Zurich University
of Applied Sciences

School of
Engineering
InIT Institute of Applied
Information Technology

MSE Master's Thesis

Exploiting Voxels and Octrees in Active Vision for Scene Understanding

Autor	Juan F. Ribera Laszkowski
Betreuung	Prof. Dr. Thilo Stadelmann
Nebenbetreuung	Dr. Giovanni Toffetti Carughi
Industriepartner	Sutter Landtechnik GmbH
Datum	30.10.2021

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

Computer vision has been a field of study since the 1960s, and over the past few years a lot of challenges, such as 2D object recognition are now considered solved problems. Nevertheless, default uncertainty in all environments is the basis of all vision problems, such as detection and recognition. The goal of this work is to analyze diverse exploration policies using reinforcement learning, and find an applicable solution for unknown 3D environments. This work will contribute to the computer vision challenges encountered by the project ”Melkroboter” project at the ZHAW.

Preface

I would like to give special thanks to Giovanni Toffeti for his continuous support during this Project work, for his great feedback and constructive discussions. I am very grateful to have been given the opportunity to explore and contribute to the ongoing research topic of computer vision and cloud robotics.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Purpose and Research Question . . . 2
1.3 Approach and Methodology . . . 2
1.4 Contributions . . . 2
1.5 Scope and Limitation . . . 3
1.6 Target Group . . . 3
1.7 Outline . . . 4

2 Foundations **5**
2.1 Related Work . . . 5
2.2 From Pixels to Voxels . . . 8
2.3 Octrees . . . 9
2.4 Scene Understanding . . . 11
2.5 3D Environments . . . 13
2.5.1 Game Engines and Simulation Environments . . . 14
2.5.2 Synthetic Data . . . 15
2.6 Reinforcement Learning . . . 16
2.6.1 Q-learning . . . 19
2.6.2 Function Approximation . . . 20
2.6.3 Proximal Policy Optimization . . . 22

3 Method **25**
3.1 Identification of the Exploration Goal . . . 26
3.2 Environment . . . 26
3.2.1 Environment Platforms . . . 27
3.2.2 Environment in Unity 3D . . . 29
3.2.3 Voxelization of the World . . . 30
3.2.4 Grid sensors and Vision from Monocular Cameras . . . 30
3.2.5 Navigation with Octrees . . . 31
3.2.6 Semantic Entropy . . . 32
3.3 Specification of the Exploration Approach . . . 32
3.3.1 Simulation of the Agent's Character . . . 33
3.3.2 Choice of Agent Observations . . . 35
3.3.3 Resetting the Environment . . . 36
3.3.4 Goal and Reward Signal . . . 37
3.3.5 Implementation . . . 37
3.4 Interpretation . . . 37
3.5 Further Use of Results . . . 39
3.5.1 Generalization to New Scenarios . . . 39
3.5.2 Cross-Platform Performance . . . 39

4 Results **40**
4.1 Environment Setup . . . 40
4.2 Pipeline Deployment . . . 41
4.3 Segmentation Network Training . . . 42
4.4 Results . . . 43
4.4.1 Research Question . . . 43
4.4.2 Deliverables . . . 44

5 Discussion **45**
5.1 Results Interpretation . . . 45
5.1.1 Research Question . . . 46
5.1.2 Generalization Scenarios . . . 46
5.1.3 Semantic Sub-modules . . . 46
5.2 Method Reflection . . . 46
5.3 Reliability . . . 46

6 Conclusion **48**
6.1 Conclusions . . . 48
6.2 Where to Go From Here? . . . 48

List of Figures **58**

List of Tables **60**

A Appendix **61**
A.1 Contact Information and Code Access . . . 61
A.2 Related Work: 3D Vision . . . 62
A.3 Computer Vision . . . 62
A.3.1 Deep Learning for Vision . . . 63
A.4 Unity 3D: Modelling the Environment . . . 67
A.5 Unity 3D: Voxelizing 3D Models . . . 67

1

Introduction

Figure 1.1: 3D Model of our Explorer Drone, from the Unity Asset Store [1].

1.1. Motivation

At the 2017 Conference on Neural Information Processing Systems, Pieter Abbeel gave a presentation of 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. Nowadays, tasks like teaching a robot how to pick a bottle of beer can be a very challenging task. 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 that are required to achieve not just perception in robots but intelligent agents [2, 3]. Moreover, DeepMind's recent publication [4] hints towards a promising future where reward maximization methods would be the only requirement to achieve 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.

Accordingly, a current project at the 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. The 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 the field of computer vision research and machine learning. Hence, the milking robot problem represents an unknown environment where not only the position of objects is of interest, but the general level of awareness of the overall environment itself.

1.2. Purpose and Research Question

In this work, the problem of uncertainty in an environment presented by the milking robot is looked from a higher view, in order to propose exploration policies that also contribute to a plethora of other use cases. More concretely, a machine learning and computer vision based solution will be used to maximize coverage in a 3D scene, using intrinsic motivation to explore new environments. Moreover, it will also take into account uncertainty in the environment, defined by temporal inconsistencies in an object detector as a supporting exploration metric. Finally, the environments developed to test the exploration policies contribute to existing benchmark environments and account for realism using Unity 3D. Regarding this task, this work tackles the following research questions:

- How can 3D scans be translated to voxel structures, which can then be further exploited for intrinsic motivation in exploration policies?
- How can uncertainty be defined through semantic entropy and be used as motivation in exploration policies?
- How can exploration policies be leveraged for a multitude of scenarios, including the milking robot problem?

1.3. Approach and Methodology

This work focuses on leveraging reinforcement learning techniques to manipulate an agent (camera) based on a set of ubiquitous environment observations (voxels). The problem that is studied within this master's thesis is the reduction of uncertainty in a 3D scene over time (maximization of exploration space policies), where uncertainty is defined by the semantic entropy observed.

The planned strategy for this project work is the following:

A state of the art analysis will be done to propose a set of exploration policies using reinforcement learning that answer the research questions. First, this analysis includes the evaluation of a variety of advancements in computer vision and machine learning fields such as 3D data representation, object detection, scene understanding, and reinforcement learning techniques for the manipulation of a vision-driven agent for exploration. Second, it includes an implementation of the proposed algorithms and the respective optimization of the agent's observations, the learning environment and the learning behavior's hyper-parameters 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 will trigger changes in the data sets (learning environment) and the diverse algorithms' parameters, in order to achieve the best possible results. The used data will be of synthetic origin. Finally, the results will be critically evaluated and compared to each other. The generalization capabilities, reliability and limits of the results will be discussed and further improvements on this field will be considered.

1.4. Contributions

To this end, 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 temporal

semantic entropy present in an object detector and the derived class complexity. The proposed pipeline is inspired by the human brain's physiology and approximates the discovery process a human would follow in an unknown environment. To this end, it leverages the concept of temporal semantic entropy proposed by [5] to define this dimension of uncertainty. Concretely, the task at hand is an exploration task for coverage maximization with a novel approach that leverages intrinsic rewards from voxel scans constructed from ubiquitous information. From the perspective of architecture design, it proposes a pipeline that can further leverage other semantic models, in an attempt to contribute to the research on semantic-curiosity-motivated exploration.

Finally, while there are 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 [6]. 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 evaluate the exploration task at hand and 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.

1.5. Scope and Limitation

Due to the time constraints, some limitations have to be laid out to ensure the work can be finished within schedule.

- The approach used will focus on the manipulation of synthetic data, namely, 3D voxels in the Unity Engine, while the manipulation of point clouds to generate these voxels will be left out. This allows the masters thesis to focus on the research on exploration policies 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, will not be taken into account.

- Methods mentioned in **??** 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 will focus on scene understanding and camera manipulation techniques using reinforcement learning in the Unity Engine.

1.6. Target Group

First, this work 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. Second, a broad audience in the industry is the public interest for this work, such as security and rescue offices, where dedicated scanning of objects, cues in a traffic accident, fires or other unexplored scenes is of special priority. This can also be extended, for example, to industries where scanning of objects for quality assurance is important. Third, as mentioned in 1.1, the industrial partner Sutter Landtechnik GmbH (SLG) is the private interest group for this research. This is because SLG not only offers sales, installation, and service of technical equipment and machinery for various tasks in agriculture, but also provides milking technology, including robots of the "Astronaut" variant [7].

1.7. Outline

The following Chapter 2 will cover the theoretical background for this work, making emphasis on computer vision, machine learning approaches, 3D object recognition, point cloud segmentation, point cloud registration, scene understanding, and reinforcement learning . Chapter 3 will describe the methodology, the nature of the data and the reinforcement learning agent's architecture. Chapter 4 will present the achieved results, and chapter 5 will examine 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. 2.5 Finally, section 2.6 introduces the basics of reinforcement learning, Q-learning and how function approximators allow agents to learn more sophisticated environments. It wraps up the theoretical background with proximal policy optimization. The following questions will be answered in this chapter:

- What are the origins of computer vision?
- What are voxels and octrees?
- What constitutes 3D environment simulators and what is the promise of synthetic data?
- What is reinforcement learning and proximal policy optimization?

2.1. Related Work

We study the problem of how to maximize exploration coverage in embodied contexts. This is related to milking robots (what are they currently capable of), 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). We survey related efforts 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. [8], 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 [9], RGB-D [10] and Thermal Imaging [11]. 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 [12] 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. [13] 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. [14] 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" [15, 16]. 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 [17, 18, 19, 20]. 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 [21, 22, 23, 24].

In the field of robotics, Lin et al. [25] 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. [26] 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 [27] 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. [28], 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, the pre-processing steps in the manipulation of image data remains at a high-information level that is difficult to manipulate and expensive to store in memory. This thesis' work proposes an efficient voxel-based approach that reduces the complexity in the environments and allows exploration and understanding of 3D environemnts, which to the best of our knowledge hasn't been exploited before for exploration tasks.

Navigation in Classical Robots. As described extensively by Chen and Zhang [29], 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 [30, 31, 32]. 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 [33] extensive literature: active SLAM has been formulated as a partially observable Markov decision process (POMDP) [34] or as choosing actions that reduce uncertainty in the environment's mapping [35]. 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 and Zhang [29] 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 [36, 37] to Chen and Zhang [29]'s learn policies on how to avoid collision and prefer open space than maximize environment coverage without leveraging from their learned maps, even mimicking SLAM techniques [38]. Similarly, many works study reinforcement-learning-based exploration [39, 40, 41, 42, 43, 44], proposing intrinsic reward functions that prefer novel states. The difference in this master's thesis work lies in the added 3D dimension and the realism of the environments. Furthermore this work proposes a variety of policy architectures using intrinsic rewards. Similar to Chen's work, this thesis studies how learning-based techniques can be exploited in real world deployments.

Other related efforts use [45] reward function based on pixel reconstruction to learn how to explore and how to solve tasks. Bai et al. [46] study Gaussian Process regression in an information-theoretic exploration method tested on simplistic environments. Kollar and Roy [47] 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 directs. 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 Perception. As described by Chaplot, active perception [48] "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 [49], amodal object detection [50], scene completion [45], and localization [51, 52]. Similar to Chaplot's work, this thesis' approach learns space exploration using a self-supervised policy and does not rely on end-task rewards, which were used in [51, 45, 50].

Intrinsic Rewards. This thesis' work is tightly related to exploration using reinforcement learning [53, 54, 39, 55]. The mentioned works set up the problem as a Markov Decision Process which attempts learn high reward paths, by motivating intrinsic rewards which direct the agents toward previously unseen [56] or unknown spaces [41] in the environment. This thesis' work not only motivates the agent to visit unknown spaces but also rewards it for "scanning" voxel structures in its 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 chaplot 2020, 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. [5], 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 [57, 58], or the vision-language [59] 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 [5]. They include: navigating to a fixed set of objects [60, 61, 57, 62, 63, 64], navigating to an object specified by language [65, 66] or by an image [67, 68], and navigating to a set of objects in order to answer a question [69, 70]. 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 on other challenges: models for FPS video games [60, 61, 62, 64] are able to train reactive agents to avoid obstacles and enemies, collective sporadic rewards scattered across the environment. Other models focus on recognizing the goal (visual perception) [67, 68], understanding the goal through language [65, 66] or understanding visual properties which make up the goal [69, 70]. Reinforcement-learning is successfully applied in these challenges but exhaustive exploration, specially in large environments, remains to be studied with more depth. This thesis' work goes in the direction of works that attempt to maximize the explored area [71, 29, 72], however the difference lies in the usage of realistic 3D scenarios in Unity and that the agent's reward function novelty is two fold: it leverages a 3D map to reward the agent based on novel spaces, and it exploits the voxels found in the environment to scan goals in more depth.

2.2. From Pixels to Voxels

Images are the ubiquitous source of information for visual tasks. As such, 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.

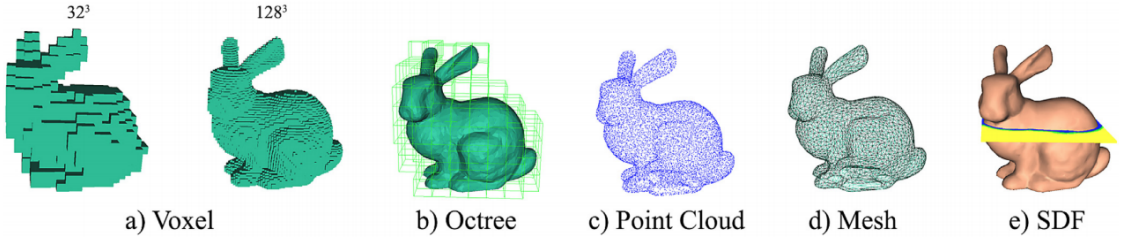

Figure 2.1: The Stanford Bunny in different 3D representations [73] .

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 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 [9]. 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 how to recover 3D information from a scene has been extensively studied [73] and 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 [74]. RGB-D image representations are therefore known as "2.5D" data [75], 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.

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 [75], 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 X 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, by down-sampling 3D scenes, one can leverage sparse voxel octrees (SVO). This work exploits SVOs to propose using low-resolution voxels for the exploration of 3D environments. The motivation for this is two-fold: first, as described by [77], in the SVO, empty space is not explicitly stored, which leads to a huge reduction of memory requirements for most scenes. Second, given voxels that do not serve any rendering purpose, they can be used as intrinsic rewards to motivate the an exploratory trajectory in a 3D environment. For further discussion on different representations of 3D data, one can refer to these outstanding surveys [75, 78].

Given an RGBD image, the process of generating a voxel grid from a triangle mesh is called voxelization [80]. A voxel contains per default a "0". 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 [81, 82] occupy themselves with point clouds from RGBD data, point cloud registration and the problem

Figure 2.2: Voxelized representation of Minas Tirith in Minecraft [76].
A multitude of voxels can also represent high-resolution data, at a high computational memory cost (image below).

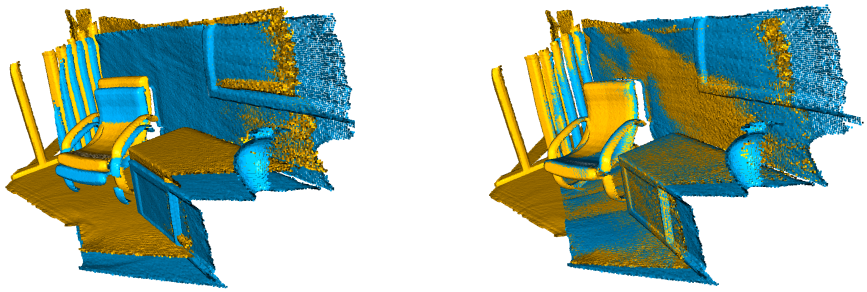

Figure 2.3: Two input point clouds (left). Result after running the ICP Algorithm until convergence (right) [79].

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 [79] for matching two point clouds together. Ultimately, a point cloud can then be voxelized or down-sampled to voxels for further processing [83].

2.3. Octrees

The octree encoding uses a hierarchical tree data structure, where analog to quadtrees, each node can have up to eight children [85]. 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

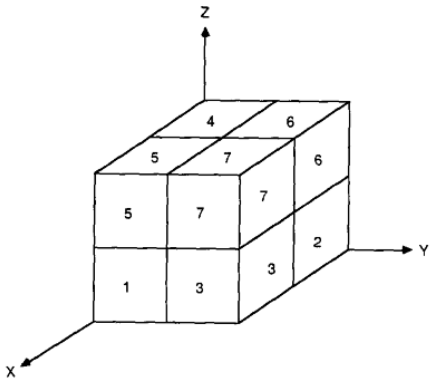

Figure 2.4: Labels of the octants of a cube (octant 0 is not shown). [84].

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.

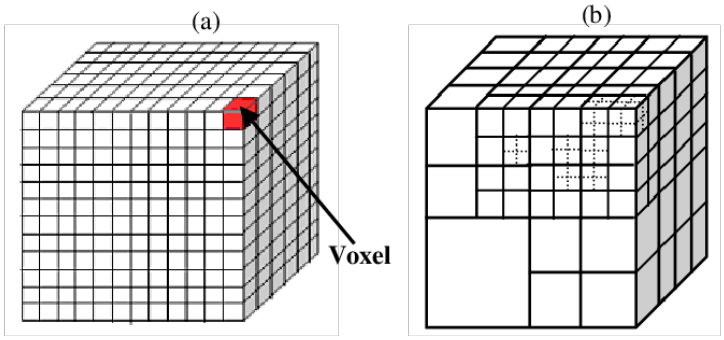

Figure 2.5: (a) Voxel grid (b) Octree representation [86].

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 complexity of using octrees and insertion and search implementations, please refer to [87, 88].

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 [89]. Second, in situations where meshes 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 [90]. Consequently, the efficient representation of high-resolution 3D models using octrees in comparison to polygon-based methods has been studied in a multitude of works such as fast collision

detection in models of over 25M polygons [91], solar radiation [92], voxelized full-waveform airborne LIDAR data [93], volume rendering methods [94], adaptive resolution SLAM [95], 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. [96] 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 [97] deep learning based methods for image classification have reached state of the art performance for 2D detection.

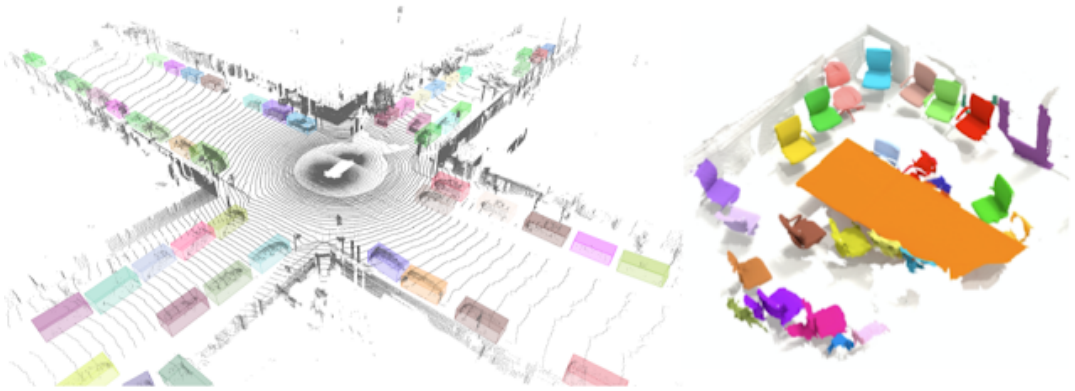

Figure 2.6: 3D object detection (left) and 3D instance segmentation (right) using Tensorflow 3D as found in [98].

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 [98], Pytorch3D [99], Torch-Points3D [100], Pytorch Geometric [101], 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 [98]. Tensorflow, for example, uses the 3D submanifold sparse U-net architecture to extract both coarse and fine features from voxels. This network, as shown in Figure 2.7, 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 than previous implementations [98].

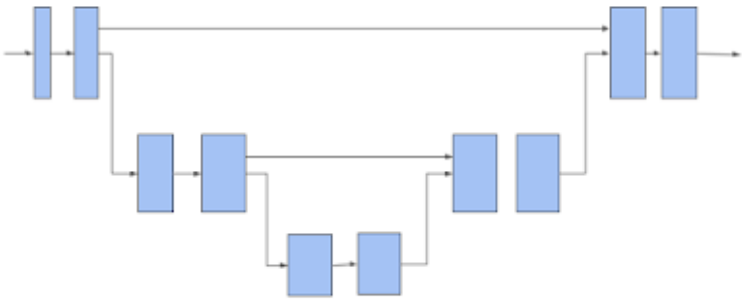

Figure 2.7: A 3D sparse voxel U-Net architecture using submanifold sparse convolutions (horizontal arrows) and submanifold sparse pooling (arrows downward) operations [**tf**].

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 [98, 99]. A great effort has gone particularly to:

- Semantic segmentation, where models predict per-voxel semantic scores using one output head, which are then mapped and used to label 3D points [98, 102, 22, 23].
- 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 [98, 103].
- 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 [98, 104].

Limitations of 3D Scene Understanding. As documented by Surendran and Hemanth [105], 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 [106]. Research [102] 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 [96] 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. [102] from ETH aids to 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 [107]. 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 [108]. 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

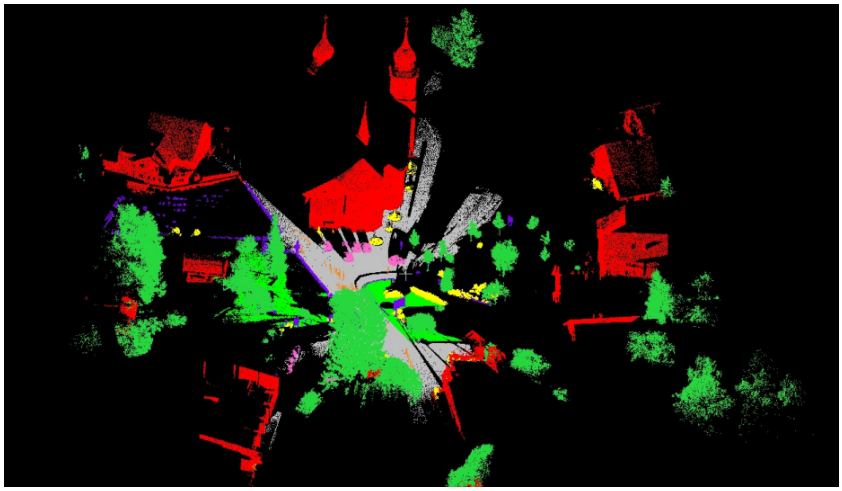

Figure 2.8: Class labels example of Semantic3D [102].

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 2.9: 3D hyper-realistic simulated desert using Unreal Engine 5 (early access) [109].

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 [110], 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 [111], 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 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. [112] 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 [1], Unreal Engine [109], MuJoCo [113], Gazebo Simulator [114] and CARLA [115]. Collins et al. [112] 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.

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

(a) (b)

Figure 2.10: Comparison of the capabilities between two simulators of creating realistic 3D environments: (a) CARLA [115], (b) Unreal Engine [116].

built on top of Unity or UE such as Airsim or CARLA were evaluated. Juliani et al. [6] 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 2.10, 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 [112, 6].

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 [117], making it important to exploit techniques that allow us to create quality datasets. In general, a data set with a small 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 motivation for synthetic data is the false belief 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[118], Gartner predicted by 2030 most of the data used in AI will be artificially 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 [109] 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

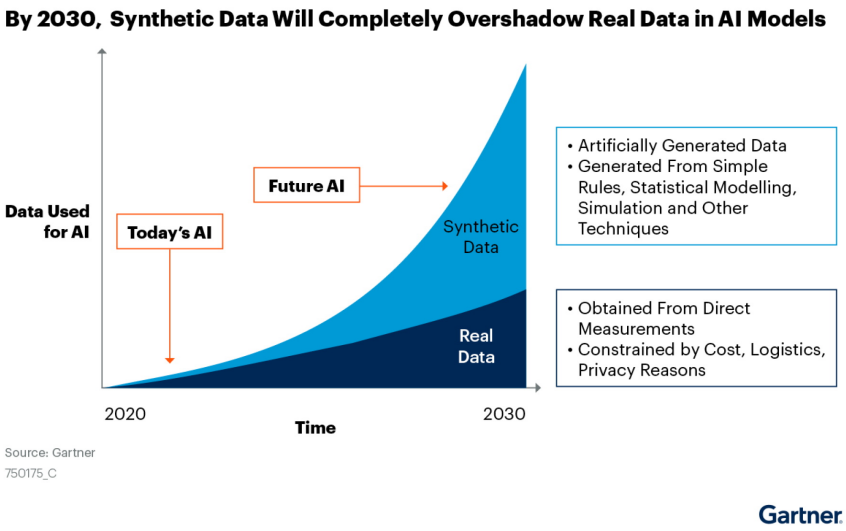

Figure 2.11: Synthetic data will become the main form of data used in AI [118].

systems, and uses artificial intelligence to reduce the technical complexity in 3D design and modelling [119]. 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.

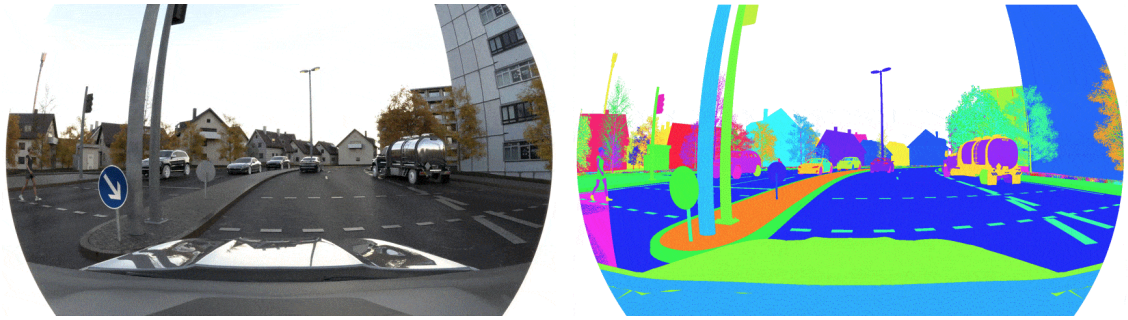

Figure 2.12: Developers can alter and randomize objects, colors, lighting, materials and poses in realistic 3D environments to quickly generate synthetic data with perfect labels [117].

There are many techniques to generate synthetic data, such as variational autoencoder-decoders algorithms, GANs and simulations [117]. The factors mentioned in section 5.1.3 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 2.12 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.

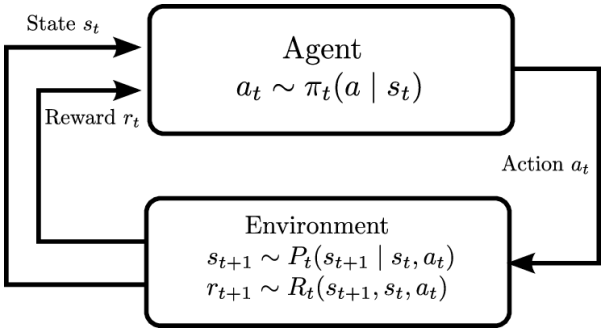

Figure 2.13: Agent-environment relationship in an MDP according to **\cite {sutton1998introduction}**: at each time step t, the agent observes the environment's current state s_t and a reward signal r_t. The agent then selects an action a_t given s_t and following policy π_t. This action changes the 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 [55].

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 [55]. 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 2.13.

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 [55].

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 [55].

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 [55].

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 chose 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 [55].

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 in the to calculate a global reward which is to be maximized [55]. 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:

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

where T is the final time step. A return can also include a discount rate γ that discourages future rewards:

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

A discount rate is used, for example, if the MDP is a continuous problem that never ends or to include uncertainty of future predictions [55].

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. [55] 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

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

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 [120].

Figure 2.14 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 negative reward. Hence, the state-value function for the cells around the pit is lower to represent that these are unwanted states.

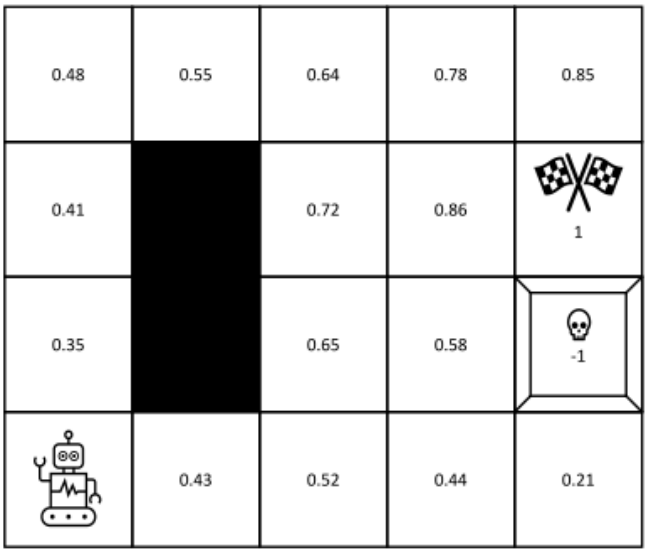

Figure 2.14: A grid-world environment [120]. Each cell represents a state with corresponding a value, which is the value of the state-value function $V(s)$. The agent gets a reward when it reaches the goal with two flags, and given a penalty when it falls down the pit with a skull.

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 [55]. 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 [55].

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 [55]. 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 [55].

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 [55]. It can be written as:

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

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:

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

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 [55]. 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 □;
    perform action A and observe R,S';
    $Q_\pi(S, A) \leftarrow Q_\pi(S, A) + \alpha \left[R + \gamma \max_a Q_\pi(S', a) - Q_\pi(S, A) \right]$
    $S \leftarrow S'$
    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 [55]. Additionally, it is one of the algorithms available in the open source deep reinforcement learning OpenAI Gym [121].

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 [55].

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 [120].

A neuron consists of an input (or data) connection, and an output (or hidden) connection, as visualized in the model in Figure 2.15. 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

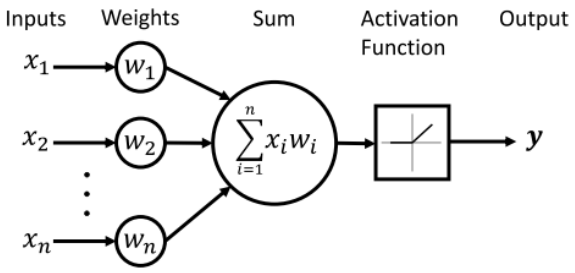

Figure 2.15: Neuron. A single neuron consists of an input (or data) connection and an output (or hidden) connection. The input x is multiplied by the training weights θ and summed. An activation function is then applied to the sum to obtain a nonlinear function $h(x, \theta)$ of the inputs.

approximation (i.e. the training set error). A common activation function, ReLU is defined as follows:

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

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 [120]. A common and efficient method to tune the weights of a neural network is batch gradient descent. Gradient descent can be defined as

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

where γ 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 [55].

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 [55].

Deep Neural Networks

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 2.16. 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, spatial information can be lost during this process, which can be necessary to understand the relationship between the elements in a scene. Convolutional neural networks have been shown to be a very effective solution to this problem. These networks process images as input and use a kernel of weights, which are also called the "filter" in convolutional neural networks, to "convolute" the image. A filter is applied over the entire input, and the filter outputs a new set of values. This process is repeated over all the different parts of the input [120]. As of today, convolutional neural networks have proven to work well in many different domains, such as image classification, object detection, and semantic segmentation.

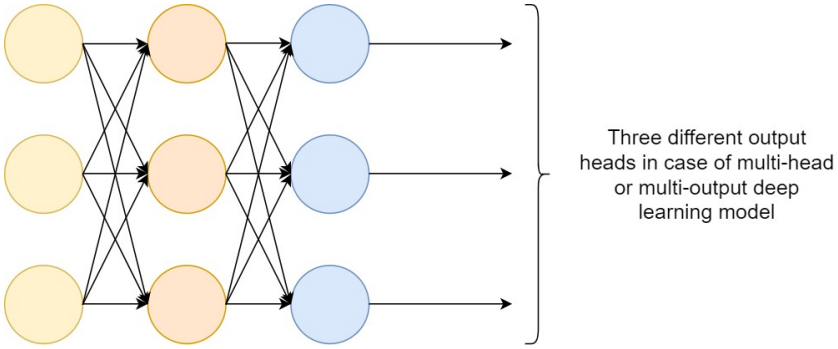

Figure 2.16: Deep Neural Network with an input layer, a hidden layer and an multiple output heads. In actor-critic reinforcement learning problems and shared parameter networks, one head can be used as the prediction from the critic and another head as the prediction from the actor [122, 55].

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. PPO overcomes a lot of challenges that other methods struggle with such as sample efficiency, overall performance, and ease of implementation and tuning capabilities [**16**, **17**]. 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.

This section introduces 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

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

These methods aim to maximize the expected return $J(\theta)$, known also as the objective function or loss, by following the parametrized policy [55]. Gradient descent (or ascent) is then used to update the weights of the network as

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

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

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

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 [55]. The respective objective function for the policy gradient estimator is therefore

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

Optimizing on L^{PG} is, however, not justified, given that it leads to destructively large policy updates [123].

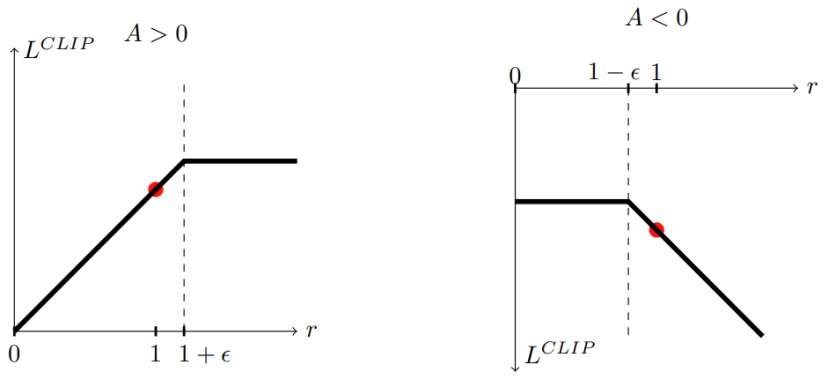

Figure 2.17: Plots showing one term (i.e., a single timestep) of the surrogate function L CLIP as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., r = 1. Note that L CLIP sums many of these terms. [123].

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

$$r(\theta) = \frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)}.$$

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

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

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 [123]. The value function loss then becomes

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

where the function $\text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)$ constraints the ratio to a maximum of 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 [123]. The advantage estimator is defined by

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

where

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

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 [123]. 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

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

where $c1$,$c2$ are coefficients, S denotes the entropy bonus and L_t^{VF} is the value function loss [123]. 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

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

where $V_\theta\left(s_t\right)$ is also known as the critic return and in some implementations is seen as $\delta_t + V_\theta\left(s_t\right)$. Similarly, the term $S_{\pi_\theta}\left(s_t\right)$ is an entropy bonus that promotes exploration [123]. Concretely, this is the Shannon entropy and is computed by

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

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 [123]. 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 $L^{\mathrm{CLIP+VF+S}}$ with respect to θ;
    update θ;
  end
end

3

Method

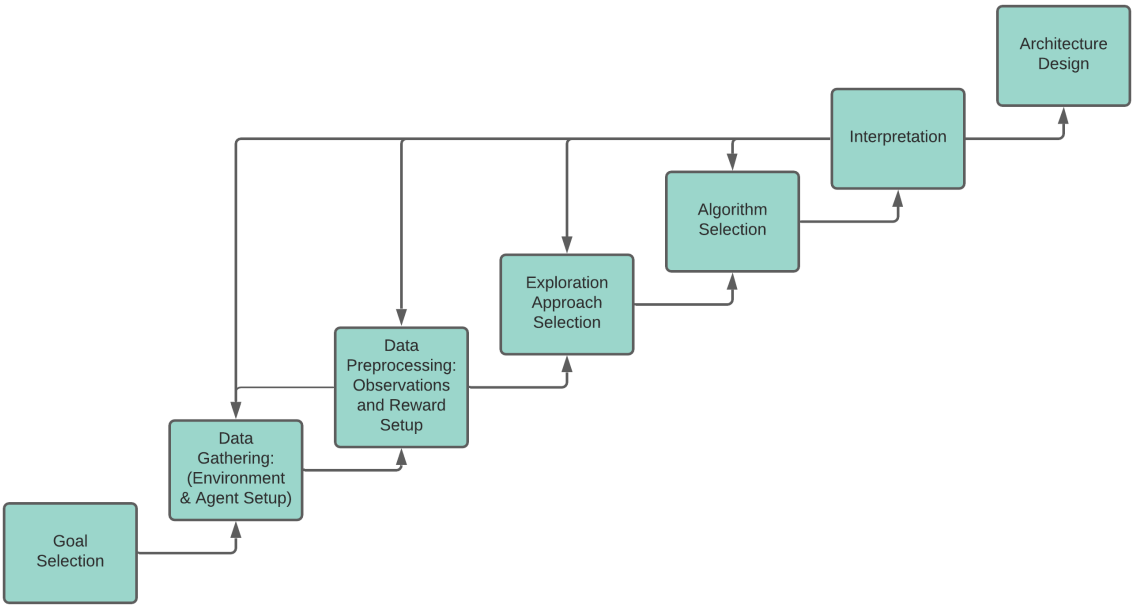

Figure 3.1: Adapted development process from "*Using machine learning methods for evaluating the quality of technical documents*", by Luckert and Schaefer-Kehnert [124]. Describes the taken steps from data collection, through algorithm selection to architecture design.

In this chapter the different steps taken to create a 3D object detector are addressed. This chapter adapts the structure proposed for machine learning algorithms proposed by Luckert and Schaefer-Kehnert [124]. The following questions are answered over the course of this chapter:

- What is essential for modeling 3D environments for exploration?
- How are voxels generated from RGBD data?
- What is Unity ML Agents for reinforcement learning?
- What are the current method's goals and limits?
- What is the method's pipeline composed of?
- What is required to apply the method?

In order to answer these questions, the structure of this chapter is illustrated as follows: Section 3.1 defines the exploration task and limits its scope. Then, the choice of a suitable 3D environment for exploration where the agent collects observations from (i.e., the data set) and the construction of the 3D environment is presented. Section 3.2 presents the 3D assets, components and variables that compose the environment and the agent. Afterwards, it explains the process of voxelization and the creation of the octree map. Finally, it outlines the sensors used in the agent for visual input and the usage of semantic entropy to define uncertainty in an environment. The choice of agent observations is defined in Section 3.3.2 and the description of the considered algorithms and techniques is covered in Section 3.3. Subsequently, the interpretation of the performance of the algorithms is described in Section 3.4. This section also tackles how the method results were evaluated, how performance was measured and which parameters were adjusted during training. Finally, Section **??** introduces the setup for the experiments to evaluate and compare learned behaviors.

3.1. Identification of the Exploration Goal

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 3D scans be translated to voxel structures, which can then be further exploited for intrinsic motivation in exploration policies?
- How can uncertainty be defined through semantic entropy and be used as motivation in exploration policies?
- How can these exploration policies be leveraged for a multitude of scenarios, including the milking robot problem?

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

1. Model and present realistic 3D scenarios and simulate the agent, enabling exploration.
2. Define the baseline exploration task in such environments.
3. Conceive visual cues that motivate in-depth exploration of objects.
4. Define visual uncertainty in an environment.

This strategy was determined because the steps for the modeling of the 3D environments is independent of the 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 of the 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 reinforcement learning algorithms will then be built on the outcome of the first step. Therefore, the definition of the environment and the agent's capabilities focuses on the first working step. The remaining steps are solved by using the modeled environments and the agent character. More concretely, the voxelization of 3D meshes as an efficient and thorough representation of goals for the agent concentrates on the second working step while the construction of an octree with the agent's trajectory and available scan radius is tackled on the third working step. Finally, the concept of uninformativeness in an environment, inspired by research in semantic curiosity [5] is addressed on the last working step.

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. 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 using Unity 3D for this work?
- How was the environment modelled in Unity 3D?
- How are voxels generated?
- How are the nodes in the octree constructed?
- How is visual uncertainty in an environment defined through semantic entropy?
- How is semantic entropy measured?

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, which includes the objects that contain it. Therefore, the agent requires: 1) motivation to navigate and explore the environment, 2) motivation to investigate objects exhaustively, and 3) a way to determine when there is uncertainty or uninformativeness while it is exploring. This last requirement is defined through semantic entropy in subsection 3.2.6. The following sections presents the 3D scene and the steps required to construct the environment states for the agent.

3.2.1. 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 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. The platform 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. The platform abstracts the environment through the **env** interface, which 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 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, but also enables the development of complex multi-agent applications such as autonomous navigation, curriculum learning, imitation learning, etc. 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 5.1.3:

- **Learning Environment** - which contains the Unity scene and all the game characters. The Unity scene provides the environment in which agents observe, act, and learn. How you set up the Unity scene to serve as a learning environment really depends on your goal. You may be trying to solve a specific reinforcement learning problem of limited scope, in which case you can

use the same scene for both training and for testing trained agents. Or, you may be training agents to operate in a complex game or simulation. In this case, it might be more efficient and practical to create a purpose-built training scene. The ML-Agents Toolkit includes an ML-Agents Unity SDK (com.unity.ml-agents package) that enables you to transform any Unity scene into a learning environment by defining the agents and their behaviors.

- **Python Low-Level API** - which contains a low-level Python interface for interacting and manipulating a learning environment. Note that, unlike the Learning Environment, the Python API is not part of Unity, but lives outside and communicates with Unity through the Communicator. This API is contained in a dedicated mlagents_envs Python package and is used by the Python training process to communicate with and control the Academy during training. However, it can be used for other purposes as well. For example, you could use the API to use Unity as the simulation engine for your own machine learning algorithms. See Python API for more information.
- **External Communicator** - which connects the Learning Environment with the Python Low-Level API. It lives within the Learning Environment.
- **Python Trainers** which contains all the machine learning algorithms that enable training agents. The algorithms are implemented in Python and are part of their own mlagents Python package. The package exposes a single command-line utility mlagents-learn that supports all the training methods and options outlined in this document. The Python Trainers interface solely with the Python Low-Level API.
- **Gym Wrapper** (not pictured). A common way in which machine learning researchers interact with simulation environments is via a wrapper provided by OpenAI called gym. We provide a gym wrapper in a dedicated gym-unity Python package and instructions for using it with existing machine learning algorithms which utilize gym.

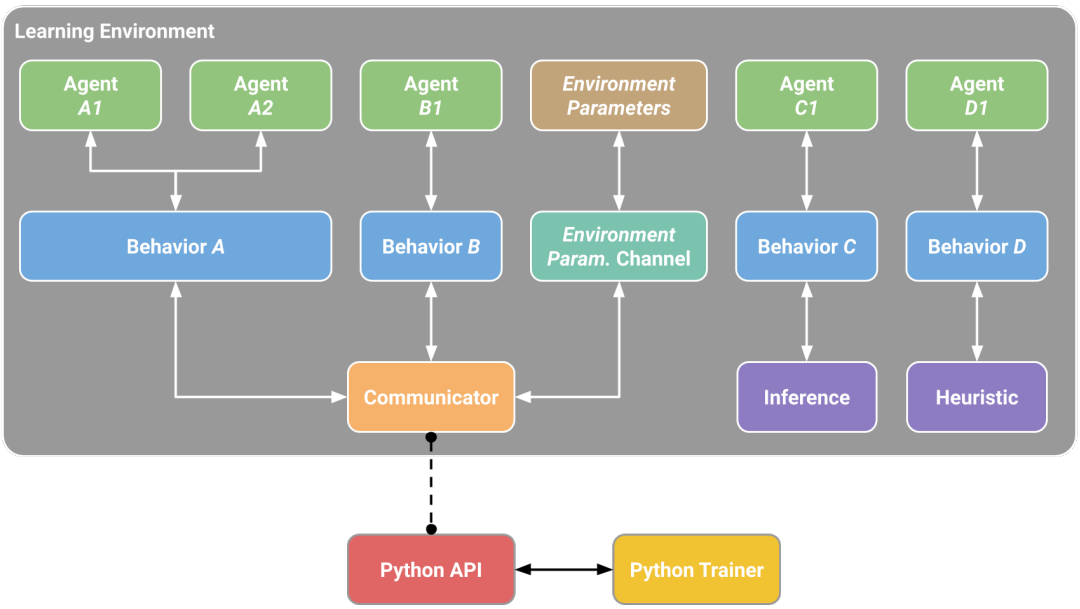

Figure 3.2: Overview of the Unity ML-Agents Toolkit, taken from [**unity_mlagents_github**].

Accordingly, the learning environment The Learning Environment contains two Unity Components that help organize the Unity scene:

- **Agents** - which is attached to a Unity GameObject (any character within a scene) and handles generating its observations, performing the actions it receives and assigning a reward (positive / negative) when appropriate. Each Agent is linked to a Behavior.
- **Behavior** - defines specific attributes of the agent such as the number of actions that agent can

take. Each Behavior is uniquely identified by a Behavior Name field. A Behavior can be thought as a function that receives observations and rewards from the Agent and returns actions. A Behavior can be of one of three types: Learning, Heuristic or Inference. A Learning Behavior is one that is not, yet, defined but about to be trained. A Heuristic Behavior is one that is defined by a hard-coded set of rules implemented in code. An Inference Behavior is one that includes a trained Neural Network file. In essence, after a Learning Behavior is trained, it becomes an Inference Behavior.

Every Learning Environment will always have one Agent for every character in the scene. While each Agent must be linked to a Behavior, it is possible for Agents that have similar observations and actions to have the same Behavior. In our sample game, we have two teams each with their own medic. Thus we will have two Agents in our Learning Environment, one for each medic, but both of these medics can have the same Behavior. This does not mean that at each instance they will have identical observation and action values.

Moreover, Note that in a single environment, there can be multiple Agents and multiple Behaviors at the same time. For example, if we expanded our game to include tank driver NPCs, then the Agent attached to those characters cannot share its Behavior with the Agent linked to the medics (medics and drivers have different actions). The Learning Environment through the Academy (not represented in the diagram) ensures that all the Agents are in sync in addition to controlling environment-wide settings.

Lastly, it is possible to exchange data between Unity and Python outside of the machine learning loop through Side Channels. One example of using Side Channels is to exchange data with Python about Environment Parameters. The following diagram illustrates the above.

3.2.2. Environment in Unity 3D

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* [125, 6]. For more information on the 3D modelling using Unity 3D, please refer to Appendix A.4.

Figure 3.3: 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.

In the setup of the scene, a variety of environment assets such as buildings, trees, rocks and foliage, were required to convey an authentic depiction of a scene. Sample screenshots of these assets, as well as of visual effects and skybox assets are given in Figure A.7. Once all the assets were allocated in a scene, the rendering pipeline could be tuned to adjust the lighting settings to achieve a more realistic rendering. Figure 3.3 shows the difference lighting makes when simulating a 3D scene, which is one of many parameters, such as colors and textures, rotations and positions, 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.

3.2.3. Voxelization of the World

Part of the exploration technique involved 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 [**everyone**], 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 a voxelized approach, which is to model the environment into cubes instead of points, as introduced in Section 2.2. This also gives more control over the shape and number of cubes in the scene, which allows the reduction of the complexity of the scene through the resolution settings. Secondly, a voxelized environment is useful for many other things besides vision, such as navigation and path planning [].

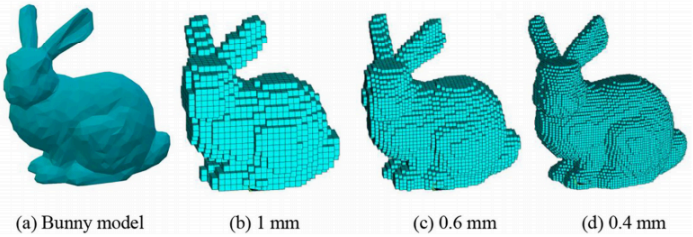

Figure 3.4: Bunny model voxelized at different resolution settings [].

Work on voxelized representations of the world have a number of advantages that this thesis work takes advantage of []. 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 is the first type of reward for our agent. There are many methods for efficient voxelization of an environment, as seen in []. 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.

3.2.4. Grid sensors and Vision from Monocular Cameras

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. It is 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 [126, 127, 128]. Among monocular cameras, the common camera is less preferred in comparison to other, more expensive, alternatives, since it is limited to an angle of view of 60 degrees and a vertical angle of view of 45 degrees. This angle constraints the amount of information fed into algorithms and adds stress to algorithms given unnecessary overhead in the feature extraction. More concretely, extracted feature points to stay in the field of vision for a short period of time, which translates to a different set of challenges and assumptions a technique must take, such as memory, tracking, etc. Accordingly, research by Davison et al. [129] 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 [130, 127].

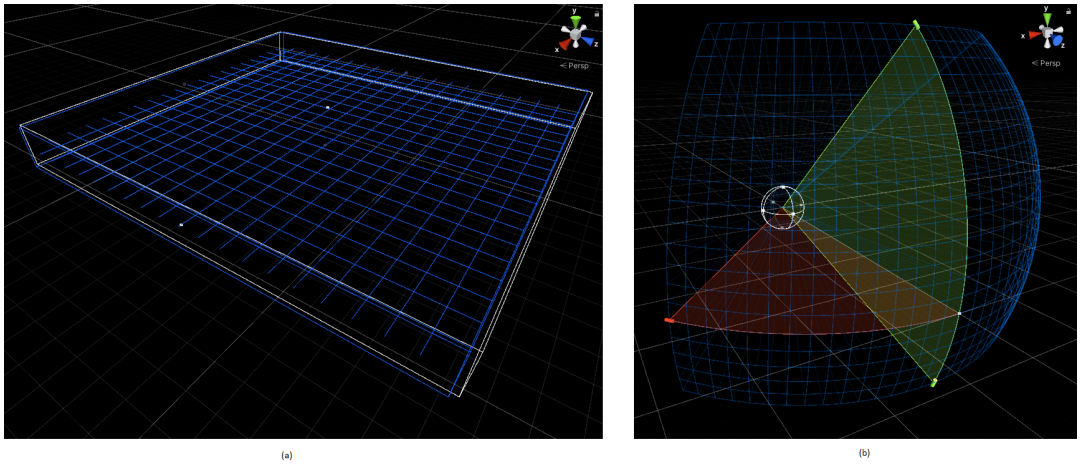

Figure 3.5: grid sensor 2D (a) and grid sensor 3D (b) by Mbaske [].

This thesis takes inspiration from these works and the benefits provided by spherical vision, to integrate the agent with a grid sensor component. The motivation behind using a grid sensor roots back to the origins of this 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 []. In 2020, the labs at Eidos-Montréal [] 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 [], 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.

The grid sensor used in this thesis is an improved version of the original grid sensor, developed by Mbaske [], under a variety of perception parameters. We then compare the performance of the agent under an ordinary monocular camera setting 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 training times in 3D environments. Since it does not depend on the rendering of visuals, training can be set to headless mode while still using high-dimensional data and the 3D creative process involving textures and lighting is completely detached from the agent's behavior. Our aim is that this also motivates further research in reinforcement learning to enable transfer learning of agent behaviors. The grid sensor by Mbaske can be found at `https://github.com/mbaske/grid-sensor`, and is visualized in Figure 3.6.

3.2.5. Navigation with Octrees

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 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 3.3.2. The reward R_o is part of the reward signal the agent receives, and it is proportional to the amount of new nodes added to the octree. The reward is further defined below in Section 3.3.4. The implementation of octrees for Unity was taken from Mbaske's explorer drone, which can be found here: `https://github.com/mbaske/ml-explorer-drone/blob/master/UnityEnv/Assets/Drone/Scripts/Octree.cs`.

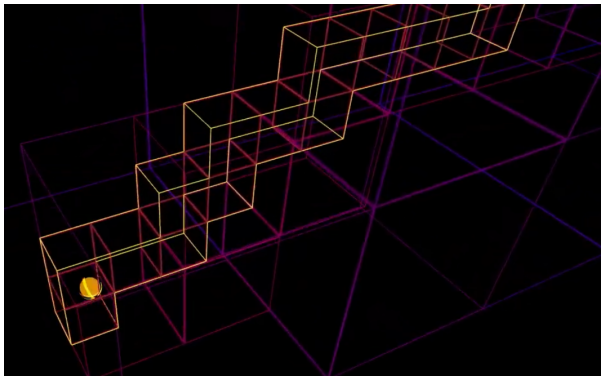

Figure 3.6: Octree Construction and Navigation in Mbaske's Explorer Drone [].

3.2.6. Semantic Entropy

Semantic entropy was defined by [**melamed**] as a way to measure semantic ambiguity, uninformativeness or as the inverse of semantic reliability. We are specially 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

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

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 and Zhang [29] and Chaplot et al. [5], 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. [5] 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 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 3.3.4 below.

3.3. Specification of the Exploration 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 on the algorithm?
- What were the main problems found during this phase?

Among all the 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 [6,8,16], its simplicity compared to other implementations [123], and general robustness. A robust algorithm shows stable performance without high dependency on hyperparameter tuning procedures or hyperparameter searches [16].

The neural network used is an implementation of the original by Schulman et al. [123] taken from [**unityML-Agents##PPO**], 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 [10-12] 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 [123].

The workflow for each trainer is presented in Figure 3.9, using Unity ML-Agents [3] as the target platform. The process was performed on a Windows 11 machine with an Intel i7-4790 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 at al., which contains three convolutional layers
- the IMPALA ResNet, which has three stacked layers, each with two residual blocks
- the ”match3” CNN by Gudmundsoon et al.
- 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 to construct an agent for a variety of practical reinforcement learning scenarios. Figure 3.8 presents an example of the visual encoders mentioned. For more information on the visual encoders, please refer to Appendix **??**.

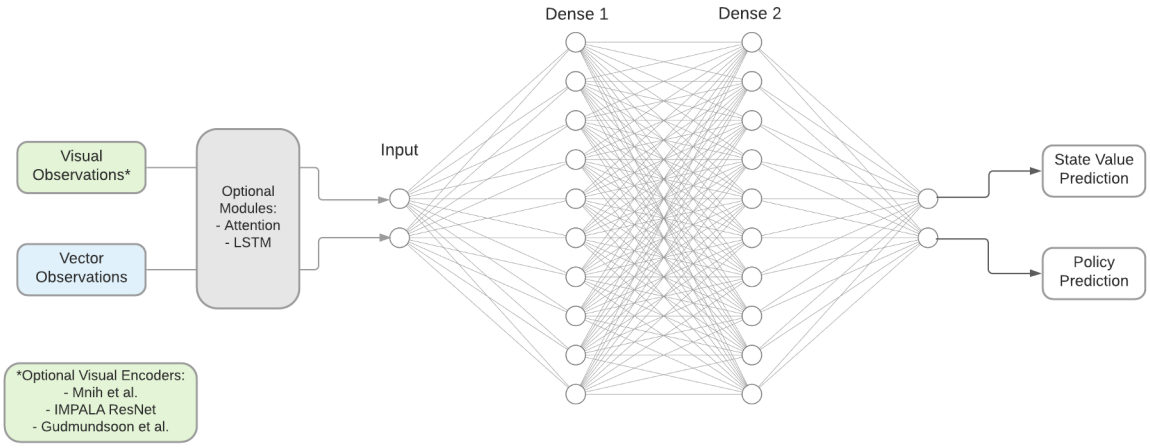

Figure 3.7: Overview of the PPO Trainer’s workflow from Unity ML-Agents [**unityml-agents**].

3.3.1. Simulation of the Agent’s Character

A set of 3D models for the agent were obtained from the Unity Asset Store [131] and Sketchfab [132] as potential avatars for the learning agent in different scenarios. Figure 3.9 visualizes some models considered for the agent. The main 3D model chosen was the Drone Bot model given the overhead complexity of animating the rigs of the other 3D models. To make the simulation more realistic, the drone was given animations, volumetric light and particle systems.

The drone counts with a scanner component which allows him to interact with the environment. More concretely, it allows him to scan voxels perceived from the environment given a set of constraints such as minimum angle and distance. The scanner component is make visible to the viewer through a *wireframe shader* as displayed in Figure 3.9. This shader, with the support of an invisible cone *collider* in

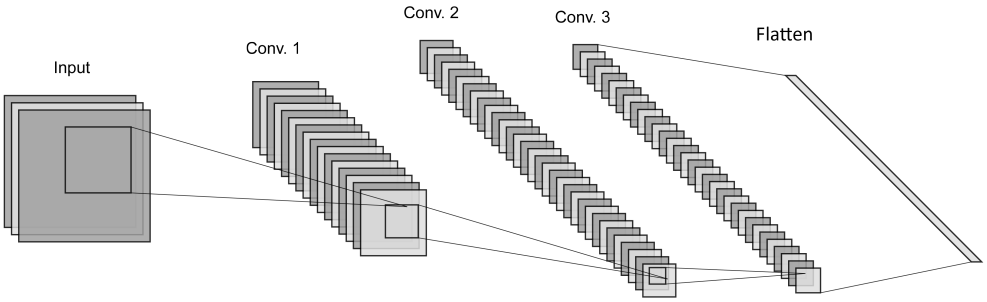

Figure 3.8: Visual encoder example, architecture by Mnih et al [**unityml-agents**].

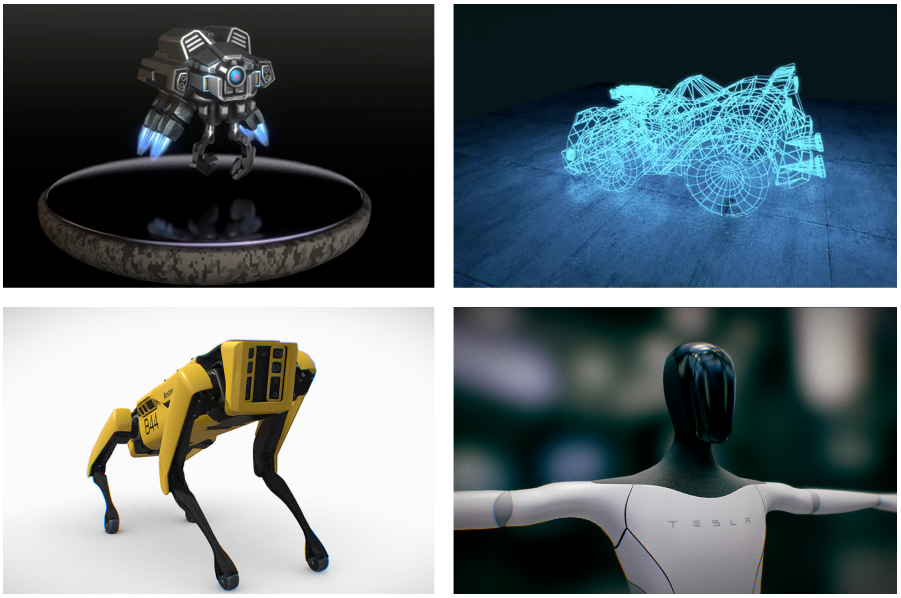

Figure 3.9: 3D Assets used to model the learning agent [**unityAssetStore**]. Drone Bot (top left), Wireframe Shader (top right), Spot Bot (bottom left), Tesla Bo (bottom right).

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.

Finally, 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.
- Physics-based movement.

The first, 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 as:

$$v_t = v_t + i_t * \eta_1$$

and

$$\tau_t = \tau_t + \mu * clip(turn_t * \beta, -1, 1) * \eta_2$$

where v_t is the agent's rigidbody velocity at timestep t, which receives a velocity change force based on input i_t at timestep t. This input consists of a *forward input*, a *vertical input*, and a *lateral input*. This force is constrained by a movement speed parameter η_1, measured in meters/second. Similarly, τ_t is the rigidbody'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 constrained by a turning speed parameter η_2.

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

$$x_t = translate(i_{xt}, i_{yt}, i_{zt}) * \eta_1$$

and

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

where x_t is the agent's position in the 3D space at timestep t, which 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, r_t is the transform's rotation of the agent at timestep t which is rotated 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.

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 component.
- A character controller script that captures and reacts to input, according to the mobility algorithm chosen.
- An agent controller script, which is required to use the Unity *ML-Agents* plugin.

The agent is further defined by a set of components, observations and parameters that enable perception of both the environment and itself. The components included in the agent are a 3D and a 2D grid sensor, enabling a visual input of the environment. Then, the metrics that the agent can perceive include its position, velocity, direction towards goals, etc., which register 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. A full list of the variables that define the agent's observations can be found below in Section 3.3.2.

3.3.2. Choice of Agent Observations

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 chosen observations for the agent briefly described:

- **Own Information**
  - The agent's position: $\mathbf{x} \in \mathbb{R}^3$.
  - The agent's rotation: $\mathbf{r} \in SO(3)$ where $\mathbf{r}$ is represented through a quaternion.
  - The agent's velocity: $\mathbf{v} \in \mathbb{R}^3$ based on the physics rigidbody metrics.
- **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 **??**. It is visualized in Figure 3.6.
  - A 2D Grid Sensor, which provides the agent with a top-down view map of his surroundings. It is visualized by Figure **??**.
- **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 any 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 constraints how far ahead can the agent scan voxels.
  - 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 realted 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: *steuernmodus*.

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 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 completeness of the information perceived and in drastic reductions of computation, memory and training costs, respectively. As mentioned in Section 3.2.4, 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 the type of exploration desired. These differences in behavior are further discussed in Section 5.

3.3.3. Resetting the Environment

In our setup, an episode can reach its end when one of two conditions are met: a) the agent scanned all voxels in the scene or b) the maximum steps were reached, and the agent ran out of time. 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.

3.3.4. Goal and Reward Signal

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

$$\max R = R_v + R_o + L(v, e),$$

where R_v is the reward given by the voxel scans, R_o is the reward from adding new nodes to the octree. $L(v, e)$ is the 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_v is 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_o is set to 0.01 per new node discovered, obtained by calculating the intersect of the current octree with the previous octree.
- $L(v, e)$ is a penalty function that grants the agent given

  $$\max L(v, e) =$$

  where blablabla

In summary, 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 policies were also covered. Following sections present the implementation of the agent and the environment, the performance interpretation, and the experiments.

3.3.5. Implementation

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 [120] 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
```

3.4. Interpretation

In order to determine the actual performance of our approach, a set of metrics need to be taken into consideration to evaluate it against a set of baselines. Following the work done by Chaplot et al. [5], the following baselines were considered suitable for this task:

- **Random Walk.** A baseline that samples action randomly.
- **Shortest Path.** This baseline provides the agent with the shortest path to the nearest voxel through an orientation vector and the 3D position of the voxel.
- **Prediction Error Curiosity.** Based on the work by Pathak et al. [41], which maximizes the error in a forward prediction model.
- **Object Exploration.** This is a naive baseline where the RL algorithm maximizes the amount of object detections given by YOLO. As pointed out by [5], this policy can learn to search for frames with more objects but not at frames with different objects over time.
- **Coverage Exploration.** Based on the work by Chen and Zhang [29], this baseline learns to maximize the total exploration.
- **Semantic Curiosity.** This baseline is based on the work by [5], and uses semantic curiosity to prefer trajectories with high temporal inconsistencies in an object detector.

We also tested the following variants to analyze the influence of different components:

- **Octree Exploration.**
- **Voxel Exploration.**
- **Octree/Voxel Exploration.**
- **Octree/Voxel Exploration.**
- **Octree/Voxel/Entropy Exploration.**
- **Octree/Voxel/Entropy/BC Exploration**
- **Octree/Voxel/Entropy/Multi-Agent Exploration**

In the evaluation of each policy, hyper-parameter tuning was also explored to evaluate and compare the sensitivity of each baseline. The default hyperparameters of the trainer are presented in Table 3.1.

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 3.1: Default hyperparameters in PPO trainer, taken from [**unitymlagentsgithub**]

In order to determine if the environment or the agent required further turning or adjustments, the average accumulated reward across all runs was the main metric to determine 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 reward. 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 [55]. On one side, the movement speed, the 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, to determine the added complexity of learning a physics model through function approximation. The interpretation of the performance was additionally evaluated through metrics and visual inspection. Firstly, inspired by the DARPA subterranean challenge, the following metrics were considered: (1) earliest time the last goal was successfully scanned, averaged across all episodes; (2) earliest time the first goal was successfully scanned, averaged across all episodes; (3) lowest average time across all scan reports, averaged across all episodes. Secondly, visual inspection allows the viewer to debug faulty behavior, such as when the agent is stuck or spinning uncontrollably. The metrics for each experiment are presented in Chapter 4 and were used to adjust the environment and tune the hyperparameters of the baselines.

3.5. Further Use of Results

As stated by [], 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. This section covers, how the agent's behaviors were used to perform cross-environment exploration.

3.5.1. Generalization to New Scenarios

The working steps to answer the research questions summarize themselves into a policy capable of exploring any proposed 3D scene. Therefore, multiple scenes need to be modelled and constructed to demonstrate the applicability of the learned behavior across a variety of scenarios. Figure 3.10 illustrates three different environment were the agent was trained.

Figure 3.10: Concept of the three proposed environments.

3.5.2. Cross-Platform Performance

In order to additionally demonstrate the value of using Unity ML-Agents, the transfer capability of the developed environments to another environment platform was tested. The platform chosen was OpenAI Gym [**openAIgym capabilities**], given that it is the go-to platform in the reinforcement learning community, with a multitude of configuration settings, algorithms and compatibility with other frameworks [**openAIgym capabilities**].

The trainer used in OpenAI Gym was the Stable Baselines [**baselies**] trainer. 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 effort required to reuse a Unity Environment in OpenAI Gym, measured by the percentage of new lines of code.

4

Results

This chapter will provide and overview of the achieved results, the environment setup, the used data and the experiment process to solve the posed research question. This chapter also adapts the structure proposed for machine learning algorithms proposed by Luckert and Schaefer-Kehnert [124]. Section 4.1 illustrates the environment setup, section 4.2 describes the deployment of the processing pipeline, section 4.3 explains the data collection and neural network training, and section 4.4 presents the pipeline results. The following questions will be answered in thoroughly:

- What was the environment setup for the artificial cow?
- What was the cloud deployment setup for the solution?
- How many images were used for training, testing and optimizing the algorithms?
- Which tools were used to acquire the metrics, create the data and perform the experiments?
- Which algorithm optimizations were done?
- What were the achieved results on pose estimation?

4.1. Environment Setup

As proof of concept for the project the test environment that was set up consisted of an artificial cow at the ZHAW and a robotic arm as shown by Figure 4.1. The hardware setup for such an environment consisted of the following:

- **Blackbox PC:** where the Robot Operating System was running the pose estimation and the robot manipulation.
- **UR10e robot:** industrial robot used in machine tending, palletizing, and packaging (6 axis).
- **Arm Flange:** the flange at the end of the arm had the camera and the milking cup.
- **Dummy cow:** artificial cow model with fake teats to test the robotic attachment.

An important aspect in the physical setup for the given task is the specific camera characteristics. A camera can have different performances as a factor to light conditions, shifting, object colors, glass panels and lower resolutions. Given the systemic error a camera can introduce, it is essential to minimize the influence of it. Therefore, a camera evaluation was carried out to compare diverse camera manufacturers and models. Appendix **??** illustrates the performance of different cameras that were available for evaluation, and presents a clear model that outperforms the rest.

Figure 4.1: Artificial cow setup at the ZHAW

4.2. Pipeline Deployment

The deployment of the components was done using docker compose files to leverage the capabilities of microservices, where every component used behaves as a separate component in the application. The benefit gotten out of microservices is that they are small in size, bounded by their context, independently developed and deployable and they allow for message-based communication. Figure 4.2 shows a sample docker-compose file, which describes the services and the volumes being deployed. In this particular example, Pose Estimator waits for ROSTeat to be succesfully deployed, whereas ROSTeat waits for ROSPlay.

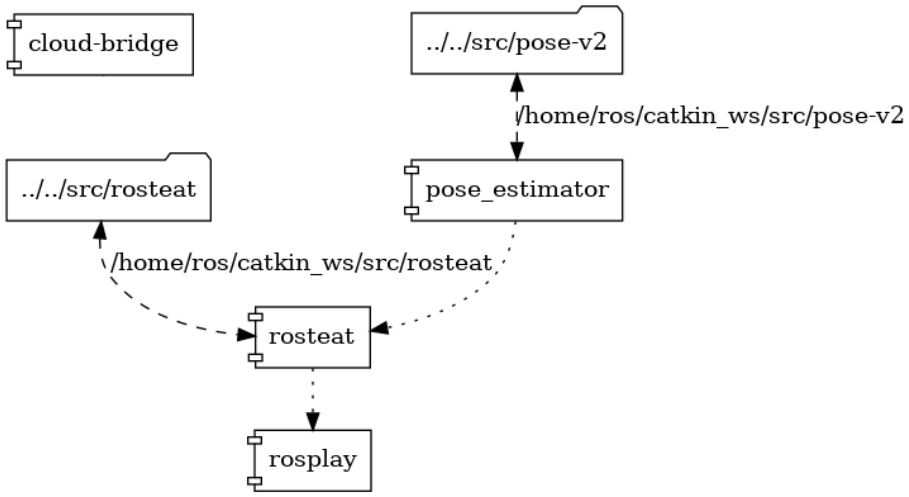

Figure 4.2: Example of a Docker-compose Deployment

The components proposed in the previous Chapter were developed both in Python 2.7 and 3.6, which was not an obstacle given the modular approach. The only problem encountered were the outdated packages given that Python 2.7 has been deprecated as of January 1st, 2020. Aditionally, the set of libraries and tools used for developing the pose estimation component in the Robot Base was the Robot Operating System (ROS). "ROS provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management" [133]. The specific versions of ROS used for the development were ROS Melodic and ROS Noetic.

4.3. Segmentation Network Training

The training data was collected using the robot's base recording procedure, which stores the set of input frames observed as a video file in ROS format (ROSbag), which could be replayed for future testing or simulations. Rviz can then be used to collect the RGB images and depth images at manually indicated timestamps. The RGB-D input received from the camera consists of images with a resolution of 640 x 480 x 4 pixels.

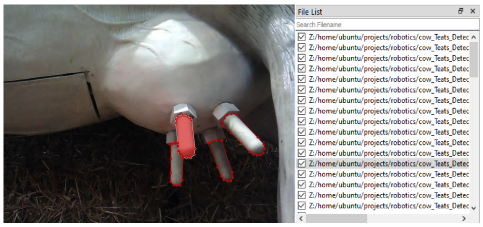

Figure 4.3: Image annotation example.

The collected realistic images are then imported into the image annotation tool LabelMe [134]. Labelme is used to manually mark the pixel-wise segmentation areas and to assign the class labels. Figure 4.3 shows a glimpse of the labelling tool. Once an image has been labelled the pixel-wise segmentation is stored as a polygon in JSON file next to the original image. As a final step, a Python script is used to merge all the individual JSON files into one, before feeding the data set into the network for training.

Number of Synthetic Training Images generated	400'000
Number of Synthetic Validation Images generated	20'000
Number of Real Training Photos collected	192
Number of Real Validation Photos collected	192
Total Number of Features	1

Table 4.1: Data sets statistics.

The training of MaskRCNN and DOPE was performed on an NVIDIA Tesla T4 GPU [135]. CUDA [136] was used to parallelize the computations both in training and testing, and therefore speed up the processing times. Table 4.1 provides an overview of the data set statistics used for training. Additionally, the docker images were configured so that the training parameters can be input as environment variables. These parameters include the data set, local weights and the remote weights locations, the epochs, score threshold, learning rate, etc. Table 4.2 show the classification errors of the methods trained

Method	**Dataset**	**Learning rate**	**Epochs**	**Accuracy**
matterport/MaskRCNN - Py2.7	Real	0.002	100	97%
matterport/MaskRCNN - Py3.6	Real	0.002	100	97%
Detectron2/MaskRCNN	Real	0.00025	2000	98%
Detectron2/MaskRCNN	Synthetic	0.00025	2000	0%

Table 4.2: Segmentation networks performance.

As shown above, the synthetic data set proved the images were not photorealistic enough to close the reality gap. Both the matterport and the Detectron2 implementations of MaskRCNN showed similar accuracy. However, the implementation from matterport in Python 2.7 had an initial average loading time of 40 seconds and the implementation in Python 3.6 had a 4 seconds average loading time. Moreover, benchmarks show the implementations from matterport have a 4x slower throughput (imgs/sec) compared to Detectron2 [137]. These drawbacks and the generally better performance of the Detectron2

implementation led to it being chosen for the segmentation task.

4.4. Results

4.4.1. Research Question

The research question tackles the pose estimation of cow teats. This section will describe the results obtained for each approach proposed, along with their respective optimizations.

Table 4.3 provides a brief summary of the results and the general performance of the used algorithms, which are discussed afterwards in more detail.

Method	Accuracy	Standard Deviation
MAV	**97%**	**0.43**
DOPE	0%	-
RANSAC	0%	-

Table 4.3: Statistics of tested methods.

The following Table 4.4 displays the results for the DOPE algorithm. DOPE uses FAT formatted data sets, which are generated using NDDS, in Unreal Engine 4. These data sets are characterized for being synthetic photorealistic images. The data set variants presented below are optimized by modifying the parameters in data set used, such as: the realism degree in the materials used for the object textures, the usage of rotation in the focus object, and the usage of obstructive objects. Aditionally, all data sets include five different photorealistic scenes: beach, studio, temple, meadow and zen garden. Surprisingly none of the data sets were realistic enough to close the reality gap, which reflected in DOPE not outputting any predictions.

Deep Object Pose		
Dataset	**Size**	**Functional**
NDDS Photorealistic (base)	260k	No
NDDS Photorealistic 2.0	80k	No
NDDS with Obstruction	80k	No
NDDS with Rotation	80k	No
NDDS Small	20k	No

Table 4.4: Overview of the best DOPE results for the respective dataset variations.

The results for the RANSAC algorithm are shown in Table 4.5. The skimage implementation was used for both RANSAC and direct ORB matching. The variants presented were surprinsingly not succesful at matching the cow teats patterns. It is suspected that RANSAC and ORB matching are not the best approach for identifying simple texture objects such as the cow teats. Figure 4.4 illustrates the suspicion and the behavior of RANSAC on both the RGB and the depth images. RANSAC was tested by varying the ORB number of keypoints, the ORB threshold and RANSAC's residual threshold as well as adding a denoising step.

RANSAC / ORB Matching				
ORB #keypoints	**ORB threshold**	**Residual Threshold**	**Denoising**	**Functional**
20	0.08	N/A (ORB matching)	No	No
200	0.08	N/A (ORB matching)	No	No
200	0.02	0.5	No	No
200	0.02	0.5	Yes	No
200	0.02	0.9	No	No
200	0.02	0.9	Yes	No
200	0.08	0.5	No	No

200	0.08	0.5	Yes	No
200	0.08	0.9	No	No
200	0.08	0.9	Yes	No

Table 4.5: Overview of the best MAV results for the respective offset-averaging mechanisms combinations.

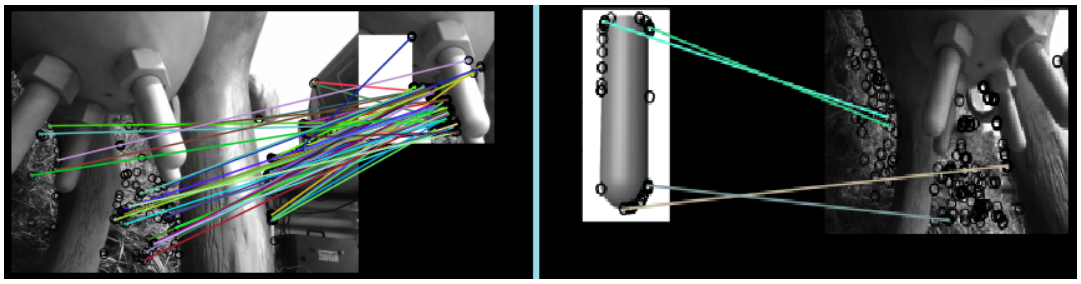

Figure 4.4: RANSAC's behavior on the cow data set.

The following table presents the results for the "MAV" algorithm. The optimizations presented are a combination of the offset and the averaging method.

MAV				
Offset	**Averaging Method**	**Average Error**	**Standard Deviation**	**Execution Time**
Fixed	Single Point	1.13	2.8	0.26 - 1.5 secs
Calculated	Single Point	1.77	3.3	0.26 - 1.5 secs
Fixed	Average: 1/3rd	**0.67**	**2.4**	0.26 - 1.5 secs
Calculated	Average: 1/3rd	1.54	2.93	0.26 - 1.5 secs
Fixed	Average: 1/10th	0.96	2.55	0.26 - 1.5 secs
Calculated	Average: 1/10th	1.51	2.87	0.26 - 1.5 secs

Table 4.6: Overview of the best MAV results for the respective offset-averaging mechanisms combinations.

4.4.2. Deliverables

The following deliverables will be handled in with this Vertiefungsarbeit:

- This work produced a pose estimation system, which is able to estimate a cow teat's pose with an average error of 0.67 cm and a standard deviation of 2.4 cm.
- This work also contributed to two other successful alternatives to the MAV algorithm that were developed by the project team for the cow teats project.
- For future research, an Unreal Engine 4 project with 5 different photorealistic scenes for a synthetic data set generation, including a cow teats data set containing over 400,000 images for pose estimation.

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 [124]. 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 best results among all algorithms were achieved by the manual manipulation of features "MAV", with a 0.67 cm error with a standard deviation of 2.46. The variations of MAV do not show substantial differences, yet they all present an average error higher than 0.5 mm with respect to the ground truth.

This contrasts with the performance shown by the other two algorithms: RANSAC and DOPE. First, RANSAC could not match the pixel descriptors for the cow teats in the RGB images as shown in Section 4.4.1. It is suspected that RANSAC could show better results if some preprocessing is done on the depth images. In other words, if the hypothesis that RANSAC cannot match figures of uniform colors from the RGB image, then a better performance could be seen when processing the depth image or the point cloud, as other research proposes [138]. Second, DOPE's performance was unexpected since it was used before for 3D recognition of chairs at the ZHAW and it showed promising results. It is suspected that DOPE's disappointing performance is originated because of the lack of photorealism in the materials of the data sets. Since the artificial cow teats materials are not a plain "gray", it was difficult to close the reality gap with simple materials. The hypothesis for improvement is that DOPE should show a better performance if the reality gap in the data set is minimized further.

An additional point of improvement would be a mechanism to reduce the offset error shown by "MAV". The correlogram shown in Appendix **??** was generated with the outputs from the algorithm, to obtain the correlation coefficients between the errors in (x, y, z) with the camera position (x, y, z) and the camera's direction (quaternion). The coefficients proved there is a strong indirect correlation between the errors and the camera metrics. Consequently, a linear model was fit to predict the errors based on the interaction of camera information variables. The model's adjusted R-squared shows that 94% of the variation in the output variables is explained by the input variables. Additionally, the overall F-statistic of the model show that the model indeed provides a better fit than the intercept-only model. Finally,

as shown in Figure 5.1, the model's residuals were analyzed. These show a constant error, constant variance, normal distribution and no influential outliers. All these cues hint towards the possibility of using a linear model to predict the error in the "MAV" algorithm to reduce the offset error in the final pose estimations.

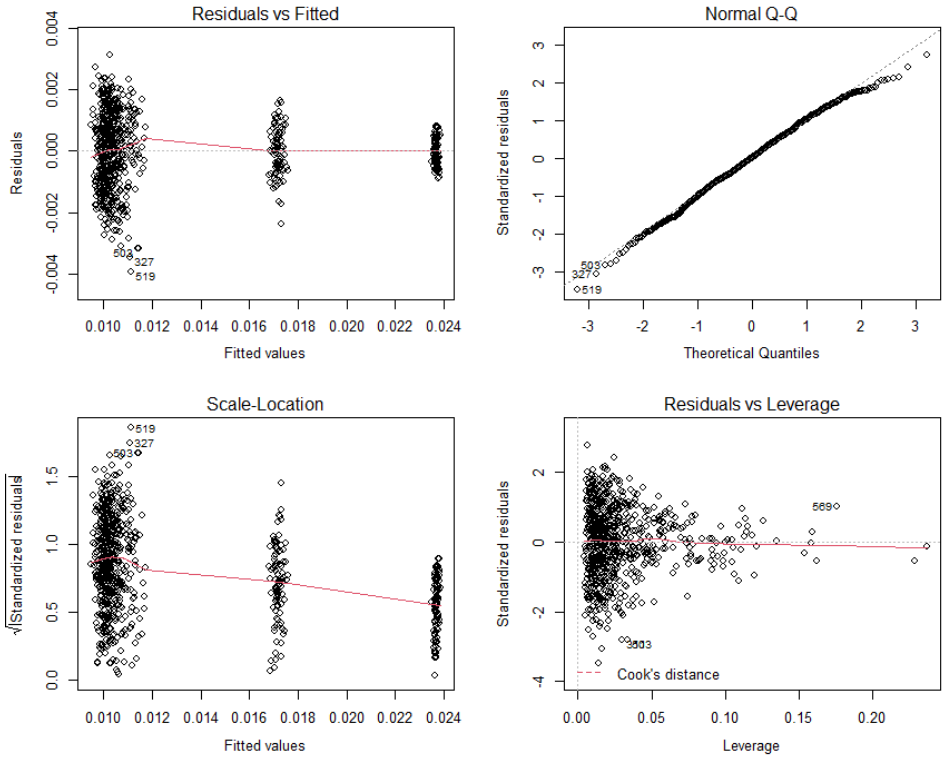

Figure 5.1: Linear model residuals of the model fit to predict the error from "MAV".

5.1.1. Research Question
5.1.2. Generalization Scenarios
5.1.3. Semantic Sub-modules

5.2. Method Reflection

The used method, adapted from the work by [124], proved to be appropriate for proposing a solution for the research question given. The structure provided allowed for the setup of different experiments without neglecting the initial goal. From the identification of the goal, through the data description and the specification and proposition of the computer vision approaches, the method proved to efficiently structure the work required to construct a pose estimation algorithm using machine learning.

5.3. Reliability

This section aims to discuss the validity and reliability of the results shown in section 4.4.1.

The achieved results by the pose estimation algorithm, specially the adaptations done in contribution based off this work, show better performance than the methods in the market for the cow milking robots. The obtained results show promising a performance for current challenges like detecting two cow teats as one. Once the error in the proposed method is reduced, the proposed pipeline could be connected to a memory system that would keep track of the cow teat positions in cases of obstruction from the suction cups, as discussed in Section 6.2.

In contrast, a few shortcomings related to the achieved results must be mentioned. First, the RANSAC algorithm analysis was limited to RGB images and should have been expanded to evaluate the information from the depth image and the point cloud. Second, a more photorealistic data set should be

generated to show DOPE's prediction capabilities and carry out a proper comparison with the "MAV" algorithm. Third, the search space of the parameters adjusted should be wider and not limited to the ones presented. In conclusion, the "MAV" algorithm answers the research question with success, being able estimate the 3D pose and direction of a cow teat in less than 2 seconds.

6

Conclusion

6.1. Conclusions

This work answered the following research question:

- How can the cow teats 3D pose be estimated under 10 seconds?

This was done by constructing a data set, training a segmentation algorithm and estimating the pose of the cow teats from the image and the predicted segementation masks. The images for the data set were collected from an artificial cow at the ZHAW using ROSBags to manually export frames at specific timestamps. The images were subsequently annotated and added to the data set. The model was then trained and tested using the generated data set. The false negatives and false positives indicated that more pictures at specific time stamps and positions with respect to the artificial cow had to be added to the data set, to increase the model's accuracy. Consequently, a pipeline was constructed for the independent deployment of the segmentation network and the pose estimation algorithms. The segmentation network would predict and publish segmentation masks for the pose estimation component to consume them and predict, in conjunction with the input images, the pose estimation of the observed cow teats. The methods tested were RANSAC and a manual manipulation approach "MAV". The latter contributed to two other approaches at the ZHAW for the pose estimation of cow teats. From the methods presented in this work, the best results were by a manual approach "MAV", which had a 0.67 cm error with a standard deviation of 2.46.

Additionally, an Unreal Engine 4 project for data set generation was constructed for further research. It contains 5 photorealistic scenes with present parameters for rotations and obstructing objects. Finally, a synthetic data set of cow teats was generated with over 400k images for pose estimation. This dataset was used to train the pose estimation algoritmh DOPE, which unfortunately could not close the reality gap and output predictions from real images.

6.2. Where to Go From Here?

This work presents a pose estimation system for cow teats using machine learning and computer vision methods. There are a few possible routes for the extension of this work to improve the performance of the achieved results. First, a different segmentation network with a faster prediction time could significantly reduce the overall performance time of the presented approach, leading to almost real-time performance. Second, a memory system could help in cases of obstruction. For example, when the suction cups are attached, the system could still remember the position of the cow teats and attempt to reattach in case one of the cups detached on movement. Third, an active vision system could significantly improve the algorithm's time to obtain precise predictions. An active vision system would control the camera position and movement to collect frames with the least amount of movements, so that the objects in the scene could be remembered and understood more efficiently.

In conclusion, the presented work extends the related research on this topic by a providing machine learning and computer vision based approach for the 3D pose and direction estimation of cow teats.

(START DISCUSSION POINTS GRABBED WHILE WRITING) Finally, given trajectories that incorporate the concept of temporal classification entropy, one can sample images to create a dataset for a semantic detector.

the work of Learning 3D Semantic Reconstruction on Octrees inspirees further work to look into octree subdivision based onn initial inherent entropy in each node.: motivate resolution of inherent uncerainty in each node by exploring it and motivate subdivisions such as in vespa2019adaptive

SOURCE: TENSORFLOW 3D ARTICLE: discussion future work, add photo of encoder In our recent paper, "DOPS: Learning to Detect 3D Objects and Predict their 3D Shapes", we describe in detail the single-stage weakly supervised learning algorithm used for object detection in TF 3D. In addition, in a follow up work, we extended the 3D object detection model to leverage temporal information by proposing a sparse LSTM-based multi-frame model. We go on to show that this temporal model outperforms the frame-by-frame approach by 7.5

Bibliography

[1] Unity Tecnologies. *Unity*. 2021. URL: https://unity.com/.

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

[3] 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.

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

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

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

[7] Lely Industries N.V. *Automatic Milking - Astronaut A5*. 2021. URL: https://www.lely.com/solutions/milking/astronaut-a5/.

[8] 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.

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

[10] 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.

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

[12] 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.

[13] Niall O'Mahony et al. "3D Vision for Precision Dairy Farming". In: *IFAC-PapersOnLine* 52.30 (2019), pp. 312–317.

[14] Wenming Cao et al. "A comprehensive survey on geometric deep learning". In: *IEEE Access* 8 (2020), pp. 35929–35949.

[15] 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.

[16] 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.

[17] 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.

[18] Karen Simonyan and Andrew Zisserman. "Two-stream convolutional networks for action recognition in videos". In: *arXiv preprint arXiv:1406.2199* (2014).

[19] Ali Diba et al. "Temporal 3d convnets: New architecture and transfer learning for video classification". In: *arXiv preprint arXiv:1711.08200* (2017).

[20] Rui Hou et al. "An efficient 3d CNN for action/object segmentation in video". In: *arXiv preprint arXiv:1907.08895* (2019).

[21] 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.

[22] 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.

[23] 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.
[24] 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).
[25] 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.
[26] Eric Jang et al. "End-to-end learning of semantic grasping". In: *arXiv preprint arXiv:1707.01932* (2017).
[27] 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.
[28] 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.
[29] Zhiqin Chen and Hao Zhang. "Learning implicit fields for generative shape modeling". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5939–5948.
[30] Richard Hartley Andrew Zisserman. "Multiple view geometry in computer vision". In: (2004).
[31] Sebastian Thrun. "Probabilistic robotics". In: *Communications of the ACM* 45.3 (2002), pp. 52–57.
[32] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.
[33] 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.
[34] 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.
[35] 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.
[36] 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.
[37] Fereshteh Sadeghi and Sergey Levine. "Cad2rl: Real single-image flight without a single real image". In: *arXiv preprint arXiv:1611.04201* (2016).
[38] Jingwei Zhang et al. "Neural slam: Learning to explore with external memory". In: *arXiv preprint arXiv:1706.09520* (2017).
[39] 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.
[40] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. "Incentivizing exploration in reinforcement learning with deep predictive models". In: *arXiv preprint arXiv:1507.00814* (2015).
[41] Deepak Pathak et al. "Curiosity-driven exploration by self-supervised prediction". In: *International conference on machine learning*. PMLR. 2017, pp. 2778–2787.
[42] 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).
[43] Manuel Lopes et al. "Exploration in model-based reinforcement learning by empirically estimating learning progress". In: *Neural Information Processing Systems (NIPS)*. 2012.
[44] Satinder Singh, Andrew G Barto, and Nuttapong Chentanez. *Intrinsically motivated reinforcement learning*. Tech. rep. MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER SCIENCE, 2005.
[45] 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.

[46] 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.
[47] 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.
[48] Ruzena Bajcsy. "Active perception". In: *Proceedings of the IEEE* 76.8 (1988), pp. 966–1005.
[49] 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.
[50] 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.
[51] Devendra Singh Chaplot, Emilio Parisotto, and Ruslan Salakhutdinov. "Active neural localization". In: *arXiv preprint arXiv:1801.08214* (2018).
[52] 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.
[53] Peter Auer. "Using confidence bounds for exploitation-exploration trade-offs". In: *Journal of Machine Learning Research* 3.Nov (2002), pp. 397–422.
[54] Thomas Jaksch, Ronald Ortner, and Peter Auer. "Near-optimal Regret Bounds for Reinforcement Learning." In: *Journal of Machine Learning Research* 11.4 (2010).
[55] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
[56] Benjamin Eysenbach et al. "Diversity is all you need: Learning skills without a reward function". In: *arXiv preprint arXiv:1802.06070* (2018).
[57] 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.
[58] Manolis Savva et al. "MINOS: Multimodal indoor simulator for navigation in complex environments". In: *arXiv preprint arXiv:1712.03931* (2017).
[59] 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.
[60] Devendra Singh Chaplot and Guillaume Lample. "Arnold: An autonomous agent to play fps games". In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.
[61] Alexey Dosovitskiy and Vladlen Koltun. "Learning to act by predicting the future". In: *arXiv preprint arXiv:1611.01779* (2016).
[62] Guillaume Lample and Devendra Singh Chaplot. "Playing FPS games with deep reinforcement learning". In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.
[63] Piotr Mirowski et al. "Learning to navigate in complex environments". In: *arXiv preprint arXiv:1611.03673* (2016).
[64] Yuxin Wu and Yuandong Tian. "Training agent for first-person shooter game with actor-critic curriculum learning". In: (2016).
[65] Devendra Singh Chaplot et al. "Gated-attention architectures for task-oriented language grounding". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.
[66] Karl Moritz Hermann et al. "Grounded language learning in a simulated 3d world". In: *arXiv preprint arXiv:1706.06551* (2017).
[67] 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.
[68] 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.
[69] Abhishek Das et al. "Embodied question answering". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1–10.
[70] 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.

[71] Devendra Singh Chaplot et al. "Learning to explore using active neural slam". In: *arXiv preprint arXiv:2004.05155* (2020).
[72] 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.
[73] 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.
[74] 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.
[75] Eman Ahmed et al. "A survey on deep learning advances on different 3D data representations". In: *arXiv preprint arXiv:1808.01462* (2018).
[76] Minecraft Middle Earth. *Minecraft Middle Earth - Minas Tirith - Cinematic Showcase*. 2020. URL: https://www.youtube.com/watch?v=KrDDHGACpok.
[77] 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.
[78] 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.
[79] Open3D. *ICP Registration*. 2020. URL: http://www.open3d.org/docs/latest/tutorial/Basic/icp_registration.html.
[80] Open3D. *Voxelization*. 2020. URL: http://www.open3d.org/docs/latest/tutorial/geometry/voxelization.html.
[81] 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.
[82] Xiaoshui Huang et al. "A comprehensive survey on point cloud registration". In: *arXiv preprint arXiv:2103.02690* (2021).
[83] Open3D. *Point Cloud*. 2020. URL: http://www.open3d.org/docs/0.7.0/tutorial/Basic/pointcloud.html.
[84] 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.
[85] 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.
[86] 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.
[87] OpenGenus IQ. *Octree data structure*. 2021. URL: https://iq.opengenus.org/octree/.
[88] Geeks for Geeks. *Octree | Insertion and Searching*. 2021. URL: https://iq.opengenus.org/octree/.
[89] Samuli Laine and Tero Karras. "Efficient sparse voxel octrees". In: *IEEE Transactions on Visualization and Computer Graphics* 17.8 (2010), pp. 1048–1059.
[90] 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.
[91] 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.
[92] 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.
[93] 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.
[94] Aaron Knoll. "A survey of octree volume rendering methods". In: *GI, the Gesellschaft für Informatik* (2006), p. 87.

[95] Emanuele Vespa et al. "Adaptive-resolution octree-based volumetric SLAM". In: *2019 International Conference on 3D Vision (3DV)*. IEEE. 2019, pp. 654–662.
[96] 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).
[97] 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.
[98] 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.
[99] Nikhila Ravi et al. "Accelerating 3D Deep Learning with PyTorch3D". In: *arXiv:2007.08501* (2020).
[100] 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.
[101] Matthias Fey and Jan E. Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.
[102] Timo Hackel et al. "Semantic3d. net: A new large-scale point cloud classification benchmark". In: *arXiv preprint arXiv:1704.03847* (2017).
[103] 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.
[104] 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.
[105] 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.
[106] 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).
[107] Google AI. *Announcing the Objectron Dataset*. 2020. URL: https://ai.googleblog.com/2020/11/announcing-objectron-dataset.html.
[108] 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.
[109] *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.
[110] *Spatial*. URL: https://spatial.io/.
[111] Asma. *What is a 3D Modeled Environment?* Feb. 2021. URL: https://it-s.com/what-is-a-3d-modeled-environment/.
[112] Jack Collins et al. "A Review of Physics Simulators for Robotic Applications". In: *IEEE Access* (2021).
[113] Roboti LLC. *MuJoCo*. 2018. URL: http://www.mujoco.org/.
[114] Open Source Robotics Foundation. *Gazebo Simulator*. 2014. URL: http://gazebosim.org/.
[115] Alexey Dosovitskiy et al. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.
[116] Pasquale Scionti. *Silentday Created with Unreal Engine 4.26 Physical Based Lighting Values*. 2021. URL: https://www.artstation.com/artwork/9m6PWo.
[117] June 8 and Gerard Andrews. *What is synthetic data?* July 2021. URL: https://blogs.nvidia.com/blog/2021/06/08/what-is-synthetic-data/.
[118] Gartner$_{Inc}$. *Maverick* Research: Forget About Your Real Data - Synthetic Data Is the Future of AI*. URL: https://www.gartner.com/en/documents/4002912.
[119] 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.

[120] Rickard Eriksson. *Deep Reinforcement Learning Applied to an Image-Based Sensor Control Task*. 2021.
[121] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.
[122] 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.
[123] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).
[124] Michael Luckert and Moritz Schaefer-Kehnert. *Using machine learning methods for evaluating the quality of technical documents*. 2016.
[125] 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.
[126] 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.
[127] Yi Zhang and Fei Huang. "Panoramic Visual SLAM Technology for Spherical Images". In: *Sensors* 21.3 (2021), p. 705.
[128] 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.
[129] 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.
[130] 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.
[131] Unity Technologies. *Unity Asset Store - The Best Assets for Game Making*. 2021. URL: https://assetstore.unity.com/.
[132] Sketchfab. *Sketchfab - The Best 3D Viewer*. 2021. URL: https://sketchfab.com/.
[133] Open Source Robotics Foundation. *ROS/Introduction - ROS Wiki*. 2021. URL: http://wiki.ros.org/ROS/Introduction.
[134] Wada Kentaro. *Image Polygonal Annotation with Python*. 2021. URL: https://github.com/wkentaro/labelme.
[135] NVIDIA. *NVIDIA T4 Tensor Core GPU for AI Inference*. 2021. URL: https://www.nvidia.com/en-us/data-center/tesla-t4/.
[136] NVIDIA. *CUDA Zone*. 2021. URL: https://developer.nvidia.com/CUDA-zone.
[137] FacebookAI. *Benchmarks - detectron2 documentation*. 2021. URL: https://detectron2.readthedocs.io/notes/benchmarks.html.
[138] Chavdar Papazov and Darius Burschka. "An efficient ransac for 3d object recognition in noisy and occluded scenes". In: *Asian Conference on Computer Vision*. Springer. 2010, pp. 135–148.
[139] Thomas Huang. "Computer vision: Evolution and promise". In: (1996).
[140] 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).
[141] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features". In: *European conference on computer vision*. Springer. 2006, pp. 404–417.
[142] Edward Rosten and Tom Drummond. "Machine learning for high-speed corner detection". In: *European conference on computer vision*. Springer. 2006, pp. 430–443.
[143] Alexander Goldenshluger, Assaf Zeevi, et al. "The Hough Transform Estimator". In: *The Annals of Statistics* 32.5 (2004), pp. 1908–1932.
[144] Frank CD Tsai. "Geometric hashing with line features". In: *Pattern Recognition* 27.3 (1994), pp. 377–389.
[145] 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).
[146] Xiaofan Zhang et al. "Skynet: a hardware-efficient method for object detection and tracking on embedded systems". In: *arXiv preprint arXiv:1909.09709* (2019).

[147] 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.
[148] 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.
[149] viso.ai. *Object Detection in 2021: The Definitive Guide*. 2021. URL: https://viso.ai/deep-learning/object-detection/.
[150] 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.
[151] 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.
[152] 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.
[153] 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.
[154] 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.
[155] 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.
[156] Vishakh Hegde and Reza Zadeh. "FusionNet: 3d Object Classification using Multiple Data Representations". In: *arXiv preprint arXiv:1607.05695* (2016).
[157] 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.
[158] 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.
[159] 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.
[160] 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.
[161] 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.
[162] 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.
[163] 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.
[164] 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.
[165] 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.

[166] 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.
[167] 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.
[168] 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.
[169] 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.
[170] 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.
[171] 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.
[172] 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.
[173] 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.
[174] 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.
[175] 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.
[176] 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.
[177] 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.
[178] 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.
[179] 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.
[180] 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.

List of Figures

1.1 3D Model of our Explorer Drone, from the Unity Asset Store [1]. 1

2.1 The Stanford Bunny in different 3D representations [73] 8
2.2 Voxelized representation of Minas Tirith in Minecraft [76]. A multitude of voxels can also represent high-resolution data, at a high computational memory cost (image below). . 9
2.3 Two input point clouds (left). Result after running the ICP Algorithm until convergence (right) [79]. 9
2.4 Labels of the octants of a cube (octant 0 is not shown). [84]. 10
2.5 (a) Voxel grid (b) Octree representation [86]. 10
2.6 3D object detection (left) and 3D instance segmentation (right) using Tensorflow 3D as found in [98]. 11
2.7 A 3D sparse voxel U-Net architecture using submanifold sparse convolutions (horizontal arrows) and submanifold sparse pooling (arrows downward) operations [**tf**]. 12
2.8 Class labels example of Semantic3D [102]. 13
2.9 3D hyper-realistic simulated desert using Unreal Engine 5 (early access) [109]. 13
2.10 Comparison of the capabilities between two simulators of creating realistic 3D environments: (a) CARLA [115], (b) Unreal Engine [116]. 15
2.11 Synthetic data will become the main form of data used in AI [118]. 16
2.12 Developers can alter and randomize objects, colors, lighting, materials and poses in realistic 3D environments to quickly generate synthetic data with perfect labels [117]. 16
2.13 Agent-environment relationship in an MDP according to **\cite {sutton1998introduction}**: 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}. 17
2.14 A grid-world environment [120]. Each cell represents a state with corresponding a 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. 19
2.15 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. 21
2.16 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 [122, 55]. 22
2.17 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. [123]. 23

3.1 Adapted development process from "*Using machine learning methods for evaluating the quality of technical documents*", by Luckert and Schaefer-Kehnert [124]. Describes the taken steps from data collection, through algorithm selection to architecture design. . . 25
3.2 Overview of the Unity ML-Agents Toolkit, taken from [**unity_mlagents_github**]. . . 28
3.3 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. 29
3.4 Bunny model voxelized at different resolution settings []. 30

3.5 grid sensor 2D (a) and grid sensor 3D (b) by Mbaske []. 31
3.6 Octree Construction and Navigation in Mbaske's Explorer Drone []. 32
3.7 Overview of the PPO Trainer's workflow from Unity ML-Agents [**unityml-agents**]. . . 33
3.8 Visual encoder example, architecture by Mnih et al [**unityml-agents**]. 34
3.9 3D Assets used to model the learning agent [**unityAssetStore**]. Drone Bot (top left), Wireframe Shader (top right), Spot Bot (bottom left), Tesla Bo (bottom right). 34
3.10 Concept of the three proposed environments. 39

4.1 Artificial cow setup at the ZHAW 41
4.2 Example of a Docker-compose Deployment 41
4.3 Image annotation example. 42
4.4 RANSAC's behavior on the cow data set. 44

5.1 Linear model residuals of the model fit to predict the error from "MAV". 46

A.1 David Marr's stages to identify the visual world. 62
A.2 McCulloch-Pitts model of a neuron. 63
A.3 The Mask R-CNN Framework for Instance Segmentation [149]. 64
A.4 Structure detail of YOLOv3.It uses Darknet-53 as the backbone network and uses three scale predictions [149]. 65
A.5 Wang et al. proposed method for category-level 6D pose and size estimation of multiple unseen objects in an RGB-D image. 65
A.6 Results of using OcCo (2020) pre-training for completion of point clouds. 67
A.7 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). 68
A.8 Voxelization of a 3D asset used with the mesh walking technique. 3D model (left) and voxelized model (right). 68

List of Tables

3.1 Default hyperparameters in PPO trainer, taken from [**unitymlagentsgithub**] 38

4.1 Data sets statistics. 42
4.2 Segmentation networks performance. 42
4.3 Statistics of tested methods. 43
4.4 Overview of the best DOPE results for the respective dataset variations. 43
4.5 Overview of the best MAV results for the respective offset-averaging mechanisms combinations. 44
4.6 Overview of the best MAV results for the respective offset-averaging mechanisms combinations. 44

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 [15], 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) [16] 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 [17] 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 [18] 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" [19]. 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. [20] 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 [21, 22, 23], 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 [24].

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 [139]. 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.

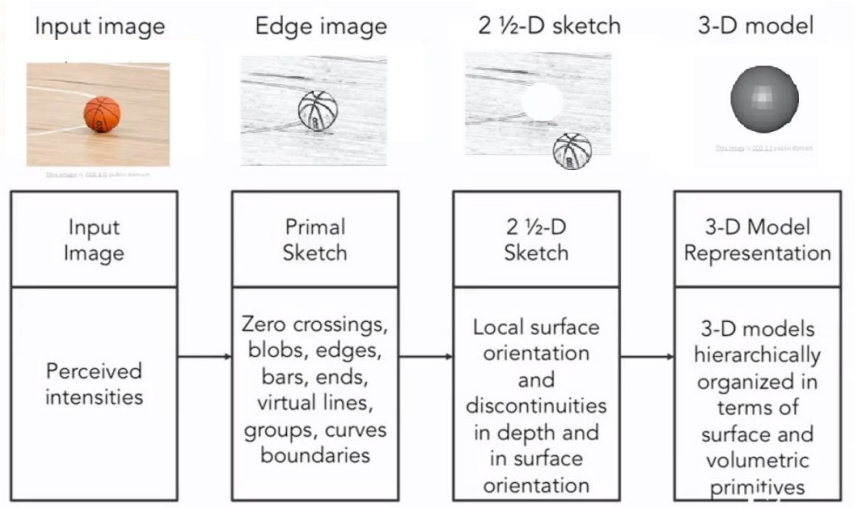

Figure A.1: David Marr's stages to identify the visual world.

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) [140]
- Speeded Up Robust Features (SURF) [141]
- Features from Accelerated Segment Test (FAST) [142]
- Hough Transforms [143]
- Geometric Hashing [144]

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. [146] 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 [147] 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.

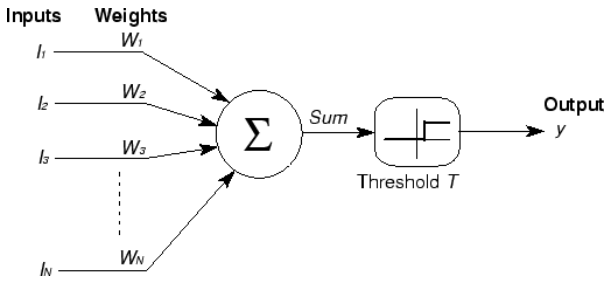

Figure A.2: McCulloch-Pitts model of a neuron.

As of August 2021, ImageNet [148] has become one of the largest high-quality public databases for image classification and the paper "ImageNet Classification with Deep Convolutional Neural Networks" [97] 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).

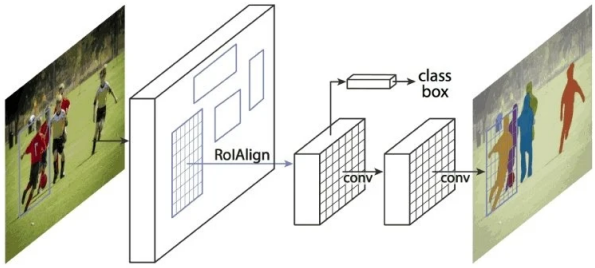

Figure A.3: The Mask R-CNN Framework for Instance Segmentation [149].

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 [149]. 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.

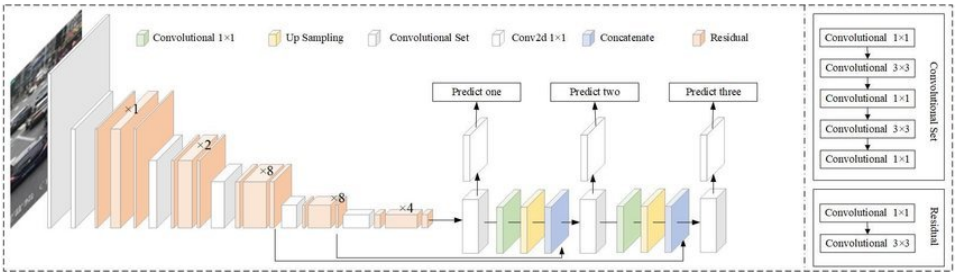

Figure A.4: Structure detail of YOLOv3.It uses Darknet-53 as the backbone network and uses three scale predictions [149].

YOLO. YOLO, in contrast, is a one-stage detection algorithm which directly predicts bounding boxes in a forward pass [149]. 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 suppression. Newer variants, such as YOLOv3 or YOLOv4 improve performance on smaller object, self-adversarial training and cross mini-batch normalization [149].

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 [150] and real data [151], 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 [152].

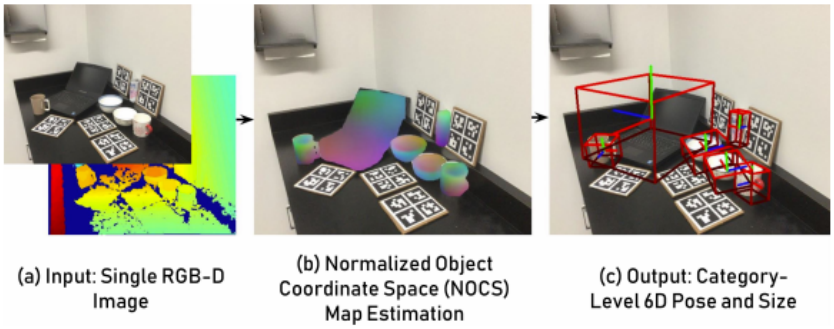

Figure A.5: Wang et al. proposed method for category-level 6D pose and size estimation of multiple unseen objects in an RGB-D image.

Sahin et al. [152] 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 [151], Vote3Deep [153] and the method by Michel et al. [154].
- **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 [155], FusionNet [156], VoxelNet [157], AVOD net [108].
- **Classification and Regression-based** approaches run both tasks mentioned above under the same architecture. Examples include MonoPSR [158], FrustumPNet [21], PointRCNN [159], SSD-6D [160] and DeepContext [161].

- **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. [162], SVMs embedded in Adaboost [163], RAPID-LR [164] and the method proposed by Ku et al. [165].
- **Point-pair feature matching** approaches were proposed by Drost et al. [166]. 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 [167].

Unfortunately, the limitations and challenges mentioned in section **??** overlap with the ones from pose estimation. Along these lines, work by Wang et al. [168] 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 [106] 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 [169] [170] [171], 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 [85], which are sparse partitioning techniques for 3D grid structures (O-CNN [172], OGN [173], OctNet [174]) to achieve high resolutions while maintaining memory efficiency. As described by Tatarchenko, Dosovitskiy, and Brox [173], 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 [175, 29] or continuous occupancy grids [176]. 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 [177] and Point-based approaches [178, 179].
- 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. [169, 180].

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.

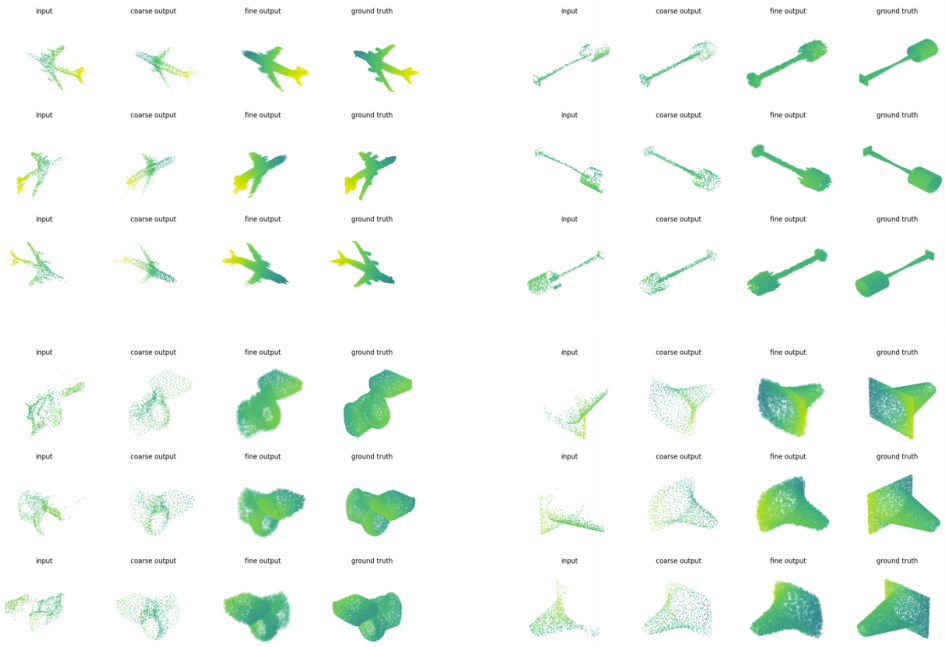

Figure A.6: Results of using OcCo (2020) pre-training for completion of point clouds.

A.4. Unity 3D: Modelling the Environment

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

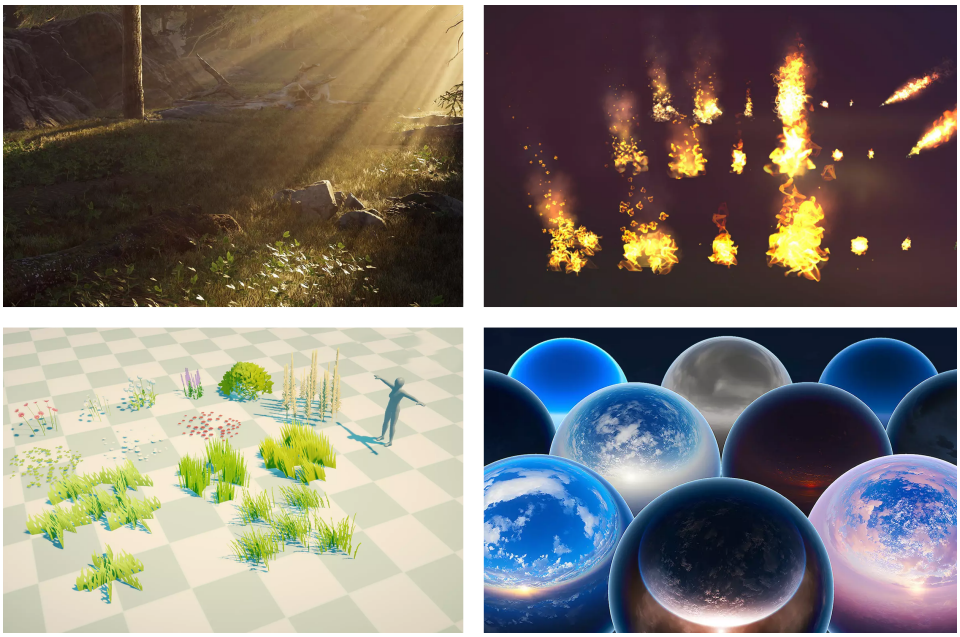

Figure A.7: Diverse Unity 3D Assets used to model 3D environments for the learning agent [**unityAssetStore**]. Wireframe Shader (top left), Drone Bot (top right), Foliage Pack (bottom left), Skyboxes (bottom right).

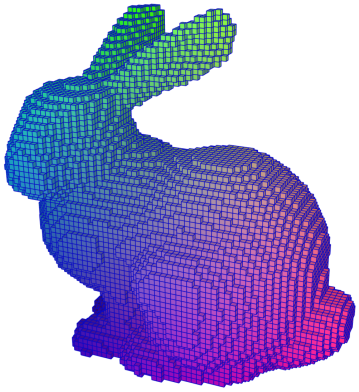

Figure A.8: Voxelization of a 3D asset used with the mesh walking technique. 3D model (left) and voxelized model (right).