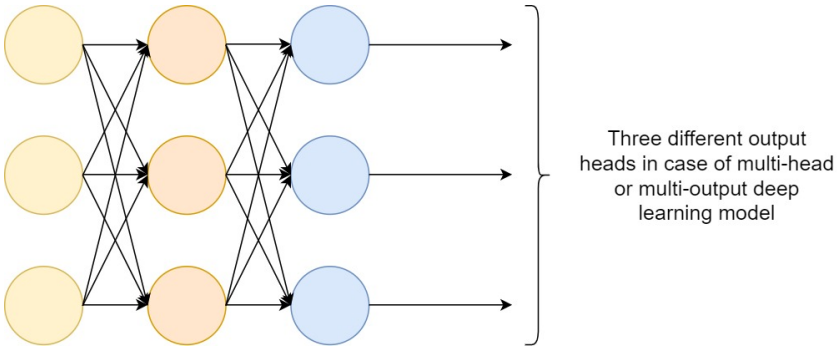


Figure 2.15: Deep Neural Network with an input layer, a hidden layer and an multiple output heads. In actor-critic reinforcement learning problems and shared parameter networks, one head can be used as the prediction from the critic and another head as the prediction from the actor [141, 75].

Deep neural networks are a type of neural network which have been shown to work well for a variety of real world problems. In general, deep neural networks are a type of neural network that uses multiple layers of interconnected neurons, as shown in Figure 2.15. In situations when the input is an image, data manipulation strategies can be used, such as vectorization of the image so that it is compatible with the input a simple neural network expects. However, the loss of spatial information during the vectorization of the data (also called flattening) can be a problem if part of the solution requires to understand the relationship between

the elements in a scene. Convolutional neural networks have therefore been shown to be a very effective solution to this problem since they are capable of processing images while keeping their matrix structure. These networks use a kernel of weights to "convolute" the image, which is also called "filter". A filter is applied over the entire input to output a new set of values. This process is repeated over all the different parts of the input and can be then fed to other types of network structures, such as fully connected layers [140]. As of today, convolutional neural networks have proven to work well in many different domains, such as image classification, object detection, and semantic segmentation.

2.6.3. PROXIMAL POLICY OPTIMIZATION

Proximal Policy Optimization (PPO), designed by OpenAI, is a reinforcement learning algorithm that is known for performing better than one of the most popular reinforcement learning algorithms: DQN (also known as the Deep Q Network algorithm; which has roots in the theory presented in Section 2.6.1). PPO overcomes a lot of challenges that other methods struggle with such as sample efficiency, overall performance, and ease of implementation and tuning capabilities [14]. Moreover, while algorithms like DQN learn from offline memory, PPO can do online learning without using a replay buffer for past experiences. In other words, it allows the agent to learn directly from the environment and discard the batch of experiences after a gradient update. Finally, PPO belongs to methods that leverage the state value function to update the policy. Actor-critic methods, as an extension of Q-learning, approximate the value function (critic) to assist in the policy (actor) updates.

The following subsections further introduce the fundamental concepts behind PPO, such as policy gradient and clipping, and the algorithm itself.

POLICY GRADIENT

Policy Gradient methods estimate the optimal parametrized action policy $\pi(a \mid s, \theta)$ (i.e. which action is taken in each state) and are typically used in large state spaces. These methods learn the policy's parameter vector θ by learning the weights of a neural network. The parametrized policy $\pi(a \mid s, \theta)$ describes a distribution which gives a probability of choosing action a given state s and parameters θ at timestep t as

$$\pi(a \mid s, \theta) = \Pr\{A_t = a \mid S_t = s, \theta_t = \theta\}. \tag{2.7}$$

These methods aim to maximize the expected return $J(\theta)$, known also as the objective function or loss, by following the parametrized policy [75]. Gradient descent (or ascent) is then used to update the weights of the network as

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)} \tag{2.8}$$

where $\widehat{\nabla J(\theta_t)}$ is an estimate of the gradient of the objective function (expected return) [**3**]. A standard gradient estimator for these methods is

$$\widehat{\nabla J(\theta_t)} = \hat{E}_t \left[\nabla_\theta \log \pi_\theta (a_t \mid s_t) \hat{A}_t \right] \tag{2.9}$$

where π_θ is the parametrized stochastic policy, $\hat{E}_t$ is the expectation across a batch of experiences and $\hat{A}_t$ is an estimator of the advantage function at timestep t [75]. The respective objective function for the policy gradient estimator is therefore

$$L^{\mathrm{PG}}(\theta) = \widehat{J(\theta)} = \hat{E}_t \left[\log \pi_\theta (a_t \mid s_t) \hat{A}_t \right]. \tag{2.10}$$

Optimizing on L^{PG} is, however, not justified, given that it leads to destructively large policy updates [14].

CLIPPING

PPO builds upon the concepts mentioned above and simplifies the concepts introduced by TRPO by proposing a clipped "surrogate" objective and retaining parallel performance.

First, the probability ratio between the policy with the current parameters and the policy with the old parameters is defined as

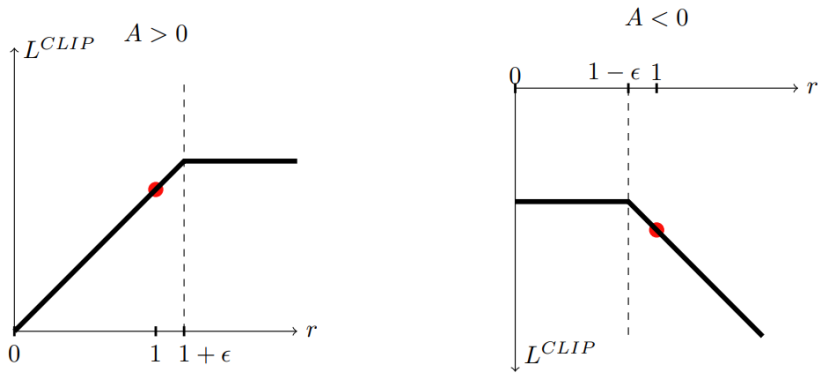


Figure 2.16: Plots showing one term (i.e., a single timestep) of the surrogate function L CLIP as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., r = 1. Note that L CLIP sums many of these terms. [14].

Then, the objective function of TRPO (on policy) is:

$$L^{\text{TRPO}}(\theta) = \mathbb{E}\left[r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a)\right]. \tag{2.11}$$

However, given that the ratio between the policies can grow to be very large, PPO constraints the ratio $r(\theta)$ to stay in a small interval around 1 through the introduction of an additional hyperparameter ϵ [14]. The value function loss then becomes

$$L^{\text{CLIP}}(\theta) = \mathbb{E}\left[\min\left(r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{\text{old}}}(s,a)\right)\right] \tag{2.12}$$

where the function $\text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)$ constraints the ratio to a maximum and a minimum. Subsequently, the objective function of PPO takes a pessimistic lower bound to the loss by keeping the minimmum of the unclipped $r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a)$ and the clipped ratio times the advantage estimator [14]. The advantage estimator is defined by

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \ldots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{2.13}$$

where

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{2.14}$$

and λ is a smoothing coefficient.

ALGORITHM

Most techniques for calculating variance-reduced advantage-function estimators leverage a learned state-value function V(s) as baseline function and PPO is no different [14]. If using a neural network architecture that shares parameters between the two predictions heads for the policy and value functions, PPO proposes to add a value function error term to the policy surrogate and an entropy term to encourage sufficeint exploration. This new loss is defined as

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S\left[\pi_\theta\right](s_t)\right] \tag{2.15}$$

where $c1$,$c2$ are coefficients, S denotes the entropy bonus and L_t^{VF} is the value function loss [14]. In other words, $L_t^{CLIP}(\theta)$ is the loss for the actor network and L_t^{VF} is the loss for the critic network. The loss of the value function used in PPO is defined as

$$L^{\text{VF}} = \left(V_\theta(s_t) - V_t^{\text{target}}\right)^2 \tag{2.16}$$

where $V_\theta(s_t)$ is also known as the critic return and in some implementations is seen as $\delta_t + V_\theta(s_t)$. Similarly, the term $S_{\pi_\theta}(s_t)$ is an entropy bonus that promotes exploration [14]. Concretely, this is the Shannon entropy and is computed by

$$S_{\pi_\theta}(s_t) = -\sum_{i=1}^{n} \pi_\theta(a_i \mid s_t)\log\left(\pi_\theta(a_i \mid s_t)\right) \tag{2.17}$$

In summary, the main idea of PPO is to avoid large policy updates during training by clipping the ratio between the old and the current policy. It successfully combines function approximation with policy optimization while showing great performance [14]. Finally, the algorithm is described in Algorithm 2 below:

Algorithm 2: PPO

initialize θ;
for *iteration i=0,1,...* **do**
  for *time step t=0,1,..., T* **do**
    sample time step with policy $\pi_{\theta,\,\text{old}}$;
    calculate advantage $\hat{A}_t$;
  end
  for *epoch k=0,1,..., K* **do**
    optimize $L^{\text{CLIP+VF+S}}$ with respect to θ;
    update θ;
  end
end

3

OCTREE- & VOXEL-DRIVEN EXPLORATION

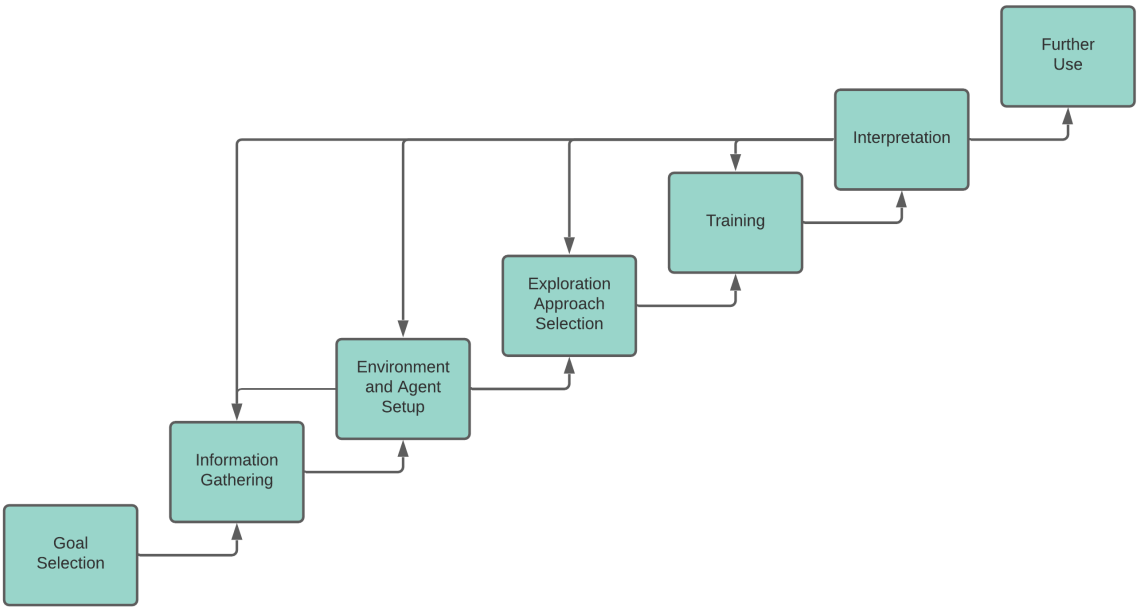


Figure 3.1: Adapted development process from "*Using machine learning methods for evaluating the quality of technical documents*", by Luckert and Schaefer-Kehnert [142]. Describes the taken steps from data collection, through algorithm selection to architecture design.

In this chapter, the different steps taken to construct the environment and the reinforcement learning agent are addressed. This chapter adapts the structure proposed for machine learning algorithms proposed by Luckert and Schaefer-Kehnert [142]. The following questions are answered during this chapter:

- What are the current method's goals and limits?
- How was the reinforcement learning environment modeled and set up?
- How is the agent-observable data constructed?
- How was the agent's character defined (3D model, movement, etc.)?
- What reinforcement learning approach was chosen?
- How is the proposed method composed?
- How can the proposed exploration method be further applied?

In order to answer these questions, the structure of this chapter is illustrated in Figure 3.1. The first step of the RL process is discussed in Section 3.1, which defines the exploration task and limits its scope. Then, Section 3.2 presents the choice and construction of a suitable 3D environment for exploration where the agent collects observations from (i.e., the data set. Afterwards, Section 3.3 explains the approach to perception, presenting the process of voxelization, the creation of the octree map, the nature of the used sensors for the visual input of the agent, and the use of semantic entropy to define uncertainty in an environment. The agent's character, 3D models, movement algorithms, choice of observations, reward signals and environment-related behaviors are presented in Section 3.4. Subsequently, the chosen reinforcement learning algorithm is covered in Section 3.5 and the interpretation of the exploration approaches to octress and voxels are described in Section 3.6. This section also tackles how the method's performance was measured and evaluated and which parameters were tuned. Section 3.7 presents the experiments, including the environment and sensors setup, and the behavior setup for object- and environment-focused agents.

Finally, Section 3.8 presents the proposed evaluation framework for the assessment and comparison of the learned behaviors, introduces the applicability of our approach in multiple scenarios, presents the transferability of the learning environment to another environment platform and proposes the comparison of the performance of the PPO with the Soft Actor-Critic algorithm for the given setup. Inspired by the structure proposed by Luckert and Schaefer-Kehnert [142] and given the iterative nature of the reinforcement learning process, this work was split into two main loops:

- The first loop includes steps 2 and 3, and involves the groundwork for the learning algorithm: it involves the manual construction of the learning environment and the agent using Unity [131] for fabricating the 3D scenes and the ML-Agents SDK [143]. This resulted in a high dependency between the 3D modeling, the simulation of a real robot and the abstractions provided by the learning platform, which demanded an iterative development process.
- The second loop includes steps 2-6, which allows the readjustment of 3D models, the learning environment, agent attributes, observations, reward signals, step size, voxel models, octree parameters, entropy definition, etc. This loop is critical for this thesis, since the main objective is the optimization of exploration strategies for unknown environments. For this purpose, the presented iterative process was used and multiple baselines were proposed to provide performance comparisons. Furthermore, the diverse algorithms performance can be inspected in real-time in Unity and therefore require readjustments in steps 2 and 3.

3.1. IDENTIFICATION OF GOALS

The predetermined goal of this project work was to propose an exploration policy that reduces uncertainty in a 3D scene. As mentioned in Section 1.2, this project work tackles the following research questions:

- How can an embodied agent increase the overall certainty about an object's characteristics, i.e., how can trajectories around objects of interest be covered to reduce the uncertainty about such objects?
- How can these objects be found in large and unknown environments by the same agent?

To answer these questions, the following practical steps were laid out:

1. Model and present a 3D environment for a learning agent to explore and find objects of interest.
2. Define the perception approach for the agent and the character baseline, which includes 3D models, movement algorithms, sensors and limitations.
3. Choose and describe the reinforcement learning approaches for two goals: a) exploration of an environment and b) exploration of objects.
4. Propose learning variants based on the influence of different observations and reward alternatives.

This strategy was determined because the steps for the 3D modeling of environments are independent of the reinforcement learning approach used. Therefore, the remaining task is to propose a reinforcement-learning-enabled pipeline that aims to solve the indicated exploration task without a priori knowledge of a 3D environment. The main challenge in the exploration pipeline lies in the choice of the diverse types of observations the environment can provide to the agent. The diverse variants of the

reinforcement learning algorithm will then be built on the outcome of the second step. Therefore, the definition of the environment, the goals and the agent's capabilities focuses on the working steps 1 and 2. The remaining steps are solved by using the modeled environments and the agent character. More concretely, the population of an octree through the agent's trajectory to solve the exploration task concentrates on the third working step. This working step also includes the exploration of objects, for which the voxelization of 3D meshes is chosen as an efficient and thorough representation of the agent's goals. We call this interest to explore goals through voxels "voxel curiosity".

Additionally, we define uninformativeness in an environment and around the objects is not only by the states that are new to the agent, but also through *semantic entropy*. This concept is inspired by research in semantic curiosity [9] and is addressed in the second working step, as uninformativeness is not only given by the modernity of a state s but also through the temporal class density in such state. Finally, in the last working step, the further use of results is presented, including the influence of the observations and rewards in the exploration policies.

An important limitation, as mentioned in section 1.5, is that this work focuses on reinforcement-learning-based exploration of an environment through ubiquitous visual information (voxels) to reduce uncertainty (defined through entropy) and does not take data sampling in the agent's trajectories or integration of further semantic modules into account, since this would go beyond the scope of this thesis.

3.2. SPECIFICATION OF THE LEARNING ENVIRONMENT

Data in a reinforcement learning problem is obtained through the environment and the diverse states the agent traverses. Therefore, in order to train and evaluate the agent, it is essential to define a 3D scene which provides information to the agent and delimits the exploration problem. The following questions will be answered in this section:

- What led to choosing Unity 3D as the preferred 3D modelling and simulation engine?
- What led to choosing *Unity ML-Agents* as the preferred environment platform?

Training a reinforcement learning agent requires the following: 1) a learning environment, 2) an agent that can observe the environment and choose actions in such environment; 3) a learning algorithm appropriate for the states present in the environment (continuous, discrete). As stated in Section 3.1, the given task is to explore an unknown environment while reducing uncertainty in the environment itself and the objects in it. Therefore, the agent requires: 1) motivation to navigate and explore the environment, 2) motivation to investigate objects exhaustively, and 3) a way to take into account if there is uncertainty or "uninformativeness" while it is exploring. This last requirement is defined through semantic entropy in subsection 3.3.4. The following sections presents the 3D scene and the steps required to construct the environment states for the agent.

3.2.1. A UNITY 3D ENVIRONMENT

The first step requires setting up the environment for the agent. The best game engines in the market right now are Unity 3D and Unreal Engine 4. Both engines allow exhaustive development of gaming environments, with support for scripting languages, and a considerable toolset for animations, triggers, physics simulations, etc. However, Unity was the engine chosen for the setup of this work given its simpler interface, bigger community, and its state-of-the-art plugin for reinforcement learning Unity *ML-Agents* [144, 10]. For more information on the 3D modelling using Unity 3D, please refer to Appendix A.4.

In the setup of a scene, a variety of environment assets such as buildings, trees, rocks, and foliage, are required to convey an authentic depiction of a scene. Sample screenshots of these assets, as well as of the visual effects and skybox assets are illustrated in appendix Figure A.6. Once all the assets are allocated in a scene, the rendering pipeline can be tuned to adjust the lighting settings to achieve a more realistic rendering if needed. Figure 3.2 shows the difference lighting makes when simulating a 3D scene, which is one of the many parameters, in addition to color, texture, rotation, position, etc., that can be randomized using a game engine. As mentioned in Section 2.5.2, domain randomization allows the creation of a dataset that includes not only corner cases but also allows the learning algorithm to generalize to unseen environments.

Given that the current approach utilizes a grid sensor, octrees and voxels to abstract the 3D environment, the realism of the scene does not play a role in the agent's performance. This setup allows the separation of

Figure 3.2: Example of a 3D scene without lighting settings (left) and the same scene with post-processing effects (right). Factors like depth, high-quality volumetric light and fog of variable density allow the construction of realistic environments. Taken from [145].

the 3D modeling work from the agent's algorithm, while still being able to extend to other use cases that can exploit Unity's scripting framework and domain randomization capabilities.

3.2.2. ENVIRONMENT PLATFORMS

An environment platform enables the development, training, and testing of reinforcement learning algorithms. It also provides the necessary interfaces to abstract the interaction between the logic running on a system and the environment that the agent is interacting with. More concretely, the environment platform provides a variety of features that allow the definition of the state of the game, the actions the agent can perform, the reward functions, and the training strategy. We introduce two environment platforms in the following sections: OpenAI Gym and Unity ML-Agents.

OPENAI GYM

OpenAI Gym [143] is a platform designed to facilitate the development of reinforcement learning agents. OpenAI Gym has been integrated into several popular reinforcement learning systems, including the open-source reinforcement learning platform OpenAI Baselines. Gym supports both training and test modes with a variety of configurations to evaluate and compare algorithms, unifying researchers in reinforcement learning to a common set of interfaces, and abstracts the environment through the **env** interface. This interface provides the following main abstractions:

- **reset(self).** This resets the environment and returns an observation.
- **step(self, action).** Transitions the environment to state s_{t+1} and returns a tuple of type: *(observation, reward, done, info)*.
- **render(self).** Renders the environment in its current timestep.

UNITY ML-AGENTS

Unity ML-Agents [143] is an RL platform which allows the construction of more environments rich in physical and sensory complexity, leveraging the Unity game engine, which is composed of a set of components that enable physics simulations, realistic renderings, etc., as described in Section 2.5. The platform not only provides an abstraction layer for the development of algorithms and sophisticated tasks like autonomous driving, but also supports the development of complex multi-agent applications, curriculum learning, imitation learning, etc [143]. This is also facilitated given their state-of-the art implementations of algorithms to easily train agents for 2D, 3D, and VR / AR applications. In contrast to OpenAI gym, the ML-Agents toolkit is composed of the following components, illustrated in Figure 3.3:

- **Learning Environment.** It encapsules the Unity scene and *game characters*. The Unity scene sets up the environment for the agent to observe, act and learn and has to be modeled and constructed by the developer. The ML-Agents plugin allows the transformation of the scene into a reinforcement learning environment. It uses the Academy component (not illustrated) to ensure that all *Agents* are synchronized and to manage the learning environment's settings.
- **Python Low-Level API.**. This is the low-level Python interface which allows communication with a

learning environment. This component is separate from Unity, in contrast to the learning environment, and uses the Communicator to interact with the Unity environment.

- **External Communicator.** This component enables the interaction between the learning environment and the Python Low-Level API.
- **Python Trainers.** The trainers contain the machine learning algorithms that realize reinforcement learning. They are implemented in Python and interface through the Python Low-Level API.
- **Gym Wrapper.** This component allows the export of learning environments to an OpenAI Gym [146] environment platform. OpenAI Gym is a common platform for developing and comparing reinforcement learning algorithms. This component is not illustrated.

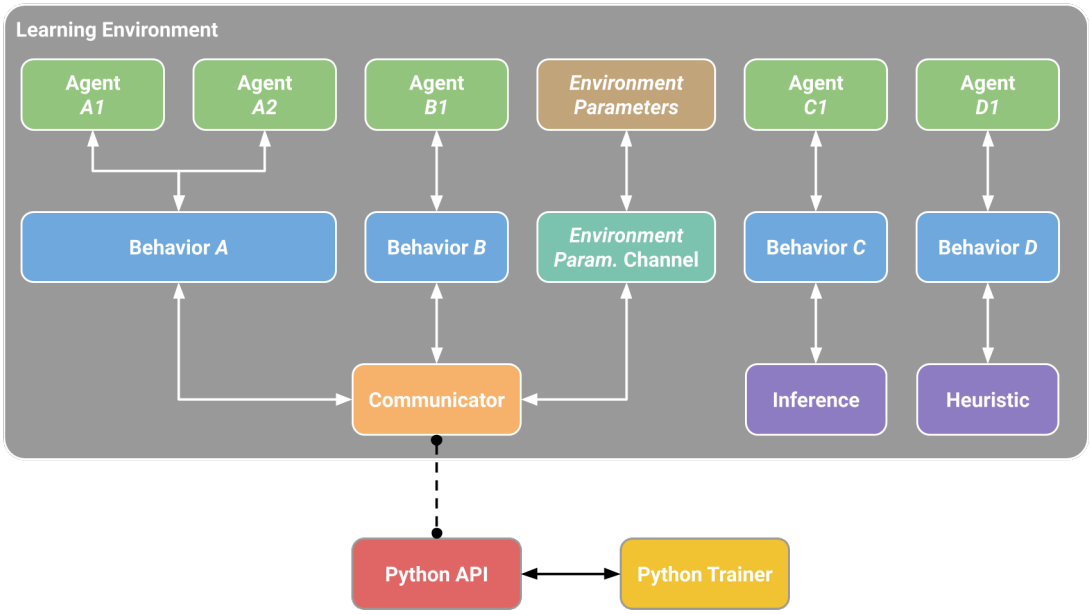


Figure 3.3: Overview of the Unity ML-Agents Toolkit, taken from [143].

Accordingly, the following components in the learning environment aid in the organization of the Unity scene:

- **Agents.** This component abstracts the provision of observations, execution of actions and assignment of rewards and penalties. It is attached to a Unity GameObject, which can be any character in the scene, and is linked to a *Behavior*.
- **Behavior.**. A Behavior component is the logic that takes observations and rewards and returns actions to execute. It also defines the essential characteristics of an agent, such as the type and number of actions the agent can take. Finally, it can be of three types: a) a *learning behavior* that communicates that a model should be created and activates the learning process, b) a *heuristic behavior* that limits itself to a set of hard-coded rules for the agent movement, or c) an *inference behavior* that loads a selected learned model and executes the learned policy in the learning environment.

Every agent in a learning environment will always have an Agent component linked to a Behavior. Similarly, an environment can have a) a single Behavior that can be linked to multiple Agents, as conceptualized in multi-agent scenarios, or b) multiple agents and multiple behaviors. Additionally, it is possible to exchange messages between Unity and Python outside of the reinforcement learning loop using *Side Channels*.

Moreover, the development of learning environments using Unity ML-Agents plugin also facilitates saving, sharing and synchronizing projects across devices and developers using Unity Collaborate [147].

Finally, Unity 3D was the preferred solution given the aforementioned extensive support toolbox for the development process, intuitive operability, active community, training mechanisms through the Unity

ML-Agents plugin, and the capability of extending to future use cases.

3.3. SPECIFICATION OF THE PERCEPTION APPROACH

The task at hand requires the agent to perceive its environment. This can be done through a set of sensors and techniques that transform the environment from 3D models to some other interpretable form. In our case, this was done through the following:

- **Voxelization** of the objects of interest, which generates an occupancy grid to simplify the shape and structure of the objects.
- A **grid sensor** component, which perceives all the 3D assets environment and feeds visual information to the agent, which is later encoded by a CNN.
- **Octree observations** that provide limited information on how much of the environment has been discovered.
- **Semantic entropy**, which provides information on how many classes are in the agent's field of view over the past few frames.

These perceptions mechanisms are presented in the following subsections.

3.3.1. VOXELIZATION OF THE WORLD

Part of the exploration technique involves providing a way for the agent to investigate objects more exhaustively. Research tends to use RGB-D images or point clouds for computer vision tasks [148, 99], which can be both computationally expensive and require a lot of space on memory. We aim to solve the problem of exhaustive exploration and the drawbacks of 3D data representations through the reduction of data volumes through a voxelized representation. Voxelization models the environment into cubes instead of points, as described in Section 2.2. Through the resolution parameter, voxels can largely reduce data volumes with low information loss and minimal overlapping [148]. This abstraction of the world is not only useful for visual purposes but also for other tasks, such as navigation and path planning.

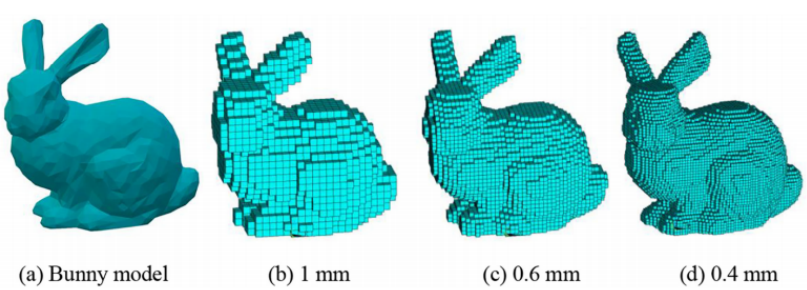


Figure 3.4: Bunny model voxelized at different resolution settings, taken from [149].

Work on voxelized representations of the world have a number of advantages in a variety of use cases that this thesis work is inspired by and takes advantage of, such as as [148, 149, 150, 151, 152]. First, they are space efficient, which makes it easier to store and use a representation of the world, and secondly, since the cubes are a finite volume, they are not restricted by the size of the camera or the resolution of the sensor. Finally, we believe that working with cubes will improve the exploration, because the representation has fewer properties and the agent has more control over the space it perceives. This final property allows the agent to be *visual-agnostic* in the navigation task and understand any kind of environment regardless of the underlying data distribution. This represents a potential bridge for transfer learning tasks for navigation, where the data distribution and preprocessing steps play a critical role in the performance of an algorithm.

In this work, we use an underlying version of a voxelized world, with an isometric representation. The main idea behind voxelization is to represent the world into a medium-sized set of cubes, each representing a specific volume. Each cube can be perceived by the agent and scanned, which awards the agent reward R^{VOX}. There are many methods for efficient voxelization of an environment, as seen in [150, 151, 152]. For more information on the voxelization process, please refer to Appendix A.5, which includes the script used to pre-generate a voxelized representation of the 3D assets in the environment.

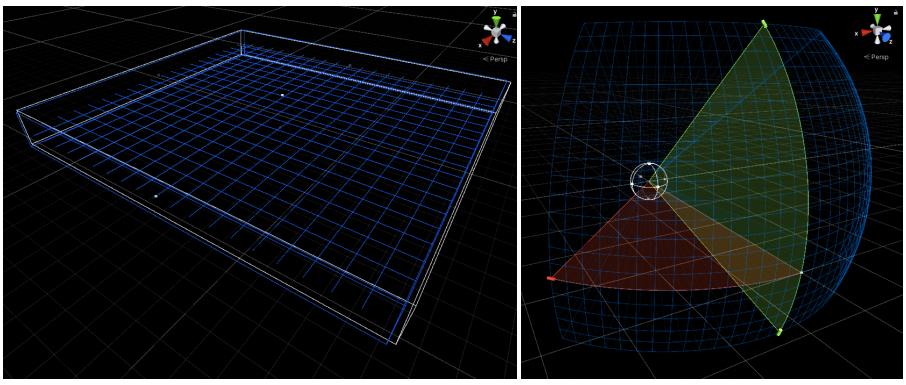
3.3.2. GRID SENSORS AND VISION FROM MONOCULAR CAMERAS

On the topic of vision, panoramic cameras are currently actively used in visual SLAM research given not only their wide range of information perception but also the fast and complete acquisition of information they enable [18, 19]. It is also claimed that spherical cameras will become the norm for driver assistance, traffic safety and autonomous navigation, where visual SLAM techniques must reduce uncertainty as quickly as possible in dynamic environments to prevent collisions [153].

In monocular cameras, common cameras are less favored than other expensive alternatives, as they are limited to 60 degrees of view. This angle limits the amount of information input into the algorithms and adds unnecessary overheads in the extraction of features. More concretely, when the extracted feature points have a short-term duration in the field of vision, a different set of challenges and assumptions a technique must be taken, such as memory, tracking, etc. Accordingly, research by Davison et al. [154] highlights that it takes less time to reduce uncertainty and correct positioning errors the longer a feature is observed continuously. For an in-depth review of visual SLAM methods using RGBD and monocular cameras, refer to [155, 153].

This thesis not only took inspiration from these works to reap the benefits provided by spherical vision but it also provides vision to the embodied agent through a grid sensor component. In a nutshell, a grid sensor enables (faster) off-render training and wider versatility in learning agents. The following paragraphs present the historical background and more details for this component.

The motivation for the use a grid sensor roots back to the origins of the component in the reinforcement learning community: with the growing interest for reinforcement learning techniques and research, a middle ground was required between model expensiveness and lengthy training times. Moreover, the testing of this middle ground had to be independent of the changing visuals across diverse video games, defined by one of the core principles of Automated Game Testing [156]. This was how the first grid sensor came to be: in 2020, the labs at Eidos-Montréal [156] innovated the state-of-the-art in perception for reinforcement learning, which at that point was limited to raycasts and cameras, with a simple 2D *grid sensor* component. Inspired by the simplification of an environment in the MinAtar paper [157], the grid sensor exploits the computational efficiency of CNNs with the generality in data structure provided by raycasts. The basic principle behind the grid sensor is to use a height x width x channel matrix of variable resolution, that queries physics information from objects in range. This keeps the spatial structure perceived and can be then fed to a CNN, a reinforcement learning algorithm, or used to perform data analysis.



(a) (b)

Figure 3.5: (a) Top-down 2D vision through 2D grid sensor (b) Spherical vision through 3D grid sensor. The used grid sensor component for Unity ML-Agents was developed by Mathias Baske [158].

The grid sensor used in this thesis is an improved version of the original grid sensor, developed by Mathias Baske [158] and is visualized in Figure 3.7. It is setup with a variety of perception parameters to customize the range, cell arc, angles, etc. Furthermore, it can provide geometric and semantic information about each object detected or be extended for other use cases. At the implementation level, the unwrapping of the grid sensor input into the 3D grid shown in Figure 3.6, is done by equirectangular projection of a single Unity object known as an *OverlapSphere* that recovers all colliders in range. The calculations for the latitude and longitude of the detected points, alongside with more information about the grid sensor, can be found at `https://github.com/mbaske/grid-sensor`.

Figure 3.6: Voxelized view of the grid sensor. The spherical vision is unwrapped through equirectangular projection and mapped into a 2D grid. Different class tags are represented with different colors and distance is shown visually through different brightness levels.

In Chapter 4 we compare the performance of the agent under an ordinary monocular camera setting of 55 degrees and a spherical, panoramic, setup. This, in conjunction with a voxelized representation of the world, conceives a training agent that is input-agnostic and can greatly increase the training time in 3D environments. Since this setup does not depend on the rendering of visuals, training in the configuration file can be set to headless (off-render) mode. This means that training runs as a normal computer process but no Graphical User Interface is shown in the computer screen, and the rendering load on the GPU can be lifted. This allows the agent to train while still using high-dimensional data and the benefits of domain randomization, such as position and orientation changes. Moreover, in terms of the development pipeline, the 3D creative process involving textures and lighting setup is completely detached from the agent's behavior training setup. Our aim is that this also motivates further research in reinforcement learning to enable effortless transfer learning of agent behaviors from training in simple environments to progressively more complex ones.

3.3.3. NAVIGATION WITH OCTREES

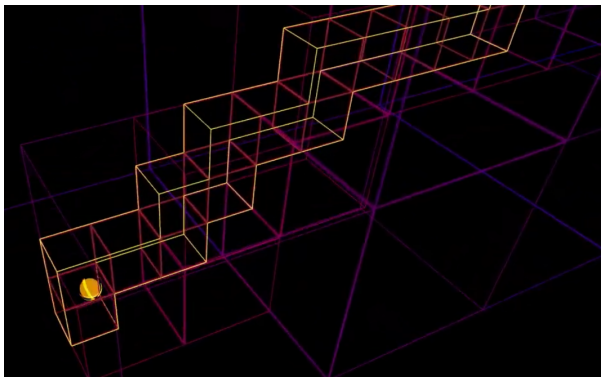


Figure 3.7: Octree Construction and Navigation in Mbaske's Explorer Drone [159].

As described in Section 2.3, octrees can be used to subdivide a scene in hierarchical nodes, providing a way to efficiently deal with large amounts of space. Since our application of octrees is not meant for rendering, the drawbacks of octrees in comparison to polygons can be disregarded. On every step, the agent adds new nodes to the octree, which can be of three types: empty nodes (scan out of range), point (scan) nodes, and nodes that have been visited. Information about the octree is part of the agent's observations, presented in Section 3.5.1. The reward R^{OCT} is part of the reward signal the agent receives, and it is proportional to the number of new nodes added to the octree. The reward is further defined below in Section 3.5.3. The implementation of octrees for Unity was taken from Baske's Explorer Drone, which can be found at [158].

3.3.4. SEMANTIC ENTROPY

In 1997, Melamed [160] defined semantic entropy as a way to measure semantic ambiguity, uninformativeness or as the inverse of semantic reliability. In this thesis, we are especially interested in the concept of entropy as the degree of randomness or disorder in an environment, in line with the definition

given in information theory, which defines entropy formally as

$$H(X) = -\sum_{x \in X} P(x_i) log P(x_i). \tag{3.1}$$

where X is a discrete random variable, with possible outcomes x_i and $P(x_i)$ is the probability that outcome x_i occurs.

Heavily inspired by recent work on exploration by Chen, Gupta, and Gupta [13] and Chaplot et al. [9], and as part of the answer to the research questions, we leverage the concept of entropy to define the uncertainty in an environment or a 3D object. Chaplot et al. [9] utilize semantic curiosity to learn exploration trajectories based on semantic maps. These semantic maps contain information about the inconsistency provided by an object detector at position P, and are used as a motivation for a reinforcement learning agent to prefer states with higher entropy. In their words, "if an object is labeled with different classes as the viewport changes, it will have high temporal entropy". Similarly, this thesis constructs a short-term memory buffer that measures the temporal inconsistencies perceived by an object detector as a way to measure semantic entropy in the agents field-of-vision. This metric, in contrast to the semantic curiosity motivation, is used to influence the linger estimator in the agent. More concretely, it acts as a discount factor in the definition of the penalty given by inactivity. The definition of the reward signal is presented in Section 3.5.3 below.

3.4. SPECIFICATION OF THE AGENT CHARACTER

An agent is simulated through 3D assets, observations collected from an environment, actions, movement algorithms and a learned policy that motivates a specific behavior. The following sections provide insight into these key areas. Furthermore, our proposal agent behavior the exploration task, is the result of a hierarchical composition of behaviors, as illustrated in Figure 3.8. This composition starts with a base DroneAgent, that sets the basic requirements for any other agent, which includes movement, sensing and interaction. Posterior layers provide additional logic to train specific policies. For example, the *SpeedDrone* takes into account variables such as angular speed, forward velocity, etc., to train an agent to minimize the speed error between its current speed and a target speed. The same logic follows for other agents such as the OctreeAgent, VoxelAgent, ShortestPathAgent, etc., where observations that belong to the specific domain of the policy to be trained are added on top of previous layers.

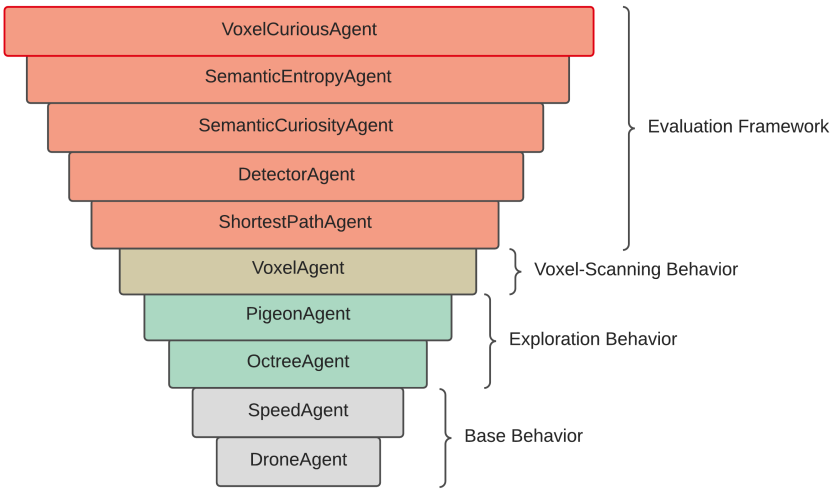


Figure 3.8: Hierarchical construction of the agent behaviors in Unity. The DroneAgent contains embodiment logic of the agent such as relative position and orientation. Higher behaviors inherit from the DroneAgent and implement speed, perception of octree nodes, perception of voxels, detections from a YOLO classifier, etc.

Given that the base drone lays the ground logic for the rest of the agents, a brief explanation of its components is due. The base agent layer introduces

- A 3D model of an agent.
- A movement algorithm.

- Basic information for embodiment such as agent's position, rotation, its velocity, etc.
- A grid sensor to enable perception of the environment, as introduced in Section 3.3.2.
- A scanner component, which enables interaction with the voxels in the environment.
- A reward signal.
- An episode reset logic.

Given that the base layer provides the foundation for other derived agent behaviors, the aformentioned elements are described in more detail in the sections below. Similarly, the base agent allows the introduction, implementation and testing of:

- a modeled 3D environment.
- an environment and goals controller.
- the diverse environment platform components.

Our environment and goals controller, also known as the *TrainingAreaController*, looks into the randomization of the environment obstacles, goals and initializes the necessary variables for the agent's future episode.

Consequently, composed agent behaviors follow the structure described for the base agent and incrementally add observations and reward signals to motivate learning a different policy. In our case, the exploratory behavior is achieved thanks to the policy provided by the OctreeAgent's behavior and the policy that motivated in-depth scanning of objects is given by the VoxelAgent's behavior. More detail is provided for each derivative agent in tables A.2 and A.3.

3D MODELS

The 3D models for the agent were obtained from the Unity Asset Store [1] and Sketchfab [161] as potential avatars for the learning agent in different scenarios. Figure 3.10 visualizes some models considered for the agent, where the Drone Bot was the chosen model, given the overhead complexity of animating the rigs of the other 3D models. To make the simulation more realistic, the drone was given a set of animations, volumetric lights and particle systems.

(a) (b) (c) (d)

Figure 3.9: Learning agent 3D model candidates: (a) Spot Bot, (b) Drone Bot, (c) Tesla Bot. (d) Wireframe Shader used for visualization of voxel scanning. Taken from [1, 161].

SCANNER

The scanner component allows the agent to scan voxels perceived from the environment given a set of constraints such as minimum angle and distance. It is made visible to a human spectator in the graphical user interface through a a special VFX feature of Unity's render engine, namely the *wireframe shader* [1], which is displayed in Figure 3.10 (d). This shader, with the support of an invisible cone *collider* [131] in the scanner component, modifies the render effect of the textures of objects, and displays a wireframe of the object's mesh in the intersection between the scanner's cone collider and the object's collider.

MOVEMENT ALGORITHMS

To provide actions to the agent, i.e., to simulate a real world robot, the key challenge was the mobility algorithm for the agent. In this work, the agent was given the capability of moving in a three-dimensional environment, so it would be able of reaching any desired location. To this end, two mobility algorithms were implemented:

- Translations-based movement with a rotating function.
- Physics-based movement.

The simpler variant was implemented using 3D translations, applied to the agent transform's local coordinate system. The second variant of the mobility algorithm was inspired by the movement model of a real robot, considering concepts such as velocity, torque and motor noise. Both approaches allow the robot to navigate its environment without the help of an explicit movement model.

Physics-based movement. This algorithm can be defined by:

$$v_t = v_{t-1} + a_t * \eta_1 \tag{3.2}$$

$$\tau_t = \tau_{t-1} + \mu * clip(turn_{t-1} * \beta, -1, 1) * \eta_2 \tag{3.3}$$

where 3.2 defines v_t, the agent's rigid body velocity at timestep t. The agent's actions accelerate its rigid body at timestep t through the *forward*, *vertical*, and *lateral* inputs. This acceleration a_t is constrained by factor η_1. Accordingly, 3.3 defines τ_t, the rigid body's torque at timestep t which is increased by a vertical unit vector μ times the turn input $turn_t$ received at timestep t. The turn input is clipped to limit the possible rotation at each timestep by the agent, and is similarly constrained by factor η_2. Our agents use constraints η_1 and η_2 of 0.1 and 0.05.

Translation-based movement. This algorithm can be defined by:

$$x_t = translate(i_{xt}, i_{yt}, i_t) * \eta_1 \tag{3.4}$$

$$r_t = rotate(\mu, clip(turn_t * \beta, -1, 1) * \eta_2 * \Delta t \tag{3.5}$$

where 3.4 defines agent's position x_t in the 3D space at timestep t, which is moved or translated based on the decomposition of the inputs received at each axis. A translation moves an object in the direction indicated by its transform's local axes. Each component from i_t is previously constrained by a movement speed parameter η, measured in meters/second and regulated for smooth transitions by a time variance parameter Δt.

Similarly, 3.5 defines the transform's rotation of the agent r_t at timestep t. This rotation is position-based and done by the game engine given a vertical unit vector and the turn input provided by the agent. As with the physics-based algorithm, the turn input is clipped to limit the possible rotation at each timestep. It is also constrained by a turning speed parameter η_2 and regulated by a time variance parameter Δt to allow smooth rotations in the game engine.

After testing these algorithms, we chose to train our agents using the physics-based algorithm, since it takes into account collisions and momentum, which resemble a real-world robot. The translation-based movement algorithm provides noiseless movement to a degree which would alienate our agent from a real world implementation.

ADDITIONAL COMPONENTS

Finally, collision parameters were also considered in the modeling of both the agent and the environment to enable the detection and avoidance of obstacles. Therefore, the learning agent consists of the following components:

- A mesh renderer that allows the visualization of the textures from the chosen 3D model.
- A physical body which includes a capsule collider and rigidbody components.
- A character controller script that captures and reacts to the input, according to the chosen mobility algorithm.
- An agent controller script, which is required to use the Unity *ML-Agents* plugin.

The agent is therefore defined by a set of components, observations, and parameters that enable perception of both the environment and itself. Firstly, the components included in the agent are a 3D and a 2D grid sensor, enabling a visual input of the environment. Secondly, the metrics that the agent can perceive include its position, velocity, direction towards goals, etc., which register the metrics of its own rigidbody and of the objects in the environment. Finally, additional parameters such as visibility constraints, scan speed constraints and sensor noise allow the agent to learn to adapt different settings and variability configurations. The choice of agent observations is described in the next section.

3.5. SPECIFICATION OF THE REINFORCEMENT LEARNING APPROACH

This section describes selection of a set of reinforcement learning baselines to compare our method to the choice of agent observations for the learning algorithm and the actual processing step. The following questions will be answered in this section:

- What reinforcement algorithm was chosen for the exploration task?
- How was the exploration approach implemented (high-level)?
- What optimization steps were taken in the algorithm?
- What were the main problems found during this phase?

Among the multitude of reinforcement learning methods, this thesis work uses Proximal Policy Optimization (PPO), which was presented in Section 2.6.3. This algorithm was chosen given its demonstrated performance in a variety of tasks [162, 163, 14, 164], its simplicity compared to other implementations [14], and general robustness. A robust algorithm shows stable performance without high dependency on hyperparameter tuning procedures or hyperparameter searches [14].

The neural network used is an implementation of the original by Schulman et al. [14] taken from [143], and it is composed of two hidden layers where the number of hidden units is part of the hyperparameter tuning process. This network was chosen given its demonstrated performance in [165, 166, 81], sample efficiency in comparison with other off-policy methods [164], and the added benefit that its reduced size allows faster training times. PPO requires the approximation of both the state-value function and the policy, which can be done with a shared-parameter network with two heads or two separate networks. We have opted for a shared-parameter network as in the original work by [14].

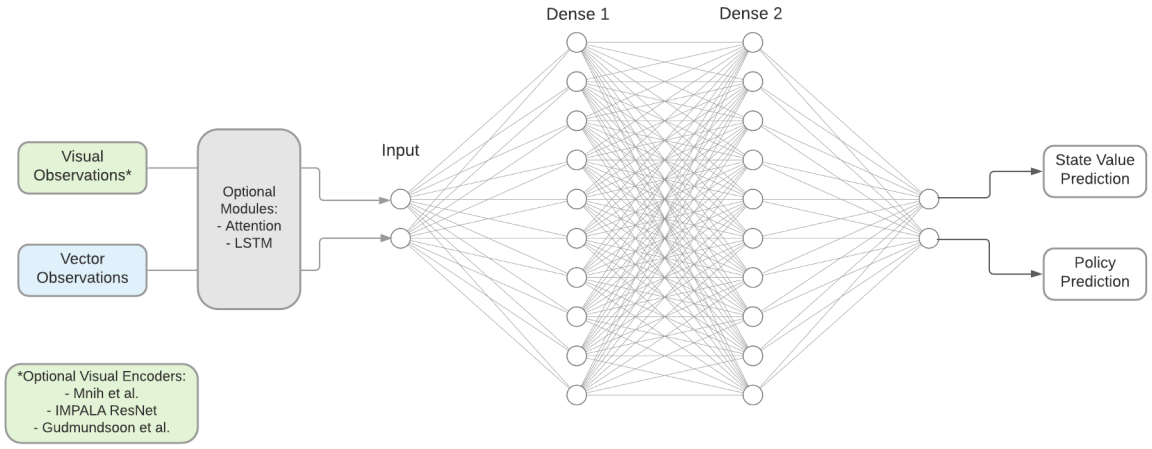


Figure 3.10: Overview of the PPO Trainer's workflow from Unity ML-Agents. Figure adapted from [143].

The workflow for each trainer is presented in Figure 3.10, using Unity ML-Agents [143] as the target platform. The development of the scenes and training of the agent was performed on a Windows 11 machine with an Intel i7-8700k CPU and 32GB RAM. The trainer can receive both visual observations, such as camera, raycasts, grid sensor inputs, and vector observations. Vector observations are any attribute set to observe, such as position, velocity, number of nodes in octree. Additionally, the visual encoder that receives the visual observations can easily be swapped for one of the following:

- A simple encoder with two convolutional layers.
- A CNN implementation by Mnih et al. [167], which contains three convolutional layers.
- The IMPALA ResNet [168], which has three stacked layers, each with two residual blocks.
- The "match3" CNN by Gudmundsson et al. [169].
- A single fully connected dense layer.

Similarly, capabilities such as attention, LSTM, curriculum learning, multi-agent setups and more can be easily changed or added in Unity to construct an agent for a variety of practical reinforcement learning scenarios [143]. Figure 3.11 presents an example of the visual encoders mentioned. For more information on visual observations in ML-Agents, please refer to [170].

The main challenge during this stage was the delay introduced by training a reinforcement learning algorithm, where observations are provided but the performance cannot be estimated until the agent has been trained. Hence, a method with good sample efficiency was also an important factor that was considered when PPO was chosen.

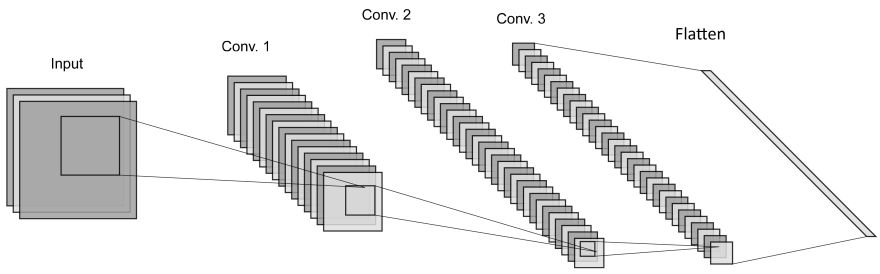


Figure 3.11: Visual encoder example, architecture by Mnih et al. Figure adapted from [167, 143].

3.5.1. CHOICE OF AGENT OBSERVATIONS

To keep the description of the agents concise, some of the chosen observations for the agents are summarized below, without clear separations of which observation belongs to which agent. For a detailed declaration of the observations provided by each derived behavior, please refer to table A.2. This section deals with the intuition, selection, and preprocessing of observations for the learning agent. Therefore, the following questions will be addressed:

- What observations were chosen for the learning agent?
- How do these observations contribute to the agent's learning?
- What were the main challenges encountered during this phase?

The selection of observations and attributes for the agent is critical for the learning method. The aim of an observation is to represent the state s_t of the environment at timestep t. The state s is a function of a set of attributes (pixels of the image, values of variables,...) and can be thought of as an event of the environment, where the values of these attributes have been measured. At each step, the agent receives an observation and updates its knowledge of the environment based on it. Therefore, in order to learn efficiently, an observation needs to contain the right amount of information. Below are the main observations for the agent presented:

- **Own Information**
  - The agent's position: $\mathbf{x} \in \mathbb{R}^3$.
  - The agent's orientation: $\mathbf{r} \in SO(3)$ where $\mathbf{r}$ is represented through a quaternion.
  - The agent's velocity: $\mathbf{v} \in \mathbb{R}^3$ based on the physics rigidbody attributes.
- **Motivation to Explore**
  - An octree as a data structure for the 3D scene.
- **Perception of the Voxelized Environment | Motivation to Explore in Depth**
  - A 3D Grid Sensor, which provides the agent with a panoramic input view, leveraging the benefits mentioned in Section 2.4. It is visualized in Figure 3.7.
- **An Inactivity Metric**
  - A linger parameter, which measures the slowness of the agent while staying in one place, given its velocity.

- **General Information about Targets perceived through the Grid Sensor**
  - An *orientation vector*, directed towards targets in view.
  - A distance vector to targets in view.

Additional attributes that compose the agent class, which are not observations include:

- **Attributes related to the scanner**
  - The scanner component that scans voxels given the scan constraint variables.
  - A scan accuracy parameter to simulate noise in the scanner.
  - A scan reload time to limit the speed of the scanner.
  - A scan range constraint.
  - A scan angle constraint.
- **Attributes related to the Grid Sensors**
  - A visibility range constraint for the objects seen by the 3D grid sensor component. This variable also intervenes in the addition of empty points to the octree
  - A visibility angle constraint for the objects seen by the 3D grid sensor component.
- **Attributes related to Mobility**
  - A parameter to specify the mobility algorithm.

One of the main problems that was encountered during this phase was finding the appropriate attributes that provide enough information to the agent about its environment while also tackling the challenge of exploration based on a different kind of input (voxels). A decisive factor to choosing this approach were the direct benefits obtained through the components and the nature of the approach. The understanding of a voxelized world abstracts the environment from the drone's perspective and makes its behavior visual-agnostic. Similarly, the grid sensors outperform both raycasts and cameras in the completeness of the information perceived and in drastic reductions of computation, memory and training costs, respectively. As mentioned in Section 3.3.2, another benefit acquired from using grid sensors is that the training environment can be executed in headless mode to relieve rendering bottlenecks. The remaining observations motivate the agent to behave differently according to different alternative secondary rewards provided, such as the minimum movement speed or a moving forward reward. These differences in behavior are further discussed in Section 5.

3.5.2. RESETTING THE ENVIRONMENT

In our setup, an episode can reach its end when one of three conditions are met: a) the agent scanned all voxels in the scene, b) the maximum steps were reached, and the agent ran out of time or c) the agent collided with an object, which earns a negative reward. Once one of these conditions is met, the environment resets, which randomizes the positions and orientations of all 3D assets in the scene. This includes the agent and all goals.

3.5.3. GOAL AND REWARD SIGNAL

Given the presented concepts of voxelization, octree navigation, and semantic entropy, the proposed agent's goal is therefore defined as

$$\max R = R^{VOX} + R^{OCT} + R^{COL} + R^{LIN} + R^{X} \tag{3.6}$$

where R^{VOX} is the reward given by the voxel scans, R^{OCT} is the reward from adding new nodes to the octree and R^{COL} is the penalty the agent receives when it collides with an object. Accordingly, the penalty R^{LIN} is granted given a linger estimator, which is a function of the velocity of the agent and the entropy perceived by the agent. The linger estimator penalizes the agent proportionally as its velocity decreases, but this factor is reduced by the entropy captured. More concretely,

- R^{VOX} is initially set to 0.01 per voxel that is scanned by the agent. A voxel counts as scanned if the scanner is within the constraints given by the scanner variables (angle, distance).
- R^{OCT} is initially set to 0.01 per new node discovered, obtained by calculating the intersect of the current octree with the previous octree.
- R^{COL} is initially set to -1, granted on collisions with objects or walls.

- R^{LIN} is a linger penalty function taken adapted [159], that penalizes the agent given

$$R^{LIN} = -t_n * S^{LIN}(e) * r_l \tag{3.7}$$

where the penalty is directly controlled by t_n, which is a metric for how many timesteps the agent is at a given octant in the octree. It is constrained to a value range of $[0,2]$ and it resets when the agent moves to a new octree node. Similarly, the strength of the penalty is given by the *lingering penalty strength* $S^{LIN}(e)$, which is defined by

$$S^{LIN}(e) = \begin{cases} \frac{1}{1+e^2} & \text{if is } \textit{semantic-entropy-agent}, \\ 1 & \text{else} \end{cases} \tag{3.8}$$

Accordingly, e, also known as *RationalizedClassEntropy*, is defined as

$$e = \Sigma_{(i)\in(M)} \left(M_t^{Sem}[i] - M_{t-1}^{Sem}[i] \right) \tag{3.9}$$

where our definition of e was taken from the semantic curiosity reward proposed by [9]. Furthermore, $\left(M_t^{Sem}\right)$ is the temporal buffer (also known as semantic map) and M is the length of the temporal buffer. The semantic map allows the agent collect the number of classes detected by the object detector over the past N timesteps. We used a temporal memory with size $N = 10$. Finally, the lingering penalty is also regulated by penalty weight r_l, set to 0.001.

- R^X encompasses a variety of other rewards and penalties that were tested during the development process, such a reward for moving towards a target, a timestep penalty, a reward for moving forward and a reward for moving at speed higher than a given minimum. Furthermore, a mechanism was implemented and analyzed to constrain these penalties if voxels are in field of view (FOV) of the agent, with the expectation that the agent is able to stay longer at certain octree nodes to scan objects.

The main challenge in the definition of the reward signal was the subtle balance between rewards and penalties. For example, large negative rewards could lead to unstable or a very slow learning process because they indicate an absence of positive learning experience [75]. To this end, the initial values proposed above are evaluated and criticized in Chapter 4.

3.5.4. IMPLEMENTATION

To summarize, the agent is constructed of a set of actions, sensors, and a scanner component that allow him to navigate, perceive, and interact with its environment. Furthermore, the agent is rewarded through the scanning of voxels and the navigation of the environment (addition of new nodes to the octree). It also takes into account the semantic entropy in the environment to modify the linger estimator's penalty, which gives the agent the chance to scan more exhaustively. The data structures, the chosen observations provided to the agent, and the learning of an exploration policy were also covered.

The environment and the agent are implemented in Unity using C# and the Unity ML-Agents platform. The workflow of a training session was adapted from [140] and is portrayed in Algorithm 3.

Algorithm 3: PPO Trainer Workflow

```
initialize agent and weights θ;
while train do
    reset environment and gather initial observation S;
    for time step t=0,1,..., T do
        let agent choose action A based on state S;
        update graphics engine according to action A;
        gather observation S′;
        calculate reward R;
        calculate advantage Â_t;
        check if episode is done;
        S < −S′;
    end
    Update weights with PPO;
end
```

3.6. INTERPRETATION

As described in the previous sections, the agent is defined through a set of observations that allow it to perceive voxels, new octree nodes, obstacles and 3D assets in the environment and the level of semantic entropy in the environment. It is also defined through a reinforcement learning agent that given the agent's observations, maximizes the reward on a given state, to solve two main tasks:

- Exploration of the Environment, achieved through the octree observations and the associated reward. The lingering penalty also plays a critical role in this behavior to prevent the agent from staying at an octree node for too long.
- Exploration of Objects (voxel-curiosity),achieved through the trajectories that the agent traverses while scanning voxels i.e., the agent is capable of looking at objects from multiple perspectives given its perception of the voxels that comprise the object of interest and the reward it obtains through the scanning of such voxels.

The set of agent behaviors we propose and analyze, that solve one or both tasks, can be categorized in the following:

- **Object-focused:** these agents have knowledge about voxels, i.e., they are capable of seeing voxels in the grid sensor. Their goal is to maximize the amount of voxels discovered in a scene.
- **Environment-focused:** refers to agents that have knowledge about octrees through information about nodes, lingering status and pigeon observations. Their goal is to maximize the space covered in a scene without direct knowledge of a map structure. From the concept of "coverage maximization", it would be similar to the work by Chen, Gupta, and Gupta [13], however we do not provide the agent with a map or information about how much it has explored so far.
- **Mixed-focus:** these agents possess both knowledge about voxels and octrees, similar to the two previous agent styles. Their goal is to find a balance between exploration of the environment and of objects.
- **Baselines:** these agents are meant to critically evaluate our approach from a research perspective and compare the usage of voxels and octrees for exploration to other relevant efforts in the reinforcement learning for exploration research. They do not posses knowledge about voxels or octrees, in order to provide insight of the value of voxels and octrees to both research questions. Following the work done by Chaplot et al. [9], the baselines considered suitable for this task are presented below and justified:
  - **Random Walk.** A baseline that samples an action randomly, used to evaluate both object- and environment-focused agents.
  - **Shortest Path.** This baseline provides the agent with the look angle and the walking angle to the nearest unvisited object. It is used to evaluate both object-focused agents given its priority for objects.
  - **Prediction Error Curiosity.** Based on the work by Pathak et al. [52], it provides the agent with intrinsic curiosity towards novel states. This is used to evaluate and environment-focused agents.
  - **Object Exploration.** This is a naive baseline where the reinforcement learning algorithm maximizes the amount of object detections given by a YOLO detector. As pointed out by Chaplot et al. [9], this policy is expected to learn to search for frames with more objects but not for frames with different objects over time.
  - **Semantic Curiosity.** This baseline is implemented based on the work by Chaplot et al. [9], and uses their proposed semantic curiosity to prefer trajectories with high temporal inconsistencies in an object detector. Following the original method, this agent maximizes the semantic curiosity reward using formula 3.9 and $\lambda_{SC} = 2.5 \times 10^{-3}$.
- **Proposed methods:** these are our proposed methods that incorporate a mixed-focused agent logic with the semantic entropy concept.
  - **Semantic Entropy.** this agent works upon the *Semantic Curiosity* method by Chaplot et al. [9] by using high temporal inconsistencies in the object detector translate to adjustments to the agents lingering penalty strength. This is expected to allow the agent not only to explore both

environments and objects, but also to prefer states with low lingering penalty and more objects. These agents implicitly observe the levels of semantic entropy through $S^{LIN}(e)$ (see 3.8).

– **Semantic Entropy as an observation.** This method is similar to the previous one by providing the semantic entropy observations to the agent, but does not adjust the lingering penalty strength.

In order to determine if the environment or the agent required further tuning or adjustments, the average accumulated reward across all runs was the initial metric to distinguish a good from a bad performance. A low or negative reward can point to a behavior that is stuck, spinning or does not explore to discover new states with higher rewards. Similarly, a high reward can indicate a good performance but also reach undesired states. For example, after scanning most of the voxels, the agent can sometimes fall into a sparse reward situation, which complicates learning [75]. On one side, the movement speed, turning speed, and constraint attributes related to the agent's perception were observed to analyze the agent's performance. On the other side, the agent's mobility algorithm was also observed and tuned to determine the added complexity of learning a physics model through function approximation.

As agents become more complex, the interpretation of the performance required more meaningful metrics, which allowed to distinguish the influences of different observations and rewards in the agents' behavior. For example:

- Total objects detected.
- Total voxel detected.
- Total octree leaf nodes discovered.
- Octree nodes discovery rate.
- Behavior of the lingering penalty
- Behavior of the Shannon entropy.

The full list of *in-depth metrics* is presented in the Appendix A.6.6. Finally, visual inspection also supported the detection of faulty behavior, such as when the agent is stuck or spinning uncontrollably.

3.7. EXPERIMENTS

3.7.1. ENVIRONMENT AND VISION SETUP

To carry out the experiments, a set of learning environments were required to train, collect metrics and test the different agent behaviors. These behaviors derivated as a result of different observations, hyperparameters, policies and reward signals. To analyze changes in performance given a larger training environment, we compare the agents' performance between a *small* environment of 32 m^2 and a *large* environment of 160 m^2. Figure 3.12 illustrates these two environments, the exploration boundaries (colliders) and the added obstacles for collision avoidance.

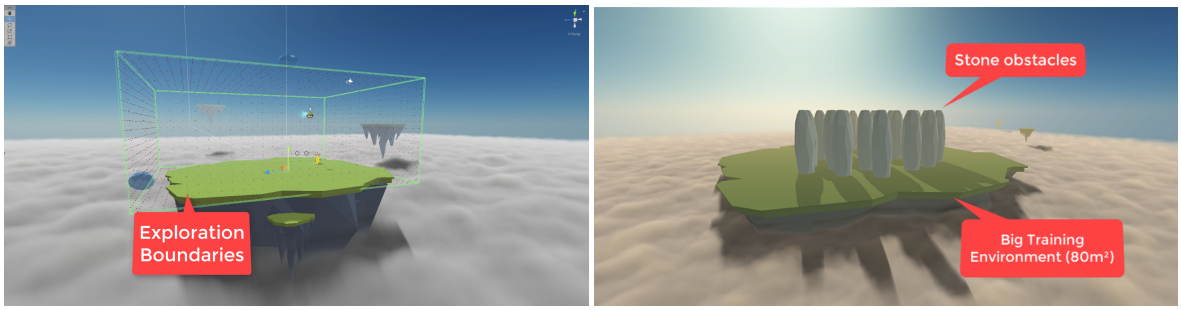


Figure 3.12: Small island ($32m2$) with exploration boundaries (left). Large environment ($160m^2$) with stone obstacles (right). 3D assets were modeled our taken from [1].

Each agent behavior presented was trained for **30 million timesteps** using PPO and variations of the hyperparameters presented in A.6.3. Following the 80-20 rule, test runs had a duration of 7.5 million timesteps. Tables A.2, A.3 present a complete overview of the different agent variants trained. The trainer's hyperparameters that were tuned are listed in the following Table 3.1. A full list with their default valuescan be found in Appendix A.6.3.

Tuned Hyperparemeters		
visual encoder type	number of epochs	learning rate
batch size	max steps	buffer size
time horizon	number of layers	hidden units
beta	lambda	epsilon

Table 3.1: Tuned hyperparameters of the PPO trainer during the training-feedback loop, taken from [143]

After the definition and setup of the environment, the next step is the setup of the vision system. As described in more detail in the previous chapter, the approach to perception was to use the grid sensors from [143, 158]. Figure 3.13 visualizes the range of the two grid sensors used in the agent and the voxelized view they produce.

Figure 3.13: Integrated grid sensor used for the agent to perceive objects, boundaries and collectibles (left). Grid sensor used by the agent as a voxel-scanner (right). Voxelized-view perceived through the grid sensors is also shown.

In our setup, one grid sensor is used to visualize elements in a panoramic fashion within a radius of 16 meters, where boundaries, obstacles and collectibles can be detected. A second and smaller grid sensor with a diameter of 6 meters acts as the scanner in the voxel-exploration task: this grid sensor mimics a "scanneable" area and has a fail-rate of 20% to provide the agent with an imperfect scanning behavior. Voxels that fall into this grid sensor's area are marked as scanned and accumulated by a counter to reward the agent at timestep t.

3.7.2. BEHAVIORS SETUP

The setup of the agent behaviors for the exploration of octrees and voxels is comprised by the set of observations and rewards provided to motivate a certain policy. The octree exploration policy was motivated by the approach presented by Chen, Gupta, and Gupta [13], where each discovered node in the octree provides a certain reward to the agent. To analyze the behavior of different octree setups, diverse experiments were run with varying from nodes of 4 m^3 to 16 m^3. Therefore, the rewarding styles for octree exploration policies vary according to the size of the octree nodes the agent can discover.

Additionally, another set of environment-focused agents take into account observations called *pigeon observations*. These observations take inspiration of the physiology used by pigeons to orient themselves in their environment, memorize routes and keep a biological compass. They consist of two vectors: one towards a geographical north and one towards a sun placed on top of the training island.

Finally, following the example given by Chaplot et al. [9], we analyze the performance of a "blind" octree variant that is curious for new states following the approach proposed by Pathak et al. [52] and then compare it to the octree methods that follow Chen, Gupta, and Gupta [13]'s approach. Moreover, as motivated by Unity Technologies [143], we go one step further and integrate Pathak et al. [52]'s parameters into the agent's model to construct our proposed explorer drone, which takes into account all three: 1) octree observations, 2) pigeon observations and 3) pathak's curiosity for new states. In summary, in addition to the information presented about the choice of agent observations in Section 3.5.1 and the baseline agents in Section 3.6, the diverse setups for environment-focused agents are a combination of the following:

- Agents with a reward for discovering 4-, 8-, 16-meter-wide octree leaf nodes.

- Agents with one or more of the following environment observations:
  - Amount of octree nodes discovered at timestep t.
  - Pigeon observations.
  - Lingering factor.
- Agents penalized given their lingering behavior.
- Agents with Pathak's curiosity module (main feature of the **Pathak** baseline agent).
- Agents with a minimum speed penalty.

It is worth clarifying that **blind** octree agents receive a reward for discovering new nodes but do not receive any of the aforementioned observations. Furthermore, mentioned in Section 3.6, Pathak's method was initially proposed for the exploration of environments but is now incorporated as a curiosity module

Similarly, the voxel exploration policy was motivated by a rewards-per-voxel fashion, where the diverse policies vary according to the reward per voxel, ranging from a 25% reward per voxel to a 100% reward per voxel. The diverse setups for object-focused agents are a combination of the following:

- Agents with a reward for discovering a voxel with a reward by a factor in the range: 25%, 50%, 75%, and 100%.
- Agents with, alternatively, a normalized voxel reward given by $R^{VOX} = \frac{voxels\_scanned}{2+voxels\_scanned}$.
- Agents with one or more of the following environment observations:
  - Vision of voxels in the grid sensor.
  - Lingering factor.
  - Pigeon observations.
  - Walk and look angles towards nearest object (main feature of **oracle** agents and the **shortest-path** baseline agent).
  - Number of classes detected by the object detector (main feature of the **object detector** baseline agent).
  - Rationalized class entropy (main feature of the **semantic curiosity** baseline agent).
  - Lingering penalty strength (observed and regulated in our **voxel-entropy** agent).
- Agents with a minimum speed penalty.
- Agents with a lingering penalty.

3.8. FURTHER USE OF RESULTS

As stated by Luckert and Schaefer-Kehnert [142], the last step of every knowledge discovery process is the application of the achieved results for further use. In this thesis work, this is of special importance, where the main focus was the exploration of unknown 3D environments and the actual performance of our approach compared to a set of baselines is carried out. This section introduces the baselines that were determined to evaluate our approach, the environments were the exploration policy is deemed most useful, and the utility provided by the development with ML-Agents to perform cross-environment exploration.

3.8.1. EVALUATION FRAMEWORK

For the evaluation of the baseline methods and the proposed methods, we constructed two variations of the open world environment. These environment serve as testing grounds for the best performing methods from the preliminary trainings. These "DARPA Unity Scenes" are identical in dimensions and modeling but they evaluate two separate behaviors. The first environment evaluates the object-exploration capabilities of the agent through a sequential three-goal setup as shown in Figure 3.14.

The DARPA subterranean challenge [15] inspired the concept for these environments, where the following metrics were collected to evaluate the performance of the object-exploration behavior:

- Time to object$_i$.
- Total time to object$_i$.
- Seconds per meter to object$_i$.
- Total seconds per meter to object$_i$.

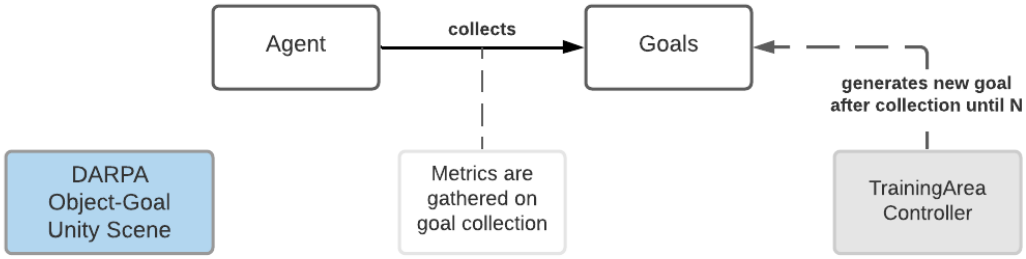


Figure 3.14: Visual diagram of our DARPA environment for the evaluation of the object-exploration capabilities of an agent in a sparse-reward setup.

The second environment evaluates the environment exploration behavior without any goals in the scene. The metrics collected are the times to achieve 10%, 20..., 90% coverage of the scene. This is meant to demonstrate that the agent is capable of exploring multiple locations in the environment. Consequently, the evaluation procedure is done through a bracket qualification system as shown in Figure 3.15. Firstly, the best models from the preliminary training are selected. Secondly, a variable analysis is done for each of the agent behaviors: object and environment exploration. This analysis provides insight into which variables are most beneficial to the agents according to the performance observed through the *in-depth* metrics. Finally, the qualifying runs are then evaluated on the DARPA environments where the relevant metrics for each research question are collected.

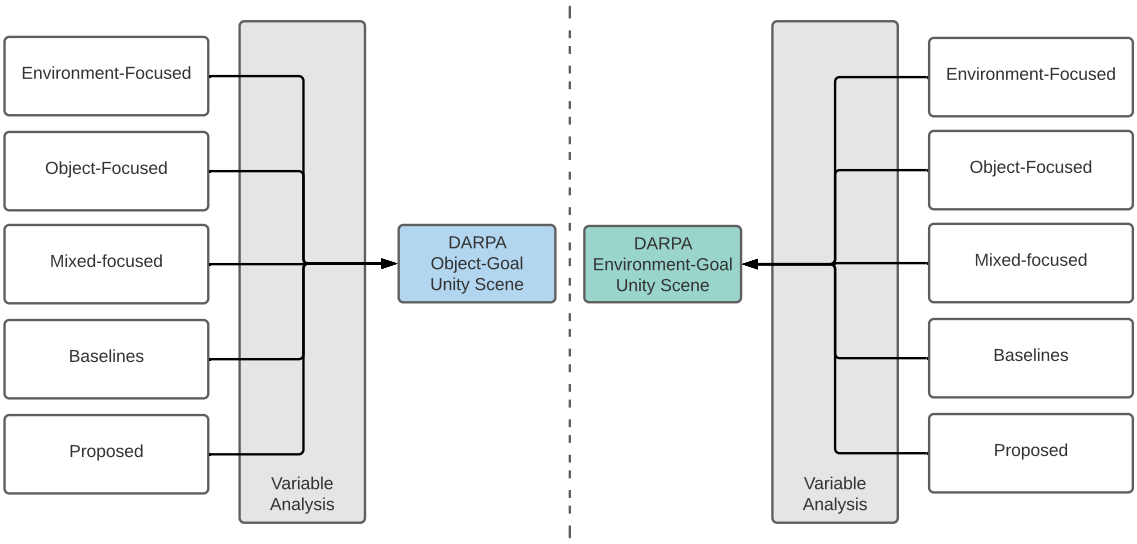


Figure 3.15: Our qualification bracket system for the evaluation framework, using the DARPA-inspired testing environment.

3.8.2. USABILITY

APPLICABLE PRACTICAL SCENARIOS

Part of the value behind the research questions also is reflected in the applicability of the proposed methods to a variety of scenarios. Therefore, multiple scenes were considered to demonstrate the generalization of our agent's behavior to new environments. Example scenes include a stable, a traffic accident, a neighborhood fire, a forest accident, etc.

CROSS-PLATFORM PERFORMANCE

Further value of using ML-Agents can be demonstrated through the capability to transfer developed environments to another environment platform. OpenAI [146] was chosen as the target platform, given that it is one of the go-to platform in the reinforcement learning community, with a multitude of configuration settings, algorithms, and compatibility with other frameworks [146].

The trainer used was the Stable Baselines 3 [17] trainer, developed by the German Aerospace Center (DLR) and the Institute of Robotics and Mechatronics (RM). Accordingly, a similar average accumulated reward and a *low transfer cost* across environments was treated as a high indicator for a high compatibility between environment platforms. The transfer cost was defined as the amount of hours and lines of code required to reuse a Unity Environment in OpenAI Gym.

ALGORITHMIC PERFORMANCE

Finally, we compare the performance of PPO to a state-of-the-art deep reinforcement learning algorithm, Soft Actor-Critic. On one hand, where vanilla RL implementations suffer from a large variance of the learned policy due to the policy update on-policy, PPO is a well-rounded algorithm that exhibits remarkable performance on a variety of tasks, as described in Sections 2.6.3 and 3.5. On the other hand, SAC is known to improve sample efficiency compared to on-policy methods and to reduce the variance through a more complex policy update (critically) and an off-policy correction step. Subsequently, SAC's off-policy training advantage is also important to analyze, because real-world problems, such as robot control, often have data from a non-replayable domain.

4

RESULTS

This chapter will provide an overview of the achieved results, the determined baselines, the setup of the experiments, the nature of the environment and the experiment process to solve the posed research question. This chapter also adapts the structure for machine learning algorithms proposed by Luckert and Schaefer-Kehnert [142]. Section 4.1 presents the experiment results, Section 4.2. Afterwards, Section 4.3. Finally the deliverables for this thesis work are presented in Section 4.4. The following questions will be answered thoroughly:

- Which observations and rewards were the most influential?
- What were the achieved results on each experiment?
- What was the setup of the best performing algorithm?

As we approach the presentation of our results, it is important to step back for a moment to look at our approach from two perspectives. The initial presented perspective looks at the problem as an object uncertainty problem, where our interest is to look at objects from multiple perspectives to reduce our *uninformativeness* about them. To this end, we proposed a reinforcement learning agent that explores environments to find such objects of interest. However, the second perspective for our proposed method is a navigation perspective, which was implicitly presented through our related works. As mentioned before, visual navigation solutions can be categorized depending if the location of the goal is known or unknown [9]. Among these, the commonly studied point-to-goal task presents two fallacies

- the location of the goal is known (either implicitly through a path or explicitly through coordinates), which is not the case in, for example, rescue missions.
- the goal is not situated very far from the target (exhaustive exploration is not required), which conceals the real exploratory performance of embodied algorithms.

Not only do traditional navigation approaches depend on map awareness, to subsequently execute some sort of path-planning algorithm, but real life problems such as rescue scenarios or dynamic environments, provide the agent with limited, changing, partial observability of the environment. This has laid the path for mapless solutions to take over classical navigation approaches [171, 172, 173]. Our approach takes this into consideration to account for the partial observability in an unknown environment, where the location of goals are also unspecified.

Moreover, traditional approaches subdivide the navigation problem into sub-modules for obstacle avoidance, map creation and then path planning. Accordingly, newer approaches avoid the pitfalls of traditional, complex model-based approach which attempt to model the dynamics of the real world and carry some sort of bias. They also solutions leverage deep learning methods to provide end-to-end solutions to the navigation problem [65, 173]. We therefore proposed a model-free method that leverages neural networks and that is capable of adapting across environments.

All of these methods, however, usually depend on the quality of the sensorial data obtained to reconstruct

the world to be navigated. Dense mapping systems are usually victims of noise, outliers and missing data, and must even balance runtime performance and reconstruction quality. Moreover, these methods either ignore moving targets or use expensive raycast operations to construct maps of the dynamic objects in the scene. [174, 173, 175] This is the pinnacle for this thesis, where we propose the usage of voxels to provide our agent with robustness to outliers, noise and missing data, and octrees for the efficient, mapless, exploration of changing environments. Furthermore, we take inspiration from the physiology of pigeons' biological compass to provide our agent with a mapless technique to orient itself in unknown environments.

Finally, as mentioned before, given the increasing momentum of reconstruction methods and synthetic data to fuel other machine learning pipelines [176, 177], we decided to exploit Unity 3D to train and test our agent in multiple 3D environments. This allows direct portability and extensibility of our agent to new methods, new scenarios, further benchmarks, etc. Without further delay, the following sections present the the training and the DARPA test results.

4.1. RESULTS

The following subsections present the results of the experiments.

Section 4.1.2 deals with the results concerning the exploration of objects and 4.1.1 deals with the results concerning the exploration of unknown spaces using octrees. Each subsection will present the best results obtained on the exploration policy variants and a brief description of the most representative details of each variant.

4.1.1. RESEARCH QUESTION 1

The first research question focuses on the reduction of uncertainty about objects, which we have posed as a voxel-curiosity task. Below, the best results for each voxel-curious variant are be presented in two subsections according to if the model has knowledge of voxels. It is worth mentioning that the training environment provided the agent 6 goals per episode situated at a distance of 5-50 meters away from the origin. In contrast, the DARPA testing environment contained a single goal located between 30 to 50 meters away from the origin. This sparse-reward testing environment evaluates the actual exploratory capabilities of an agent. For information on the run names, please refer to the previous Section 3.7.2 and Appendix A.6.4.

AGENTS WITH KNOWLEDGE OF VOXELS

The following Table 4.1 shows a brief overview of the results at the top 50 percentile based on the average amount of objects scanned. More information about all methods are presented in the tables afterwards.

Method	Episode Length %	Average Total Objects Scanned	Standard Deviation
voxel++025	94	1.56	0.22
voxel++100-nospeed	98	1.52	0.24
voxel++100-nospeed-nolinger	97	1.73	0.20
voxel++100-nospeed-nolinger-oracle	96	1.33	0.20
voxel++100-nospeed-nolinger-oracle-8	99	1.77	0.21
voxel++100-nospeed-nolinger-pigeon-oracle-8	97	1.30	0.24
voxel++100-nospeed-oracle-4	100	1.64	0.19
voxel++100-nospeed-oracle-8-pigeon	93	1.68	0.23
voxel-entropy++100-nospeed	97	2.10	0.20
voxel-entropy++100-oracle	96	1.27	0.20
voxel-entropy++100-oracle-nospeed	98	1.59	0.20

Table 4.1: Overview of the best object-focused runs with knowledge of voxels respect to the *Episode Length* and the *Total Objects Scanned* metrics.

As mentioned before, a small episode length indicates that the agent is overwhelmed by the penalties and the fastest way to minimize the reward function is to end the episode. An object is marked as scanned once all voxels that belong to it are scanned by the agent.

AGENTS WITHOUT KNOWLEDGE OF VOXELS

The following Table 4.2 presents the aggregated results for the agents that get rewarded for finding voxels but are not able to see them in the grid sensor view. These agents must rely on their other information to find objects and their voxels, such as shortest path angles (look angle and walk angle), the output of an object detector, the temporal semantic map in their trajectories [9], the semantic entropy in the environment, or no information at all (blind agent).

Method	Episode Length %	Average Total Objects Scanned	Standard Deviation	Total Detections Count
random-agent	89 %	0.14	0.15	10
shortest-path	97	0.06	0.04	34
semantic-curiosity	99	0.06	0.05	73
semantic-entropy	39	0.13	0.17	21
blind-agent-voxel-constrained	100	0.31	0.11	31
blind-agent-voxel	99	0.33	0.10	40
object-detector	96	0.35	0.33	55
object-detector-pure	93	0.94	0.25	38

Table 4.2: Overview of the best object-focused runs without knowledge of voxels. Total Detection Counts are in tens of thousands.

More detailed tables of voxel-aware agents can be found in Appendix A.6.8.

4.1.2. RESEARCH QUESTION 2

This section deals with the results related to question 2, which focuses on the exploration coverage of a 3D space given octrees as the 3D data structure for the space. The first two tables present these results through the octree discovery reward.

AGENTS WITH KNOWLEDGE OF OCTREES

Table 4.3 summarizes the results for the agents that are aware of octrees for the given task. This can be through feedback of how many nodes were discovered each timestep or through pigeon observations.

Method	Episode Length %	Octree Leaf Nodes Visited	Octree Leaf Nodes Visited %	Visited Volume (m^3)
octree-4	92	87.16	2.18	348
octree-4-pigeon	17	10.20	2.04	81
octree-4-pigeon-pathak	22	12.32	2.46	98
octree-8	43	22.90	4.58	183
octree-16	4	1.69	2.71	27
octree-16-constrained	100	12.70	20.32	203
octree-16-constrained-pigeon	100	20.35	32.56	325
octree-16-constrained-pigeon-pathak	100	6.07	9.71	97

Table 4.3: Overview of the environment-exploration focused runs with knowledge of octrees.

AGENTS WITHOUT KNOWLEDGE OF OCTREES

Pathak's method [52] usage is two-fold. First, without knowledge of octrees, it is used as one baseline to compare the exploration performance to our method. Secondly, it is also used as a curiosity module for other agents, as suggested by Unity ML-Agents [143], given that it motivates the agent to prefer newer states.

The results for the environment-focused methods that have no knowledge of octrees are presented below in Table 4.4.

Method	Episode Length %	Octree Leaf Nodes Visited	Octree Leaf Nodes Visited %	Visited Volume (m^3)
random-agent	100	0.01	0	0
blind-octree-explorer	91	86.40	2.16	345
blind-octree-explorer-nospeed	95	91.82	2.30	367
blind-octree-explorer-nospeed-pathak	91	88.52	2.21	354
pathak-4	90	79.74	1.99	318
pathak-8	65	30.48	6.10	243
pathak-16	4	1.66	2.66	26
pathak-16-constr-nospeed	100	1.01	1.62	16
pathak-16-constr-nospeed-nolinger	100	19.05	30.48	304

Table 4.4: Overview of the environment-exploration focused agents that have no knowledge of octrees.

The performances can be contrasted based on the number of scan nodes, which originate from objects of interest or obstacles (stones) in the environment, the number of leaf nodes and the lingering behavior. The $160m^2$ environment consists of 40, 20, 10 octree nodes on the XZ-axis when segmented with node sizes of $4m^3$, $8m^3$ and $16m^3$, respectively. Since the vertical space is not useful for our use case (objects are not situated at different heights), the Y-axis is limited to 5, 2.5 and 1.25 nodes, respectively. This is equal to 3D volumes of 4000, 500 and 62.5 nodes, respectively. Table A.6.10 presents more details for the results for all environment-focused agents.

4.1.3. MIXED-FOCUS AGENTS

This sections presents the variant of agents that have both knowledge of the environment and voxels. Information about the environment can be present in the form of octree nodes or *pigeon observations*. We propose an adapted F1-score [178] to calculate the harmony mean between the total scanned objects and the visited volume. This provides a way to evaluate agents based on their performance on both tasks using only one statistic. The following Table 4.5 displays the results for the mixed-focus agents with their respective episode length, average amount of scanned objects, F1-score and percentage of octree leaf nodes visited.

Method	Episode Length %	Total Objects Scanned	F1-score	Visited Volume (m^3)
explorer-entropy-4	97	0.54	0.21	310
explorer-entropy-4-nolinger-noTrainEntropy	97	0.17	0.08	281
explorer-entropy-4-noTrainEntropy	98	0.00	0.00	358
explorer-entropy-8	88	1.96	0.30	151
explorer-entropy-8-nolinger	96	1.77	0.24	117
explorer-entropy-8-nolinger-noTrainEntropy	98	1.83	0.25	120
explorer-entropy-8-noTrainEntropy	92	1.65	0.27	139
explorer-entropy-16	95	0.22	0.05	31
explorer-entropy-16-nolinger	100	0.78	0.14	74
explorer-entropy-16-nolinger-noTrainEntropy	97	1.18	0.16	79
explorer-entropy-16-noTrainEntropy	99	0.23	0.05	32

Table 4.5: Brief overview of the mixed-focused runs, with respect to the *Total Objects Scanned*, *F1-score* and *Leaf Nodes Visited*.

More details related to these agents can be found in Appendix A.6.11.

4.2. EVALUATION FRAMEWORK

This section presents the performance of the best performing agents that reached the *DARPA* evaluation challenge presented in Section 3.8.1. These are done in the DARPA evaluation scenarios, where they test

- how good the agent is at finding objects and fully exploring them (abbr: obj-goal).
- how good the agent is at maximizing octree exploration coverage (abbr: env-goal).

To this end, the evaluation candidates, as presented in Figure 3.15, were selected from the following agent categories:

- object-focused
- environment-focused
- mixed-focused
- baselines (such as the shortest-path agent, object detector agent, etc.)

The test candidates are shown the following Table 4.6.

Method	**F1-score**	**Total Objects Scanned**	**Visited Volume (m^3)**	**DARPA Scenario**
blind-octree-explorer-constrained-nospeed	0.00	0.00	367	env-goal
octree-16-constrained	0.00	0.00	203	env-goal
octree-16-constrained-pigeon	0.00	0.00	325	env-goal
octree-4	0.00	0.00	348	env-goal
voxel++025	0.26	1.43	150	both
voxel++100-nospeed	0.25	1.47	144	obj-goal
voxel++100-nospeed-nolinger	0.25	1.62	138	obj-goal
voxel++100-nospeed-oracle-8-pigeon	0.25	1. 57	12	obj-goal
shortest-path	0.02	0.06	63	obj-goal
object-detector-pure	0.18	0.94	110	obj-goal
semantic-curiosity	0.02	0.06	54	env-goal
semantic-entropy	0.05	0.13	63	obj-goal
voxel-entropy++100-nospeed	0.29	2.10	147	env-goal
explorer-entropy-16	0.05	0.22	31	both
explorer-entropy-8	0.29	1.96	151	env-goal

Table 4.6: Best runs across all agent categories with their respective relevant metrics and the DARPA scenario in which they will be tested. For example, *voxel++025* is tested in the object-exploration scenario. White letters represent the highest runs from their category.

Given that the general performance of agents has already been registered during training (for both object and environment exploration), not all the best runs are candidates for both tests. Therefore, agents that have no motivation for voxels, such as *octree-4*, are not included in the candidates for the Object-Test. Similarly, voxel-focused runs such as *voxel++025* are not considered candidates for the Environment-Test.

As mentioned in Section 3.8.1, one DARPA environment evaluates the object-exploratory performance of the agent, by situating a goal randomly in the scene until the agent has collected 3 goals. The times to each goal and the seconds per meter are collected to evaluate how fast the agent reaches the goal. This also evaluates which run is the fastest to the first goal, and which one is the first one to reach the last goal.

Finally, each test is considered an episode, where each episode can run up to 40 thousand timesteps and a total of 7.5 million timesteps were tested. On the object exploration environment, a test (episode) is completed when all three goals were collected. On the octree exploration environment, the test is completed when 100% of the environment was discovered.

Table 4.8 displays the results for the voxel-goal task, with the different times to each three objects, and the average seconds per meter.

Method	Avg. Time to Goal #1	Avg. Time to Goal #2	Avg. Time to Goal #3	Avg. Time to Goal	Avg. s/m btw. Goals
voxel++025	234	294	-	-	4
voxel++100-nospeed	145	187	63	132	2
voxel++100-nospeed-nolinger	233	164	-	-	3
voxel++100-nospeed-oracle8-pigeon	91	108	130	110	2
shortest-path	132	-	-	-	2
object-detector-pure	197	353	-	-	4
semantic-entropy	321	-	-	-	5
voxel-entropy++100-nospeed	105	128	81	105	2
explorer-entropy-8	82	106	80	89	1
explorer-entropy-16-nolinger	193	116	162	157	2

Table 4.7: Overview of the results in the DARPA objects-environment for the best voxel-focused finalist runs. The *Average Time to Goal* is not reported for runs that were not able to scan all 3 goals.

The second DARPA environment evaluates the environmnet exploration capabilities of each agent, by measuring the time it takes to discover 10%, 20%,..., 100% of half of 3D volume. This allows the objective evaluation of how fast and how much actual space the agent discovers. We only evaluate half of the volume since objects of interest are located at the ground level and our memory constraint of 15 GB limits the maximum amount of timesteps to 40 thousand. Table 4.8 shows the results for the best methods tested in the octree-goal task with the Time to Coverage (TTC) relevant percentages of the environment.

Method	TTC 10%	TTC 20%	TTC 30%	TTC 100%
blind-agent-explore-constrained-nospeed	256	591	755	-
octree-16-constrained	83	387	793	-
octree-16-constrained-pigeon	69	-	-	-
octree-16-constrained-pigeon-nospeed	44	117	180	461
octree-4	287	690	-	-
octree-4-constrained-pigeon-nospeed	307	665	-	-
voxel++025	-	-	-	-
semantic-curiosity	-	-	-	-
voxel-entropy++100-nospeed	-	-	-	-
explorer-entropy-16	93	249	296	-
explorer-entropy-16-constrained	43	90	146	510
explorer-entropy-16-constrained-noTrainEntropy	139	349	512	-
explorer-entropy-8	122	332	622	-
explorer-entropy-8-constrained	118	268	422	-

Table 4.8: Overview of the results in the DARPA exploration-environment for the best octree-focused finalist runs.

4.3. FURTHER EVALUATIONS

4.3.1. SMALL ENVIRONMENT RESULTS

The corresponding results for the large environment were presented in Sections 4.1.1 and 4.1.2 respectively. This section presents the best performances for the voxel exploration task and the octree exploration task.

As mentioned before, the small environment has an area of 32 m^2 (whereas the large environment counts with 160 m^2), which is segmented into 8, 4, 2 octree nodes on the XZ-axis when segmented with node sizes of $4m^3$, $8m^3$ and $16m^3$, respectively. Since vertical space is not essential for the task at hand (objects are not allocated at different heights), we limited the Y-axis is limited to 4, 2 and 1 nodes, respectively. This is equal to 3D volumes of 256, 32 and 4 nodes, respectively. Additionally, the agent's grid sensor is also adjusted to a smaller radius of 8 meters. The best runs for the voxel-curious behaviors in the small environment are shown in Table A.8.

Method	Total Objects Scanned	Standard Deviation
blind-agent-explore-constrained	0.00	0.00
octree-4	0.00	0.00
voxel++025	4.76	1.84
voxel++050	1.49	2.37
voxel++075	2.62	0.47
voxel++100	9.42	0.88
voxel	3.03	0.39
shortest-path	0.00	0.00
object-detector-nospeed	8.71	0.01
object-detector	0.01	0.56
semantic-curiosity	0.16	0.12
semantic-entropy	0.00	0.01
voxel-entropy++050	9.19	0.00
voxel-entropy++100	9.04	0.41
voxel-entropy-normalized	0.00	0.66

Table 4.9: Overview of the results for the runs in the small environment, with respect to the *Total Objects Scanned* metric.

The following Table 4.10 presents the runs with the best performance in the octree exploration task.

Method	Octree Leaf Nodes Visited	Visited Volume (m^3)	Standard Deviation
blind-agent-explore-constrained	35.63	142	0.00
octree-4	71.27	285	0.00
voxel++025	30.52	122	0.01
voxel++050	24.12	96	0.00
voxel++075	26.65	106	0.00
voxel++100	30.42	121	0.00
voxel	35.58	142	0.00
shortest-path	3.16	12	0.00
object-detector-nospeed	28.47	113	0.00
object-detector	14.62	58	0.00
semantic-curiosity	21.56	86	0.00
semantic-entropy	2.98	11	0.00
voxel-entropy++050	31.44	125	0.01
voxel-entropy++100	32.36	129	0.00
voxel-entropy-normalized	30.02	120	0.00

Table 4.10: Overview of the results for the runs in the small environment, with respect to the *Octree Discovery Reward* metric.

More information of the results for the the small environment can be found in Figure A.6.12

4.3.2. MONOCULAR VISION

This section presents the performance obtained by the panoramic grid sensor and a *55 degrees* wide grid sensor. This latter, narrower, grid sensor represents ordinary cameras that have a smaller field of view [179]. Furthermore, we also present the performance results for a wide, circular, voxel scanner that resembles the behavior of an agent that utilizes a smaller radius of the panoramic grid sensor's view. The following Table 4.11 therefore compares the total objects scanned, the look direction, the octree leaf nodes to distinguish key performance differences based on the camera and scanner system.

Method	Episode Length %	Total Objects Scanned	F1-score	Visited Volume (m^3)
explorer-8	100	1.96	0.35	151.04
ev_smallcam.explorer-8	81	0.00	0.00	278.32
ev_bigscanner.explorer-8	94	0.92	0.28	184.64
ev_bigscanner.explorer-16	99	0.83	0.23	139.18

Table 4.11: Overview of the a subset of the explorer agents with different vision setups: small camera vision (55 degrees), panoramic camera vision (360 degrees) and a panoramic scanner (360 degrees wide and 30 degrees wide north and south).

4.3.3. PPO VS SAC

To compare the sample efficiency of PPO and SAC, we trained the best PPO-models using SAC. The following Table 4.12 displays the peformance of such models and of experiments to demonstrate the sample efficiency of PPO. We measure the number of episodes and the number of actions to the state transition in the environment. In addition, we analyze the training speed in our experiments.

Method	Episode Length %	Total Objects Scanned	Visited Volume (m^3)
explorer-8	86	1.96	151.04
explorer-8-sac	63	0.74	104.04
octree-16-constrained-pigeon	100	0.00	325.61
octree-16-constrained-pigeon-sac	100	0.01	123.23
voxel++100-nospeed-nolinger	97	1.62	138.43
voxel++100-nospeed-nolinger-sac	94	0.09	23.81

Table 4.12: Overview of a subset of runs trained using PPO and SAC to compare their performance.

4.3.4. APPLICABLE PRACTICAL SCENARIOS

(a) Dreamscape Scenario. (b) City Neighborhood. (c) Snowy Forest. (d) Summer Forest.

Figure 4.1: Modelled 3D scenarios to present the practical aspect of the proposed methods. 3D models subject to copyright from Unity Technologies [1].

Multiple 3D scenes have been collected and setup to demonstrate the applicability of our method in practical real world scenarios. Given the time constraint, deeper demonstrations of these specific use cases are out of scope. Figure 4.1 illustrates the training environment and different environments which were used to demonstrate the octree explorer agent's versatility to multiple environments.

Concretely, the 3D environments that were used to test the portability of our approach are:

- **Dreamscape.** A fantasy forest scene.
- **Forest.** An open nature scene with obstacles and a bus that was drove off as part of an accident. It is presented in three weathers: sunny, snowy and snowy overcast.
- **City Neighborhood.** A city scene where fires could be prevented with firefighter drones.

Similarly , the voxel-curious agent showed visual agnostic performance to other voxelized objects such as bus and house, which are part of the environments above.

4.3.5. CROSS-PLATFORM COMPATIBILITY

This section presents the results of the metrics determined to evaluate the effort of transferring the Unity ML-Agents environment to an OpenAI Gym compatible implementation. We define a compatible implementation as one that is ready to be trained and produce relevant results.

The development metrics are displayed in Table 4.13. The agent reward is reported for the training of the *voxel-agent++100* after 20M timesteps. It is negative given that the the voxel rewards are sparse and the movement speed and lingering penalties are dense signals.

Metric	Value
Lines of code	1500
Time required to transition the environment	4h
Time required to integrate to WandB	20h
Time required to implement resuming of experiments	40h
Time required for other features	40h
Agent reward after 20M timesteps (baselines)	-300
Agent reward after 20M timesteps (mlagents)	-374

Table 4.13: Overview of cross-platform effort measurement metrics.

4.4. DELIVERABLES

This work produced three types of exploration-capable agents: environment-focused, object-focused and mixed-focused. Accordingly, the following deliverables will be handed in with this master thesis:

- The environment-focused agent is able to explore twice as much 3D space than non-exploratory models ($358m^3$ versus $147m^3$ per episode). This is also shown in the evaluation framework where the best candidate agents are able to cover 100% of the required octree nodes.
- The object-focused agent is able to scan at least 2 objects per episode on average. This translates to a scanning speed of 2500 timesteps per object. This performance is also demonstrated in the evaluation framework where the best candidate agents are able to scan all 3 required goals within the limited timesteps.
- Finally, the mixed exploration agent is able to keep up with the 2 objects per episode performance and explore an average volume of $151m^3$, which is comparable to the performance shown by our object-focused agents. Moreover, it outperforms object-focused agents and performs comparable to the environment-focused agents in the evaluation framework.
- A set of baselines are also handed in, used to analyze the performance of traditional methods for exploration (random actions, shortest path) and other state-of-the-art-inspired methods (object detection maximization, semantic curiosity, semantic entropy).
- The trained models for our proposed method in ONNX formats, including all other variants for voxel and octree exploration and an out-of-the-box Unity-ready reinforcement learning environment capable of reproducing the experiments and results.
- For future research, a set of Unity scenes that demonstrate the versatility of the reinforcement learning agent. These scenes further allow the creation of future benchmarks for other synthetic data models and use cases, which are not limited to machine learning approaches.

5

DISCUSSION

This chapter will discuss the obtained results, the used methodology, the validity and the reliability of the experiments, adapting the structure proposed by Luckert and Schaefer-Kehnert [142]. Section 5.1 will look into the results, describing what has been achieved, as well as indicate the main problems concerning the experiments. Section 5.2 will reflect on the research task and discuss whether the right method was chosen to solve the given task. Finally, Section 5.3 discusses the validity of the datasets that were used and the overall experiment setup. Based on these validity remarks, this chapter will clarify the reliability of the experiments' results. Therefore, the following questions will be tackled:

- What conclusions can be taken from the presented results?
- Was the chosen method appropriate for the task?
- What benefits and shortcomings have been identified related to the presented work?
- What is the validity and reliability of the used data sets and the presented results?

5.1. RESULTS INTERPRETATION

The following sections present the results for the object-focused agents and environment-focused agents. Mixed-focus agents that explore both octrees and objects are discussed further below in Section 5.1.5. The results are presented from each knowledge-based perspectives. Voxel knowledge is provided when voxel elements are visible in the grid sensor view to the agent, Similarly, octree knowledge is provided through octree node observations, pigeon observations or observations of the lingering metric. This should allow the validation of our approach against other methods.

5.1.1. OBJECT EXPLORATION WITH KNOWLEDGE OF VOXELS

In general, the best performing object-focused agent without any semantic information has "oracle" knowledge about the nearest goal. As expected, this shows that oracle observations (walk and look angles towards the closest goal) contribute towards finding objects faster, since these agents have more information about the environment. Surprisingly, the best agent is run *voxel-entropy++100-nospeed*, which scanned an average of 2.10 objects per episode with a standard deviation of 0.2. This agent has vision of voxels and also observes the level of entropy in the scene around it through the observation *LingeringPenaltyStrength* (see equation 3.8). This agent is closely followed by oracle run *voxel++100-nospeed-nolinger-oracle-8*, which scans an average of 1.77 objects per episode, and its non-oracle variant, *voxzel+100-nospeed-nolinger*, which scanned 1.73 objects per episode.

Overall, the training results show that the best object exploration results were achieved by the agents that had a *sparse high voxel reward function*. In other words, the best performances were achieved when the agents can focus on the scanning of voxels without the distraction of other reward functions. Accordingly, the mechanism to reduce the minimum speed penalty and the lingering penalty, if voxels were found in the agent's field of vision (FOV), did not provide the expected performance in *run63++075*, nor *run63++100*. However, even though *run63++025* had the influence of the minimum speed penalty and the lingering

penalty, its performance was almost as high as the aforementioned runs with 1.56 objects scanned per episode. This indicates the pressure to change the behavior in the presence of voxels was relieved through a smaller voxel reward (25%).

These results points towards the hypothesis that a high voxel reward function diverges from the reward signal of these penalties. Therefore balancing these two priorites can be challenging for the agent if the voxel reward is set as a priority by the domain experts. It can also be inferred that simpler agents with clear goals and fewer distractions perform better at scanning the objects they need.

An additional point of interest is the fact that some agents prefer to move to a newer object before fully scanning the first one. In situations where the training setup allocates multiple objects close to each other, the agent chooses to initially scan the visible sides from each object instead of scanning the objects sequentially. Even though this behavior could be adjusted with a reward for finishing the full scan of an object, it is also explained by the pressure the agent has from the movement penalties. An alternative solution would be to reward the agent for moving at a slower speeds when voxels are in the agent's FOV.

On a separate note, the shortest-path agents did not perform as expected and were greatly out-shined by the oracle agents. The difference between these two agents is that oracle runs have shortest-path observations, but do not receive a penalty from the walk and look errors. Visual inspection showed that the penalty for the walk and look errors caused the shortest-path agent to fly above goals to try to minimize these errors, instead of focusing on scanning the voxels. This also indicates that the walk and look penalties were too high in comparison to the rewards given for scanning voxels. Finally, the FOV-reduction mechanism did not improve these agent's performance within the 30M training timesteps, which suggests that the behavior adjustment should be looked at from a different perspective, such as action masking [143]. This would limit the agents behavior at a "hardware" level.

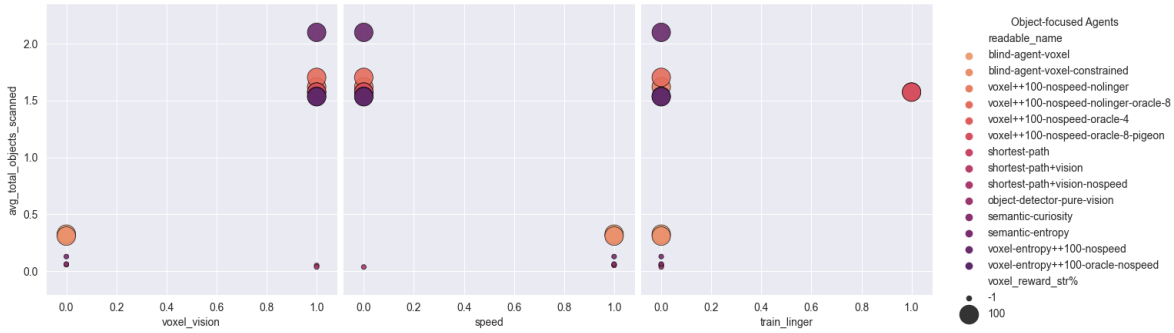


Figure 5.1: Comparison of the influence of relevant variables on the total amount of objects scanned, for a subset of the voxel-focused runs. The variables for *pigeon* and *pathak* are always set to 1 (active).

It is also important to consider the question of what observations, rewards, and constraints have the greatest impact on the effectiveness of agents in solving particular tasks. Based on the performance of runs below the 25th percentile and above the 75th percentile, Figure A.12 visualizes the influence of the most relevant variables. Results show that voxel-aware agents without the minimum speed requirement perform better when the voxel reward is over 75% strength, which corresponds to the performance discussed above. Finally, the constraint variable confirms that only agents without a normalized voxel reward signal saw sufficient value in the collection of voxels to overcome the penalties of environment. For example, *blind-agent-voxel* only explores a volume of 77 m^3, which, on visual inspection, is the space it covers spinning in circles, minimizing the minimum speed penalty. For more detailed plots on the variable influence for all runs please refer to Appendix A.6.5.

In summary, the best performing agent achieves an average of 2.1 total objects scanned while only visiting $147m^3$ of the environment per episode. This agent is also characterized for performing better when removing the minimum speed penalty and, tentatively, the lingering penalty. It is worth reminding that goals are situated at the ground level of the environment and not at different heights, which does not motivate the agent to explore remote locations of the environment to find the goals.

5.1.2. OBJECT EXPLORATION WITHOUT KNOWLEDGE OF VOXELS

This section discusses the results for the agents that receive a voxel reward but are not directly able to see voxels in their grid sensor view.

While we expected the *semantic-curiosity* agent to be the best run, the best *blind* agent is *object-detector-nospeed* with an average of 0.94 objects scanned per episode. The *semantic-curiosity* agent however has the highest detections count per episode: visual inspection shows that the agent moves towards objects to scan them but is not as accurate as the *object-detector-nospeed*. We suspect this occurs because the *RationalizedClassEntropy* (see equation 3.9) uses temporal information across the past 10 frames, whereas the *object-detector-nospeed* use of the immediate output of the object detector translates to more recent signals.

Overall, the results show that these agents struggle to scan voxels of objects without their explicit vision. On the one hand, the shortest path agent was overwhelmed by the penalties from the walk and look errors, scanning an average of 0.06 objects per episode. On the other hand, the semantic curiosity agent prefers states that maximize Chaplot's rationalized class entropy [9]. Such states are usually too far from voxels and prefer locations where multiple objects are in the agent's camera view (given the late update received from the temporal buffer). Such performances were achieved similar scores as the random agent. Additionally, fully blind agents, which do not observe any information nor penalties, performed better with 0.39 objects scanned per episode.

Concluding, among agents that do not have knowledge of voxels, the object detector agent performed the best with 0.94 objects scanned per episode. In order to improve the performance of these methods, we suggest that the penalties and reward strengths would have to be restructured.

5.1.3. ENVIRONMENT EXPLORATION WITH KNOWLEDGE OF OCTREES

This section explores the results for the agents that are rewarded for exploring new locations in the explorable environment, with observations such as the amount octree nodes discovered or pigeon observations.

The best performing octree aware agent was *octree-4*, which discovers $348m^3$ of the environment in the maximum 5000 timesteps given. It is closely followed by *octree-16-constrained-pigeon* which discovers $325m^3$ of the environment. It is worth noting that each different octree node size provides a different amount of granularity about the positions in the environment: unlike a 16-meter-wide leaf node, a 4-meter-wide leaf node requires an agent to visit more precise locations in the environment. This is observed in the amount of nodes that constitute the explorable 3D volume at each different dimension (4000, 500 and 62.5). In essence, exploratory capabilities do not vary with the size of the octree node. However, bigger octree node sizes promote the agent to visit more dispersed locations of the environment. Whereas the performance of *octree-4* accounts for 2.18% of the 4000 leaf nodes in the environment, *octree-16-constrained-pigeon* covers almost 33% of the 62.5 leaf nodes in the 5000 timesteps given.

Another positive point of interest is that the pigeon observations provided a positive impact on exploration missions and helped agents better orient themselves in a mapless environment and cover more different locations, as shown by run *octree-16-constrained-pigeon*'s 33% coverage Interestingly, Pathak's curiosity module did not provide added value to run *octree-16-constrained-pigeon-pathak*, bringing its coverage percentage to 9.5%.

Furthermore, the constraint variable proved itself useful to evaluate the performance agents that end the episode too soon. This occurred when the rewards were too low or the penalties too high, such as in the case of runs *octree-16* and *octree-4-pigeon*.

These results suggest that octrees are appropriate navigation data structures to reward the agent for navigating new locations in an environment. It is important to highlight that the agent does not have access to the octree nodes nor some kind of map. Our best octree-only agent is able to explore $348m^3$ of the environment in 132 seconds using uniquely the reward per each new octree node visited. We are certain that this coverage would be higher given a higher episode limit, as shown in the to-be-discussed test results.

5.1.4. ENVIRONMENT EXPLORATION WITHOUT KNOWLEDGE OF OCTREES

This section discusses the results for agents that are rewarded for visiting new octree nodes, yet do not have any information about the number of octree nodes discovered nor spatial information such as pigeon

observations. This covers blind agents and those based on Pathak's method [52]. The latter is a 1-1 implementation of the paper and is provided through Unity ML-Agents' curiosity module [143].

Surprisingly, the best performing agent is *blind-octree-explorer-nospeed*, which explores $364m^3$ of the environment within 5000 timesteps. This suggests that the observations that *octree-4* perceives (the number of octree nodes discovered) does not provide substantial change in the awareness of the octree. Furthermore, these agents provide an explanation to the early-exiting of the environment by the octree agents: the minimum speed penalty overshadows the rewards provided by the discovered octree nodes.

The best performing Pathak baseline is *pathak-4*, which achieves $318m^3$, which suggests that pathak does not contribute to the discovery of new octree nodes. Interestingly, the removal of the lingering penalty in *pathak-16-constrained-nospeed-nolinger* ($304m^3$ explored) shows an increase in performance compared to *pathak-16-constrained-nospeed* ($16m^3$ explored), which suggests that the lingering penalty is not compatible with Pathak's reward function.

These results broaden our understanding of the feasibility of octrees for the exploration of environments and the influence of the minimum speed penalty, the lingering penalty and Pathak's curiosity method. Overall, the results for the fully blind agents suggest that octrees are an appropriate data structure for motivating a agent to explore the environment. It is safe to say that an explorer agent does not necessarily need a minimum speed constraint and that the lingering penalty's value function conflicts with Pathak's method when applied to exploration.

5.1.5. MIXED-FOCUSED EXPLORATION

This section deals with the results from the mixed-focused agents. These agents balance the exploration of both objects and the environment. The baseline mixed-agent incorporates the voxel-entropy agent, removes the minimum speed requirement, and adds octree-node observations, pigeon observations and Pathak's curiosity module. This set of results incorporate an F1-score metric, calculated from the total objects scanned and the octree leaf nodes visited metrics. This allows the use of a single metric to evaluate the effectiveness of agents in balancing motivation between two high-level tasks.

According to our F1-score, the best mixed-focus agent is the *explorer-entropy-8*, with a 0.3 score. This result, however, comes at the expense of exploring less of the environment ($151m^3$) and scanning more objects (1.96). The best mixed-agent for exploration is agent *explorer-entropy-4-noTrainEntropy*, which covers $358m^3$ of the environment. This result shows that the agent prioritizes the exploration of the environment over the scanning of voxels. This is because the octree reward is higher and is present at every timestep, whereas the voxel reward is sparse.

Another point of interest is that the results show that agents the adjustment of the *LingeringPenaltyStrength* (see equation 3.8) did contribute to both the exploration of the environment and of objects. Concretely, *explorer-entropy-8-noTrainEntropy* is the runner-up agent with an average of 1.65 objects per episode and explores $139m^3$ of the environment. Furthermore, the lingering penalty positively influenced the performance mixed-focus agent: *explorer-entropy-8-nolinger* explored less of the environment ($117m^3$) and less objects per episode(1.77).

Taken together, these findings demonstrate that mixed-focus agents are capable of balancing the object- and environment-exploration tasks. In addition, it can safely be said that Pathak's curiosity module provided by ML-Agents helps agents find new states, but it would be important to analyze the effects of fine-tuning its hyperparameters. Finally, even though counter-intuitive, the removal of the minimum speed requirement suggests that noise (unnecessary) rewards (and observations) should be avoided, as they translate into slower convergence times and they mask which observations actually matter for the task at hand. We hope that these findings will reach RL practitioners, specially those who were also inspired by "Reward is enough"[7], so that observations and rewards will be carefully deconstructed, analyzed, and evaluated for their actual impact on an agent's behavior.

5.1.6. COMPARISON OF THE TWO RESEARCH QUESTIONS

Regarding the setup of the two research questions, each research question (RQ) tackled a different expected policy in the reinforcement learning agent. While RQ1 focuses on the exploration of objects from multiple angles, RQ2 deals with the exploration of unknown environments, in which objects of interest could be found. The usage of Unity 3D provided a platform for the implementation and practical testing, in addition to the

theoretical argumentation of our approach. Furthermore, the separation of methods based on knowledge about octrees or voxels, allowed the critical, unbiased, feasibility evaluation of our proposed approach. The expected results were that agents trained with knowledge of voxels and octrees would balance both policies and even outperform single-task agents and the chosen blind-baselines, following Silver's claim Silver et al. [7]. These expectations were confirmed by the experiments, achieving over 2 scanned objects per episode while covering a minimum volume of $147m^3$ without repetition.

From the perspective of RQ1, the our best mixed-focus agent shows an increase in the amount of objects scanned by at least a factor of two in comparison to the best chosen baseline (object detector). Moreover, this agent also shows an increase of 24% compared to the best voxel-oracle run, *voxel++100-nospeed-nolinger-oracle-8*. In comparison to our best voxel-focused run, *voxel-entropy++100-nospeed*, our best mixed-focus agent is 7% behind with 1.96 objects scanned. The respective results for the different algorithms are influenced the most by the minimum speed penalty, which added pressure on the agents to continue moving even though they needed to slow down to scan objects.

From the point of view of research question 2, the best octree-focused agent *octree-4* explores 5% than the best baseline agent *blind-octree-explorer-nospeed*. This points out that the amount of nodes discovered per timestep does not provide much "awareness" of the surrounding octree to the agent. This is one of the assumptions taken in this thesis work as part of the scope limitation that form part of the further steps. In other words, providing the octree agent with more information about the surrounding octree nodes (partial observability) would have expanded the scope of this work to analyze more aspects such as efficiency and performance variations given the three different types of octree nodes our data structure stores (scan points, visited points, out-of-range points). This allows for the continuation of this work to analyze the various possibilities that octrees can enable.

Accordingly, our best mixed-focus agent explores 59% less the best environment-focused agent *blind-octree-explorer-nospeed*. Similarly, Pathak's baseline *pathak-4* falls 8% behind our best environment-focused agent and 111% ahead of the best mixed-focus agent, This is expected due to the fact that mixed-focus agents must sacrifice their limited timesteps given, in order to explore the environment's objects.

The respective results for the different agents that explore the environment are most influenced by the minimum speed penalty and the environment-reset-constraint variable. The removal of the former allowed agents to explore locations that required a slower speed, such as corners or around obstacles. Furthermore, though many of the early-stopping situations were caused by the minimum speed penalty, the addition of the environment-reset-constraint refused the agent the possibility of early-stopping episodes as a solution to the minimization of penalties, such as observed on 16-meter-wide subdivisions of the octree.

It is important to remember that in the case of RQ1, we can only provide an approximate comparison in performance with relation to other state-of-the-art methods. The original methods were not implemented in the Unity game engine, and given the limited time scope, our implementations are not 1-1 copies of the original works. If the authors would be willing to adjust their solution to be tested in the Unity game engine, we can provide a much more accurate performance comparison. In the case of research question 2, our performance comparisons are reliable, since Pathak's curiosity module was implemented by the Unity ML-Agents development team as an identical copy of the method proposed by the original paper.

Concluding, the experiments confirmed the expectations of octrees and voxels being able to balance the exploration of environments and objects. The in-depth analysis of the influence of observations and rewards in a reinforcement learning agent allowed the construction of an efficient, embodied explorer drone.

5.1.7. EVALUATION FRAMEWORK

The proposed framework evaluates the performance of the agents based on two separate tasks, each corresponding to one research question. To this end, inspired by the DARPA Subterranean Challenge, two test environments were developed. One environment tests the agent's time-to-goal for three goals situated sequentially after the previous goal has been discovered. The second environment evaluates how much time each environment-curious agent takes to discover a proportion of the environment, up to 100% coverage. Given the limited time scope, only the best training runs from four categories were tested: object-focused, environment-focused, baselines, and mixed-focus. For the tests, the length of each episode was extended from 5'000 to 40'000 timesteps. It is also worth mentioning that the episode length was not able to be set to infinite, since that simulations that run for too long without stopping required more computer memory than

what we had available. Therefore, the percentages reported are for half of the explorable vertical space.

The results for the object-discovery task show that not all agents were able to reach the third goal within the given timesteps. Agents *explorer-entropy-8*, *voxel-entropy++100-nospeed* and *voxel++100-nospeed* reached the third goal after 277, 357 and 264 seconds, respectively. They each have an average time between goals of 88, 92 and 119 seconds, and take an average of 1, 2 and 2 seconds/meter, respectively. These results show that the introduction of entropy as an awareness signal shows an impact in between the second and third best agents. They also show that our mixed-focus agent explores the environment faster and has overall better performance.

Concerning the environment exploration task, the results clearly show that not all agents are capable of exploring newer locations in the environment. Firstly, given that baselines displayed a poor performance in the previous sections, it was expected that the baselines would not perform well in the test environments. Secondly, from the perspective of voxel-focused agents, it was expected that they are not able to explore much of an environment without goals. On visual inspection, the agents were stuck spinning around obstacles. Thirdly, in terms of the mixed-focus agents, it was expected that they are able to perform to some extent in this test. Results show that they are capable of exploring the environment, but they are 11% slower than the best agent for this task. Finally, results clearly show that a variant of the environment-focused agents was able to outperform all other agents. Concretely, run *octree-16-constrained-pigeon-nospeed*, which uses 16-meter wide leaf nodes, was able to fully traverse half of the the environment (31 octree nodes) within 7.68 minutes. Its runner up, mixed-focus agent *explorer-entropy-16-constrained* achieves the same result after 8.5 minutes.

It can be therefore safely assumed that given an infinite episode (or an episode with at least 80 thousand timesteps), these agents would explore all of the environment after an average 16 minutes, covering therefore a volume of 62.5 octree nodes or a $100m^3$.

5.2. METHOD REFLECTION

The used method to structure this thesis, adapted from the work by [142], proved to be appropriate for proposing a solution for the research questions given. The format provided allowed for the setup of different experiments without neglecting the initial goal. From the identification of the goals for each task, through the environment description and the specification and proposition of the reinforcement learning approaches, the method proved to efficiently structure the work required to construct an octree-voxel-curious explorer agent.

Similarly, in terms of octree-focused agents, future steps would include the analysis of the performance of an *octree-no-speed-pigeon* agent, given that results suggest that the addition of pigeon observations and the removal of the minimum speed requirement improved the exploratory performance of agents.

Overall, the results of this work prove that octrees and voxels are suitable for finding a solution to the two research questions. To address analysis of possible reward signals capable of solving these tasks, multiple observations and reward signals were developed. These were then evaluated separately to determine their influence in the agent's behavior. Furthermore, combining the environment-focused agents with the object-focused agents allowed the final mixed-focused agents to balance both exploratory behaviors and reach outstanding performance in the evaluation framework. Finally, the abstraction of the complexities in the environment and of objects allowed the agent to be visual-agnostic and transfer the learned behaviors across the proposed practical scenarios from Section 4.3.4. This demonstrates that our proposed method is capable of adapting to newer environments.

Furthermore, our method can be used as more than just an agent for navigation and exploration, but also as a baseline in the generation and evaluation of 3D object-centric datasets such as Objectron [125], where the trajectories around objects to collect multiple perspectives about its features are essential for a thorough representation. Accordingly, in terms of the practical scenarios that are part of our contributions, it was important for us to cover three aspects in the creation of benchmarks, in order to continue the work laid out by Juliani et al. [10]:

- the contribution of data, which in our case comprise our Unity 3D environment setups.
- the provision of a baseline, which constitutes the diverse agents that are shipped with our deliverables.

- a new set of questions that will motivate further research in navigation, point-to-goal, dataset generation, online model training, reinforcement learning and active vision tasks.

Along these lines, it was most important for this thesis that the research carried out would distance itself with the common lack of reproducibility and dependency problems that clogs scientific research projects.

To this end, not only does the usage of Unity 3D enabled the immediate reproduction of the experiments, but the ML-Agents plugin also proved to be a great toolbox for developing reinforcement learning agents. Concretely, it greatly simplified the development process with an intuitive UI, thorough documentation, an active and supportive community, and an extensive set of tools and modules such as attention, memory, curiosity, imitation learning, curriculum, and more. Ultimately, we consider ML-Agents an essential solution for developing reinforcement-learning-based solutions. The plugin, in cohesion with Unity, provided efficient means to model, evaluate and improve models and debug errors during the execution of the scientific method.

Additionally, as part of the further use of the results, the evaluation of compatibility with as OpenAI Gym could be easily done with the OpenAI Gym wrapper provided with ML-Agents. The wrapper enabled the loading of our Unity-developed environments into a Jupyter notebook and Python scripts, where an agent could be trained using an external neural network from OpenAI Baselines [17]. While the Unity OpenAI wrapper permitted a smooth environment transfer, the results presented in Section 4.3.5 did not justify the overall usage of OpenAI Gym. More concretely, development functions such as resuming of experiments, logging with Weights and Biases [180], debugging, etc., negatively affected the overall development process, which puts ML-Agents ahead of the curve. Even though Unity ML-Agents is currently limited to a set of trainers like PPO and SAC, it provided increased versatility and plug-and-play integration of a multitude of modules, such as attention, memory, curiosity, behavioral cloning, auto-curriculum, and many more. Therefore, we consider the use of Unity ML agents to be more appropriate for reinforcement learning tasks as of current writing.

Similarly, in small-scale environments, the results show that the longer an agent has to move to reach a goal, the greater the real difference in each method's effectiveness. Concretely, large-scale environments prove that some agent behaviors are not motivated to explore to find rewards. Instead, these agents choose to move in circles in a small part of the environment, minimizing the minimum speed penalty. This indicates that these agents depend on the domain randomization to spawn a goal near them. In contrast, in small-scale environments octree agents are more predisposed to display similar performance, with at least an average of 23% of the environment explored by all agents. This proves that these methods are very effective in scanning voxels of objects in the vicinity, such as *run65-pure* demonstrates after visual inspection. However, this does not translate to a policy that can be used in a practical environment. We can safely claim that "small environments" conceal the subtleties that could distinguish one method from another, since their performance metrics would not be too far apart. Therefore, in order to properly assess exploration policies, we recommend that agents must face not only various scenarios with obstacles, different heights, shapes, and colors but also larger environments that resemble the real world.

On top of that, the panoramic vision agents proved to outperform agents equipped with an ordinary monocular vision of 55 degrees. Concretely, while the *smallcam* agent explored $278m^3$ of the environment without scanning objects, its *panoramic* alternative scanned 1.96 objects and explored $151m^3$ of the environment per episode. These results support the claim by Zhang and Huang [153] and research by Davison et al. [154], presented in Section 3.3.2, that monocular cameras require more complex models for target tracking, feature remembrance, additional noise reduction, etc. It is therefore safe to say, that panoramic cameras should be preferred in navigation tasks to avoid the unnecessary overhead small FOVs add in scientific research.

In terms of future steps, the following points were collected during the creation of this work, as great opportunities to continue research in reinforcement learning:

- real-time voxelization of the objects of interest to capture limitations in terms of computing, memory and noise, while also considering efforts with event-driven cameras, inspired various works by Scaramuzza, such as [181, 182] and Grinvald [175].
- a parallel-agent environment where multiple agents could be trained simultaneously. This is a relatively straight forward task using ML-Agents but given the limited time frame it could not be implemented

given our setup. A separate but related point is a multi-agent learning and coordination, inspired by works by Jacques, such as [183].

Furthermore, given that voxel-focused blind agents do not perform as well as voxel-aware agents, we thought it would be interesting to do a preliminary analysis on the agent behavior if these methods were looked as more than competitors. Concretely, instead of considering the methods proposed by related efforts as reward signals, we consider part of the future steps the evaluation of the contribution of these methods as agent observations, as shown in Table 5.1.

Method	**Episode Length %**	**Average Total Objects Scanned**	**Standard Deviation**
shortest-path	97	0.06	0.04
semantic-curiosity	100	0.08	0.05
semantic-entropy	49	0.16	0.17
object-detector	97	0.47	0.33
voxel++100-nospeed-**object-detector**	99	1.27	0.16
voxel++100-nospeed	99	1.51	0.20
voxel++100-nospeed-**semantic-entropy**	95	1.71	0.20
voxel++100-nospeed-nolinger-**oracle**-8	99	1.85	0.20
voxel-entropy++100-nospeed	96	2.10	0.20
voxel++100-nospeed-**semantic-curiosity**	97	2.21	0.19

Table 5.1: Overview of baselines' as reward signals and as observations in voxel-aware agents, including a simple voxel-agent *voxel++100-nospeed* and the best performing voxel-focused agent *voxel-entropy++100-nospeed*.

These preliminary results show that Chaplot's semantic curiosity [9] improves the performance of the previous best run *voxel-entropy++100-nospeed* by 5%. Given that our mixed-focus agents implicitly observe the levels of entropy (through the *LingeringPenaltyStrength*), it would be part of our further steps to analyze their performance when observing entropy directly through Chaplot's semantic entropy. This would incorporate evaluations in the DARPA environments, the analysis of the size of the semantic memory buffer and the review of the stacked observations in the ML-Agents hyperparameters.

Additionally, as part of our preliminary future steps, an alternative agent with a panoramic-like scanner was implemented and trained to evaluate its performance. While the expected behavior was for this agent to scan objects faster, the results showed that a panoramic scanner not only allowed the agent to scan objects faster and but also to explore more of the environment (18%). We believe that this behavior can be further improved by providing a reward for finishing object scans.

A test pilot in our future work includes the integration of our method to the Milking Robot project mentioned in Section 1.1. In this concrete example, the agent would be the grounded robot arm and the cow teats would be voxelized to constitute the object of interest. Our method would therefore would be able to reduce the uncertainty in the amount of cow teats given obstructions by exploring all reachable voxels and octree nodes given limitations given by the robotic arm joints. In this scenario, the previously proposed enhanced octree observations, that are part of our future work, would allow the arm to prefer nodes that contain scan points over empty or out-of-reach nodes.

In conclusion, the proposed use of octree for environmental exploration and voxels for object exploration has allowed agents to adapt successfully to new environments, to cover large spaces and to perform better than other proposed baselines. Moreover, the evaluation framework clearly distinguishes the performance of the different types of agents. Unity also demonstrated that it has a comprehensive toolkit for implementing concept demonstrations and reinforcement learning agents. Further work includes parsing voxels in real time, taking into account visual input noise and measuring the real-time performance of our methods.

5.3. RELIABILITY

This section aims to further discuss the validity and reliability of the results shown in section 4.1. The achieved results show substantial improvement over the taken baselines for both object and environment exploration.

Using voxels as a reward signal for our reinforcement learning agent proved to be valid for the coverage of trajectories around objects of interest to reduce the uncertainty about such objects. In the future, we plan to voxelize objects in real-time to determine the algorithmic and computational limitations of the voxelization process. Furthermore, we plan to study the regulation of the amount of scanned voxels required for an object to be marked as scanned.

Similarly, using octrees a data structures for the exploration of newer spaces proved to be a valid strategy to find such objects of interest, given the efficiency of octrees at representing sparse point clouds, specially at lower resolutions. In the future, we plan to dynamically adjust the size of the octree nodes for the agent to observe more accurate distance information about its surroundings, such as in distances in small passages in caves.

Finally, PPO proved to be a valid reinforcement learning algorithm for an exploration drone, since the obtained results were on par with the results achieved by a human-in-the-loop. One of the main drawbacks, however, is the high computational cost of the PPO algorithm, due to the large number of samples that are needed for it to converge to a good solution. For this reason, we trained for 30 million timesteps and tested on 7.5 million timesteps. On a similar note, results for SAC-trained agents show that the SAC algorithm required higher training times in the former. Concretely, SAC took an average of 11h/M timesteps as compared 3.2h/M timesteps when using PPO. Moreover, on closer look, SAC agents preferred to restart the episodes, even for agents in a plentiful-reward setup. This indicates that PPO was a robust and valid reinforcement learning strategy, and that SAC needs more fine-tuning to improve its performance. In the future, we plan to analyze the agent's performance in dynamic, multiple-object and multiple-agent scenarios.

6

CONCLUSION

6.1. CONCLUSIONS

This work answered the following research questions:

- How can an embodied agent increase the overall certainty about an object's characteristics, i.e., how can trajectories around objects of interest be covered to reduce the uncertainty about such objects?
- How can these objects be found in large and unknown environments by the same agent?

This was done by modeling and implementing multiple 3D environments in Unity 3D. These environments were used to train and test several reinforcement learning agents aimed at exploring both objects and environments. Firstly, objects were explored through the scanning of the voxels that composed the object in the 3D space. Secondly, agents learned to explore the environment through extrinsic rewards of the new octree nodes discovered. The resulting agents were visual agnostic and could adapt to multiple environments. Their explorative behavior can also be adapted to a multitude of further scenarios, from product quality assurance drones to rescue, police and firefighter drones. Additionally, data collection of 3D scenes is a clear use case for the explorative behavior of the drone, enabling the creation of thorough point cloud datasets in any kind of environment. Given the problematic of small environments proposed by previous methods, our learning environments were $160m^2$ wide and provided an explorable volume of $1000m^3$. Our approach was tested against baselines that answered the research questions without direct knowledge of voxels or octrees, respectively. Furthermore, the two testing grounds were inspired by the DARPA subterranean challenge to collect data about the time taken to discovery and how much of the environment was explored. Our results show that our exploration drone outperformed oracle-agents and the proposed baselines in the object-exploration DARPA test by 24% and a factor of two, respectively. Furthermore, a variant of our mixed-focus agent showed comparable performance to the best environment-focused agent, exploring 100% of the tested volume after 510 seconds and achieved an 8% increase in the space covered when compared to Pathak's exploration baseline.

Accordingly, we provided the Unity code for this thesis work and three Unity environments for further work on this project, benchmarks and reproduction of the results. The Unity assets are subject to copyright and require a license to be downloaded and used. We hope that this line of work reaches many people in the community, as the work by Meta Research (Detic) [184], Plenoxels [176] and Unity's recent acquisition of Ziva Dynamics [185] continue to lay the path for the future of synthetic data and machine learning.

6.2. WHERE TO GO FROM HERE?

This work presents a reinforcement learning setup for agents to explore objects and environments. There are a few possible routes for the extension of this work to improve the performance of the achieved results. First, our model-free approach uses synthetic data which greatly simplifies the problematics of the real world [65]. Our current Unity setup provides noise-free information about the environment through the grid sensor; this is one of the points we would like to tackle in our future work steps which involves the creation of the voxelized environment from the depths of the unity cameras, bringing our method closer to a production

setup. Second, the implementation of a dynamic, moving, environment would allow us to evaluate the obstacle avoidance capabilities of our agent and, most importantly, the claim that model free approaches suffer given their simplification of the dynamics of an environment [65]. Third, the usage of our model to collect object data and train an object detector, would allow us to directly compare our approach to Chaplot's work [9]. Accordingly, the connection to other semantic sub-modules would also bring our agent closer to production. Finally, a number of other improvements would benefit the project such as 1) the evaluation of our mobility algorithms and our F1-metric, 2) hyper-parameter sensitivity tests, 3) analysis of attention and memory modules in the Unity ML-Agents plugin, as well as 4) the implementation of a Unity environment that allows curriculum learning using the provided tools by ML-Agents [143].

In conclusion, the presented work extends the related research on this topic by a providing a reinforcement learning approach for the exploration of objects and environments through autonomous navigation and Unity simulated agents.

BIBLIOGRAPHY

[1] Unity Technologies. *Unity Asset Store - The Best Assets for Game Making*. 2021. URL: https://assetstore.unity.com/.

[2] DULAL, SAURAB. *octree Construction and Nearest Neighborhood Search (NSS)*. 2018. URL: https://dulalsaurab.github.io/computerscience/octree-construction-and-nns/.

[3] Wüstefeld, Tobias. *3D Pixel / Voxel*. 2010. URL: http://www.bilderzucht.de/blog/3d-pixel-voxel/.

[4] Pieter Abbeel. *Deep Learning for Robotics (NIPS 2017 Keynote)*. 2017. URL: https://youtu.be/MBQTu8P1N4M?t=87.

[5] Bing Cai Kok and Harold Soh. "Trust in robots: Challenges and opportunities". In: *Current Robotics Reports* (2020), pp. 1–13.

[6] Deepak Trivedi et al. "Soft robotics: Biological inspiration, state of the art, and future research". In: *Applied bionics and biomechanics* 5.3 (2008), pp. 99–117.

[7] David Silver et al. "Reward is enough". In: *Artificial Intelligence* (2021), p. 103535.

[8] Yunlong Song et al. "Autonomous drone racing with deep reinforcement learning". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 1205–1212.

[9] Devendra Singh Chaplot et al. "Semantic curiosity for active visual learning". In: *European Conference on Computer Vision*. Springer. 2020, pp. 309–326.

[10] Arthur Juliani et al. "Unity: A general platform for intelligent agents". In: *arXiv preprint arXiv:1809.02627* (2018).

[11] Nicolas Borla et al. "Teat Pose Estimation via RGBD Segmentation for Automated Milking". In: *CoRR* abs/2105.09843 (2021). arXiv: 2105.09843. URL: https://arxiv.org/abs/2105.09843.

[12] Meier Ralph and Roost, Dano. *Active Scene Understanding from Image Sequences for Next-Generation Computer Vision*. 2020. URL: https://www.zhaw.ch/storage/shared/upload/ba20_toff_01_ralph_dano_final.pdf.

[13] Tao Chen, Saurabh Gupta, and Abhinav Gupta. "Learning exploration policies for navigation". In: *arXiv preprint arXiv:1903.01959* (2019).

[14] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[15] DARPA. *Subterranean Challenge*. 2021. URL: https://www.subtchallenge.com/.

[16] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[17] German Aerospace Center (DLR) - Institute of Robotics and Mechatronics (RM). *PyTorch version of Stable Baselines, reliable implementations of reinforcement learning algorithms*. 2021. URL: https://github.com/DLR-RM/stable-baselines3.

[18] Róbert-Adrian Rill and Kinga Bettina Faragó. "Collision Avoidance Using Deep Learning-Based Monocular Vision". In: *SN Computer Science* 2.5 (2021), pp. 1–10.

[19] Christian Wojek et al. "Monocular visual scene understanding: Understanding multi-object traffic scenes". In: *IEEE transactions on pattern analysis and machine intelligence* 35.4 (2012), pp. 882–897.

[20] Abhishesh Pal et al. "Algorithm design for teat detection system methodology using TOF, RGBD and thermal imaging in next generation milking robot system". In: *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. IEEE. 2017, pp. 895–896.

[21] Terabee. *Time of Flight Principle*. 2021. URL: https://www.terabee.com/time-of-flight-principle.

[22] Leandro Cruz, Djalma Lucio, and Luiz Velho. "Kinect and rgbd images: Challenges and applications". In: *2012 25th SIBGRAPI conference on graphics, patterns and images tutorials*. IEEE. 2012, pp. 36–49.

[23] Association for Advancing Automation. *What is Thermal Imaging?* 2016. URL: https://www.automate.org/blogs/what-is-thermal-imaging.

[24] Akanksha Rastogi and Beom Sahng Ryuh. "Teat detection algorithm: YOLO vs. Haar-cascade". In: *Journal of Mechanical Science and Technology* 33.4 (2019), pp. 1869–1874.

[25] Niall O'Mahony et al. "3D Vision for Precision Dairy Farming". In: *IFAC-PapersOnLine* 52.30 (2019), pp. 312–317.
[26] Wenming Cao et al. "A comprehensive survey on geometric deep learning". In: *IEEE Access* 8 (2020), pp. 35929–35949.
[27] NV Kartheek Medathati et al. "Bio-inspired computer vision: Towards a synergistic approach of artificial and biological vision". In: *Computer Vision and Image Understanding* 150 (2016), pp. 1–30.
[28] Mohammad K Ebrahimpour et al. "Ventral-dorsal neural networks: object detection via selective attention". In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2019, pp. 986–994.
[29] Akiko Wagatsuma et al. "Locus coeruleus input to hippocampal CA3 drives single-trial learning of a novel context". In: *Proceedings of the National Academy of Sciences* 115.2 (2018), E310–E316.
[30] Karen Simonyan and Andrew Zisserman. "Two-stream convolutional networks for action recognition in videos". In: *arXiv preprint arXiv:1406.2199* (2014).
[31] Ali Diba et al. "Temporal 3d convnets: New architecture and transfer learning for video classification". In: *arXiv preprint arXiv:1711.08200* (2017).
[32] Rui Hou et al. "An efficient 3d CNN for action/object segmentation in video". In: *arXiv preprint arXiv:1907.08895* (2019).
[33] Charles R Qi et al. "Frustum pointnets for 3d object detection from rgb-d data". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 918–927.
[34] Charles R Qi et al. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.
[35] Charles Ruizhongtai Qi et al. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in neural information processing systems* 30 (2017), pp. 5099–5108.
[36] A Nivaggioli, JF Hullo, and G Thibault. "USING 3D MODELS TO GENERATE LABELS FOR PANOPTIC SEGMENTATION OF INDUSTRIAL SCENES." In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* 4 (2019).
[37] Yunzhi Lin et al. "Using synthetic data and deep networks to recognize primitive shapes for object grasping". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 10494–10501.
[38] Eric Jang et al. "End-to-end learning of semantic grasping". In: *arXiv preprint arXiv:1707.01932* (2017).
[39] Ricson Cheng, Arpit Agarwal, and Katerina Fragkiadaki. "Reinforcement learning of active vision for manipulating objects under occlusions". In: *Conference on Robot Learning*. PMLR. 2018, pp. 422–431.
[40] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *The International Journal of Robotics Research* 37.4-5 (2018), pp. 421–436.
[41] Richard Hartley Andrew Zisserman. "Multiple view geometry in computer vision". In: (2004).
[42] Sebastian Thrun. "Probabilistic robotics". In: *Communications of the ACM* 45.3 (2002), pp. 52–57.
[43] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.
[44] Cesar Cadena et al. "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age". In: *IEEE Transactions on robotics* 32.6 (2016), pp. 1309–1332.
[45] Ruben Martinez-Cantin et al. "A Bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot". In: *Autonomous Robots* 27.2 (2009), pp. 93–103.
[46] Henry Carrillo, Ian Reid, and José A Castellanos. "On the comparison of uncertainty criteria for active SLAM". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 2080–2087.
[47] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. "Learning to fly by crashing. In 2017 IEEE". In: *RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3948–3955.
[48] Fereshteh Sadeghi and Sergey Levine. "Cad2rl: Real single-image flight without a single real image". In: *arXiv preprint arXiv:1611.04201* (2016).
[49] Jingwei Zhang et al. "Neural slam: Learning to explore with external memory". In: *arXiv preprint arXiv:1706.09520* (2017).
[50] Jürgen Schmidhuber. "A possibility for implementing curiosity and boredom in model-building neural controllers". In: *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*. 1991, pp. 222–227.

[51] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. “Incentivizing exploration in reinforcement learning with deep predictive models”. In: *arXiv preprint arXiv:1507.00814* (2015).
[52] Deepak Pathak et al. “Curiosity-driven exploration by self-supervised prediction”. In: *International conference on machine learning*. PMLR. 2017, pp. 2778–2787.
[53] Justin Fu, John D Co-Reyes, and Sergey Levine. “Ex2: Exploration with exemplar models for deep reinforcement learning”. In: *arXiv preprint arXiv:1703.01260* (2017).
[54] Manuel Lopes et al. “Exploration in model-based reinforcement learning by empirically estimating learning progress”. In: *Neural Information Processing Systems (NIPS)*. 2012.
[55] Satinder Singh, Andrew G Barto, and Nuttapong Chentanez. *Intrinsically motivated reinforcement learning*. Tech. rep. MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER SCIENCE, 2005.
[56] Dinesh Jayaraman and Kristen Grauman. “Learning to look around: Intelligently exploring unseen environments for unknown tasks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1238–1247.
[57] Shi Bai et al. “Information-theoretic exploration with Bayesian optimization”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 1816–1822.
[58] Thomas Kollar and Nicholas Roy. “Trajectory optimization using reinforcement learning for map exploration”. In: *The International Journal of Robotics Research* 27.2 (2008), pp. 175–196.
[59] Rui Zeng et al. “View planning in robot active vision: A survey of systems, algorithms, and applications”. In: *Computational Visual Media* (2020), pp. 1–21.
[60] Ruzena Bajcsy. “Active perception”. In: *Proceedings of the IEEE* 76.8 (1988), pp. 966–1005.
[61] Phil Ammirato et al. “A dataset for developing and benchmarking active vision”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1378–1385.
[62] Jianwei Yang et al. “Embodied amodal recognition: Learning to move to perceive objects”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 2040–2050.
[63] Devendra Singh Chaplot, Emilio Parisotto, and Ruslan Salakhutdinov. “Active neural localization”. In: *arXiv preprint arXiv:1801.08214* (2018).
[64] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. “Active markov localization for mobile robots”. In: *Robotics and Autonomous Systems* 25.3-4 (1998), pp. 195–207.
[65] Xin Wang et al. “Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 37–53.
[66] J Irving Vasquez-Gomez, L Enrique Sucar, and Rafael Murrieta-Cid. “View/state planning for three-dimensional object reconstruction under uncertainty”. In: *Autonomous Robots* 41.1 (2017), pp. 89–109.
[67] Narcıs Palomeras et al. “Autonomous exploration of complex underwater environments using a probabilistic next-best-view planner”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1619–1625.
[68] Devendra Singh Chaplot and Guillaume Lample. “Arnold: An autonomous agent to play fps games”. In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.
[69] Devendra Singh Chaplot et al. “Gated-attention architectures for task-oriented language grounding”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.
[70] Devendra Singh Chaplot et al. “Learning to explore using active neural slam”. In: *arXiv preprint arXiv:2004.05155* (2020).
[71] Devendra Singh Chaplot et al. “Neural topological slam for visual navigation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 12875–12884.
[72] Tegg Taekyong Sung et al. “Exploring Navigation using Deep Reinforcement Learning”. In: *International Journal of Applied Engineering Research* 13.19 (2018), pp. 14447–14450.
[73] Peter Auer. “Using confidence bounds for exploitation-exploration trade-offs”. In: *Journal of Machine Learning Research* 3.Nov (2002), pp. 397–422.
[74] Thomas Jaksch, Ronald Ortner, and Peter Auer. “Near-optimal Regret Bounds for Reinforcement Learning.” In: *Journal of Machine Learning Research* 11.4 (2010).
[75] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
[76] Benjamin Eysenbach et al. “Diversity is all you need: Learning skills without a reward function”. In: *arXiv preprint arXiv:1802.06070* (2018).
[77] Saurabh Gupta et al. “Cognitive mapping and planning for visual navigation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2616–2625.

[78] Manolis Savva et al. “MINOS: Multimodal indoor simulator for navigation in complex environments”. In: *arXiv preprint arXiv:1712.03931* (2017).
[79] Peter Anderson et al. “Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3674–3683.
[80] Alexey Dosovitskiy and Vladlen Koltun. “Learning to act by predicting the future”. In: *arXiv preprint arXiv:1611.01779* (2016).
[81] Guillaume Lample and Devendra Singh Chaplot. “Playing FPS games with deep reinforcement learning”. In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.
[82] Piotr Mirowski et al. “Learning to navigate in complex environments”. In: *arXiv preprint arXiv:1611.03673* (2016).
[83] Yuxin Wu and Yuandong Tian. “Training agent for first-person shooter game with actor-critic curriculum learning”. In: (2016).
[84] Karl Moritz Hermann et al. “Grounded language learning in a simulated 3d world”. In: *arXiv preprint arXiv:1706.06551* (2017).
[85] Yuke Zhu et al. “Target-driven visual navigation in indoor scenes using deep reinforcement learning”. In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 3357–3364.
[86] Abhishek Das et al. “Embodied question answering”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1–10.
[87] Daniel Gordon et al. “Iqa: Visual question answering in interactive environments”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4089–4098.
[88] Parrot. *ANAFI Ai | The 4G robotic UAV*. 2022. URL: https://www.parrot.com/en/drones/anafi-ai.
[89] Kuan Fang et al. “Scene memory transformer for embodied agents in long-horizon tasks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 538–547.
[90] George Fahim, Khalid Amin, and Sameh Zarif. “Single-view 3D reconstruction: a survey of deep learning methods”. In: *Computers & Graphics* 94 (2021), pp. 164–190.
[91] Michael Firman. “RGBD datasets: Past, present and future”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2016, pp. 19–31.
[92] Eman Ahmed et al. “A survey on deep learning advances on different 3D data representations”. In: *arXiv preprint arXiv:1808.01462* (2018).
[93] Minecraft Middle Earth. *Minecraft Middle Earth - Minas Tirith - Cinematic Showcase*. 2020. URL: https://www.youtube.com/watch?v=KrDDHGACpok.
[94] Tim McGraw. “High-quality real-time raycasting and raytracing of streamtubes with sparse voxel octrees”. In: *2020 IEEE Visualization Conference (VIS)*. IEEE. 2020, pp. 21–25.
[95] Anastasia Ioannidou et al. “Deep learning advances in computer vision with 3d data: A survey”. In: *ACM Computing Surveys (CSUR)* 50.2 (2017), pp. 1–38.
[96] Open3D. *ICP Registration*. 2020. URL: http://www.open3d.org/docs/latest/tutorial/Basic/icp_registration.html.
[97] Open3D. *Voxelization*. 2020. URL: http://www.open3d.org/docs/latest/tutorial/geometry/voxelization.html.
[98] Chaochuan Jia et al. “A new fast filtering algorithm for a 3D point cloud based on RGB-D information”. In: *PloS one* 14.8 (2019), e0220253.
[99] Xiaoshui Huang et al. “A comprehensive survey on point cloud registration”. In: *arXiv preprint arXiv:2103.02690* (2021).
[100] Open3D. *Point Cloud*. 2020. URL: http://www.open3d.org/docs/0.7.0/tutorial/Basic/pointcloud.html.
[101] Homer H Chen and Thomas S Huang. “A survey of construction and manipulation of octrees”. In: *Computer Vision, Graphics, and Image Processing* 43.3 (1988), pp. 409–431.
[102] Donald Meagher. *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer*. Electrical and Systems Engineering Department Rensselaer Polytechnic, 1980.

[103] Abderrahim Saaidi, Khalid Satori, et al. “Multi-view passive 3D reconstruction: comparison and evaluation of three techniques and a new method for 3D object reconstruction”. In: *2014 International Conference on Next Generation Networks and Services (NGNS)*. IEEE. 2014, pp. 194–201.

[104] OpenGenus IQ. *Octree data structure*. 2021. URL: https://iq.opengenus.org/octree/.

[105] Geeks for Geeks. *Octree | Insertion and Searching*. 2021. URL: https://iq.opengenus.org/octree/.

[106] Samuli Laine and Tero Karras. “Efficient sparse voxel octrees”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.8 (2010), pp. 1048–1059.

[107] Enrico Gobbetti, Fabio Marton, and José Antonio Iglesias Guitián. “A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets”. In: *The Visual Computer* 24.7 (2008), pp. 797–806.

[108] Jan Elseberg, Dorit Borrmann, and Andreas Nüchter. “One billion points in the cloud–an octree for efficient processing of 3D laser scans”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 76 (2013), pp. 76–88.

[109] Francisco Javier Melero, Ángel Aguilera, and Francisco Ramón Feito. “Fast collision detection between high resolution polygonal models”. In: *Computers & Graphics* 83 (2019), pp. 97–106.

[110] Jianming Liang and Jianhua Gong. “A sparse voxel octree-based framework for computing solar radiation using 3d city models”. In: *ISPRS International Journal of Geo-Information* 6.4 (2017), p. 106.

[111] Milto Miltiadou et al. “A Comparative Study about Data Structures Used for Efficient Management of Voxelised Full-Waveform Airborne LiDAR Data during 3D Polygonal Model Creation”. In: *Remote Sensing* 13.4 (2021), p. 559.

[112] Aaron Knoll. “A survey of octree volume rendering methods”. In: *GI, the Gesellschaft für Informatik* (2006), p. 87.

[113] Emanuele Vespa et al. “Adaptive-resolution octree-based volumetric SLAM”. In: *2019 International Conference on 3D Vision (3DV)*. IEEE. 2019, pp. 654–662.

[114] Gurjeet Singh et al. “FotonNet: A HW-Efficient Object Detection System Using 3D-Depth Segmentation and 2D-DNN Classifier”. In: *arXiv preprint arXiv:1811.07493* (2018).

[115] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[116] Mahyar Najibi et al. “Dops: Learning to detect 3d objects and predict their 3d shapes”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11913–11922.

[117] Nikhila Ravi et al. “Accelerating 3D Deep Learning with PyTorch3D”. In: *arXiv:2007.08501* (2020).

[118] Thomas Chaton et al. “Torch-Points3D: A Modular Multi-Task Frameworkfor Reproducible Deep Learning on 3D Point Clouds”. In: *2020 International Conference on 3D Vision (3DV)*. IEEE. 2020. URL: %5Curl%7Bhttps://github.com/nicolas-chaulet/torch-points3d%7D.

[119] Matthias Fey and Jan E. Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.

[120] Timo Hackel et al. “Semantic3d. net: A new large-scale point cloud classification benchmark”. In: *arXiv preprint arXiv:1704.03847* (2017).

[121] Li Jiang et al. “Pointgroup: Dual-set point grouping for 3d instance segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 4867–4876.

[122] Charles R Qi et al. “Deep hough voting for 3d object detection in point clouds”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 9277–9286.

[123] Ranjini Surendran and D Jude Hemanth. “Scene Understanding Using Deep Neural Networks—Objects, Actions, and Events: A Review”. In: *International Conference on Innovative Computing and Communications: Proceedings of ICICC 2019, Volume 1*. Vol. 1087. Springer Nature. 2020, p. 223.

[124] Xianfeng Han, Hamid Laga, and Mohammed Bennamoun. “Image-based 3d object reconstruction: State-of-the-art and trends in the deep learning era”. In: *IEEE transactions on pattern analysis and machine intelligence* (2019).

[125] Google AI. *Announcing the Objectron Dataset*. 2020. URL: https://ai.googleblog.com/2020/11/announcing-objectron-dataset.html.

[126] Yuan Tian et al. “Occlusion and Collision Aware Smartphone AR Using Time-of-Flight Camera”. In: *International Symposium on Visual Computing*. Springer. 2019, pp. 141–153.

[127] *Unreal Engine 5 is now available in Early Access!* URL: https://www.unrealengine.com/en-US/blog/unreal-engine-5-is-now-available-in-early-access.
[128] *Spatial*. URL: https://spatial.io/.
[129] Asma. *What is a 3D Modeled Environment?* Feb. 2021. URL: https://it-s.com/what-is-a-3d-modeled-environment/.
[130] Jack Collins et al. "A Review of Physics Simulators for Robotic Applications". In: *IEEE Access* (2021).
[131] Unity Tecnologies. *Unity*. 2021. URL: https://unity.com/.
[132] Roboti LLC. *MuJoCo*. 2018. URL: http://www.mujoco.org/.
[133] Open Source Robotics Foundation. *Gazebo Simulator*. 2014. URL: http://gazebosim.org/.
[134] Alexey Dosovitskiy et al. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.
[135] Pasquale Scionti. *Silentday Created with Unreal Engine 4.26 Physical Based Lighting Values*. 2021. URL: https://www.artstation.com/artwork/9m6PWo.
[136] June 8 and Gerard Andrews. *What is synthetic data?* July 2021. URL: https://blogs.nvidia.com/blog/2021/06/08/what-is-synthetic-data/.
[137] Gartner$_{I}$nc. *Maverick* Research: Forget About Your Real Data - Synthetic Data Is the Future of AI*. URL: https://www.gartner.com/en/documents/4002912.
[138] Scott Belsky. *Announcing Adobe Substance 3D: Tools for the next generation of creativity*. 2021. URL: https://blog.adobe.com/en/publish/2021/06/23/announcing-adobe-substance-3d-tools-for-the-next-generation-of-creativity.html.
[139] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge, 1998.
[140] Rickard Eriksson. *Deep Reinforcement Learning Applied to an Image-Based Sensor Control Task*. 2021.
[141] Sovit Ranjan Rath. *Multi-Head Deep Learning Models for Multi-Label Classification - DebuggerCafe*. https://debuggercafe.com/multi-head-deep-learning-models-for-multi-label-classification/. (Accessed on 08/25/2021). Jan. 2021.
[142] Michael Luckert and Moritz Schaefer-Kehnert. *Using machine learning methods for evaluating the quality of technical documents*. 2016.
[143] Unity Technologies. *Unity ML-Agents Toolkit*. 2021. URL: https://github.com/Unity-Technologies/ml-agents.
[144] Asaf Eldad. *Unity vs Unreal, What Kind of Game Dev Are You?* 2021. URL: https://www.incredibuild.com/blog/unity-vs-unreal-what-kind-of-game-dev-are-you.
[145] Lexie. *[RELEASED] Hx Volumetric Lighting*. 2016. URL: https://forum.unity.com/threads/released-hx-volumetric-lighting.420343/.
[146] OpenAI. *OpenAI Gym: A toolkit for developing and comparing reinforcement learning algorithms*. 2021. URL: https://github.com/openai/gym.
[147] Unity Technologies. *Create together with Unity Collaborate, part of Unity Teams*. 2021. URL: https://unity.com/unity/features/collaborate.
[148] Yuxing Xie, Jiaojiao Tian, and Xiao Xiang Zhu. "Linking points with labels in 3D: A review of point cloud semantic segmentation". In: *IEEE Geoscience and Remote Sensing Magazine* 8.4 (2020), pp. 38–59.
[149] Yong Zhou et al. "Voxelization modelling based finite element simulation and process parameter optimization for Fused Filament Fabrication". In: *Materials & Design* 187 (2020), p. 108409.
[150] Zhao Dong et al. "Real-time voxelization for complex polygonal models". In: *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings*. IEEE. 2004, pp. 43–50.
[151] Charles Loop, Cha Zhang, and Zhengyou Zhang. "Real-time high-resolution sparse voxelization with application to image-based modeling". In: *Proceedings of the 5th High-performance Graphics Conference*. 2013, pp. 73–79.
[152] Sergio Orts-Escolano et al. "Holoportation: Virtual 3d teleportation in real-time". In: *Proceedings of the 29th annual symposium on user interface software and technology*. 2016, pp. 741–754.
[153] Yi Zhang and Fei Huang. "Panoramic Visual SLAM Technology for Spherical Images". In: *Sensors* 21.3 (2021), p. 705.
[154] Andrew J Davison et al. "MonoSLAM: Real-time single camera SLAM". In: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007), pp. 1052–1067.
[155] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. "Visual SLAM algorithms: a survey from 2010 to 2016". In: *IPSJ Transactions on Computer Vision and Applications* 9.1 (2017), pp. 1–11.

[156] Shih, Jeffrey and Travnik, Jaden and Martel, Eric. *How Eidos-Montréal created Grid Sensors to improve observations for training agents*. 2020. URL: https://blog.unity.com/technology/how-eidos-montreal-created-grid-sensors-to-improve-observations-for-training-agents.

[157] Kenny Young and Tian Tian. "Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments". In: *arXiv preprint arXiv:1903.03176* (2019).

[158] Mathias Baske. *Grid Sensors for Unity ML-Agents - Version 2.0*. 2021. URL: https://github.com/mbaske/grid-sensor.

[159] mbaske. *Unity Machine Learning Controlled Explorer Drone*. 2020. URL: https://github.com/mbaske/ml-explorer-drone.

[160] I Dan Melamed. "Measuring semantic entropy". In: *Tagging Text with Lexical Semantics: Why, What, and How?* 1997.

[161] Sketchfab. *Sketchfab - The Best 3D Viewer*. 2021. URL: https://sketchfab.com/.

[162] Christopher Berner et al. "Dota 2 with large scale deep reinforcement learning". In: *arXiv preprint arXiv:1912.06680* (2019).

[163] Ilge Akkaya et al. "Solving rubik's cube with a robot hand". In: *arXiv preprint arXiv:1910.07113* (2019).

[164] Chao Yu et al. "The surprising effectiveness of mappo in cooperative, multi-agent games". In: *arXiv preprint arXiv:2103.01955* (2021).

[165] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[166] Michał Kempka et al. "Vizdoom: A doom-based ai research platform for visual reinforcement learning". In: *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE. 2016, pp. 1–8.

[167] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.

[168] Lasse Espeholt et al. "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1407–1416.

[169] Stefan Freyr Gudmundsson et al. "Human-like playtesting with deep learning". In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE. 2018, pp. 1–8.

[170] Unity Technologies. *Learning Environment Design Agents | Unity ML-Agents*. 2022. URL: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Design-Agents.md#visual-observations.

[171] André Brandenburger, Diego Rodriguez, and Sven Behnke. "Mapless Humanoid Navigation Using Learned Latent Dynamics". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 1555–1561.

[172] Linhai Xie, Andrew Markham, and Niki Trigoni. "Snapnav: Learning mapless visual navigation with sparse directional guidance and visual reference". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 1682–1688.

[173] Mark Pfeiffer et al. "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations". In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 4423–4430.

[174] Silvan Weder et al. "Routedfusion: Learning real-time depth map fusion". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 4887–4897.

[175] Margarita Grinvald et al. "TSDF++: A Multi-Object Formulation for Dynamic Object Tracking and Reconstruction". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 14192–14198.

[176] Alex Yu et al. "Plenoxels: Radiance Fields without Neural Networks". In: *arXiv preprint arXiv:2112.05131* (2021).

[177] Andrews, Gerard. *What is Synthetic Data? | NVIDIA Blogs*. 2021. URL: https://blogs.nvidia.com/blog/2021/06/08/what-is-synthetic-data/.

[178] Wood, Thomas. *F1 Score Defintion | DeepAI*. 2022. URL: https://deepai.org/machine-learning-glossary-and-terms/f-score.

[179] Cicala, Roger. *The Camera Versus the Human Eye*. 2012. URL: https://petapixel.com/2012/11/17/the-camera-versus-the-human-eye/.

[180] Weights Biases. *Weights Biases - Developer tools for ML*. 2022. URL: https://wandb.ai/.

[181] Nico Messikommer et al. "Bridging the Gap between Events and Frames through Unsupervised Domain Adaptation". In: *IEEE Robotics and Automation Letters* (2022).

[182] Manasi Muglikar, Diederik Paul Moeys, and Davide Scaramuzza. "Event Guided Depth Sensing". In: *2021 International Conference on 3D Vision (3DV)*. IEEE. 2021, pp. 385–393.
[183] Kamal K Ndousse et al. "Emergent social learning via multi-agent reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 7991–8004.
[184] Xingyi Zhou et al. "Detecting Twenty-thousand Classes using Image-level Supervision". In: *arXiv preprint arXiv:2201.02605* (2022).
[185] Marc Whitten. *Welcome, Ziva Dynamics! | Unity Blog*. 2022. URL: https://blog.unity.com/news/welcome-ziva-dynamics.
[186] Thomas Huang. "Computer vision: Evolution and promise". In: (1996).
[187] Ebrahim Karami, Mohamed Shehata, and Andrew Smith. "Image identification using SIFT algorithm: Performance analysis against different image deformations". In: *arXiv preprint arXiv:1710.02728* (2017).
[188] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features". In: *European conference on computer vision*. Springer. 2006, pp. 404–417.
[189] Edward Rosten and Tom Drummond. "Machine learning for high-speed corner detection". In: *European conference on computer vision*. Springer. 2006, pp. 430–443.
[190] Alexander Goldenshluger, Assaf Zeevi, et al. "The Hough Transform Estimator". In: *The Annals of Statistics* 32.5 (2004), pp. 1908–1932.
[191] Frank CD Tsai. "Geometric hashing with line features". In: *Pattern Recognition* 27.3 (1994), pp. 377–389.
[192] Anuja Bhargava and Atul Bansal. "Fruits and vegetables quality evaluation using computer vision: A review". In: *Journal of King Saud University-Computer and Information Sciences* (2018).
[193] Xiaofan Zhang et al. "Skynet: a hardware-efficient method for object detection and tracking on embedded systems". In: *arXiv preprint arXiv:1909.09709* (2019).
[194] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.
[195] Jia Deng et al. "ImageNet: A Large-scale Hierarchical Image Database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.
[196] viso.ai. *Object Detection in 2021: The Definitive Guide*. 2021. URL: https://viso.ai/deep-learning/object-detection/.
[197] Saurabh Gupta et al. "Aligning 3D models to RGB-D images of cluttered scenes". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 4731–4740.
[198] Buyu Li et al. "Gs3d: An efficient 3d object detection framework for autonomous driving". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 1019–1028.
[199] Caner Sahin et al. "A review on object pose recovery: from 3d bounding box detectors to full 6d pose estimators". In: *Image and Vision Computing* (2020), p. 103898.
[200] Martin Engelcke et al. "Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1355–1361.
[201] Frank Michel et al. "Global hypothesis generation for 6D object pose estimation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 462–471.
[202] Danfei Xu, Dragomir Anguelov, and Ashesh Jain. "Pointfusion: Deep sensor fusion for 3d bounding box estimation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 244–253.
[203] Vishakh Hegde and Reza Zadeh. "FusionNet: 3d Object Classification using Multiple Data Representations". In: *arXiv preprint arXiv:1607.05695* (2016).
[204] Yin Zhou and Oncel Tuzel. "Voxelnet: End-to-end learning for point cloud based 3d object detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4490–4499.
[205] Jason Ku, Alex D. Pon, and Steven L. Waslander. "Monocular 3D Object Detection Leveraging Accurate Proposals and Shape Reconstruction". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019. DOI: 10.1109/cvpr.2019.01214. URL: https://doi.org/10.1109%2Fcvpr.2019.01214.

[206] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. "Pointrcnn: 3d object proposal generation and detection from point cloud". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 770–779.

[207] Wadim Kehl et al. "Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1521–1529.

[208] Yinda Zhang et al. "Deepcontext: Context-encoding neural pathways for 3d holistic scene understanding". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1192–1201.

[209] Stefan Hinterstoisser et al. "Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes". In: *Asian conference on computer vision*. Springer. 2012, pp. 548–562.

[210] Reyes Rios-Cabrera and Tinne Tuytelaars. "Discriminatively trained templates for 3d object detection: A real time scalable approach". In: *Proceedings of the IEEE international conference on computer vision*. 2013, pp. 2048–2055.

[211] E. Munoz et al. "Fast 6D pose estimation for texture-less objects from a single RGB image". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2016. DOI: 10.1109/icra.2016.7487781. URL: https://doi.org/10.1109%2Ficra.2016.7487781.

[212] Jason Ku et al. "Joint 3D Proposal Generation and Object Detection from View Aggregation". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2018. DOI: 10.1109/iros.2018.8594049. URL: https://doi.org/10.1109%2Firos.2018.8594049.

[213] Bertram Drost et al. "Model globally, match locally: Efficient and robust 3D object recognition". In: *2010 IEEE computer society conference on computer vision and pattern recognition*. Ieee. 2010, pp. 998–1005.

[214] Mustafa Mohamad, David Rappaport, and Michael Greenspan. "Generalized 4-Points Congruent Sets for 3D Registration". In: *2014 2nd International Conference on 3D Vision*. IEEE, Dec. 2014. DOI: 10.1109/3dv.2014.21. URL: https://doi.org/10.1109%2F3dv.2014.21.

[215] He Wang et al. "Normalized Object Coordinate Space for Category-Level 6D Object Pose and Size Estimation". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019. DOI: 10.1109/cvpr.2019.00275. URL: https://doi.org/10.1109%2Fcvpr.2019.00275.

[216] Xinchen Yan et al. "Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction without 3D Supervision". In: *Advances in neural information processing systems*. 2016, pp. 1696–1704.

[217] Christopher B Choy et al. "3d-r2n2: A unified approach for single and multi-view 3d object reconstruction". In: *European conference on computer vision*. Springer. 2016, pp. 628–644.

[218] Jiajun Wu et al. "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling". In: *Advances in neural information processing systems* 29 (2016), pp. 82–90.

[219] Peng-Shuai Wang et al. "O-cnn: Octree-based convolutional neural networks for 3d shape analysis". In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–11.

[220] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. "Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2088–2096.

[221] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. "Octnet: Learning deep 3d representations at high resolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 3577–3586.

[222] Jeong Joon Park et al. "Deepsdf: Learning continuous signed distance functions for shape representation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 165–174.

[223] Lars Mescheder et al. "Occupancy networks: Learning 3d reconstruction in function space". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4460–4470.

[224] Jhony K Pontes et al. "Image2mesh: A learning framework for single image 3d reconstruction". In: *Asian Conference on Computer Vision*. Springer. 2018, pp. 365–381.

[225] Andrey Kurenkov et al. "Deformnet: Free-form deformation network for 3d shape reconstruction from a single image". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 858–866.

[226] Haoqiang Fan, Hao Su, and Leonidas J Guibas. “A point set generation network for 3d object reconstruction from a single image”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 605–613.

[227] Amir Arsalan Soltani et al. “Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1511–1519.

[228] Fathi, Alireza. *Google AI Blog: 3D Scene Understanding with Tensorflow 3D*. 2021. URL: https://ai.googleblog.com/2021/02/3d-scene-understanding-with-tensorflow.html.

List of Figures

List of Tables

A

APPENDIX

A.1. CONTACT INFORMATION AND CODE ACCESS

If you have any questions regarding this work, feel free to contact me at j.francisco.ribera@gmail.com. The code for the 3D environments in unity with the MLAgents code will be available at https://github.com/Ademord/ma-unity.

A.2. RELATED WORK: 3D VISION

Several works have been grabbing inspiration from biological vision [27], where the authors discuss the foundations of existing computational studies modelling biological vision, considering three classical computer vision tasks from a biological perspective: image sensing, segmentation and optical flow. They describe how computer vision engineers limit their scopes to the distributions described in a dataset and the set of stimuli that their architectures describe. Along these lines, a new framework called Ventral-Dorsal Networks (VDNets) [28] follows the brain's physiology by separating the "what" from the "where". The authors use a top down saliency analysis to identify irrelevant image regions, i.e., a selective attention mechanism that masks out noise and unimportant background information in the image.

The concept of information over time is equally important. On one hand, researchers [29] discovered that there is a part of the brain that reacts to "familiarity", such as familiar faces, but is separated from long or short term memory. On the other hand, there are works [30] that serve as a great example that leverage the temporal dimension for action recognition or for accurate classification. One of them, proposes to embed a "Temporal Transition Layer" in the DenseNet architecture, creating their proposed "Temporal 3D ConvNet" [31]. They also provide a technique to leverage learned features from 2D ConvNets, so that a pretrained weights from a 2D CNN can be transfered to a 3D CNN (this technique is not limited to RGB models). Similarly, Hou et al. [32] conduct a detailed ablation study to identify the individual contributions of the components of their proposed 3D CNN framework for doing action and object segmentation in video, using separable filters to reduce the computational burden of standard 3D convolutions. Other research in this direction includes PointNet [33, 34, 35], which is able to localize objects in large-scene point clouds with high efficiency and high recall, regardless of heavy occlusion or with very sparse points. However their method struggles with multi-instance scenarios, pose estimation accuracy in sparse point clouds, and 3D detection if the 2D detector faces strong occlusion. Similarly, applicability of these methods in industries is as important, for example by leveraging panoptic segmentation in industrial panoramas for carrying out inventories [36].

A.3. COMPUTER VISION

Computer vision has its origins in the 1960s, when computer models were conceptualized in an attempt to simulate human perception. Back then, computer vision was studied following a "blocks" approach (polyhedra), as introduced by Larry Roberts [186]. Later, David Marr proposed a set of required stages for carrying out image understanding. His goal was to reach a 3D understanding of the models in a scene. Figure A.1 illustrates Marr's algorithm to identify the visual world.

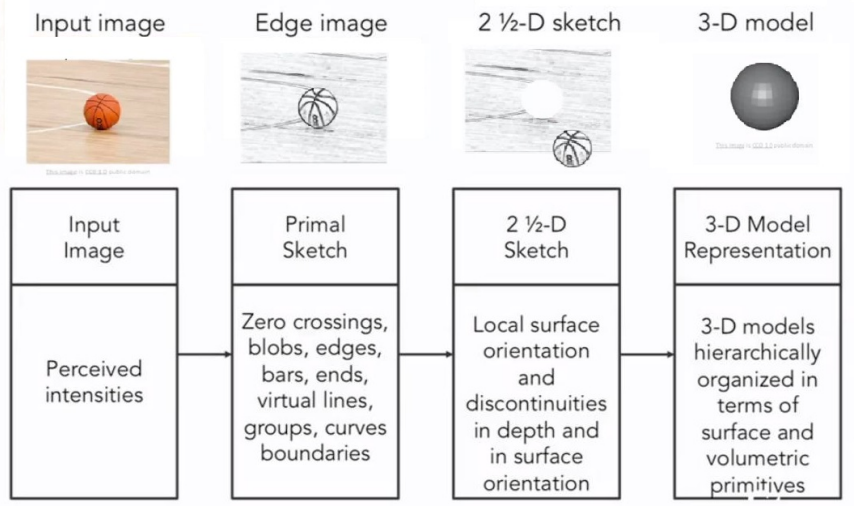


Figure A.1: David Marr's stages to identify the visual world.

Throughout the years, however, most comvision applications did not require complete 3D object models of the world. Nevertheless, Marr's paradigm shaped the future of computer vision for the years to come. The methods developed until circa 2000 analyzed images in terms of simple geometric features, such as a combination of lines. The basic initial goal for computer vision was to achieve object segmentation,

separating the background from salient objects. Later, the analysis of pixels and more complex features allowed to identify colors, shapes and objects in images. When the same features were seen in multiple images, an image could be classified by approximation. These collection of features for classification was called a "bag of words". Consequently, traditional machine learning approaches such as SVMs, boosting methods and graph models eased the usage of features for classification. Examples of features are:

- Scale Invariant Feature Transform (SIFT) [187]
- Speeded Up Robust Features (SURF) [188]
- Features from Accelerated Segment Test (FAST) [189]
- Hough Transforms [190]
- Geometric Hashing [191]

Nowadays, traditional computer vision techniques are used in aerospace field, industrial automation, security inspections, intelligent transportation systems, security and transportation systems Bhargava and Bansal are mostly used to perform simple tasks where machine learning is excessive or in situations where there are memory constraints (microcontrollers). However, machine learning methods for computer vision not only outperform classical computer vision in some tasks, but have also shown progress in classical computer-vision-dominated fields, such as embedded systems, as discussed by Zhang et al. [193] in “Skynet: a hardware-efficient method for object detection and tracking on embedded systems”.

A.3.1. DEEP LEARNING FOR VISION

In 1943, following the ambition of trying to simulate the way the human brain is made up of a collection of simple units, McCulloch and Pitts [194] developed a model of a neuron. They called this neuron MCP, which contributed to development of the neural networks we know today. Since around 2010, the rise of both public data and GPU hardware availability have allowed deep learning techniques to outperform traditional machine learning methods and reach super-human performance in a variety of tasks such as object detection, semantic segmentation, motion tracking, human pose estimation, and action recognition.

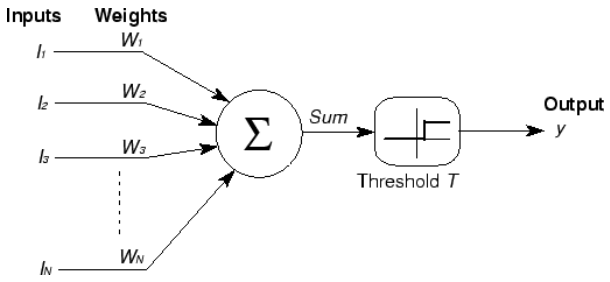


Figure A.2: McCulloch-Pitts model of a neuron.

As of August 2021, ImageNet [195] has become one of the largest high-quality public databases for image classification and the paper “ImageNet Classification with Deep Convolutional Neural Networks” [115] has been cited over 85000 times. Other breakthroughs in the field, such as the alleviation of the vanishing gradients problem, new regularization techniques and the surge of frameworks for deep learning development, contributed to bringing machine and deep learning to the state we know of today. Consequently, deep learning techniques have defined themselves as the state-of-the-art for the task of object detection, which is of special interest for this work and further sections.

OBJECT DETECTION

Object detection involves localizing and classifying objects in a 2D image. Similarly, object segmentation is an extension of the recognition problem, which involves assigning labels to the specific pixels in a given image that belong to the detected objects.

This section describes the problem of object detection as a way to infer semantics from the environment, which is an important concept for the later definition of semantic entropy in Chapter 3. Methods for inferring semantics from the environment include:

- Object Detection
- Object Segmentation
- Instance Segmentation
- Panoptic Segmentation
- Natural language descriptors of visual descriptors

Given the time constraint to experiment with different kinds of semantic entropy, this thesis will focus on 2D object detection methods for acquiring semantics from the environment. Therefore, two relevant algorithms will be described: Mask-RCNN and YOLO. For historical relevancy, some traditional machine learning methods for 2D semantics are worth mentioning, such as: the Viola-Jones detector, HOG detectors, deformable part models, SVMs, Random Forests, K-means clustering. Deep learning architectures have however evolved extensively over the past few years and become the norm. Additionally:

- Relevant two-stage detectors include: RCNN and SPPNet (2014), Fast RCNN and Faster RCNN (2015), Pyramid Networks/FPN (2017), G-RCNN (2021).
- Relevant one-stage object detection algorithms include: YOLO (2016), SSD (2016), RetinaNet (2017), YOLOv3 (2018), YOLOv4 (2020) and YOLOR (2021).

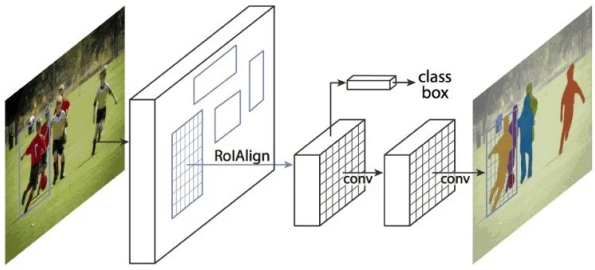


Figure A.3: The Mask R-CNN Framework for Instance Segmentation [196].

Mask-RCNN. Mask-RCNN's architecture (2017) is a two-stage object detector came to be as a sequential evolution of other architectures, where previous iterations tackled given bottle necks to improve both accuracy and speed of the model. The first step in these detectors involves a region proposal method. The second step involves object classification and bounding box regression based on features obtained during previous step [196]. The sequence of models that paved the road for Mask-RCNN are the following:

- **R-CNN:** a "selective search" algorithm proposes bounding boxes and features are obtained using a deep convolutional neural network (for example, AlexNet). Object classifications are then made with linear SVMs.
- **Fast R-CNN:** unifies the feature detector, and the bounding box predictor approach into a single model, but the region of interests are still part of the input. The shared computation showed speed improvements.
- **Faster R-CNN:** unifies the region proposal algorithm into the CNN model. This model merges a RPN (region proposal network) with Fast R-CNN.
- **Mask R-CNN:** extends the previous model to add pixel-level image segmentation. A small fully connected network was added that per region of interest outputs a segmentation mask.

YOLO. YOLO, in contrast, is a one-stage detection algorithm which directly predicts bounding boxes in a forward pass [196]. This type of detectors tend to be faster and structurally simpler in exchange for less accuracy: YOLO is 1000x faster than R-CNN and 100x faster than Fast-RCNN.

YOLO, as shown in Figure A.4, is a convolutional neural network which divides an input into cells or regions, and scores objects in these regions based on their similarity to predefined classes. For each region a set of anchor boxes is then predicted with a confidence score. Final boxes are then filtered via non-maximum

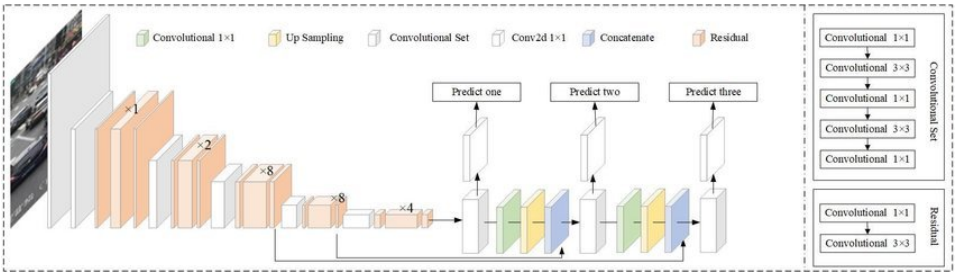


Figure A.4: Structure detail of YOLOv3.It uses Darknet-53 as the backbone network and uses three scale predictions [196].

suppression. Newer variants, such as YOLOv3 or YOLOv4 improve performance on smaller object, self-adversarial training and cross mini-batch normalization [196].

POSE ESTIMATION IN 3D

Pose estimation is an extension of the 3D object recognition problem. It involves predicting a 3D translation and a 3D rotation. These types of methods have been trained using both synthetic [197] and real data [198], labeled with the 6D poses of the objects. Additionally, these methods can be 2D supervision-based or 3D supervision-based. 2D supervision-based methods utilize the information in the RGB image to predict a 3D bounding box, whereas 3D supervision-based methods directly detect 3D bounding boxes from stereo images, RGB-D cameras or LIDAR sensors [199].

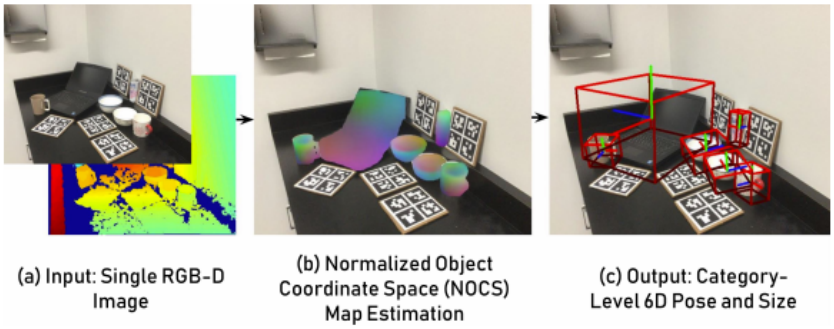


Figure A.5: Wang et al. proposed method for category-level 6D pose and size estimation of multiple unseen objects in an RGB-D image.

Sahin et al. [199] propose a discriminative categorization criteria to organize pose estimation techniques, as follows:

- **Classification-based** approaches leverage 2D information to retrieve a 3D bounding box. Then, a refinement stage is added, such as random forests, to obtain the 6D pose of the object. Examples include: GS3D [198], Vote3Deep [200] and the method by Michel et al. [201].
- **Regression-based** approaches are similar to classification based approaches but they run another neural network directly over the predicted 3D bounding box to regress the 6D pose. Examples include: PointFusion [202], FusionNet [203], VoxelNet [204], AVOD net [108].
- **Classification and Regression-based** approaches run both tasks mentioned above under the same architecture. Examples include MonoPSR [205], FrustumPNet [33], PointRCNN [206], SSD-6D [207] and DeepContext [208].
- **Template matching** approaches look at an image using sliding windows and compare obtained features with a database of pre-defined feature templates, and the pose parameter is given to the window with the closest similarity to the template matched. Examples include: the LINEMOD adaptation from Hinterstoisser et al. [209], SVMs embedded in Adaboost [210], RAPID-LR [211] and the method proposed by Ku et al. [212].
- **Point-pair feature matching** approaches were proposed by Drost et al. [213]. The idea is to store point-pair features in a hash table in a separate "offline" stage. During prediction, potential matches then

are obtained by comparing to a global model representation. Finally, these matches vote on the pose parameters. These methods underperform given objects with similar features, occlusion and sensor noise [214].

Along these lines, work by Wang et al. [215] attempts to push pose estimation techniques to being able to generalize to unknown objects, removing the dependency to knowing an object's 3D model previously. Figure A.5 illustrates a glimpse to the method proposed by Wang et al.

3D METHODS

Han, Laga, and Bennamoun [124] provide a thorough review of recent research for 3D object reconstruction.

A brief overview of the methods that have been successful at 3D object recognition and reconstruction is presented below:

- The majority of works pre-2018 use voxelized representations [216] [217] [218], which allow the representation of arbitrary topology considering both the surface and the internal structure of object. Representation types of data are: volumetric (based on 3D voxel grids), surface-based (meshes, point clouds) and intermediate (3D reconstruction is done based off 2D information from RGB images).
- Since convolutions and overall processing of 3D images has a high memory requirement, volumetric techniques leverage octrees [102], which are sparse partitioning techniques for 3D grid structures (O-CNN [219], OGN [220], OctNet [221]) to achieve high resolutions while maintaining memory efficiency. As described by Tatarchenko, Dosovitskiy, and Brox [220], Octree Generating Networks reduce memory requirements for reconstruction from 4.5 GB to 0.29 GB. These techniques, however, are complex to implement which impacts their reproducibility and further research.
- Other approaches learn continuous Signed Distance Functions [222, 13] or continuous occupancy grids [223]. These methods have a lower memory requirement and can reconstruct the 3D object at the desired resolution.
- Methods that look at multiple frames for reconstructing representations outperform single-view methods for they collect more information about the silhouette of the objects.
- Techniques that try to reconstruct primarily the surface of a seen object (surface-based techniques) outperform volumetric methods by a small margin. These are: Mesh-based approaches [224] and Point-based approaches [225, 226].
- Methods that utilize 2D supervision to reconstruct 3D objects improved their performance since 2017, and are outperformed by 3D supervision-based approaches by a small margin. [216, 227].

Similarly, Han, Laga, and Bennamoun provide a set of research directions based on the challenges and limitations that 3D object recognition currently faces. These areas of improvement can be summarized as follows:

- Bigger training data sets since deep learning methods depend heavily on them.
- Generalization to unseen objects, and not just specific-domain models.
- Methods that allow for 3D reconstruction in higher resolutions, such as reconstruction of hairs, fur and plants.
- Combined approaches for both recognition and reconstruction in a single method.
- Domain specialized methods that can perform well also on unseen objects, but of a specific domain, say wild animals, which are not easy to label.
- Reconstruction of 3D video streams where frames are missing and information is passed from future frames to previous ones.
- Full 3D scene understanding, which requires detection, recognition, reconstruction, spatial awareness and robustness to unseen objects.

A.4. UNITY 3D: MODELLING THE ENVIRONMENT

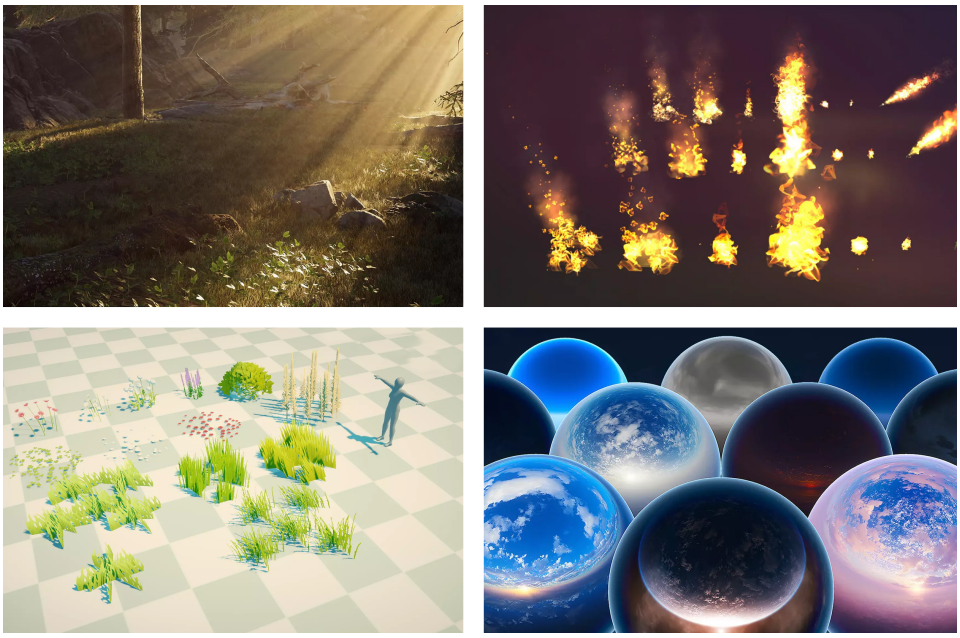


Figure A.6: Diverse Unity 3D Assets used to model 3D environments for the learning agent [**unityAssetStore**]. Wireframe Shader (top left), Drone Bot (top right), Foliage Pack (bottom left), Skyboxes (bottom right).

The modelling of 3D assets was done using Blender , which is a 3D modelling tool with an exhaustive toolset. In order to migrate self-made assets to Unity, the 3D models were exported from Blender using the FBX format. FBX format is a popular xml-based file format, and it is a good choice for exporting complex models that could have many subparts

Moreover, it allows the storage of position, UV and normal data with different topologies, which enables features like accurate subdivision surfaces. The rest of the 3D assets, including shaders and particles, were either obtained through the Unity Asset store , or manually created using the *Unity Shader Graph* Materials were also acquired through the Adobe Substance 3D Source [], which is part of the Adobe 3D Creative Suite mentioned in Section 2.5.2.

Unity allows each object to be used as a *prefab* object, which allows the linking of multiple objects to one single reference. This link provides the opportunity to control the properties of the objects from one single model. For example, a model could be shared by multiple agents and all its parts could be used to define their 3D features, agent-specific characteristics and behaviors.

A.5. UNITY 3D: VOXELIZING 3D MODELS

The current voxelization process involves pre-voxelizing the 3D assets instead of doing it real time. The pre-voxelization of the assets was done using the following script Voxelizer script which can be found in our Github repository linked in Appendix A.1. +

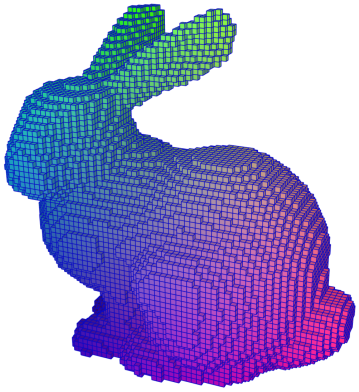


Figure A.7: Voxelization of a 3D asset used with the mesh walking technique. 3D model (left) and voxelized model (right).

As part of our further steps, we plan to look at [174], [117], [228, 116] and [120] for the manipulation of 3D clouds and their respective voxelization. This should cover the analysis of noise, outliers and empty spaces in the cloud generated, and bring our agent closer to a production-ready state.

A.6. UNITY ML-AGENTS

A.6.1. PPO TRAINER: HYPERPARAMETERS

Hyperparemeter	**Default Value**	**Typical Range**
visual encoder type	simple	-
normalize	false	-
learning rate	3e-4	1e-5 - 1e-3
learning rate schedule	linear	linear, constant
batch size	1024	512 - 5120
max steps	500000	5e5 - 1e7
buffer size	10240	2048 - 409600
time horizon	64	32 - 2048
number of layers	2	1-3
hidden units	128	32-512
beta	5.0e-3	1e-4 - 1e-2
epsilon	0.2	0.1 - 0.3
lambda	0.95	0.9 - 0.95
number of epochs	3	3 - 10

Table A.1: Default hyperparameters in PPO trainer, taken from [143]

A.6.2. UNITY PROJECT CLASS DIAGRAM

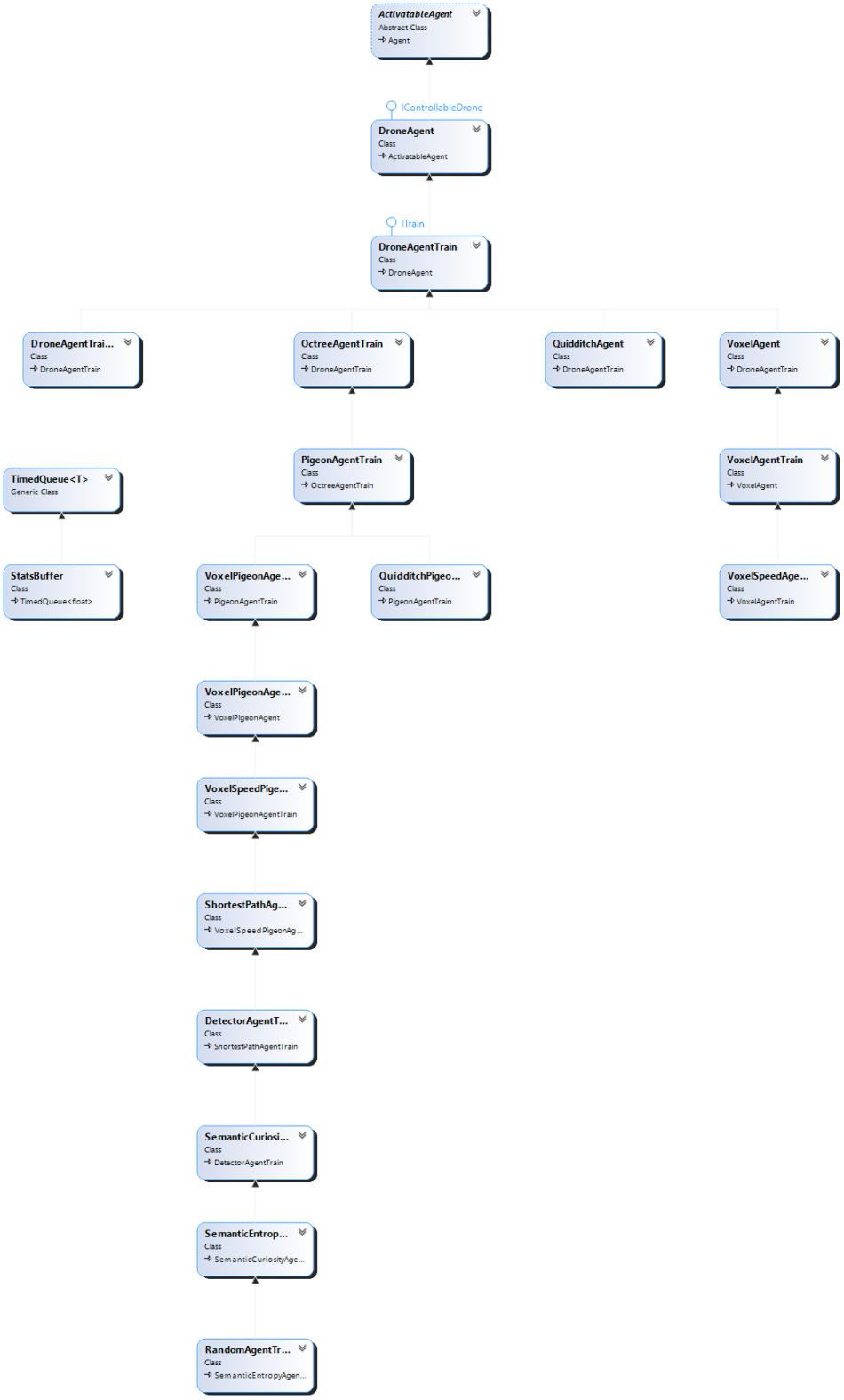


Figure A.8: Class Diagram for the C# implementation for this thesis shows the hierarchical behavior inheritance in the implementation of the agents. Generated using Visual Studio 2019.

A.6.3. EXAMPLE TRAINING CONFIGURATION FILE

```yaml
# config file: run63++100.yaml
behaviors:
  Drone:
    trainer_type: ppo
    hyperparameters:
      batch_size: 1024
      buffer_size: 10240
      learning_rate: 0.0003
      beta: 0.01
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 2
      vis_encode_type: simple
      # no lstm
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
      curiosity:
        gamma: 0.99
        strength: 0.02
        network_settings:
          hidden_units: 256
        learning_rate: 0.0003
    keep_checkpoints: 5
    max_steps: 20000000
    time_horizon: 64
    summary_freq: 30000
```

```yaml
environment_parameters:
  # 0:base, 1:open, 2:fire, 3:forest
  EnvironmentType: 0
  steuernModus: 1
  m_EnableVFX: 0
  m_EnableTrainDebuggingLogs: 0
  # base
  m_TrainMovingForward: 1
  m_TrainTargetSpeed: 1
  # octree
  leafNodeSize: 4
  m_AddOctreeObservations: 0
  m_TrainOctreeDiscovery: 0
  m_TrainLingerPolicy: 0
  m_AddPigeonObservations: 0
  # voxel
  m_TrainVoxelCollection: 1
  allowedToSeeVoxels: 1
  m_VoxelRewardStrength: 1
  NormalizeVoxelReward: 0
  # enabled for all voxel-behaviors
  m_SpeedSensitivityToTargetsInFOV: 1
  # shortest
  m_AddShortestPathObservations: 0
  m_TrainShortestPath: 0
  # detector
  m_LoadDetector: 0
  m_AddDetectorObservations: 0
  m_TrainObjectDetectionMaximization: 0
  NormalizeDetectionsReward: 1
  # curiosity
  m_AddSemanticCuriosityObservations: 0
  m_TrainSemanticCuriosity: 0
  # entropy
  m_AddSemanticEntropyObservations: 0
  m_TrainSemanticEntropy: 0
```

A.6.4. AGENT BEHAVIORS UI

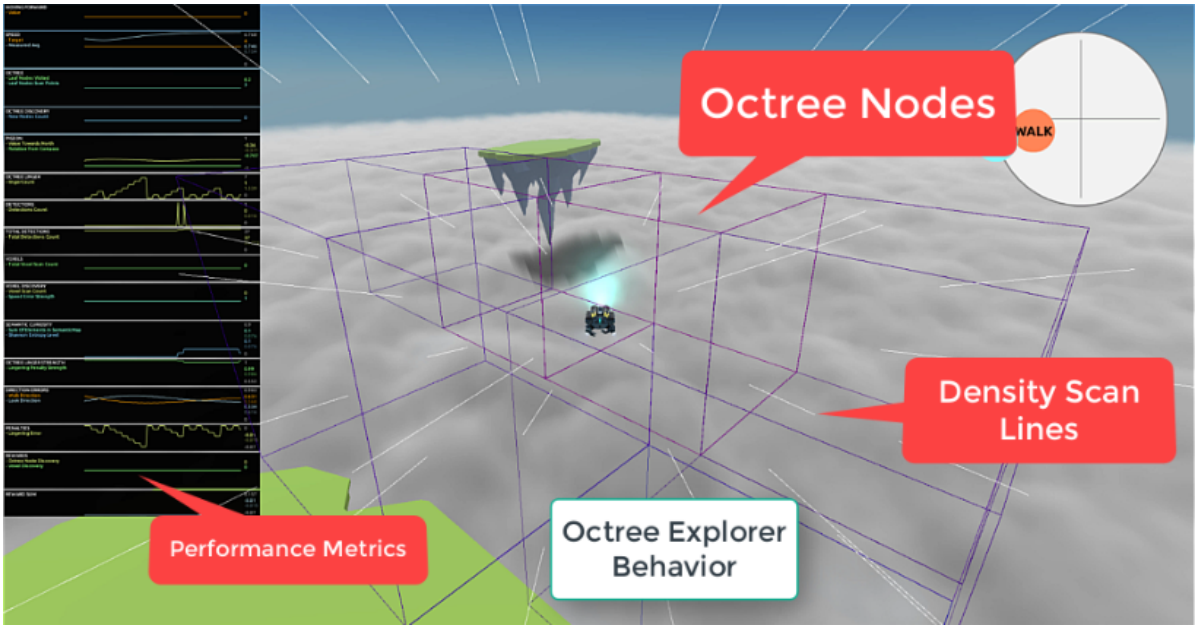


Figure A.9: Octree nodes and agent perspective in the training environment.

Figure A.10: Agent perspective in the training environment of the voxelized scan goal, including the nearest object compass, object detector camera, voxels-only camera and performance metrics.

A.6.5. INFLUENCE OF VARIABLES

These charts also suggest that the lingering penalty has a smaller influence on the performance of agents, regardless of different strengths of the voxel reward.

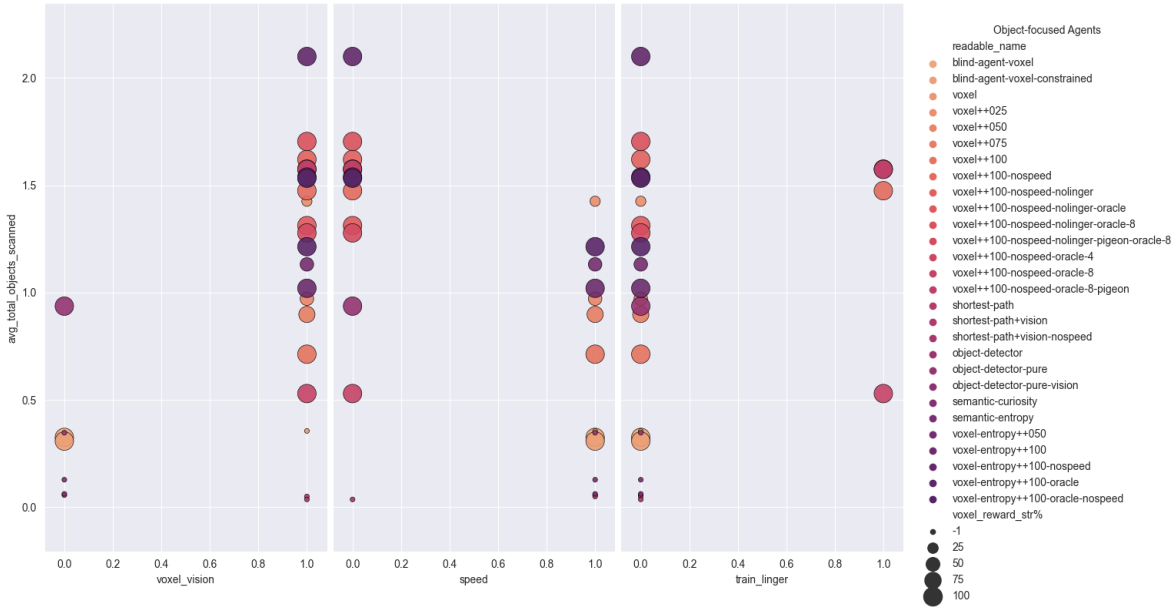


Figure A.11: Comparison of the influence of relevant variables on the total amount of objects scanned for all object-focused runs. The variables for *pigeon* and *pathak* are always set to 1 (active).

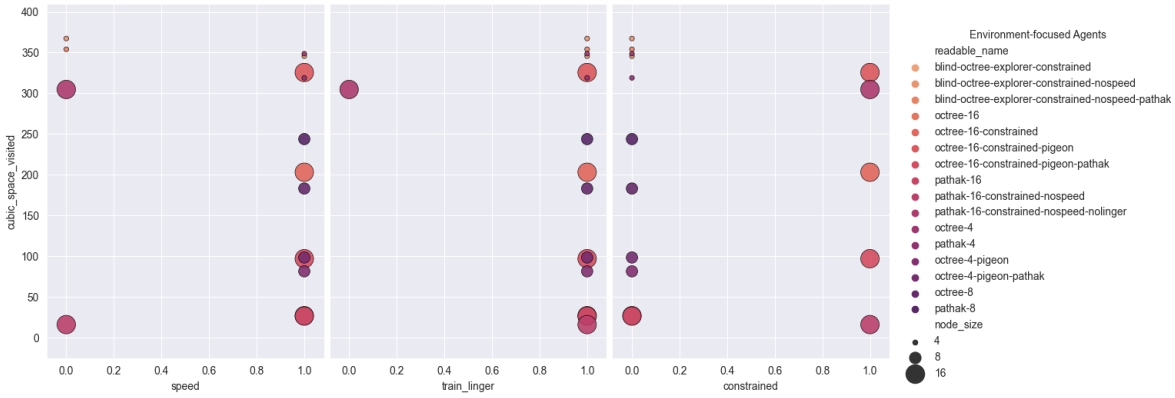


Figure A.12: Comparison of the influence of relevant variables on the visited volume for all environment-focused runs.

A.6.6. IN-DEPTH METRICS

The following in-depth metrics were collected per agent

- name
- speed
- linger
- pigeon
- pathak
- constrained
- node_size
- train_octree_discovery
- train_voxel_discovery
- voxel_reward_str
- voxel_vision
- oracle
- train_entropy
- sac
- n_leafNodes
- n_leafNodes_y
- leafNodeVolume
- %leafNodesExplored
- %scanNodesExplored
- visual_analysis
- %episode_length
- avg_total_objects_scanned

A.6.7. NAMING

Following what was explained in Section 3.7.2, the run names that have a number such as *run63++025* indicate that the voxel reward has a weight factor of 25%, whereas *run63++100* provides the agent a 100% of the voxel reward. This allowed the analysis of different influences of voxel reward values and their impact on the agent's behavior. Furthermore, *-constrained* indicates that the training environment manager will not reset the episode if the agent hits a boundary wall.

Similarly, following what was presented in equation 3.8, agents that are labeled with the -entropy suffix, such as *voxel-entropy++100*, observe the variable *lingering-penalty-strength* and accordingly modify the lingering penalty at each timestep. This should enable agents to navigate longer in locations with high entropy. In contrast, agents with the suffix *-noTrainEntropy* only introduce observe the strength factor but do not modify the lingering penalty.

Given its performance, the baseline for mixed-focus agents is also known as *voxel-entropy-nospeed-octree-pigeon-pathak*. As mentioned in A.6.4, the variants of this baseline are created varying the octree leaf node size, the lingering penalty and the "noTrainEntropy" suffix, which refers to agents that receive entropy observations yet do not modify the lingering penalty based on the entropy levels.

The following tables further explain the names of each agent in terms of rewards and observations.

Agent Definitions by Observations

	Obs. per Baseline								
Baseline	Octree Obs.	Voxel Vision	Pigeon Obs.	Speed Sensitivity to FOV	Shortest Path Obs.	Detector Obs.	Semantic Curiosity Obs.	Semantic Entropy Obs.	Comment
Random Walk									Simple agent with obs: speed and position in 3D space.
Octree Agent	x		x						DA with discovered octree nodes count each timestep.
Voxel (R_VOX norm)		x		x					DA that observes voxels with a normalized reward.
Voxel (R_VOX str: 0.25)		x		x					DA that observes voxels with a reward strength of 25%
Voxel (R_VOX str: 0.50)		x		x					DA that observes voxels with a reward strength of 50%
Voxel (R_VOX str: 0.75)		x		x					DA that observes voxels with a reward strength of 75%
Voxel (R_VOX str: 1.00)		x		x					DA that observes voxels with a reward strength of 100%
Shortest Path				x	x				DA with walk angle and look angle to target.
Shortest Path + Vision		x		x					DA with voxel vision.
Object Detection				x		x			Simple agent with input from object detector.
Object Detection + pure				x					Simple agent without speed constraint.
Semantic Curiosity				x			x		DA with rationalized class density each timestep.
Semantic Entropy				x					DA with lingering penalty strength present reduction.
Explorer (R_VOX str: 0.5)	x	x	x	x				x	DA as above. Voxel reward strength of 50%
Explorer (R_VOX str: 1.0)	x	x	x	x				x	DA as above. Voxel reward strength of 100%

Table A.2: Diverse agent behavior variants and the observations they receive. "DA" stands for Derived Agent. All agents observe their own information, such as 3D position, orientation, etc.

AGENT DEFINITIONS BY REWARDS

	Trained Functions							
Baseline	Speed	Octree Nodes Discovery	Voxel Discovery	Shortest Path	Object Detection	Semantic Curiosity	Semantic Entropy	Comment
Random Walk	x							Simple that moves at a minimum target speed.
Octree Agent	x	x						DA that explores new (octree) locations.
Voxel (R_VOX norm)	x		x					"" ""
Voxel (R_VOX str: 0.25)	x		x					"" ""
Voxel (R_VOX str: 0.50)	x		x					"" ""
Voxel (R_VOX str: 0.75)	x		x					"" ""
Voxel (R_VOX str: 1.00)	x		x					"" ""
Shortest Path	x		x	x				DA that learns to move to the closest target.
Shortest Path + Vision	x		x	x				DA with voxel vision.
Object Detection	x		x		x			DA that maximizes object detections.
Object Detection + pure			x		x			DA with no minimum speed penalty.
Semantic Curiosity	x		x			x		DA that maximizes the temporal class density.
Ultron (Semantic Entropy)	x		x				x	Voxel-DA with a lingering penalty regulator.
Explorer (R_VOX str: 0.5)	x	x	x				x	Explorer DA for voxels and octrees.
Explorer (R_VOX str: 1.0)	x	x	x				x	Explorer DA for voxels and octrees.

Table A.3: Diverse agent behavior variants and the rewards they receive. "DA" stands for Derived Agent.

A.6.8. VOXEL-FOCUSED AGENTS

Table A.4 shows the results of relevant object-focused agents that are aware of voxels, using the total detections, the walk error and the look error. These last two errors represent the directional error to the nearest target for the agent; an agent may walk towards a target but look in another direction.

Method	Total Detections Count	Walk Error	Look Error
voxel	61	0.48	0.42
voxel++025	42	0.46	0.37
voxel++050	42	0.48	0.47
voxel++075	37	0.47	0.46
voxel++100	36	0.47	0.44
voxel++100-nospeed	31	0.46	0.41
voxel++100-nospeed-nolinger	42	0.47	0.41
voxel++100-nospeed-nolinger-oracle	48	0.43	0.41
voxel++100-nospeed-oracle-4	39	0.39	0.38
shortest-path+vision	36	0.42	0.43
shortest-path+vision-nospeed	52	0.20	0.22
object-detector-pure-vision	30	0.46	0.42
voxel-entropy++050	28	0.47	0.47
voxel-entropy++100-oracle-nospeed	38	0.44	0.38

Table A.4: Overview of the voxel-aware agents with respect to the *Total Detections*, the *Walk Error* and *Look Error* metrics. These errors are with respect to the direction to the nearest target. An agent can walk in one direction and look towards another one.

The following Table A.5 presents further details for these agents.

A.6.9. Voxel-focused Agents

Method	F1-score	Average Total Objects Scanned	Visited Volume (m^3)	Octree Leaf Nodes Visited	Octree Lingering Penalty Count	Octree Lingering Penalty Strength
shortest-path+vision-nospeed	0.02	0.04	33.29	8.32	98.00	0.91
shortest-path+vision	0.02	0.05	79.22	19.80	11.96	0.96
shortest-path	0.03	0.06	63.70	15.92	14.42	0.96
semantic-curiosity	0.03	0.06	54.12	13.53	8.41	0.86
semantic-entropy	0.05	0.13	63.50	15.87	4.78	0.95
blind-agent-voxel-constrained	0.12	0.31	112.86	14.11	19.51	0.96
blind-agent-voxel	0.12	0.33	77.84	9.73	24.91	0.95
object-detector	0.14	0.35	158.48	39.62	13.40	0.91
voxel	0.14	0.36	115.75	28.94	10.47	0.93
voxel++100-nospeed-oracle-8	0.18	0.53	91.58	11.45	33.79	0.95
voxel++100	0.23	0.71	116.50	29.13	23.72	0.96
voxel++075	0.27	0.90	118.24	29.56	11.07	0.96
object-detector-pure	0.27	0.94	110.38	27.59	44.95	0.95
voxel++050	0.27	0.97	106.95	26.74	23.25	0.95
voxel-entropy++100	0.31	1.02	131.65	32.91	23.51	0.95
voxel-entropy++050	0.34	1.13	151.31	37.83	21.44	0.94
voxel-entropy++100-oracle	0.33	1.21	123.13	30.78	18.79	0.95
voxel++100-nospeed-nolinger-pigeon-oracle-8	0.31	1.28	100.24	25.06	37.67	0.95
voxel++100-nospeed-nolinger-oracle	0.32	1.31	100.71	25.18	39.59	0.94
voxel++025	0.40	1.43	150.49	37.62	29.88	0.95
voxel++100-nospeed	0.40	1.47	144.93	36.23	15.87	0.96
voxel-entropy++100-oracle-nospeed	0.38	1.53	124.80	31.20	24.39	0.95
object-detector-pure-vision	0.35	1.54	105.10	26.28	15.57	0.93
voxel++100-nospeed-oracle-8-pigeon	0.40	1.57	137.12	17.14	34.60	0.94
voxel++100-nospeed-oracle-4	0.32	1.58	86.76	21.69	9.66	0.96
voxel++100-nospeed-nolinger	0.41	1.62	138.43	34.61	29.76	0.95
voxel++100-nospeed-nolinger-oracle-8	0.32	1.70	85.04	21.26	17.45	0.96
voxel-entropy++100-nospeed	0.48	2.10	147.69	36.92	25.97	0.91

Table A.5: Overview of all voxel-aware agents for the object exploration task, with respect to the *Octree Scan Points*, *Octree Leaf Nodes Visited*, *Average Total Objects Scanned*, *Lingering Penalty Count* and *Lingering Penalty Strength* metrics. As mentioned before, the lingering penalty strength is regulated by the semantic entropy variable. Oracle agents observe the look and walk angle towards the nearest target, hence they are considered a baseline of an agent with "global awareness".

A.6.10. Octree-focused Agents

Method	Episode Length %	Leaf Nodes XZ-Axis	Leaf Nodes Y-Axis	Node Volume	Octree Leaf Nodes Visited	Visited Volume (m^3)	New Nodes Discovery Rate	Lingering Count
pathak-16-constrained-nospeed	100	10	1.25	62.50	1.01	16.19	0.02	3.26
pathak-16	4	10	1.25	62.50	1.66	26.63	0.56	9.69
octree-16	4	10	1.25	62.50	1.69	27.08	0.60	10.02
octree-4-pigeon	17	20	2.50	500.00	10.20	81.57	1.88	7.06
octree-16-constrained-pigeon-pathak	100	10	1.25	62.50	6.07	97.06	0.04	7.47
octree-4-pigeon-pathak	22	20	2.50	500.00	12.32	98.52	1.79	8.06
octree-8	43	20	2.50	500.00	22.90	183.19	1.60	6.88
octree-16-constrained	100	10	1.25	62.50	12.70	203.16	0.09	6.40
pathak-8	65	20	2.50	500.00	30.48	243.85	1.37	7.01
pathak-16-constrained-nospeed-nolinger	100	10	1.25	62.50	19.05	304.76	0.20	17.19
pathak-4	90	40	5.00	4000.00	79.74	318.97	6.30	3.56
octree-16-constrained-pigeon	100	10	1.25	62.50	20.35	325.61	0.22	5.76
blind-octree-explorer	91	40	5.00	4000.00	86.40	345.62	7.01	3.35
octree-4	92	40	5.00	4000.00	87.16	348.64	6.59	3.45
blind-octree-explorer-nospeed-pathak	91	40	5.00	4000.00	88.52	354.09	6.65	3.17
blind-octree-explorer-nospeed	95	40	5.00	4000.00	91.82	367.29	6.86	2.84

Table A.6: Overview of all environment-focused runs.

A.6.11. Mixed-Focus Agents

Method	Episode Length	Objects Scanned	F1-score	Octree Leaf Nodes Visited	Visited Volume (m^3)	Look Direction	Detections Total Count	Shannon Entropy
explorer-entropy-4-noTrainEntropy	98	0.00	0.00	2.24	358.06	0.66	13733.84	0.00
explorer-entropy-16	95	0.22	0.05	3.14	31.36	0.41	343075.19	0.16
explorer-entropy-16-noTrainEntropy	99	0.23	0.05	3.21	32.13	0.42	456362.79	0.14
explorer-entropy-4-nolinger-noTrainEntropy	97	0.17	0.08	1.76	281.55	0.30	62897.76	0.06
explorer-entropy-16-nolinger	100	0.78	0.14	7.46	74.64	0.43	90385.70	0.10
explorer-entropy-16-nolinger-noTrainEntropy	97	1.18	0.16	7.98	79.76	0.42	335268.83	0.13
explorer-entropy-4	97	0.54	0.21	1.94	310.53	0.64	194669.37	0.08
$ev_sac.explorer-entropy-8$	90	0.96	0.22	3.63	145.33	0.41	93265.69	0.11
explorer-entropy-8-nolinger	96	1.77	0.24	2.95	117.86	0.39	420789.33	0.14
explorer-entropy-8-nolinger-noTrainEntropy	98	1.83	0.25	3.02	120.73	0.41	378857.56	0.12
explorer-entropy-8-noTrainEntropy	92	1.65	0.27	3.49	139.49	0.42	458449.95	0.15
explorer-entropy-8	88	1.96	0.30	3.78	151.04	0.41	421204.65	0.14

Table A.7: Overview of all mixed-focused runs.

A.6.12. Small Environment Results (Table)

Method	Episode Length	Total Objects Scanned	F1-score	Visited Volume (m^3)	Look Error	Total Detections Count	Rationalized Class Entropy
blind-agent-explore-constrained	100	0.00	0.00	142.53	0.35	0.00	0.00
octree-4	100	0.00	0.00	285.08	0.64	0.00	0.00
shortest-path	100	0.00	0.00	12.64	0.37	0.00	0.00
semantic-entropy	100	0.00	0.00	11.93	0.35	655758.30	0.28
voxel-entropy-normalized	100	0.00	0.00	120.08	0.45	312259.79	0.10
object-detector	100	0.01	0.00	58.48	0.19	637355.56	0.64
semantic-curiosity	100	0.16	0.02	86.24	0.43	606310.02	0.30
voxel++050	100	1.49	0.11	96.49	0.28	0.00	0.00
voxel++075	100	2.62	0.16	106.60	0.32	0.00	0.00
voxel	100	3.03	0.20	142.31	0.20	0.00	0.00
voxel++025	100	4.76	0.23	122.10	0.18	0.00	0.00
object-detector-nospeed	100	8.71	0.28	113.87	0.18	1282254.18	0.53
voxel++100	100	9.42	0.30	121.67	0.17	0.00	0.00
voxel-entropy++050	100	9.19	0.30	125.76	0.16	1227545.28	0.52
voxel-entropy++100	100	9.04	0.31	129.43	0.17	1142712.42	0.49

Table A.8: Overview of all the small environment runs.

A.6.13. SMALL ENVIRONMENT RESULTS (PLOTS)

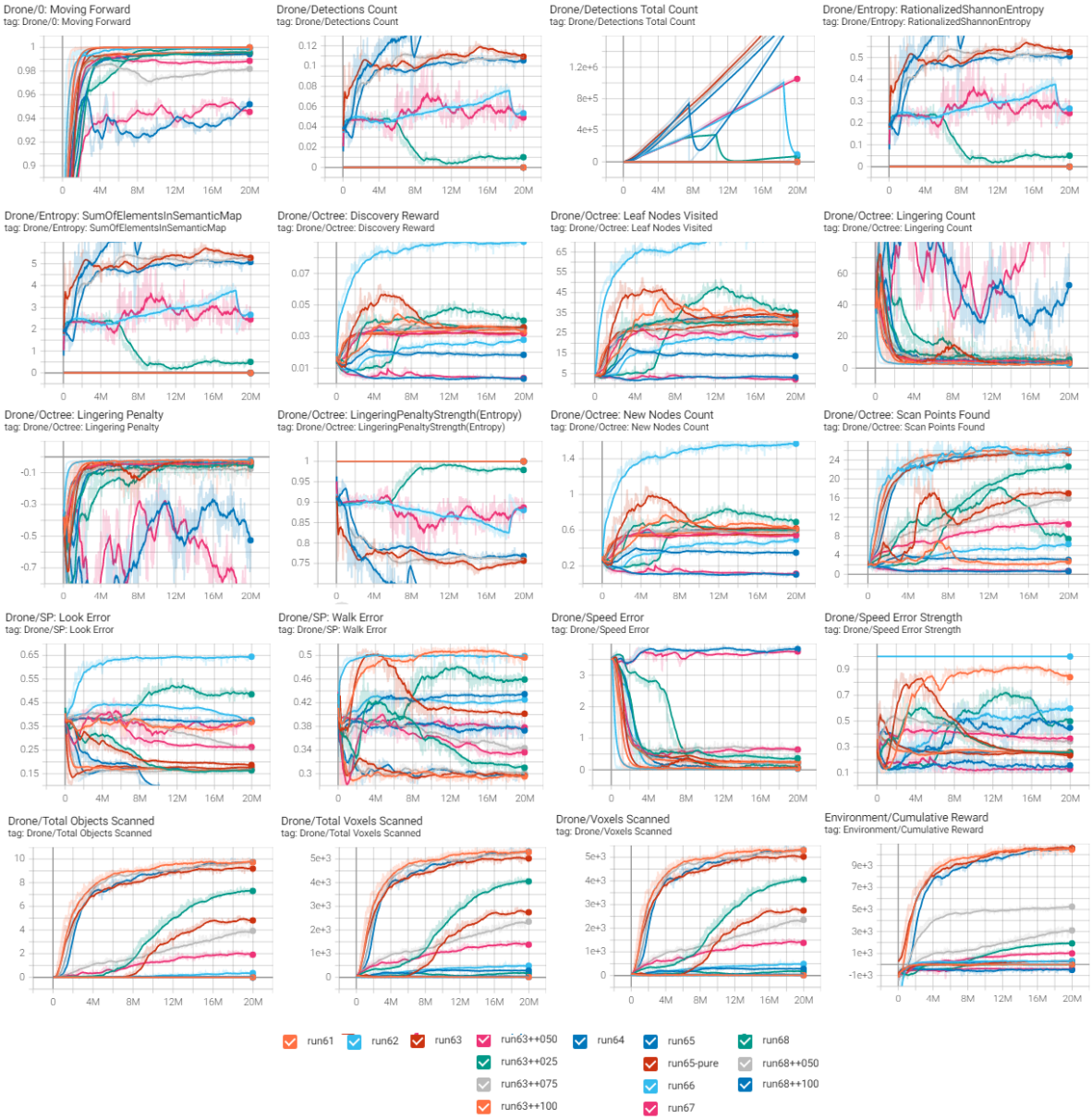


Figure A.13: Small environment (32 m^2) results.) .