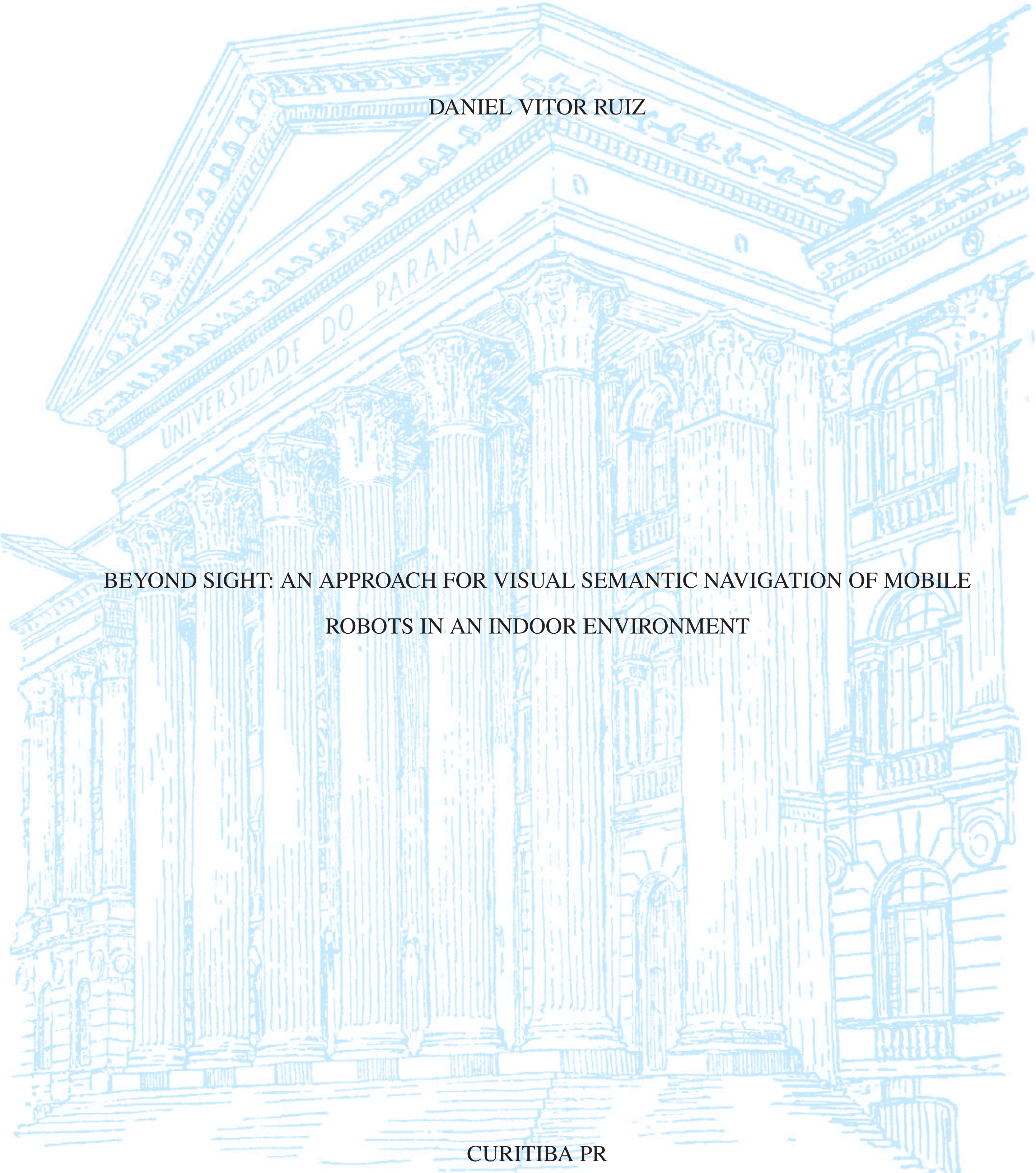
UNIVERSIDADE FEDERAL DO PARANÁ

DANIEL VITOR RUIZ



BEYOND SIGHT: AN APPROACH FOR VISUAL SEMANTIC NAVIGATION OF MOBILE ROBOTS IN AN INDOOR ENVIRONMENT

CURITIBA PR

2021

DANIEL VITOR RUIZ

BEYOND SIGHT: AN APPROACH FOR VISUAL SEMANTIC NAVIGATION OF MOBILE ROBOTS IN AN INDOOR ENVIRONMENT

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Eduardo Todt.

CURITIBA PR

2021

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

R934b Ruiz, Daniel Vitor

Beyond sight [recurso eletrônico]: an approach for visual semantic navigation of mobile robots in an indoor environment / Daniel Vitor Ruiz. – Curitiba, 2021.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2021.

Orientador: Eduardo Todt.

1. Robótica. 2. Web semântica. 3. Algoritmos. I. Universidade Federal do Paraná. II. Todt, Eduardo. III.Título.

CDD: 025.0427

Bibliotecária: Vanusa Maciel CRB- 9/1928

MINISTÉRIO DA EDUCAÇÃO
SETOR DE CIENCIAS EXATAS
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO INFORMÁTICA -
40001016034P5

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **DANIEL VITOR RUIZ** intitulada: **BEYOND SIGHT: AN APPROACH FOR VISUAL SEMANTIC NAVIGATION OF MOBILE ROBOTS IN AN INDOOR ENVIRONMENT**, sob orientação do Prof. Dr. EDUARDO TODT, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.
A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 22 de Fevereiro de 2021.

Assinatura Eletrônica
22/02/2021 16:24:58.0
EDUARDO TODT
Presidente da Banca Examinadora

Assinatura Eletrônica
22/02/2021 16:31:08.0
FERNANDO SANTOS OSORIO
Avaliador Externo (INSTITUTO DE CIêNCIAS MATEMáTICAS E DE COMPUTAçãO)

Assinatura Eletrônica
22/02/2021 16:39:49.0
EDUARDO JAQUES SPINOSA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
22/02/2021 16:17:51.0
JOAO ALBERTO FABRO
Avaliador Externo (UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ)

To my family and friends for their encouragement and support.

ACKNOWLEDGEMENTS

I want to thank my advisor Eduardo Todt for his expertise, patience, and guidance; my coworkers and colleagues for providing a welcoming workspace where ideas could always be discussed and where discoveries and progress were encouraged to be explored and published, which helped me grow academically as a researcher. I am especially grateful to Bruno A. Krinski and Gabriel Salomon for working with me on our academic publications.

I want to thank the Coordination for the Improvement of Higher Education Personnel (CAPES) for the Masters scholarship. I also gratefully acknowledge the founders of the publicly available datasets and simulators used for this work, and the support of NVIDIA Corporation with the donation of the GPUs used for this research.

We must perceive in order to move, but
we must also move in order to perceive.

James J. Gibson

RESUMO

Com o crescimento da automação, os veículos não tripulados tornaram-se um tema de destaque, tanto como produtos comerciais quanto como um tópico de pesquisa científica. Compõem um campo multidisciplinar de robótica que abrange sistemas embarcados, teoria de controle, planejamento de caminhos, localização e mapeamento simultâneos (SLAM), reconstrução de cenas e reconhecimento de padrões. Apresentamos neste trabalho uma pesquisa exploratória de como a fusão dos dados de sensores e algoritmos de aprendizagem de máquinas, que compõem o estado da arte, podem realizar a tarefa chamada Navegação Visual Semântica que é uma navegação autônoma utilizando observações visuais egocêntricas para alcançar um objeto pertencente à classe semântica alvo sem conhecimento prévio do ambiente. Para realizar experimentos, propomos uma encarnação chamada VRIBot. O robô foi modelado de tal forma que pode ser facilmente simulado, e os experimentos são reproduzíveis sem a necessidade do robô físico. Três diferentes *pipelines* EXchangeable, AUTOcrat e BEyond foram propostos e avaliados. Nossa abordagem chamada BEyond alcançou a 5ª posição entre 12 no conjunto val_mini do Habitat-Challenge 2020 ObjectNav quando comparada a outros resultados relatados na tabela classificativa da competição. O resultado da pesquisa mostra que a fusão de dados em conjunto com algoritmos de aprendizado de máquina são uma abordagem promissora para o problema de navegação semântica.

Palavras-chave: Navegação-visual-semântica. SLAM. Aprendizado-profundo. Navegação-Autônoma. Segmentação-semântica.

ABSTRACT

With the rise of automation, unmanned vehicles became a hot topic both as commercial products and as a scientific research topic. It composes a multi-disciplinary field of robotics that encompasses embedded systems, control theory, path planning, Simultaneous Localization and Mapping (SLAM), scene reconstruction, and pattern recognition. In this work, we present our exploratory research of how sensor data fusion and state-of-the-art machine learning algorithms can perform the Embodied Artificial Intelligence (E-AI) task called Visual Semantic Navigation, a.k.a Object-Goal Navigation (ObjectNav) that is an autonomous navigation using egocentric visual observations to reach an object belonging to the target semantic class without prior knowledge of the environment. To perform experimentation, we propose an embodiment named VRIBot. The robot was modeled in such a way that it can be easily simulated, and the experiments are reproducible without the need for the physical robot. Three different pipelines EXchangeable, AUTOcrat, and BEyond, were proposed and evaluated. Our approach, named BEyond, reached 5th rank out of 12 on the val_mini set of the Habitat-Challenge 2020 ObjectNav when compared to other reported results on the competition's leaderboard. Our results show that data fusion combined with machine learning algorithms are a promising approach to the semantic navigation problem.

Keywords: Visual-semantic-navigation. Deep-Learning. SLAM. Autonomous-navigation. Semantic-segmentation.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

CNN	Convolutional Neural Network
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
DBN	Dynamic Bayesian Network
DD-PPO	Decentralized Distributed Proximal Policy Optimization
Deep-RL	Deep Reinforcement Learning
DNN	Deep Neural Network
DoF	Degrees of Freedom
DWA	Dynamic Window Approach
E-AI	Embodied Artificial Intelligence
EKF	Extended Kalman Filter
EmbodiedQA	Embodied Question Answering
FCN	Fully Convolutional Network
FoV	Field of View
FPS	Frames Per Second
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
IMU	Inertial Measurement Unit
IPC	Inter-Process Communication
LSTM	Long Short-Term Memory
MLP	Multilayer Perceptron
MPC	Model Predictive Control
NLP	Natural Language Processing
ObjectNav	Object-Goal Navigation
PointNav	Point-Based Navigation
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
ResNet	Residual Network
RL	Reinforcement Learning
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
ROI	Region of Interest
ROS	Robot Operating System

RTAB-Map	Real-Time Appearance-Based Mapping
SBC	Single Board Computer
SLAM	Simultaneous Localization and Mapping
SOD	Salient Object Detection
SPL	Success weighted by inverse Path Length
SVM	Support Vector Machine
TEB	Timed-Elastic Band
TSDF	Truncated Signed Distance Function
UAV	Unmanned Aerial Vehicle
UFPR	Federal University of Paraná/*Universidade Federal do Paraná*
UGV	Unmanned Ground Vehicle
UKF	Unscented Kalman Filter
URDF	Unified Robot Description Format
VIO	Visual-Inertial Odometry
VLN	Vision-and-Language Navigation
VQA	Visual Question Answering
VRI	Vision, Robotics and Image

LIST OF SYMBOLS

η	eta, a normalization term
Φ	upper case phi, used to represent a set of transformations
ϕ	lower case phi, a particular transformation
π	lower case pi, a policy network
σ	lower case sigma, denotes the sigmoid function

CONTENTS

1 INTRODUCTION

Advances in the miniaturization of integrated circuits in the last half-century brought to reality embedded systems such as smartphones, which in many cases include in a single device, components like monocular/stereo digital cameras, Inertial Measurement Unit (IMU), microphones, speakers, high-resolution touch screen, biometric fingerprint scanner, infrared proximity sensor, multi-core Central Processing Unit (CPU) along with a Graphics Processing Unit (GPU), sometimes also called General Purpose Graphics Processing Unit (GPGPU). Those innovations combined with the lowering hardware costs broadened the horizon of what is possible to achieve in robotics, an application area where power consumption, size, and weight of each component are essential. As automation progresses, mobile robots are expected to navigate autonomously.

Traditionally autonomous navigation use point-based goals in a specific coordinate space. However, with the progress on pattern recognition, there is increasing interest in creating smarter robots that can do more than identifying similarities between observations, e.g., detecting objects and estimating their locations without having their prior explicit coordinates (Cadena et al., 2016).

Similarly, the pattern recognition field is going through a paradigm shift from 'internet AI' based on static datasets, e.g., ImageNet (Deng et al., 2009), COCO (Lin et al., 2014), VQA (Antol et al., 2015), to Embodied Artificial Intelligence (E-AI) where agents act within realistic environments, by active perception, long-term planning, learning from interaction, while grounded in an environment (Savva et al., 2019).

The domain of Embodied AI, in which agents learn to complete tasks through interaction with their environment from egocentric observations, has experienced substantial growth with the advent of Deep Reinforcement Learning (Deep-RL) and increased interest from the computer vision, Natural Language Processing (NLP), and robotics communities. This growth has been facilitated by the creation of a large number of simulated environments, such as AI2-THOR (Deitke et al., 2020), Habitat (Facebook AI Research, 2019), iGibson (Xia et al., 2020), and CARLA (Dosovitskiy et al., 2017), tasks like point navigation, instruction following, and embodied question answering, and associated leaderboards (Weihs et al., 2020), see chapter 4.

The combination of deep learning and autonomous navigation is a growing research field (Crespo et al., 2020), with methods employing different strategies, supervised learning/imitation learning, unsupervised learning, reinforcement learning (Weihs et al., 2020). Those methods aim to solve increasing abstract tasks from image target navigation, where instead of a point, an image of the goal is provided, the agent must reach a position where its viewpoint matches the goal. Target-driven visual navigation, e.g., Object-Goal Navigation (ObjectNav) and AreaNav, instead of a target image, a target class or instance is provided as a goal, not the point location of it but its class/instance label. For further details on E-AI tasks see section 2.15.

In this work our focus is the E-AI task known as Visual Semantic Navigation, a.k.a as Semantic Navigation or ObjectNav. A formal definition of ObjectNav is presented at section 2.16. Figure 1.1 illustrates the problem.

The key concept to expand from Point-Based Navigation (PointNav) to ObjectNav is semantics, which can be further divided for our purposes into scene understanding, semantic understanding, and semantic reasoning. For further details on scene understanding, object detection, semantic segmentation, and instance segmentation, see section 2.8. For further details on semantics and learning-based methods, see section 2.9.

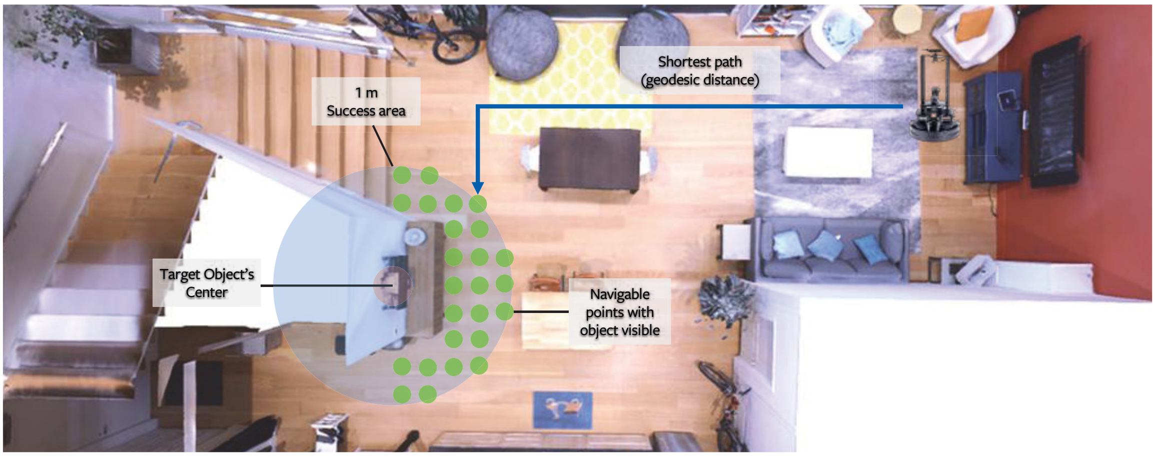


Figura 1.1: ObjectNav Evaluation: The goal object, in this case, a monitor, is highlighted in pink. All points within 1m to the object surface (highlighted as a blue circle) are considered potential success goal poses. Only navigable locations from where the object can be seen with proper orientation of the camera (highlighted in green) are considered from this area. These points cumulatively form the 'success zone,' locations where the episode would be considered successful if the agent called 'stop.' The blue arrows show the shortest path from the agent's (random) starting pose and the closest point in the success zone as predicted by a path planning oracle (Batra et al., 2020).

1.1 OBJECTIVES

Our primary goal in this work is to introduce semantic information into navigation pipelines and exploit it so that mobile robots can progress from traditional point-based navigation to the E-AI task of Visual Semantic Navigation/ObjectNav.

In this work we proposed and evaluated three complete pipelines composed of hierarchical algorithms, here referred to as the building blocks. Each complete pipeline was considered a single entity and evaluated as a single algorithm. Our pipelines are: EXchangeable, AUTOcrat, and BEyond. Our objectives in this work are:

- An overview of the state-of-the-art of E-AI Semantic Navigation. See sections 2.15, 3, 4.

- VRIBot, an embodiment that is modular and can be easily assembled with Commercial Off-The-Shelf (COTS) components to achieve a similar robot to popular ones such as LoCoBot (Carnegie Mellon University, 2019) and TurtleBot (Wise and Foote, 2010). Additionally, the embodiment can be simulated in simulators compatible with Unified Robot Description Format (URDF) (Open Robotics, 2019) and Robot Operating System (ROS) (Open Robotics, 2018), and the agent's internal logic can be used in real and simulated environments. See section 5.1.

- For our experiments, we propose a methodology to scrutinize the relation between episode settings and the agent's performance. In the literature, it is common to only report the average metrics for the entire dataset, which contains several completely different scenes. Sometimes the performance per class is also reported (Mousavian et al., 2019; Shen et al., 2019; Fang et al., 2019). We believe that this traditional way of running and reporting experiments for ObjectNav overlook several nuances, such as how the episode distance interferes with the performance. We investigate if the episode's distance is a property similar to how the size of the object interferes in object detection. See section 6.2.1.

- Our goal prediction network that can bridge dedicated modules of semantic segmentation and motion planning (see section 5.5.2). We hypothesize that by separating semantics from the motion planning, we could achieve better performance with BEyond (see section 5.5) than AUTOcrat (see section 5.4), for the experiments, see chapter 6 and the discussion on chapter 7. By introducing a dedicated module for semantic reasoning, a modular approach such as our BEyond pipeline can be an alternative to end-to-end policies that attempt to map observations to actions directly.

- Our promising approach named BEyond reached 5th rank out of 12 on the val_mini set of the Habitat-Challenge 2020 ObjectNav when compared to other reported results on the leaderboard. See section 7.1.

- The code related with this work is publicly available at `https://github.com/VRI-UFPR/BeyondSight`

2 THEORETICAL FOUNDATION

In this chapter, we discuss what is Simultaneous Localization and Mapping (SLAM), its history, the concepts used with it, such as belief, dealing with data uncertainty, Bayes filter, and Markov's assumption. An illustrative example is presented in section 2.2 to clarify some of the intuition behind the math used for SLAM. Then in section 2.3, we provide a discussion about different approaches for data fusion. We briefly discuss Hidden Markov Model (HMM) and graph-based SLAM, along with Real-Time Appearance-Based Mapping (RTAB-Map), a particular method used in this work.

In section 2.6 are presented the different types of kinematic models such as holonomic and nonholonomic. Then, in section 2.7 is presented how motion planning is done with model-based algorithms. Next, we talk about scene understanding and how it can be divided into classification, detection, semantic segmentation, and instance segmentation problems. We discuss how semantics can be exploited to create smarter agents and how learning-based methods are particularly interesting for this. Subsequently, the taxonomy proposed by Chen et al. (2020) to represent the confluence of Deep Learning and mobile robots are briefly discussed. Some essential concepts of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are also presented.

The chapter ends with a discussion about Embodied Artificial Intelligence (E-AI) navigation and its ramifications, along with essential definitions such as what constitutes an episode, what makes an episode successful, the metrics used for evaluation, and their limitations.

2.1 SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)

Autonomous robots operating in the real world must be able to navigate in large, unstructured, dynamic, and unknown spaces. However, to go somewhere, it is desirable first to know where it stands. This is where SLAM comes in handy since it provides a robot with the capability to build and update a map of its surrounding environment and simultaneously localize itself in it.

SLAM is a class of algorithms that intend to determine an agent's location in an unknown environment and simultaneously create a map using landmarks to minimize error on the estimation. SLAM has a long history with its beginnings dating back to 1986. A thorough historical review of the first 20 years of the SLAM problem is given by the two surveys (Durrant-Whyte and Bailey, 2006; Bailey and Durrant-Whyte, 2006).

Cadena et al. (2016) defined two distinct ages on SLAM research: **the classical age** (1986-2004) with the introduction of the main probabilistic formulations for SLAM, including approaches based on Extended Kalman Filters, Rao-Blackwellised Particle Filters, and maximum likelihood estimation. Moreover, it delineated the basic challenges connected to efficiency and robust data association. The **algorithmic-analysis age** (2004-2015), partially covered by Dissanayake et al. (2011). The study of SLAM's fundamental properties, including observability, convergence, and consistency. In this period, sparsity's key role in efficient SLAM solvers was also understood, and the main open-source SLAM libraries were developed. With the rise of deep learning and its increased adherence to robotics, one could argue that now we are going towards an **age of spatial machine intelligence**, a term used by Chen et al. (2020).

Before going into further details about SLAM algorithms, some key concepts need to be discussed. First, a formal definition of *state* is required. Let us consider the state a mathematical model for the collection of all relevant aspects of the robot and its environment that can impact the future, in the context of localization. Certain state variables are *dynamic* and tend to change over time, such as the robot *pose* and the whereabouts of people in the vicinity of a robot. Others tend to remain *static*, such as the location of walls in buildings. The state is denoted by x, and the state at time t is denoted by x_t.

Throughout this document, we will refer to an embodied agent in the world, i.e., a mobile robot platform, or in simulation, i.e., a virtual robot, simply as an agent. An agent's *pose* comprises its location and orientation relative to a global coordinate frame. Rigid mobile robots possess six such state variables, a.k.a six Degrees of Freedom (DoF), three for their Cartesian coordinates, and three for their angular orientation (pitch, roll, and yaw). Orientation is more commonly represented with a quaternion since it can avoid gimbal lock (Perumal, 2011). For rigid mobile robots confined to planar environments, the pose is usually given by three variables, a.k.a 3DoF, its two location coordinates in the plane, and its heading direction (yaw).

A crucial notion from here onwards is that robot motion and sensors have some uncertainty associated with them, and the only way of inferring the state is through them. Sensors can be divided into *proprioceptive* and *exteroceptive*. The former type deals with the system's internal information; the wheel odometer is an example of such a sensor, while the latter deal with external data such as the luminosity of the surrounding environment. The uncertainty derives from unmodeled variables that interfere with the measurements. e.g., when using a wheel encoder, wheel slippage can interfere with the internal measure of the speed of the robot resulting in a difference between the linear velocity measured and the real one.

A robot is likely to overshoot or undershoot a target location slightly. Furthermore, some variables can be complex to predict and account for properly. Thus, we modeled all data concerning the robot as ***probabilistic***, with *stochastic* elements rather than simply *determinist*. See section 2.3 for further discussion on data fusion and different approaches when dealing with uncertain data.

The terms *environment* and *world* was used throughout this work interchangeably when referring to a two or three-dimensional workspace containing a set of simple, closed, untraversable obstacles, which the robot is not allowed to overlap. For the sake of simplicity, the area outside the obstacle is considered homogeneous and equally easy to navigate. Examples of cluttered environments may include offices, human-made structures, and urban environments.

A robot *trajectory* can be defined as a sequence of poses in time, or rather a set of dynamic variables on a sequence of states. When navigating without external information, a.k.a *dead-reckoning* — the process of calculating one's current position by using a previously determined position and advancing that position based upon known or estimated speeds over elapsed time and course without the aid of landmarks— accumulative errors can quickly drift the estimated trajectory from the real one. So, to counter accumulative errors, it is desirable to perform a *loop closure*. This refers to the process when a robot detects that it reached a previously known area and thus can correct its current localization with its real position according to its internal representation, which could be a *map*.

A *map* is a representation of aspects of interest, such as the position of *landmarks*, and obstacles, describing the environment in which the robot operates. Here *landmark* refers to a distinguishable static object or feature that can be recognized reliably. A map is often required to support other tasks; for instance, a map allows precise path planning and intuitive visualization for a human operator. Maps can be parametrized as a set of spatially located landmarks by dense representations like occupancy grids, surface maps, or raw sensor measurements. Figure 2.1 illustrates three typical dense map representations for 3D and 2D: multilevel surface maps, point clouds, and occupancy grids.

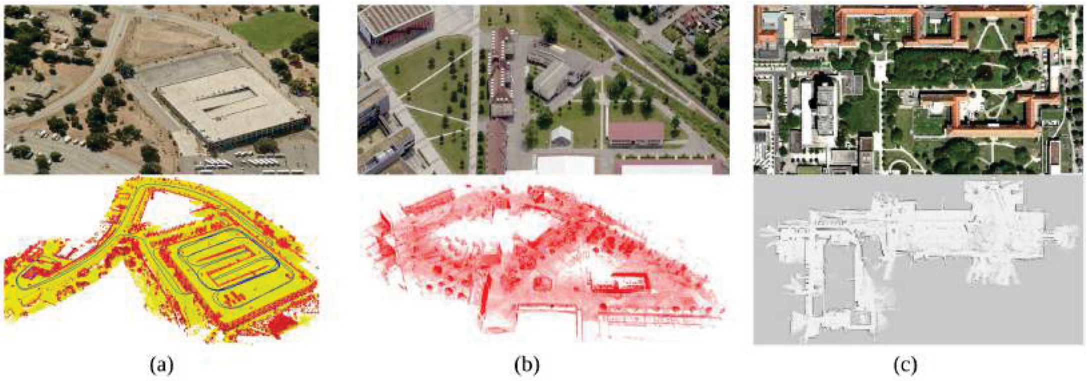


Figura 2.1: Examples of 3D Maps. Subfigure (a): a 3D map of the Stanford parking garage acquired with an instrumented car (bottom) and the corresponding satellite view (top). Subfigure (b): a Point cloud map acquired at the University of Freiburg and relative satellite image. Subfigure (c): an Occupancy grid map acquired at the hospital of Freiburg, with a bird's eye view of the area (top) and the occupancy grid representation (bottom). The gray areas represent unobserved regions. The white part represents traversable space while the black points indicate occupied regions (Grisetti et al., 2010).

As stated before, solving the SLAM problem consists of estimating the robot trajectory and the map of the environment as the robot moves in it. The robot is assumed to move in an unknown environment, along a trajectory described by a sequence of poses, modeled as random variables, $x_{1:t} = \{x_1, \ldots, x_t\}$ (from now on we will usually refer the notation of state x when referring to a pose since pose is the variable of the state that we are most interest during SLAM). While moving, it acquires a sequence of odometry measurements $u_{1:t} = \{u_1, \ldots, u_t\}$ and perceptions of the environment $z_{1:t} = \{z_1, \ldots, z_t\}$. We will distinguish data acquired from odometry, from data acquired by other sensors, being the former referred to as *control data* and the latter as *measurement data*.

To actually solve the problem, it is necessary to estimate the posterior probability of the robot's trajectory $x_{1:t}$ and the map m of the environment given all the measurements plus an initial position x_0:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}, x_0) \tag{2.1}$$

The initial position is *prior* knowledge, which can be arbitrarily defined as the origin of the coordinate system. Estimating the posterior at equation 2.1 is only tractable under some assumptions, namely the static world assumption and the *Markov assumption* (Thrun et al., 2005). As the name suggests, the static world assumption assumes that the only dynamic agent in the world is the robot and all other elements are static. The Markov assumption entails that a state x_t is *complete*, meaning that knowledge of past states, measurements, or controls carry no additional information that would help us predict the future more accurately. In other words, x_t is said to be complete if it is robust enough to describe the world without any additional information other than itself. Temporal processes that meet these conditions are commonly known as *Markov chains*. The Equation 2.2 illustrates the implication of a complete state on the posterior computation, while not worrying about the map; the equality is an example of *conditional independence*, and $p(x_t|x_{t-1}, u_t)$ is known as *state transition probability*.

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{2.2}$$

One might also want to model the process by which measurements are being generated. Again, if x_t is complete, we have an important conditional independence:

$$p(z_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(z_t|x_t) \tag{2.3}$$

In other words, the state x_t is sufficient to predict the (potentially noisy) measurement z_t. Knowledge of any other variable, such as past measurements, controls, or even past states, is irrelevant if x_t is complete. The probability $p(z_t|x_t)$ is called the *measurement probability* or the *observation model*. The observation models the *likelihood* of a measurement z_t given a state x_t. However, states cannot be directly obtained, only inferred.

So, the concept of *Belief* is also relevant to present here, since it refers to the robot's internal knowledge about the state of the environment. Synonyms for belief in the literature are the terms *state of knowledge* and *information state*. Belief arises from inference on the data, so there is no guarantee that it is perfectly equal to the true state. We will denote belief over a state variable x_t by $bel(x_t)$, which is an abbreviation for the posterior:

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{2.4}$$

The most general algorithm for calculating beliefs is given by the recursive *Bayes filter* algorithm (Thrun et al., 2005). Which can be described by the Equation 2.5:

$$bel(x_t) = \eta \underbrace{p(z_t|x_t)}_{\text{observation}} \int \underbrace{p(x_t|x_{t-1}, u_t)}_{\text{transition}} \underbrace{bel(x_{t-1})}_{\text{recursion}} dx_{t-1} \tag{2.5}$$

By definition, time is assumed to be continuous, and thus state, control, and observation data are also continuous. Where η is a normalization term to ensure that value stay between $[0, 1]$. A somewhat simplified *Bayes filter* is the *Kalman filter*, which operates under further assumptions, such as, the random distribution is always a Gaussian, and the transitions are linear functions; a nonlinear version of the standard Kalman filter is known as Extended Kalman Filter (EKF) (Welch and Bishop, 1995). The *Bayes filter* algorithm can be discretized, leading to a family of algorithms known as *particle filtering* which are Monte Carlo based (Thrun et al., 2005).

2.2 WALL FOLLOWER EXAMPLE

Here we present an example to clarify the intuition behind the math of probabilistic localization. This example is adapted from Thrun et al. (2005). Our focus will be simply self-localization instead of SLAM. This entails that the agent has prior knowledge about the environment and merely needs to determine its position. Imagine a world where there is a hallway with three identical doors. A simple mobile robot exists in this world. It can move forward along the hall, and it has only one sensor. The sensor is facing the wall, and it measures if there is a door or not. However, its measurements are imprecise, with random noise. The implication is that the sensor can measure a door when there is not and vice-versa. So, how can the robot know its location? Through belief. More precisely, it can use the concept of belief and a Bayes filter to estimate its state, consequently its position. To do so, it starts with a prior of a uniform distribution. This is the *base case* for the recursive part of our filter, see figure Figure 2.2 (a).

For the sake of simplicity our *observation model*, which dictates our *likelihood function*, follows a Gaussian distribution. So, starting from a prior, the robot takes an action u_1 which is: do not move, in sequence, it makes a measurement z_1, this information is pumped into the Bayes

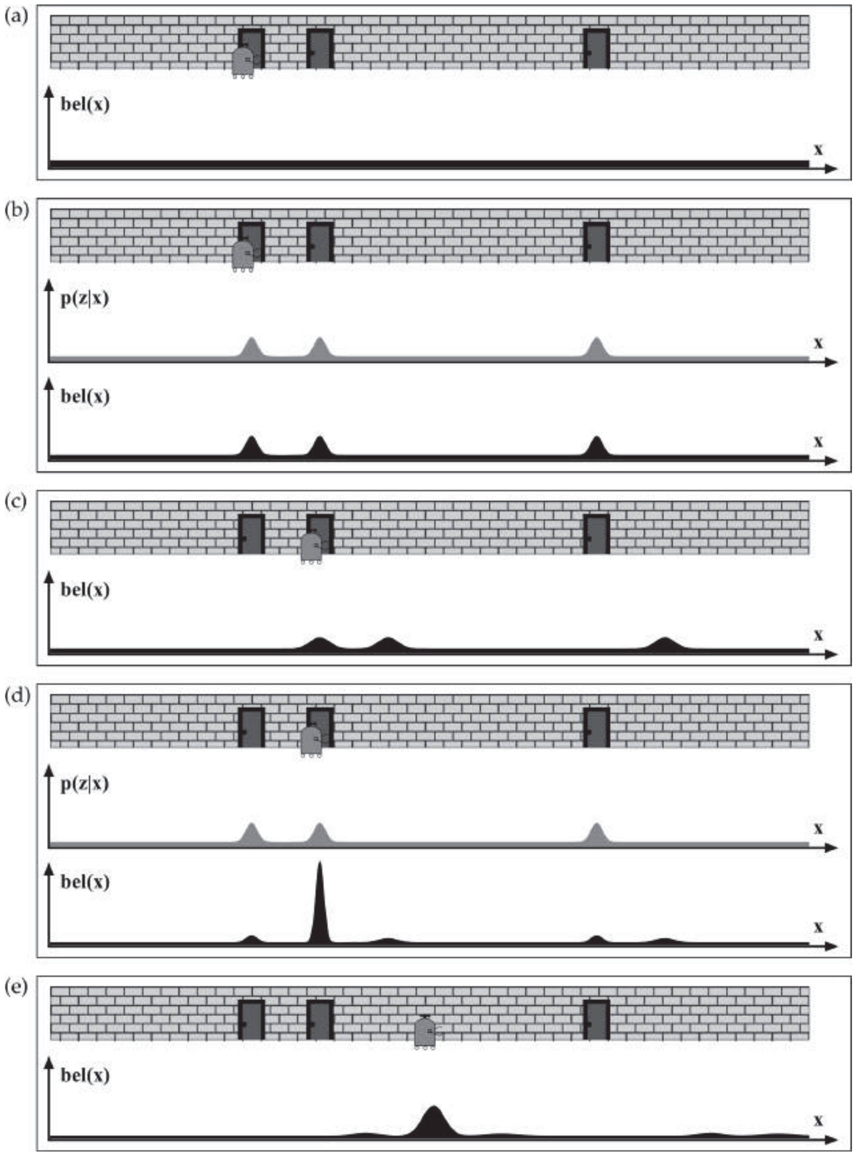


Figura 2.2: A wall follower robot in a hallway. The X-axis represents the states, while the Y-axis represents the measurement probability in gray and the belief in black (Thrun et al., 2005).

filter, and its belief is updated, see figure Figure 2.2 (b). In reality, actions and measurements can happen before or after each other, but here we assume a precise order.

Assuming that the robot observes a door, based on its measurements, the robot places a higher probability near doors and a lower probability near walls. Notice that this distribution has three peaks, each corresponding to one of the world's indistinguishable doors. Thus, by no means does the robot know where it is yet. Instead, it has three, distinct hypotheses which are each equally plausible given the sensor data, so the robot can be in any of these positions. Then, the robot takes another action u_2, which is: move forward. Again this is incorporated into the belief leading to a situation illustrated in Figure 2.2 (c). The distribution is shifted through a *deterministic shift* in the direction of motion. However, a spread on the distribution happens, smoothing each hypothesis. This is due to the introduced uncertainty by the action, meaning that the robot knows that it moved forward, but it cannot perfectly say how much. At its new location, the robot then measures z_2 leading to an updated belief, see Figure 2.2 (d). Note that even though a high probability is now associated with a location, there are other non-zero hypotheses still. The ability to maintain several low-probability hypothesis peaks is essential for attaining robustness.

The probability of the robot being at the second door is higher since the robot has prior knowledge of the three doors' location. It has previously detected that it was in the vicinity of a door, then it moved forward a little. Even though it cannot be sure how much it has moved, it then detects a second door. It knows that there are two doors close to each other and one far ahead, so its belief is that it is at the second door. The robot repeats its cycle of actions and measurements, the state at time t is illustrated at Figure 2.2 (e).

2.3 DATA FUSION

Multisensor data fusion is a technology that enables combining information from several sources in order to form unified information, hopefully, better than the individual ones. Data fusion systems are now widely used in various areas such as sensor networks, robotics, video and image processing, and intelligent system design (Sheridan, 1991).

Dealing with imperfect data is the most challenging problem of data fusion systems, and thus extensive research work has focused on this issue (Sheridan, 1991). There are several mathematical theories available to deal with data imperfection, such as probability theory (Durrant-Whyte and Henderson, 2008), fuzzy set theory (Zadeh, 1965), possibility theory (Zadeh, 1978), rough set theory (Pawlak, 1992), and Dempster-Shafer evidence theory (DSET) (Shafer, 1976). Most of these approaches are capable of representing specific aspect(s) of imperfect data. For example, a probabilistic distribution expresses data uncertainty, fuzzy set theory can represent vagueness of data, and evidential belief theory can represent uncertain as well as ambiguous data.

There is a tendency towards the use of probabilistic localization in mobile robotics, as pointed by Magrin et al. (2019). Furthermore, the Kalman filter is one of the most popular fusion methods mainly due to its simplicity, ease of implementation, and optimality in a mean-squared error sense. It is a very well established data fusion method whose properties are deeply studied and examined both theoretically and in practical applications. However, when dealing with non-linear system dynamics, one usually has to resort to approximation techniques. For instance, the EKF (Welch and Bishop, 1995) and Unscented Kalman Filter (UKF) (Julier and Uhlmann, 1997), which are extensions of the Kalman filter, are applicable to non-linear systems.

Nowadays, learning-based methods are a popular approach for data fusion, especially when dealing with sensors such as cameras, where a single observation can have megabytes of data. Deep Neural Networks (DNNs) have achieved impressive results when dealing with data fusion (Meng et al., 2020). CNNs have been extensively employed as encoders to produce a latent representation that is exponentially lower in size than the raw input. Fully Convolutional Networks (FCNs) encode and infer meaning on data without fully connected layers, see section 2.11. Temporal sequences that are usually handled with a Partially Observable Markov Decision Process (POMDP) can use RNNs, see section 2.13. Semantics and learning-based methods are further discussed at section 2.9.

2.4 HIDDEN MARKOV MODEL (HMM) AND GRAPH-BASED SLAM

A convenient way to describe SLAM is using a HMM, a type of Dynamic Bayesian Network (DBN). A Bayesian network describes a stochastic process as a directed graph. The graph has one vertex for each random variable in the process, and an arc between two vertices models a conditional dependence between them, see Figure 2.3.

The DBN is characterized by a *state transition model* and by an *observation model*, being the transition model $p(x_t|x_{t-1}, u_t)$ and the observation model $p(z_t|x_t, m_t)$ (Grisetti et al., 2010). While a DBN highlight the temporal aspect of SLAM, an alternative representation

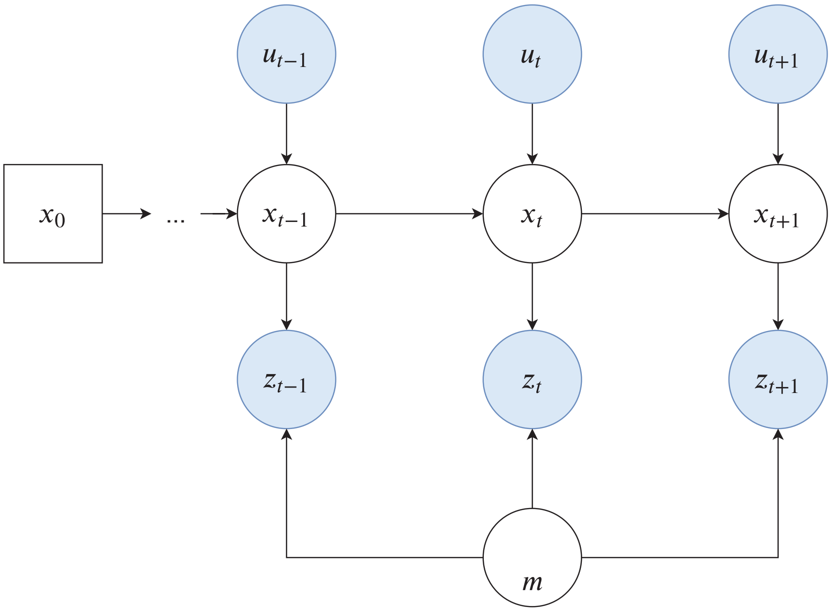


Figura 2.3: The dynamic Bayes network of the SLAM process. Blue vertices indicate the observable variables (here $z_{1:t+1}$ and $u_{1:t+1}$) and white vertices are the hidden variables. The hidden variables $x_{1:t+1}$ and m model the robot's trajectory and the map of the environment, respectively. Adapted from (Grisetti et al., 2010).

so-called graph-based SLAM highlight its underlying spatial structure. The intuitive idea behind this approach is to use a graph whose vertices correspond to the robot's poses at different points in time and whose edges represent constraints between the poses. The latter is obtained from observations of the environment or from movement actions carried out by the robot. Once such a graph is constructed, the map can be computed by finding the vertices' spatial configuration that is most consistent with the measurements modeled by the edges.

In a graph-based SLAM, the problem is divided into two tasks: constructing the graph from the raw measurements (graph construction) and determining the most likely configuration of the poses given the graph's edges (graph optimization). The graph construction is usually called the front-end, while the graph optimization back-end. Examples of Graph optimizers: Tree-based netwORk Optimizer (TORO) (Grisetti et al., 2009), general framework for graph optimization (g2o) (Kümmerle et al., 2011), and GTSAM (Dellaert, 2012). Figure 2.4 illustrates this process.

2.5 REAL-TIME APPEARANCE-BASED MAPPING (RTAB-MAP)

Among the various SLAM methods, we chose the Real-Time Appearance-Based Mapping (RTAB-Map) due to its support of multiple sensors and its policy to ensure real-time constraints. RTAB-Map is a graph-based SLAM method with many incarnations throughout the years. We focus on the version proposed by Labbé and Michaud (2019), which encompasses the previous improvements. It addresses multi-session mapping, and it is based on a global Bayesian loop closure detector, meaning that the detector employs a discrete Bayes filter. The loop closure detector uses a bag-of-words approach, with image features such as ORB (Mur-Artal et al., 2015), to determine how likely a new measurement comes from a previous location or a new location. When a loop closure hypothesis is accepted, a new constraint is added to the map's graph. Then a graph optimizer minimizes the errors in the map. A memory management approach is used to limit the number of locations used for loop closure detection and graph optimization so that real-time constraints on large-scale environments are respected.

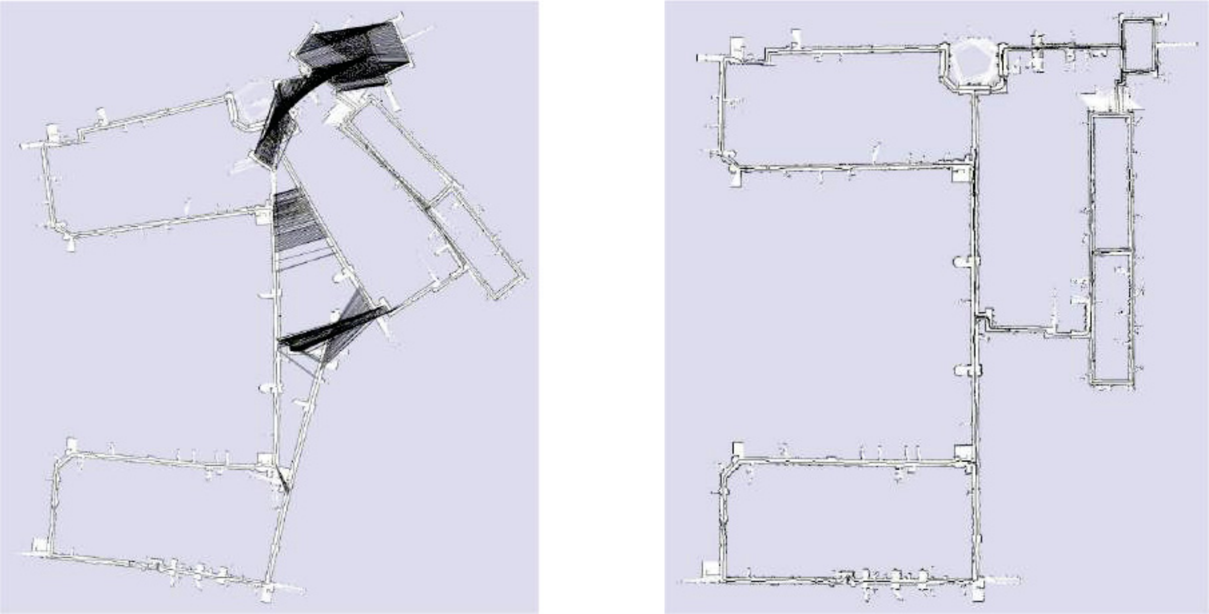


Figura 2.4: Example of graph-based SLAM. On the left is the initial graph created, and on the right is the graph after optimization (Grisetti et al., 2010).

2.6 KINEMATIC MODELS

A kinematic model describes a robot's motion in mathematical form without considering the forces that affect motion and concerns itself with the geometric relationship between elements (Hoy et al., 2015).

Many types of vehicles must operate in unknown environments, such as ground vehicles, Unmanned Aerial Vehicles (UAVs), surface vessels, and underwater vehicles. Most vehicles can be generally categorized into two types of kinematic models *holonomic* and *nonholonomic*, while the latter can be further divided into *unicycle*, and *bicycle* (Hoy et al., 2015). We use the term kinematic to describe models based on more abstracted control inputs.

Holonomic kinematics: These describe vehicles that have control capability in any direction. Holonomic kinematics are encountered on helicopters and certain types of wheeled robots equipped with omnidirectional wheels. Holonomic motion models have no notion of body orientation for path planning purposes, and only the Cartesian coordinates are considered. However, orientation may become a consideration when applying the resulting navigation law to real vehicles, though this is decoupled from planning.

Unicycle kinematics: These describe vehicles that are associated with a particular angular orientation, which determines the direction of the velocity vector. A turning rate constraint limits changes to the orientation. Unicycle models can be used to describe various types of vehicles, such as differential drive wheeled mobile robots and fixed-wing aircraft.

Bicycle kinematics: These describe a car-like vehicle with a steerable front wheel separated from a fixed rear wheel. Kinematically this implies that the maximum turning rate is proportional to the vehicle's speed. This places an absolute bound on the curvature of any path the vehicle may follow regardless of speed. This constraint necessitates high-order planning to navigate in confined environments.

2.7 MOTION PLANNING

Since an agent abides by physical limitations, arbitrary translations do not happen instantly, i.e., no teleportation. A motion planning module is then necessary to find possible paths between points and choose one of them. In this work, we explored two different approaches for motion planning, model-based and learning-based. We first discuss the classical approach, which is model-based, and the learning-based approach is discussed in Section 3.3.

In the classical approach, we first perform SLAM using the agent's sensors data to create an occupancy map for the environment and estimate the agent's current pose in it. This map can be modeled as a graph. This allows us to model the shortest unobstructed path between the agent and a point goal as a graph search for the optimal sequence of vertices transitions. This kind of path planner is here referred to as a global planner, and graph search algorithms like Dijkstra's algorithm (Dijkstra, 1959), A* (Hart et al., 1968), D* Lite (Koenig and Likhachev, 2005), and fast marching (Garrido et al., 2006) can be used.

While a global planner abstracts the agent's constraints and provides a high-level path as a series of vertices transitions, a local planner considers the effect of the robot's mass and velocity. For example, a sharp turn that, from a global perspective, is the optimal path may not be physically feasible because of the robot's inertia. Thus, the local planner's concern is not to find the globally optimal path but rather a local optimal and doable one within the robot's sensing range. Since the control and planning issues that appear in this model seem quite similar to those faced by a jogger in the city environment, this trajectory planning using only sensor information was initially termed the *Jogger's Problem* (Alvarez et al., 1998; Lumelsky and Shkel, 1995).

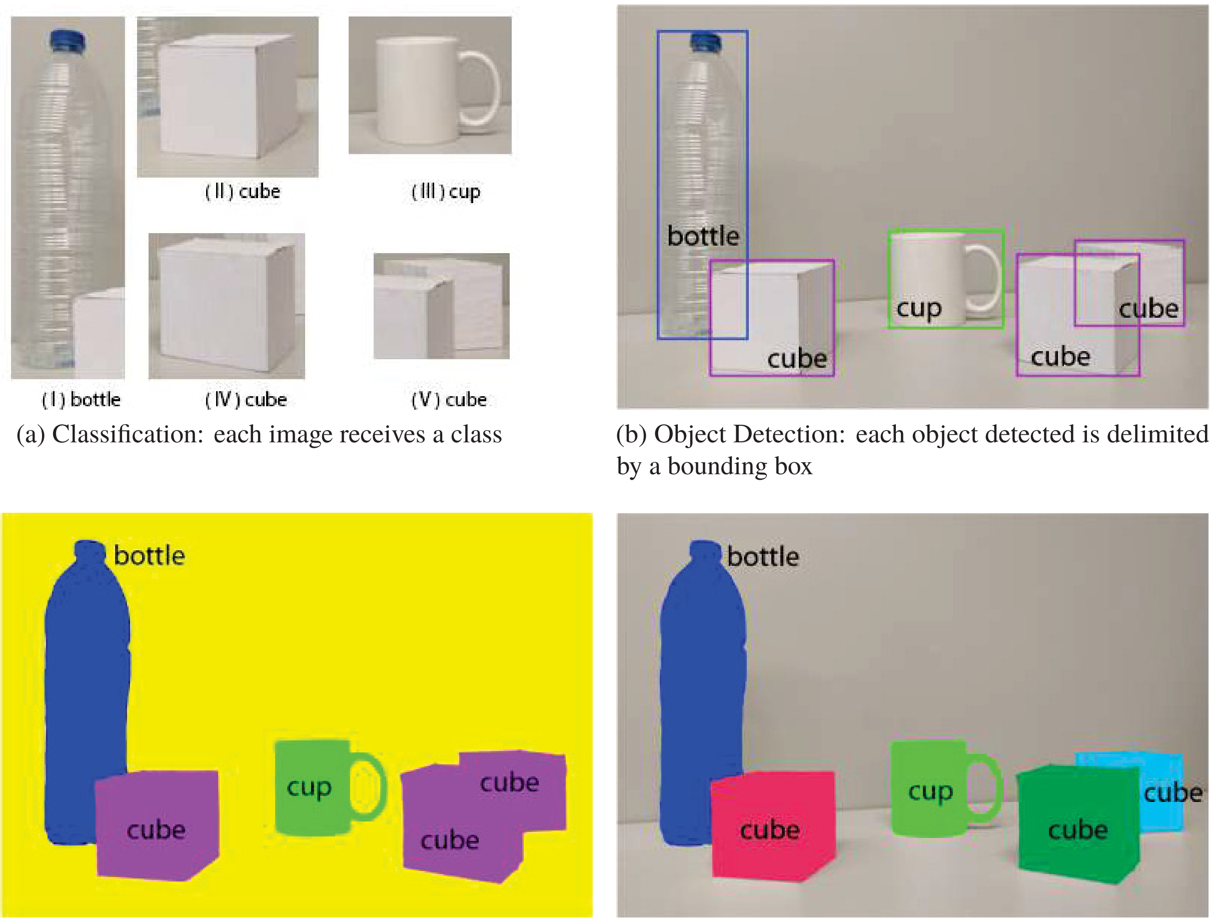
In essence, the local planner computes a series of feasible velocities throughout time to move the agent roughly along the path proposed by the global planner while avoiding collisions. Collision avoidance happens under the assumption that the agent moves at a relatively low speed and can fully stop within the sensoring range. Examples of local planners are the Dynamic Window Approach (DWA) (Fox et al., 1997; Seder et al., 2005), and Timed-Elastic Band (TEB) (Rösmann et al., 2012).

2.8 SCENE UNDERSTANDING

Human beings receive a large amount of data through their senses. Nevertheless, the human brain learns how to handle all this data. For example, it is capable of recognizing objects in a scene by their format, food by its smell and taste, and other people by their voice (Tripathi, 2017). Machine Learning is a subfield of Artificial Intelligence that aims to make computers learn from data without the need to be explicitly programmed, i.e., data-driven instead of model-based. Machine learning attempts to model the learning process taking inspiration from how humans learn (Watt et al., 2020).

Scene understanding is one of the computer vision tasks that got a major performance boost with state-of-the-art machine learning (Schmidhuber, 2015). The scene understanding problem can be divided into classification, detection, semantic segmentation, and instance segmentation problems (Garcia-Garcia et al., 2017).

The classification is the ability to distinguish different types of objects (Watt et al., 2020). Figure 2.5a presents a classification example. Each image receives a class (classification). The detection aims to find Region of Interests (ROIs) where the objects are located, and those regions are delimited by a set of points containing at least a part of each object detected, as presented in Figure 2.5b. The semantic segmentation aims to cluster regions in the image that belongs to the same object (Thoma, 2016). In semantic segmentation, every pixel in the image



(a) Classification: each image receives a class

(b) Object Detection: each object detected is delimited by a bounding box

(c) Semantic Segmentation: each pixel in the image receives a label

(d) Instance Segmentation: each pixel of a different instance receives a label

Figura 2.5: Differences between image classification, object classification, semantic segmentation, and instance segmentation problems. Subfigures (b) and (c) share the semantic color labels, blue for the bottle, magenta for the cube, and green for the cup. In Subfigures (b), the pixels classified as background are yellow. In Subfigure (d), the colors represent instances instead of the semantic classes, dark blue for the only bottle, bright green for the only cup, and red, dark green, and light blue for the cubes. Adapted from (Garcia-Garcia et al., 2017).

receives a classification according to its category. Figure 2.5c presents a segmentation example with the scene segmented in four categories: cup, cube, bottle, and background. The semantic segmentation does not differentiate object instances. The instance segmentation is a step further from semantic segmentation and aims to identify the different instances of objects, giving a label (classification) to each segmented object (Garcia-Garcia et al., 2017). Different from the semantic segmentation example presented in Figure 2.5c, each object instance receive a different color to represent instance segmentation in Figure 2.5d. The classification problem and its derivatives (detection, semantic segmentation, and instance segmentation) can be divided into a pipeline of three steps: collecting data; designing features if it is model-based or training if it is data-driven, and testing (Watt et al., 2020).

2.9 SEMANTICS AND LEARNING-BASED METHODS

In recent years, there is a growing interest in adding high-level information to several robotics applications to achieve more capable robots. By employing methods with higher generalization, the robot can better react to events that it has never seen before. Following this trend, the mobile robotics field is starting to include semantic information in navigation tasks, leading to a new concept: **semantic navigation**. This type of navigation brings closer the human way of understanding the environment with how the robots understand it (Crespo et al., 2020).

Typically, to extract useful information from a raw sensor input, feature extraction is performed, and later on, pattern recognition is performed in the extracted features. Previously this required a lot of feature engineering from a software developer. This refers to what is known today as **handcrafted features**, features descriptors such as ORB (Rublee et al., 2011) fall into this category. Machine learning proposes to learn the useful patterns based on the data itself instead of using a handcrafted model.

Deep Learning is a type of machine learning algorithm that employs deep neural networks as feature extractors, classifiers, and regressors that attempt to perform various tasks by learning from examples (Watt et al., 2020). Recently, deep learning-based approaches have won many machine learning competitions and challenges, even achieving superhuman visual results in some domains (Schmidhuber, 2015). Some research areas in which such approaches have considerably outperformed traditional ones – based on handcrafted features – are license plate recognition (Laroca et al., 2018, 2021), image-based automatic meter reading (Laroca et al., 2019, 2020; Salomon et al., 2020b), neural architecture search (Alves and d. Oliveira, 2020), segmentation of medical images (Alves et al., 2018), and biometrics (Zanlorensi et al., 2018; Lucio et al., 2019; Salomon et al., 2020a). Learning-based methods are usually divided in **supervised learning**, **unsupervised learning**, and **reinforcement learning** (Meng et al., 2020).

In this work, we consider **semantic understanding** the ability to infer meaning and context between individual pieces of data and reach an interpretation of the data by exploiting previous knowledge. Semantic understanding is closely related to scene understanding, where the latter focuses on determining what is in a scene and where, and the former focuses on the relations between components in a scene. We illustrate this concept with a short example.

Imagine an agent that receives a digital image as input. The agent can, via computation, determine that three distinct cabinets, a television, a wooden floor, and a wall, lie within the image. This step is scene understanding. Because of these objects and their spatial arrangement, the agent concludes that the image is of a living room. In this case, the agent has some semantic understanding since it used its previous knowledge that images contain objects to infer that some pixels form an object, each object is distinct and not a single entity due to factors like spatial placement and color, it can infer that if there are a wall and a floor, then it could be a room and so on. By inferring meaning and context, then forming an interpretation, it applies its knowledge to understand the visual observation.

A naive solution to obtain a crude semantic understanding would be simply including an object detection algorithm in the agent's navigation pipeline. If the agent sees an object of the target class, it should mark the object's location and navigate to it. After all, object detection achieved impressive results with deep learning, and this approach has been used by works such as (Reis et al., 2019).

Nevertheless, why would this solution be naive? The answer is not so simple. In a short distance where the task has a short-horizon, the solution can solve the task, but in reality, the agent will not always start close to the target object. For instance, if the agent starts in a faraway room from any of the desired objects, this room could be a bathroom, and the agent could be searching for a laptop. None of the detected objects in this room will be of any help if they are merely detected. So, the agent should employ some exploration heuristic until he detects one of the desired objects? It would be better than simply coupling object detection, but it would be akin to a brute force search. Data was gathered, analyzed, and classified about the environment but not exploited in the search.

The missing component in this solution is **semantic reasoning**, a way to exploit the semantic understanding to take action. Semantic-based reasoning is a sub-area of the vast area of Knowledge Representation and Reasoning in the field of Artificial Intelligence that deals with solving complex problems, like having a dialogue in natural language or inferring a person's mood (Cadena et al., 2016). A model-based approach for semantic reasoning would employ ontologies, but in this work, we will focus on learning-based methods.

2.10 CONFLUENCE OF DEEP LEARNING AND MOBILE ROBOTS

We talked about how semantics are particularly useful for agents and how learning-based methods are more promising than model-based ones when dealing with semantics. This raises an important question: how those concepts should be integrated? In this section, we briefly discuss how deep learning has been introduced into localization and mapping by several researchers.

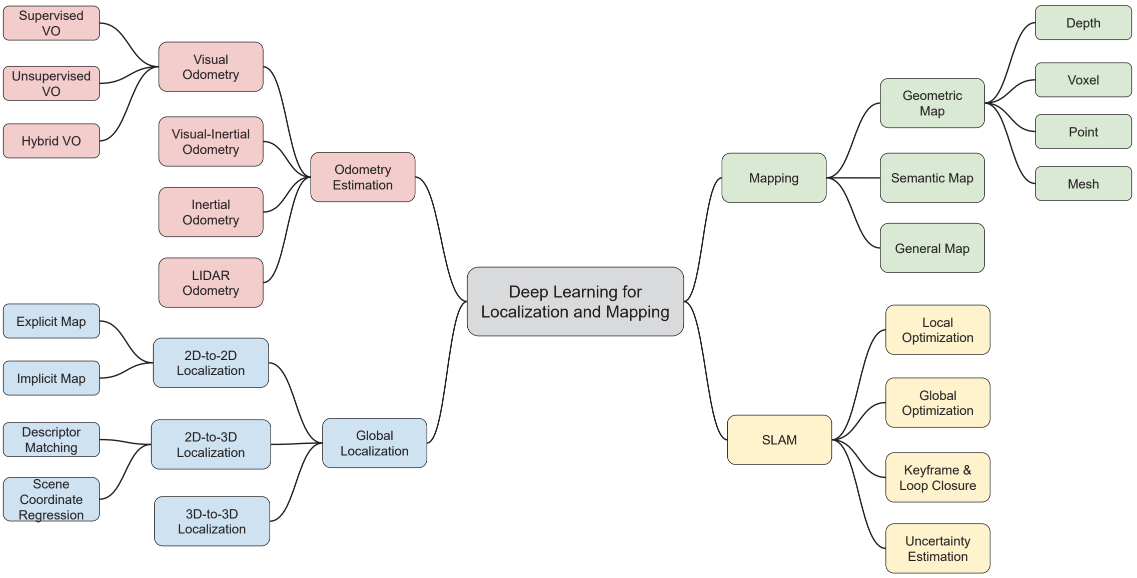


Figura 2.6: Taxonomy ramifications of Deep learning for Localization and Mapping. VO stands for Visual Odometry. Adapted from (Chen et al., 2020).

Chen et al. (2020) provided an extensive survey on this subject and proposed a taxonomy for such, see figure 2.6. They divided the integration into four broad areas: odometry estimation, mapping, global localization, and SLAM. Figure 2.7 illustrates how those areas are integrated on what Chen et al. (2020) call a spatial machine intelligence system.

Odometry estimation, especially Visual Odometry and LiDAR Odometry, are some the most popular areas to employ Deep Learning (Saeedi et al., 2018; Caminal et al., 2018; Patel et al., 2019), which consists of replacing handcrafted features with features encoded by a CNN. Mapping includes works on 3D reconstruction, depth estimation, and semantics. Pure SLAM includes NeuralSLAM (Zhang et al., 2017). A proper discussion of the related works is in chapter 3. For a discussion about end-to-end Deep Reinforcement Learning (Deep-RL) agents that perform SLAM, semantic reasoning, and navigation see section 3.3.

2.11 CONVOLUTIONAL NEURAL NETWORK (CNN)

CNNs were originally proposed by LeCun et al. (1998), expanding on the successful use of backpropagation on multi-layer neural networks. However, only with AlexNet (Krizhevsky

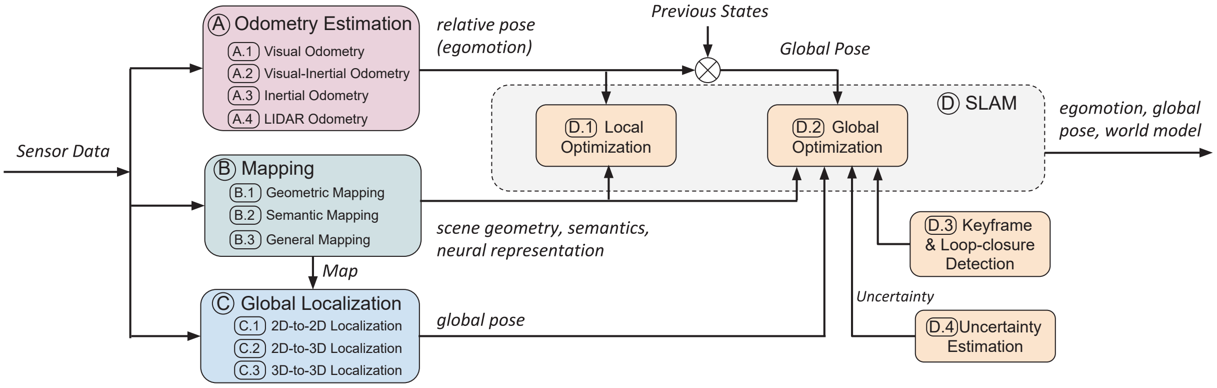


Figura 2.7: High-level conceptual illustration of a spatial machine intelligence system. Adapted from (Chen et al., 2020).

et al., 2012), convolutional neural networks regained popularity. As several subsequential improvements were proposed by many authors its renown grew, causing the favored Support Vector Machine (SVM) to fell in popularity. Advancements on CNNs led to a plethora of complex architectures known as backbones, a.k.a feature extracting networks, such as VGG (Simonyan and Zisserman, 2014), Residual Network (ResNet) (He et al., 2016), DarkNet (Redmon and Farhadi, 2017), and MobileNet (Howard et al., 2017). Figure 2.8 presents an illustration of how typically a CNN is composed for classification tasks.

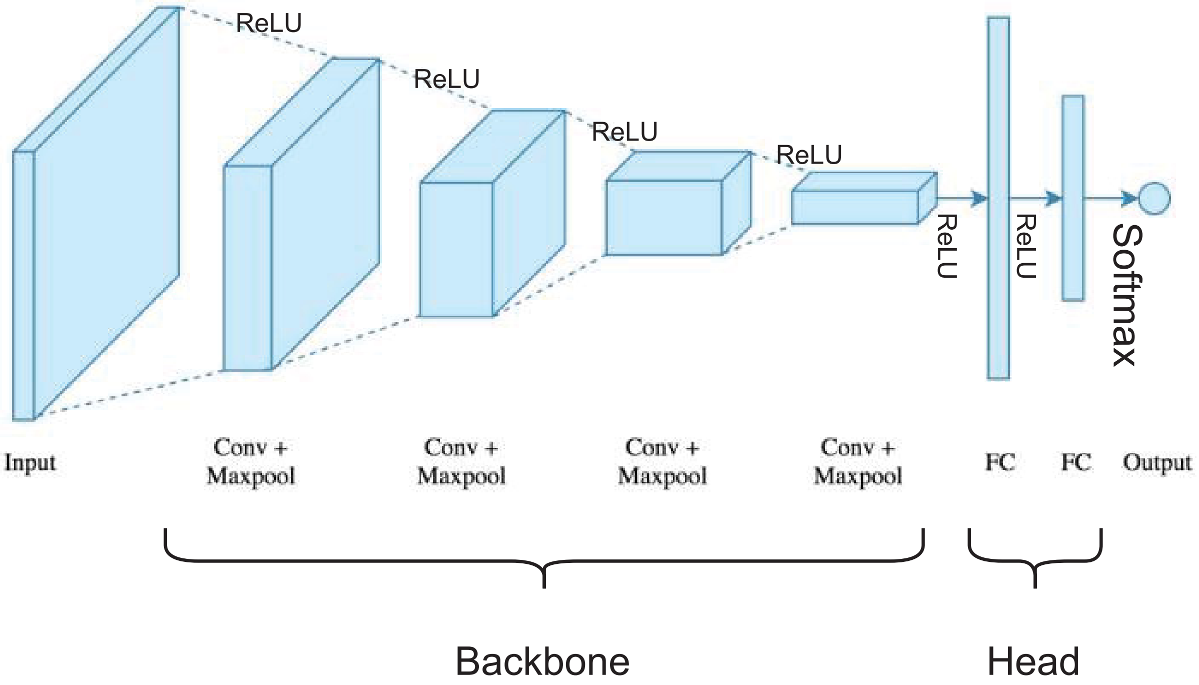


Figura 2.8: Illustration of a Simple CNN. FC stands for Fully connected layer. Where the backbone is responsible for feature extraction, i.e., encoding the input to a latent representation, and the head typically performs classification or regression. ReLU is the activation function in this example. Maxpool stands for Max Pooling, a type of pooling operation. Conv stands for Convolution. Softmax is a normalization function, typically used when performing classification, where the desired output vector should sum to 1.

A common term within machine learning is latent space representation or just latent representation. It refers to an encoded array with a smaller number of dimensions than the original data but that still holds all the important information.

Each convolutional layer comprises a set of independent filters and their biases. The bias shifts the decision boundary away from the origin and does not depend on any input value. The bias is an important concept inherited from the Perceptron that is the basis for any artificial neural network. Those filters, which are used to perform convolutions with the input, have trainable weights that are updated together with their biases during training via backpropagation. The number of stacked filters is known as *Depth*. The output of a convolutional layer is referred to as *feature map*.

Since convolutions are linear operations, activation functions are used such as Sigmoid, Hyperbolic Tangent (Tanh), ReLu and Leaky ReLU to ensure non-linearity, which is essential to handle complex problems where there is not a guarantee of a linear solution. Sigmoid is defined as $\sigma(x) = 1/(1 + e^{-x})$ and normalize to the [0,1] interval. Hyperbolic Tangent (Tanh) can also used as an activation but causes normalization on the [-1,1] range. The Rectified Linear Unit (ReLU) is defined as $f(x) = max(0, x)$. In other words, the activation is simply threshold at zero. Leaky ReLU is defined as $f(x) = max(0.1x, x)$.

Pooling: Its function is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network. A Pooling layer operates on each feature map independently. The most common approach used in pooling is max pooling. Other variations exist, such as average pooling. Pooling can also be replaced by more convolutional layers, known as Depthwise Separable Convolutions (Sandler et al., 2018).

Padding: the addition of filler values on the perimeter of the input, those values are typically zeros and are added to preserve pixels at the margins. We refer to the number of rows and columns traversed per slide as the *Stride*. Multilayer Perceptron (MLP) refers to a feedforward network made of fully connected layers. That is, each neuron in one layer is connected to all neurons in the next layer, constituting a complete directed graph.

2.11.1 Head

In Deep Neural Networks (DNNs) the region that produces the output is called the head. A DNN can have multiple heads, as it is typically the case when dealing with multitasking. In general, classification is done using a MLP, which is essentially a series of fully connected layers. Long et al. (2015) proposed a FCN architecture for semantic segmentation and demonstrated that other architectures could be converted to fully convolutional as well. The main difference of FCN to conventional DNNs is that the head does not use fully connected layers. Instead, it uses convolutional layers like in the encoding part. An example of DNN with multiple heads is the work of Nekrasov et al. (2019), see figure 3.3.

Although it is possible to train a backbone from scratch, i.e., starting with random weights, it is common to perform a pre-train and reuse some learned weights as the starting point. By starting from learned weights, the convergence happens in fewer iterations than from scratch. To pre-train the backbone, a head is attached at the end to perform a broad task such as image classification on the ImageNet (Deng et al., 2009), since it has 1000 classes and millions of examples. The head and backbone learn together via backpropagation of the calculated loss based on the loss function. The loss function depends on the problem a regression problem could employ Root Mean Square Error (RMSE) and a classification one Cross-entropy loss. After the initial training, it is common to replace the head with one fitting the actual desired task and perform transfer learning, a type of fine-tuning to a new task. The reusability of a trained model is a coveted property of CNNs.

2.11.2 Residual Blocks

He et al. (2016) introduced the concept of residual blocks, the core idea behind the ResNet architecture, a macrostructure that contains multiple convolutional layers within. The main idea is that the resulting values of shallower layers are summed with the result of deeper layers and thus propagated throughout the layers. Figure 2.9 illustrates the residual blocks.

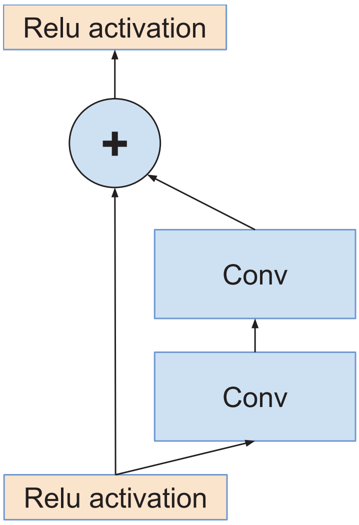


Figura 2.9: Residual Block. Note how previous information bypasses two convolutional stages and is summed with the results of sequential convolutions (He et al., 2016).

2.12 ENCODER-DECODER

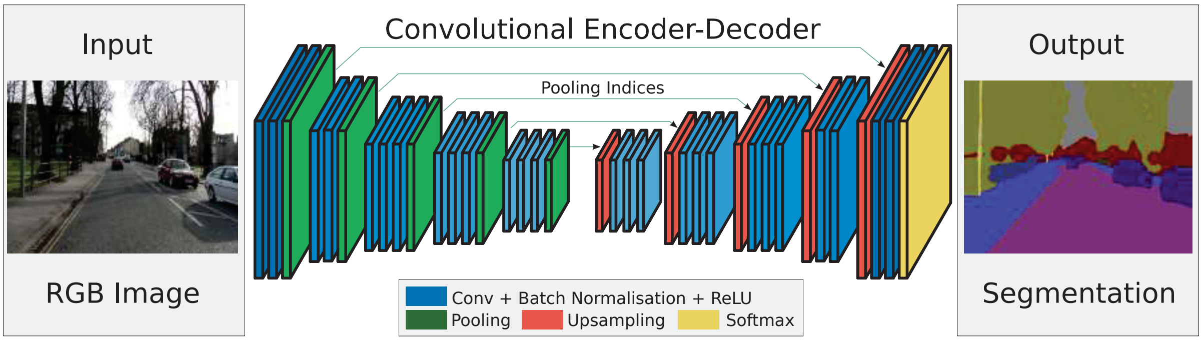


Figura 2.10: An illustration of the SegNet architecture, a example of encoder-decoder. There are no fully connected layers, and hence it is only a convolutional structure. The encoder summarizes the input data to latent representation. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification (Badrinarayanan et al., 2017).

The idea behind the encoder-decoder architecture for DNNs is that the encoder part reduces the input dimensionality to a latent representation, then this representation is passed to a decoder that creates the output. Each module can be trained separately. Different decoders can use the same latent representation created by an encoder. An Autoencoder is a particular type of encoder-decoder that is usually employed in unsupervised learning. If linear activations functions are used, or only a single sigmoid hidden layer, then the optimal solution to an autoencoder is strongly related to principal component analysis (PCA) (Bourlard and Kamp, 1988). Figure 2.10 presents SegNet (Badrinarayanan et al., 2017), an example of encoder-decoder.

2.13 RECURRENT NEURAL NETWORK (RNN)

A shortcoming of traditional neural networks is that they lack a mechanism to retain temporal information. To address this issue, RNNs were designed to work with temporal sequences. They are networks with loops in them, allowing information to persist, see figure 2.11. A stands for a RNN cell, x_t is a input at time t, and h_t the output hidden state at time t.

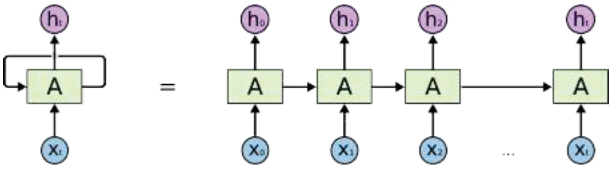


Figura 2.11: An unrolled recurrent neural network. A stands for a RNN cell, x_t is a input at time t, and h_t the output hidden state at time t (Olah, 2015).

Long Short-Term Memorys (LSTMs) were originally proposed by Hochreiter and Schmidhuber (1997) as a type of RNN. They are explicitly designed to avoid the long-term dependency problem that was common on earlier RNNs. All RNNs have the form of a chain of repeating modules of neural network.

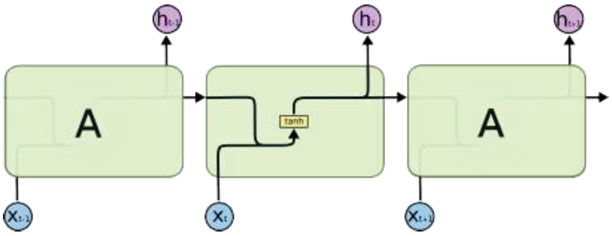


Figura 2.12: The repeating module in a standard RNN contains a single layer, for example, a hyperbolic tangent (tanh) layer (Olah, 2015).

In standard RNNs, those repeating modules will have a very simple structure, such as a single tanh layer, see figure 2.12. LSTMs instead have four layers in each module, see figure 2.13.

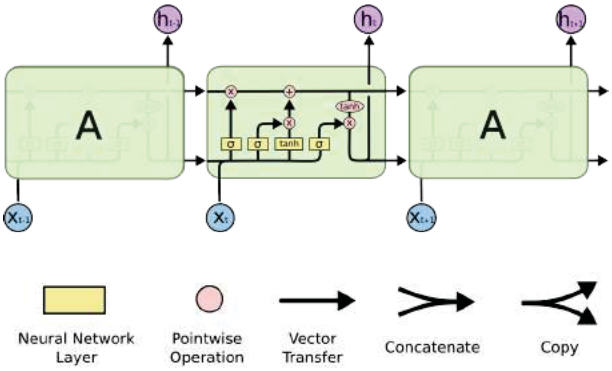


Figura 2.13: LSTM diagram. Each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are trainable neural network layers. Lines merging denote concatenation, while a line forking denotes its content being copied and the copies going to different locations (Olah, 2015).

The core concept of LSTMs is the cell state, denoted by C_t, it can be altered by structures called *gates*, see figure 2.14.

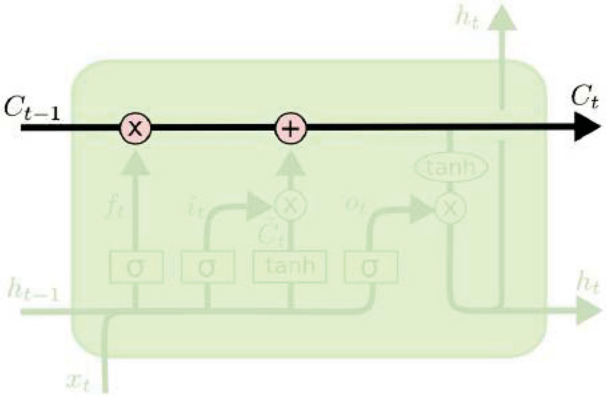


Figura 2.14: Information flow of the cell state within a LSTM. Note how the cell state C_{t-1} is transformed into the C_t (Olah, 2015).

Gates are a way to let information through optionally. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation, see figure 2.15. The output values of the sigmod layer fall into the [0,1] range and act as weights for the pointwise operations with the cell state.

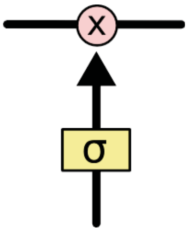
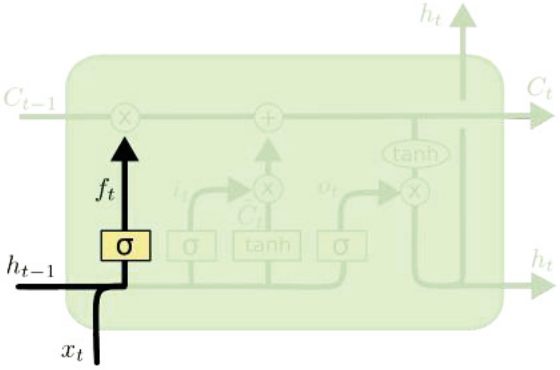


Figura 2.15: Gates within a LSTM (Olah, 2015).

The forget gate layer is responsible for controlling how much should be kept from the previous cell state, see figure 2.16. It looks at h_{t-1} and x_t, and outputs values between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents 'completely keep this' while a 0 represents 'completely get rid of this.'

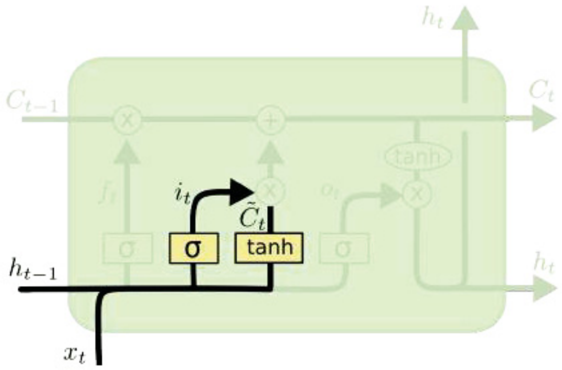


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figura 2.16: Forget Gate Layer. σ stands for the sigmoid function. h stands for the hidden state. x_t stands for the input at time t. W_f stands for the weights of the forget layer. b_f stands for the bias of the forget layer (Olah, 2015).

The input gate layer decides how much new information should be included in the cell state, see figure 2.17. It is composed of two steps. First, a sigmoid layer to determine the weight

i_t and a tahn layer creates a temporary cell state $\widetilde{C}_t$ those are combined in a pointwise operation used to create an update value for the cell state.

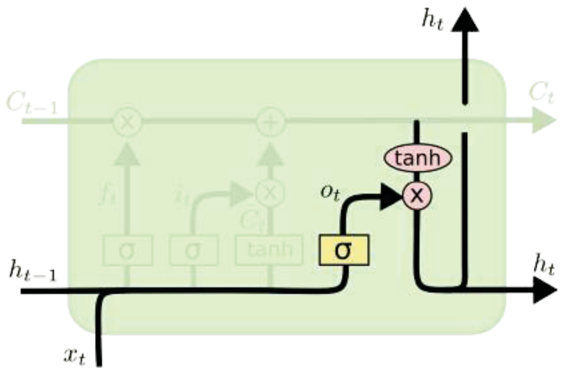


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figura 2.17: Input Gate Layer. (Olah, 2015).

Finally the output gate layer can transform the cell state C_t into the hidden state h_t, see figure 2.18.



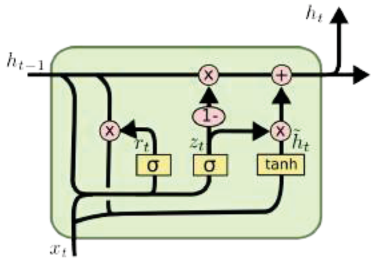
$$o_t = \sigma\left(W_o \; [\, h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Figura 2.18: Output Gate Layer. (Olah, 2015).

The main difference between HMMs and RNNs is that HMMs operate under the complete state assumption, where the past is irrelevant, while RNNs can learn patterns that occur throughout a sequence since they have a mechanism to keep previous information.

Gated Recurrent Units (GRUs) (Cho et al., 2014) combines the forget and input gates into a single ‘update gate.’ It also merges the cell state and hidden state and makes some other changes. The resulting model is simpler than standard LSTM models and has been growing increasingly popular.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figura 2.19: GRU diagram (Olah, 2015).

The LSTM network can be organized into an architecture called the Encoder-Decoder LSTM that allows the model to support variable-length input sequences and predict or output variable-length output sequences. One of the earliest works to use an Encoder-Decoder LSTM was Srivastava et al. (2015). A brief discussion about the issues concerning LSTMs is in section 8.1.

2.14 DECENTRALIZED DISTRIBUTED PROXIMAL POLICY OPTIMIZATION (DD-PPO)

Decentralized Distributed Proximal Policy Optimization (DD-PPO) (Wijmans et al., 2019) is an extension of the Proximal Policy Optimization (PPO) (Schulman et al., 2017). Both methods were designed to be trained using Deep-RL. The idea is to have a policy net as the backbone and a head layer that acts as the critic. The policy net has a separate encoder for each sensor. RGB-D can use variations of the ResNet as an encoder. The latent representation of each sensor is concatenated and fed to a RNN, the RNN encode temporal information and state. The output of the policy net is fed to the critic head that decides the final predicted action. In this work, we used the implementation and weights publicly available with the Habitat Simulator.

2.15 EMBODIED AI NAVIGATION

Navigation in three-dimensional environments is an essential capability for an embodied agent to function in the physical world. Animals, including humans, can traverse cluttered dynamic environments efficiently and deliberately in pursuit of distant goals. This ability holds true even in previously unseen environments (Anderson et al., 2018a).

To produce an agent with a similar ability, a hierarchy of sub-problems must be addressed first. Imagine walking up to a home robot and asking, ‘Can you check if my key is on my desk? Moreover, if so, bring it to me.’ In order to be successful, such a robot would need a wide range of skills — **visual perception** and **classification**, to observe and determine if the image contain an object of interest; **language understanding**, to break down a natural language sentence into subject, action and additional linguistic components and its relations within the context; **episodic memory** construction (what is where?); **reasoning**, **planning**, **navigation**, and so on (Batra et al., 2020; Sun et al., 2017). Note that beyond the computational sub-problems, a real mobile robot has additional problems related to electrical and mechanical engineering.

A plethora of sub-problems must be solved to complete this particular task, and some form of hierarchy exists between them. Instead of solving the whole problem of finding a key on a desk, grabbing it, and bring it to a dynamic target that is highly complex, let us focus on the computational perspective of searching for and navigating to an object (‘find key’), an embodied navigation task known as Object-Goal Navigation (ObjectNav) (Anderson et al., 2018a), sometimes called Visual Semantic Navigation (Yang et al., 2018; Deitke et al., 2020).

A fundamental challenge in robotics, artificial intelligence, and language processing is a sequential prediction: to reason, plan, and make a sequence of predictions or decisions to minimize accumulated cost, achieve a long-term goal, or optimize for a loss acquired only after many predictions (Sun et al., 2017).

Since navigation has received a surge of scientific research recently, Anderson et al. (2018a) categorized embodied navigation tasks via the description of the navigation goal:

- PointGoal: The agent must direct itself to a given nonvisual target destination specified using coordinates a point in space, e.g., $(x, y, z)^\top$, avoiding obstacles and walls as it navigates.

- ObjectGoal: Similarly, the agent must avoid obstacles and reach a semantically distinct object instance, e.g., ‘desk.’

- AreaGoal: The agent must avoid obstacles and reach a semantically distinct area, e.g., ‘bedroom.’

This effort also resulted in a set of recommendations, such as the *Success weighted by inverse Path Length (SPL)* metric for evaluation and the use of a ‘stop’ action by agents to indicate completion of an episode.

The combination of deep learning and autonomous navigation is a growing research field (Crespo et al., 2020) with methods employing different strategies, supervised learning/imitation learning, unsupervised learning, reinforcement learning (Weihs et al., 2020). Those methods aim to solve increasing abstract tasks, such as Image target navigation, Target-driven visual navigation, Embodied Question Answering, and Vision-and-Language Navigation.

Image target navigation, where instead of a point, an image of the goal is provided, the agent must reach a position where its viewpoint matches the goal. Target-driven visual navigation (e.g., ObjectNav and AreaNav), instead of a target image, a target class or instance is provided as a goal, not the point location of it but its class/instance label.

Embodied Question Answering (EmbodiedQA), a variation of Visual Question Answering (VQA)— where an agent is spawned at a random location in a 3D environment and asked a question (‘What color is the car?’). In order to answer, the agent must first intelligently navigate to explore the environment, gather necessary visual information through first-person (egocentric) vision, and then answer the question (Das et al., 2018).

Vision-and-Language Navigation (VLN) where it requires turning relatively general natural-language instructions into robot agent actions on the basis of the visible environment. This requires extracting value from two very different types of natural-language information. The first is object description, e.g., ‘table,’ ‘door,’ each presenting as a tip for the agent to determine the next action by finding the item visible in the environment, and the second is action specification, e.g., ‘go straight,’ ‘turn left,’ which allows the robot to directly predict the next movements without relying on visual perceptions (Anderson et al., 2018b). Figure 2.20 illustrates the difference between VLN and VQA.

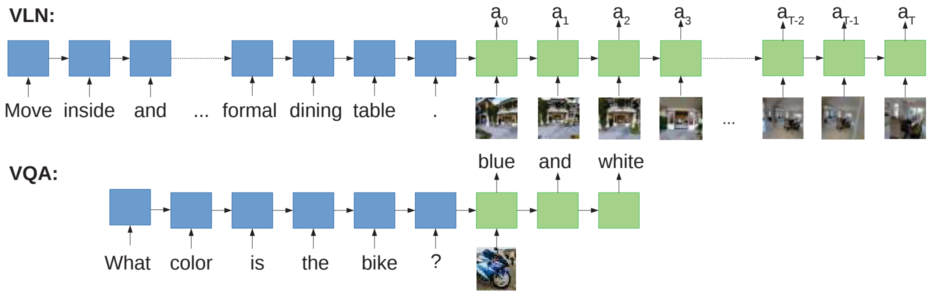


Figura 2.20: Differences between Vision-and-Language Navigation (VLN) and Visual Question Answering (VQA). Both tasks can be formulated as visually grounded sequence-to-sequence transcoding problems. However, VLN sequences are much longer, and some of the words produce cues for exploration. In blue are the individual words and their intermediary representation forming the input sequence. In green are the sequence of output actions $\langle a_0, a_1, \dots a_T \rangle$ given a language sequence and visual observations (Anderson et al., 2018b).

2.16 OBJECTNAV TASK DEFINITION

ObjectNav is defined as the task of navigating to an object (specified by its category label) in an unexplored environment. In particular, the agent is initialized at a random **pose**— A pose is composed of location and orientation— in an environment and asked to find an instance of an object category, e.g.,'chair' by navigating to it (Batra et al., 2020). In this context, the sequence of actions, observations, and intermediary states throughout time denoted by an initial state, and a final one is called **episode**, see Figure 2.21. The blue arrow in Figure 2.21 shows the shortest path from the agent's (random) starting pose and the closest point in the success zone, according to a path-planning oracle. According to Batra et al. (2020), three particular descriptions are fundamental and should be provided:

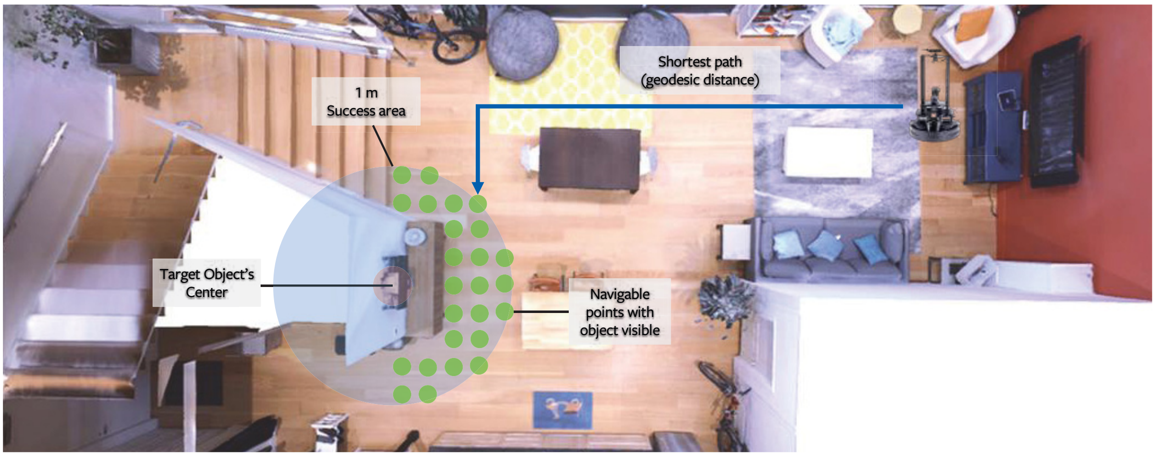


Figura 2.21: ObjectNav Evaluation: The goal object, in this case, a monitor, is highlighted in pink. All points within 1m to the object surface (highlighted as a blue circle) are considered potential success goal poses. Only navigable locations from where the object can be seen with proper orientation of the camera (highlighted in green) are considered from this area. These points cumulatively form the 'success zone,' locations where the episode would be considered successful if the agent called 'stop.' The blue arrows show the shortest path from the agent's (random) starting pose and the closest point in the success zone as predicted by a path planning oracle (Batra et al., 2020).

- **Object Finding and Evaluation:** Why are we trying to find objects, and what constitutes successful navigation? See section 2.16.1.

- **Embodiment:** What is the agent specification, in terms of actions and observations? See section 5.1.

- **Environments:** What environments (object categories, scenes) should be used? See chapter 4.

2.16.1 Object Finding and Evaluation

It is crucial to start with the question — why is the agent trying to find objects? Finding then navigating to objects can be seen as the first step to interacting with objects. It is a fundamental perception and navigation milestone that forms a core building block of any intelligent embodied agent interacting with the world (opening containers, picking up objects, opening doors along the way, clearing obstacles out of a navigation path, responding to questions (Das et al., 2018; Gordon et al., 2018; Anderson et al., 2018b; Batra et al., 2020)).

To perform an evaluation disentangled from any specific downstream task, specific evaluation protocols were used. They measure the agent's ability to navigate to the object and not the ability of object detection or framing of the object in the agent's view at the end of the episode. Thus, according to to Batra et al. (2020) an agent has successfully navigated to the goal object if all of the following conditions are satisfied:

- **Intentionality**: The agent emits a STOP action at a location, indicating that it 'believes' it has reached the object.
- **Validity**: The agent is at a navigable location given the agent's dimensions and the environmental constraints (true by construction during actual navigation but not necessarily true for a randomly sampled point on the mesh).
- **Proximity**: The agent's location, represented as a point, e.g., center of mass, is within a Euclidean distance shell around the object surface. The exact size of this shell will vary depending on the size of the robot and its reach; a reasonable starting point may be $r = 1$m, which may be tightened over the years as the community makes progress on the task. The object surface may be approximated by an axis-aligned or oriented 3D bounding box for computation efficacy.
- **Visibility**: The object is within the field of view of the agent. The exact specification of 'within' will depend on the agent's sensor specification (field of view, pose), action space (whether direct changes to the camera roll/pitch/yaw are allowed), and the kinds of objects being considered (small table-top objects or larger furniture objects). However, true visibility could be ignored for the sake of simplicity and task disentanglement, resulting in a case where the agent does not actually need to be viewing the object at the stopping location and operate under **oracle-visibility** – i.e., assume access to an oracle that is able to optimally frame the object in the agent's camera view without changing the agent's location – i.e., by in-place turning the agent, making changes to roll/pitch/yaw of the camera. In this setting, the agent's goal is to reach locations from where the goal object **can** be viewed, but not necessarily visible from the pose the agent stops in. Oracle-visibility is a proxy for 'the agent is close enough to interact with the object, and the navigation module may now hand control over to an interaction module.' This criterion is more suitable for scenarios that the focus is on pathfinding instead of both pathfinding and object framing.

2.17 METRICS FOR OBJECTNAV EVALUATION

Determining if an episode was successful is not enough to evaluate the planner's efficiency. Thus, additional metrics to the success rate should be employed, such as the SPL. ObjectNav-SPL may be defined analogous to PointNav-SPL (Anderson et al., 2018a). The key difference is that the shortest path is computed to the object instance closest to the agent start location. Thus, if an agent spawns close to '$chair_1$' but stops at a distant '$chair_2$', it will succeed (because it found a '$chair$'), but with a low SPL (because the agent path is much longer compared to the oracle path).

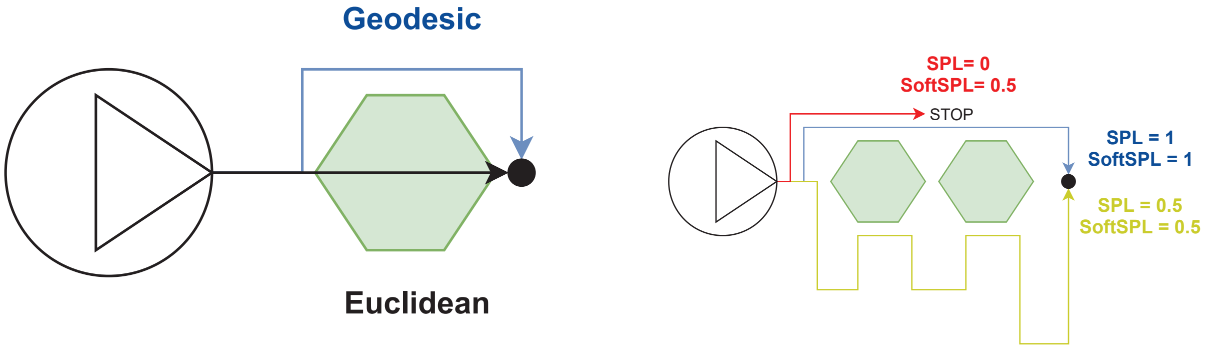
The formal definition of ObjectNav-SPL is as follows: for the i^{th} episode, S_i denotes navigation success. $S_i = 1$ if and only if the agent has succeeded, otherwise $S_i = 0$, the criteria for success was defined in Section 2.16.1. l_i denotes the length of the shortest path to the closest instance of the object, and p_i denotes the length of the actual path taken by the agent. Those lengths are typically measured in geodesic distance and the shortest path computed by a path planning oracle. The metric is defined by Equation 2.6. The geodesic distance is the shortest path between two points constrained by a navigable surface, see figure 2.22.

$$\text{SPL}_i = S_i \cdot \frac{l_i}{\max(p_i, l_i)} \tag{2.6}$$

The final evaluation metric is the average over all episodes: SPL $= (1/N)\sum_i \text{SPL}_i$. Since SPL proposal, some critical issues with the SPL metric have been found, more details on Section 2.17.1. To counter some of those issues, Datta et al. (2020) proposed a metric called SoftSPL. Let d_{init} and d_T denote the geodesic distances to target upon episode start and termination, respectively. The SoftSPL for an episode is defined in Equation 2.7.

$$\text{SoftSPL}_i = (1 - \frac{d_T}{d_{init}}) \cdot \frac{l_i}{\max(p_i, l_i)} \tag{2.7}$$

l_i and p_i are the lengths of the shortest path and the path taken by the agent. SoftSPL replaces the binary success term in SPL with a 'soft' value that indicates the progress made by the agent towards the goal. Note that 'progress' can be negative if the agent ends up farther away from the goal than where it started.



(a) Geodesic vs Euclidean distance

(b) SPL vs SoftSPL

Figura 2.22: Illustration of the differences between Geodesic Distance and Euclidean Distance in Subfigure (a). The Euclidean distance, in black, ignore navigable constrains such as obstacles and walls while the Geodesic distance, in blue, is limited by those constrains while computing the shortest path between points. Illustration of the differences between SPL and SoftSPL in Subfigure (b). Three different paths were illustrated. In blue is the agent follows the same shortest path as the oracle resulting in both SPL=1 and SoftSPL=1. In red the path ends earlier due to a STOP action of the agent, this causes the SPL to be zero since it is a failure but SoftSPL is 0.5 since it represents the partial progress made by the agent. In yellow the agent takes a longer path than the optimal one but ultimately reaches the goal making a successful episode causing the SPL and SoftSPL to be 0.5. The circle represent the agent, the triangle the agent's orientation, and the green hexagons represent obstacles.

2.17.1 SPL's Main Issues

In the following, we briefly discuss some of the issues of using the SPL metric for evaluation as pointed out by Batra et al. (2020).

- **All failures are not equal.** A navigation episode is deemed unsuccessful if the agent stops outside a pre-determined radius r around the goal. However, this results in no distinction between major failures and minor mistakes and gives no indication of partial progress. An agent navigating 30m to reach the goal and stopping at an approximately r distance from it vs. one that refuses to move from the start position are equally unsuccessful, both achieving an SPL of 0. SoftSPL (Datta et al., 2020) does not suffer from this particular issue and can be used instead of SPL for this case.

- **High Variance.** The binary nature of success introduces high variance in the average SPL computation, SPL $= (1/N)\sum_i \text{SPL}_i$. As a thought experiment, consider N nearly identical episodes – i.e., $\frac{l_i}{\max(p_i, l_i)}$ are nearly identical $\forall i$. However, due to (say) actuation noise, the agent's stopping behavior is stochastic – sometimes the agent stops just outside the success criteria ($> r$) and sometimes just inside ($< r$); i.e., S_i can be modeled as a Bernoulli random variable with probability p. In this scenario, the paths traversed by the agent are nearly identical, so intuitively this should result in $Var(SPL) \approx 0$. However, in fact, SPL has a high variance of $p \cdot (1 - p) \cdot \left[(1/N) \sum_i \text{SPL}_i\right]^2$. While this is an example exaggerated to illustrate a point, we do indeed observe in practice high variance in SPL estimates, particularly in settings with noisy actuation. This results in the need to evaluate an unnecessarily large set of episodes to gain confidence in the SPL sample estimates (Batra et al., 2020).

- **No penalty for turning.** SPL_i only accounts for the distance traveled by the agent along the path it takes. On the one hand, this is a good idea because this path's length is not tied to the agent's action space and can be compared across different robots. However, one potentially unintended side-effect is that p_i is unaffected by an agent rotating/turning in-place. This means that SPL does not penalize the agent turning in place. Thus, an agent could always construct a panorama by turning 360° before moving. An alternative would be to measure the number of actions or energy used by the agent (Xia et al., 2020). A set of path comparison metrics in the context of VLN are presented by Jain et al. (2019).

- **No cross-dataset comparison.** Different length paths provide different degrees of sloppiness available to the agent — short paths require a strict adherence to achieve high SPL, while long paths allow significant deviations while still achieving high SPL. See Figure 2.23. Thus, SPL cannot be used for comparisons across different datasets or different portions of a dataset with significantly different optimal path lengths. A similar issue exists for other tasks such as object detection when for example, performances on small and large objects are compared.

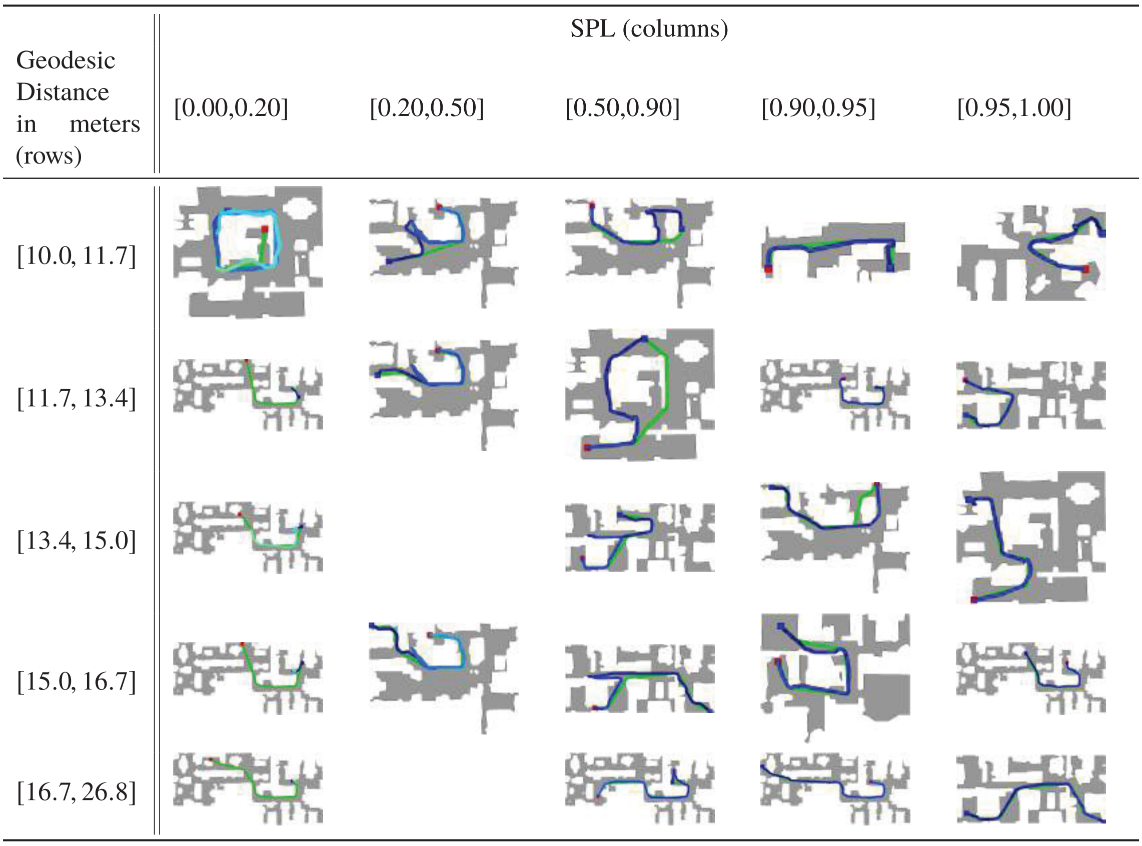


Figura 2.23: Example episodes broken down by geodesic distance between agent's spawn location and target (on rows) vs. SPL achieved by the agent (on cols). Gray represents navigable regions on the map, while white is non-navigable. The agent begins at the blue square and navigates to the red square. The green line shows the shortest path on the map (or oracle navigation). The blue line shows the agent's trajectory. The color of the agent's trajectory changes from dark to light over time. Note the example on the first row, first column, the evaluated planner failed to reach the episode's goal, marked by the red square, resulting in an SPL of 0, the planner looped through approximately the same path, the advance of time is denoted by the change in lightness on the blue path from dark to light. The first column contains examples with very low SPL, meaning that the planner did an early stop. Low path adherence is more penalized in shorter trajectories by the SPL metric (Wijmans et al., 2019).

2.18 CHAPTER SUMMARY

In this chapter, we reviewed SLAM, its history, the concepts used with it such as belief, dealing with data uncertainty, Bayes filter, Markov's assumption. An example of a wall-follower robot within a corridor with three doors was presented to illustrate self-localization. a brief discussion of different approaches for dealing with data uncertainty and data fusion was presented next.

Then, a brief discussion of the different types of kinematic models such as holonomic and nonholonomic was presented. In this work, we focus on nonholonomic kinematics using our VRIBot, see section 5.1. How motion planning is done with model-based algorithms, and how it can be divided into global, which are usually graph pathfinding algorithms and local planners that can use a type of Model Predictive Control (MPC) to predict velocities.

Then we talked about scene understanding and how it can be divided into classification, detection, semantic segmentation, and instance segmentation problems. We discussed how semantics could be exploited to create smarter agents, i.e., semantic understanding and semantic reasoning, and how learning-based methods are particularly interesting for this due to impressive results in other applications. We presented the taxonomy proposed by Chen et al. (2020) to represent the confluence of Deep Learning and mobile robots. And, some essential concepts of CNNs and RNNs.

In this chapter, we introduced some of the Embodied AI navigation tasks, which in its majority employ egocentric visual observations from a single camera as input, an important distinction since navigation research had employed throughout the years GPS, sonar, LiDAR, WiFi, pressure-sensitive bumpers and other sensors to navigate to a point goal successfully. The main task that we are concerned about is called ObjectNav or Visual Semantic Navigation, where an agent must find and navigate to an instance of an object belonging to the semantic target class in an unknown environment and without the aid of artificial markers such as fiducials markers. The definitions for a successful episode were presented, along with the metrics commonly used on the field.

3 RELATED WORK

In this chapter, we review relevant works that deal with the 3D reconstruction of environments that mobile agents can use. Our review elucidates a tendency to employ semantic segmenting to enhance those maps and/or improve odometry. Additionally, we present a brief discussion of how Embodied Artificial Intelligence (E-AI) have been proposed to incorporate the recent advances of pattern recognition into complex tasks that were unfeasible with traditional model-based algorithms and how there is fragmentation on E-AI tasks. Moreover, we deliberate how and why Deep Reinforcement Learning (Deep-RL) have been favored for the E-AI tasks. Moreover, at the end of the chapter, we introduce mid-level visual representations and how they can be used to train effective visuomotor policies.

3.1 SEGMENTATION AND 3D RECONSTRUCTION DECOUPLED FROM NAVIGATION

The introduction of Kinect and other consumer-grade depth cameras enabled more researchers to work on producing dense, accurate 3D maps. KinectFusion (Newcombe et al., 2011) introduced an algorithm that estimates the pose of a moving sensor and uses the Truncated Signed Distance Function (TSDF) (Curless and Levoy, 1996) to store the reconstruction. An improvement to this technique is VoxelHashing (Nießner et al., 2013), which proposes a hierarchical hashing approach to store and access voxels. Recent developments include BundleFusion (Dai et al., 2017), which performs on-the-fly surface reintegration in real-time.

Semantic labeling may be solved more accurately by considering additional information from multiple views of an individual object within a scene. Furthermore, being able to correctly recognize objects within a scene may provide a context for a robot that needs to navigate a 3D scene. Performing online semantic segmentation along with 3D reconstruction has been actively studied in the past few years: (Finman et al., 2013; Herbst et al., 2014; Hermans et al., 2014; Martinović et al., 2015; Tateno et al., 2015; Dai and Nießner, 2018). SemanticPaint (Valentin et al., 2015), employs InfiniTAMv2 (Kähler et al., 2015) to perform live reconstruction and DenseCRF (Krähenbühl and Koltun, 2011) for segmentation. Vineet et al. (2015) present a system for large-scale semantic reconstruction, with impressive results on the KITTI dataset (Geiger et al., 2012). Nguyen et al. (2018) propose an annotation tool that integrates both 2D and 3D segmentation while providing a means to correct any inaccuracies the automatic segmentation system might have produced. SemanticFusion (McCormac et al., 2017) is built upon ElasticFusion (Whelan et al., 2015) to perform SLAM and extends the image semantic segmentation CNN proposed by (Noh et al., 2015).

Semantic SLAM algorithms assist with scene understanding by producing a labeled map of the environment. Labels identify the classes of objects present in a scene. SemanticFusion (McCormac et al., 2017) is an algorithm that produces labelled dense 3D reconstructions. It contains a CNN, based on the work of (Noh et al., 2015), which performs frame-by-frame segmentation and runs in parallel with ElasticFusion (Whelan et al., 2015). The CNN has trained on the NYURGB-Dv2 (Silberman et al., 2012) dataset, using 13 semantic classes. The 2D predictions of this neural network are projected onto the map of ElasticFusion and fused with the existing geometry. Similar to SemanticFusion, ORB-SLAM2-CNN (Saeedi et al., 2018) is based on ORB-SLAM2 (Mur-Artal and Tardós, 2017), projecting the segmentation of a modified version of MobileNet (Howard et al., 2017) to label the keypoints of the ORB-SLAM2-generated map. Thus, the key difference from SemanticFusion is that this algorithm produces a labeled sparse map rather than a dense one. The work of Bujanca et al. (2019) called SLAMBench 3.0 provides a benchmark of several SLAM algorithms, including semantic ones. Table 3.1 presents SLAM algorithms included in SLAMBench 3.0.

Tabela 3.1: SLAM algorithms included in SLAMBench 3.0. Adapted from (Bujanca et al., 2019)

Algorithm	**Type**	**Sensors**	**Implementations**	**Authors**
ORB-SLAM2	Sparse	RGB-D, Stereo, Monocular	C++	(Mur-Artal and Tardós, 2017)
OKVIS	Sparse	Stereo, IMU	C++	(Leutenegger et al., 2015)
SVO	Sparse	Monocular	C++	(Forster et al., 2014)
MonoSLAM	Sparse	Monocular	C++, OpenCL	(Davison et al., 2007)
PTAM	Sparse	Monocular	C++	(Klein and Murray, 2007)
BundleFusion	Dense	RGB-D	CUDA	(Dai et al., 2017)
ElasticFusion	Dense	RGB-D	CUDA	(Whelan et al., 2015)
InfiniTAM	Dense	RGB-D	C++, OpenMP, CUDA	(Kähler et al., 2015)
KinectFusion	Dense	RGB-D	C++, OpenMP, OpenCL, CUDA	(Newcombe et al., 2011)
LSD-SLAM	Semi-Dense	Monocular	C++, PThread	(Engel et al., 2014)
SemanticFusion	Dense, semantic	RGB-D	CUDA	(McCormac et al., 2017)
ORB-SLAM2-CNN	Sparse, semantic	Monocular	C++	(Saeedi et al., 2018)
DynamicFusion	Dense, non-rigid	RGB-D	CUDA	(Newcombe et al., 2015)
FLaME	Depth estimation	Monocular	C++	(Greene and Roy, 2017)

Caminal et al. (2018) argued that RGB-D cameras are not feasible to reconstruct large outdoor environments due to lighting conditions and low depth range. Thus, they focused on a LiDAR only approach. Their approach uses the RTAB-Map SLAM algorithm (Labbé and Michaud, 2019), and it is compared to Kintinuous (Whelan et al., 2012). Additionally, they presented a comparison between Depth only from an RGB-D camera, no color information was used, and a 3D LiDAR on the KITTI Dataset (Geiger et al., 2013).

Milioto and Stachniss (2019) proposed a tool called Bonnet, an open-source training and deployment framework for semantic segmentation in robotics using Convolutional Neural Networks (CNNs). It provides a modular approach to simplify the training of a semantic segmentation CNN independently of the used dataset and the intended task and facilitates the integration with ROS for the inference.

Patel et al. (2019) propose a tight coupling between semantic segmentation and Visual-Inertial Odometry (VIO) to improve Simultaneous Localization and Mapping (SLAM). Their approach uses a Dilated Fully Convolutional Network (FCN) to perform semantic segmentation. The predictions of segmentation are propagated to sparse ORB features (Mur-Artal et al., 2015), see Figure 3.1, in turn, the semantic ORB features are used in VIO and to perform SLAM akin to ORB-SLAM2 (Mur-Artal and Tardós, 2017). Their method was tested on their own Unmanned Ground Vehicle (UGV), see Figure 3.2.

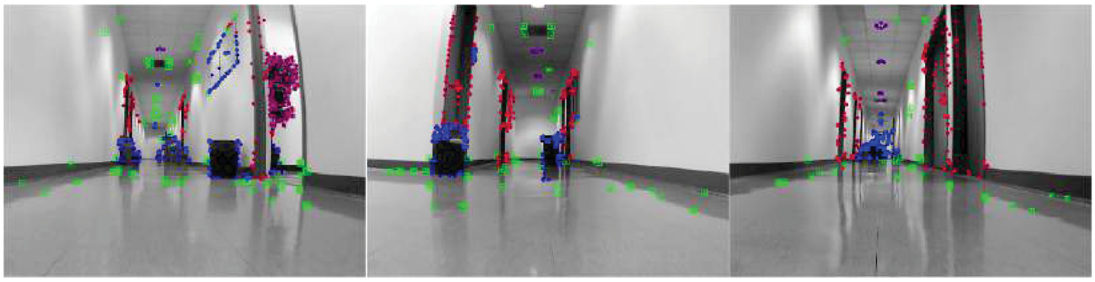


Figura 3.1: Resulting semantic ORB features. Coloring distinguish the semantic class of each detected ORB feature (Patel et al., 2019).

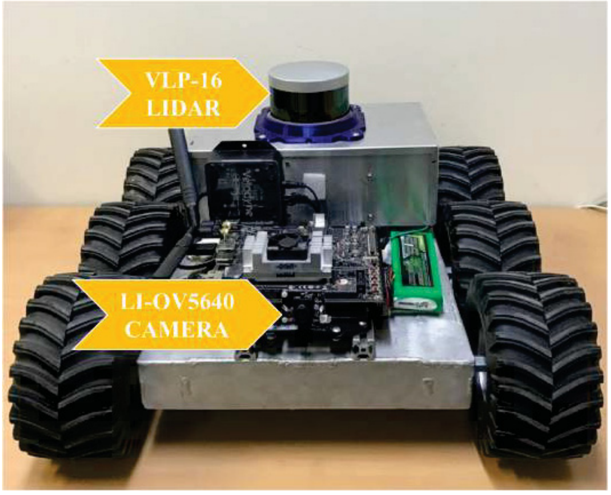


Figura 3.2: Patel et al. (2019)'s Unmanned Ground Vehicle (UGV) uses a platform based on NVIDIA Jetson, a LiDAR (VLP-16), an RGB camera (LIOV5640), and a MEMS Inertial Measurement Unit (IMU) (LSM330DLC).

Wu et al. (2019) focuse on LiDAR only segmentation, and address the one of the issues of supervised point cloud segmentation: limited realistic labeled LiDAR data. They proposed an unsupervised domain adaptation from synthetic data acquired by simulators such as GTA-V, and demonstrate that training with synthetic data using domain adaptation improved the performance of the segmentation.

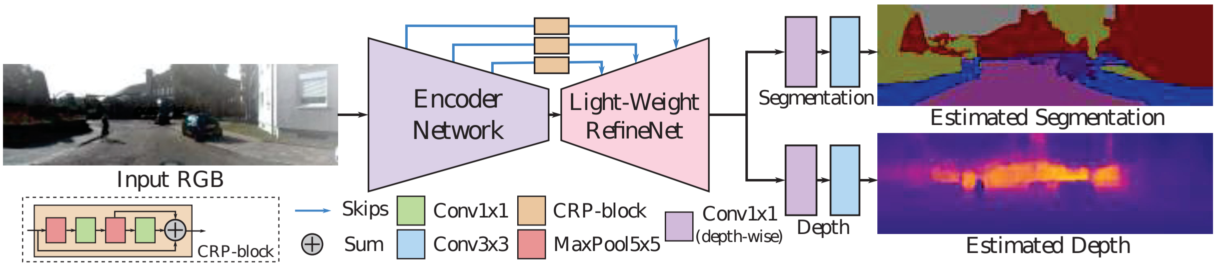


Figura 3.3: General network structure for joint semantic segmentation and depth estimation. Each task has only two specific parametric layers, while everything else is shared. (Nekrasov et al., 2019)

Envisioning deploying of deep learning models to robotics, Nekrasov et al. (2019) proposed a single model that can perform multiple tasks at once: depth estimation and semantic segmentation using RGB images, see figure 3.3.

Grinvald et al. (2019) presented a robotic oriented online scene reconstruction that performs semantic segmentation. Their focus is on RGB-D camera data. As a basis for their segmentation, they employ a Mask R-CNN (He et al., 2017) on the RGB frame and further refined it using the Depth information from the RGB-D camera. Their approach incrementally builds volumetric object-centric maps during online scanning with a localized RGB-D camera. Figure 3.4 presents their proposed pipeline, and Figure 3.5 shows a result obtained by their ABB YuMi robot running their method. Their method is said to run at approximately 1 Hz on 640x480 input.

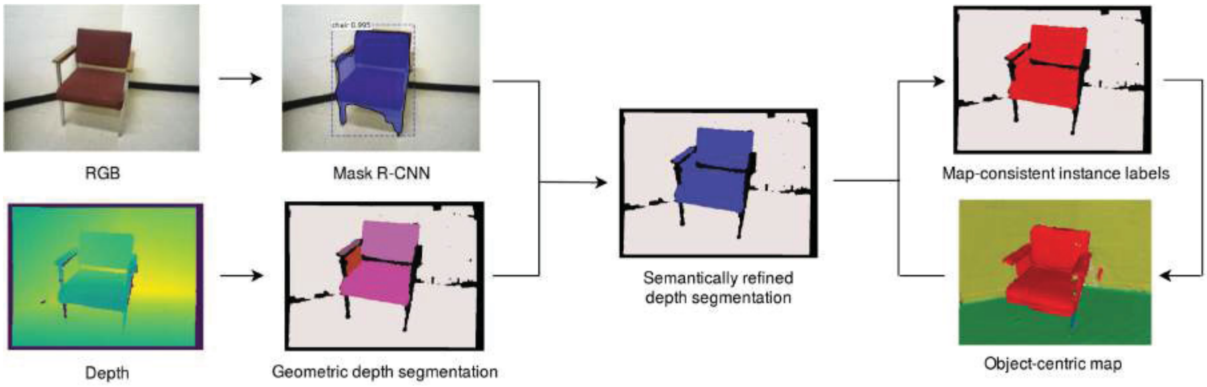

RGB
Mask R-CNN
chair 0.995
Depth
Geometric depth segmentation
Semantically refined depth segmentation
Map-consistent instance labels
Object-centric map

Figura 3.4: The individual stages of the proposed approach for incremental object-level mapping are illustrated here with an example. At each new frame, the incoming RGB image is processed with the Mask R-CNN network to detect object instances and predict each semantically annotated mask. Simultaneously, a geometric segmentation decomposes the depth image into a set of convex 3D segments. The predicted semantic mask is used to infer class information for the corresponding depth segments and refine non-convex objects' over-segmentation by grouping segments by the object instance they belong to. Next, a data association strategy matches segments predicted in the current frame to their corresponding instance in the global map to retrieve each map-consistent label. Finally, dense geometry from the resulting local map and segmentation information from the current frame are integrated into the global map volume. (Grinvald et al., 2019)

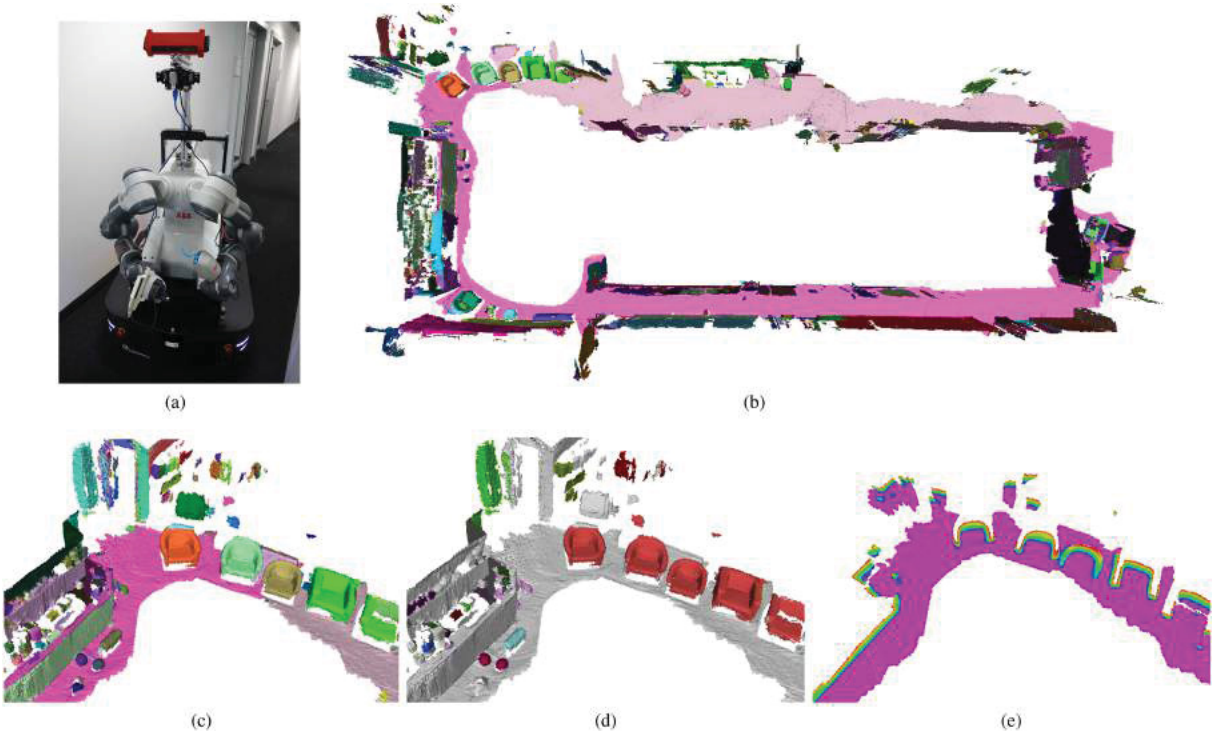


Figura 3.5: Subfigure (a) shows the robotic platform used for the online mapping experiment of an office floor. The map is reconstructed from RGB-D data recorded with two Primesense cameras mounted on an ABB YuMi robot attached to a Clearpath Ridgeback mobile base. The final map is shown as a mesh in Subfigure (b). Subfigure (c) shows a detail of the map where individual objects identified in the scene are represented with different colors. The corresponding semantic categories of the recognized instances are shown in Subfigure (d). Subfigure (e) shows a single horizontal slice at 1 m height of the reconstructed TSDF grid with magenta indicating observed free space. (Grinvald et al., 2019)

3.2 EMBODIED ARTIFICIAL INTELLIGENCE

The domain of Embodied AI, in which agents learn to complete tasks through interaction with their environment from egocentric observations, has experienced substantial growth with the advent of deep reinforcement learning and increased interest from the computer vision, Natural language processing (NLP), and robotics communities. This growth has been facilitated by the creation of a large number of simulated environments (such as AI2-THOR, Habitat, iGibson, and CARLA (Dosovitskiy et al., 2017)), tasks (like point navigation, instruction following, and embodied question answering), and associated leaderboards. While this diversity has been beneficial and organic, it has also fragmented the community: a huge amount of effort is required to do something as simple as taking a model trained in one environment and testing it in another (Weihs et al., 2020). Figure 3.6 is an analysis provided by Weihs et al. (2020) and displays the growth and fragmentation of E-AI in recent years.

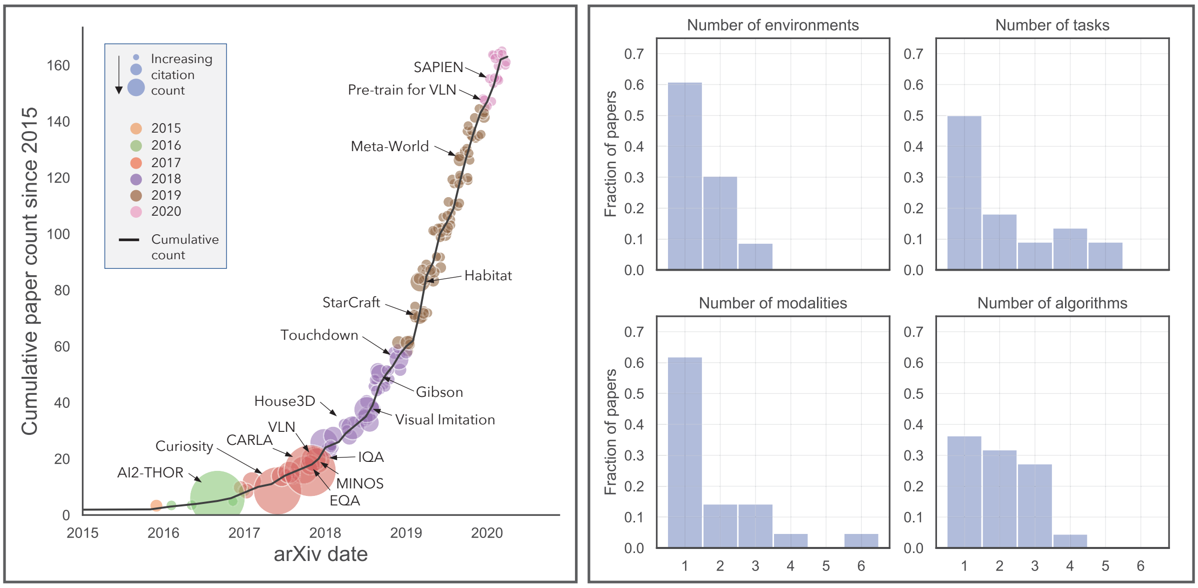


Figura 3.6: Growth and fragmentation of Embodied AI (E-AI). *Left*— the cumulative number of papers published on arXiv since 2015 which were identified as being in the E-AI domain. The number of publications has dramatically increased in recent years. *Right*— Weihs et al. (2020) manually annotated the 20+ most-cited E-AI papers, then plotted the histograms of the frequencies with which these papers ran experiments on multiple environments, with multiple tasks, multiple modalities, and multiple algorithms. The large frequency of works with only a single task, environment, and modality evaluated suggests large barriers to comprehensive evaluation. While several papers evaluate multiple algorithms, we noticed little standardization - some compare imitation with reinforcement learning, some compare A3C (Mnih et al., 2016) and PPO (Schulman et al., 2017), others try Q-learning (Watkins and Dayan, 1992). Moreover, these represent the most cited papers of the past years, likely making this analysis not representative of a randomly selected paper. This analysis used the S2ORC (Lo et al., 2020). Figure and analyses by Weihs et al. (2020).

3.3 DEEP REINFORCEMENT LEARNING BASED NAVIGATION

Deep-RL has been favored for E-AI tasks (Sax et al., 2020; Fang et al., 2019; Wijmans et al., 2019) with an end-to-end design, where a single module uses these egocentric visual observations to infer an action, instead of a hierarchy of modules, e.g., visual odometry, SLAM, path planning. In those studies, semantic reasoning was arguably learned implicitly during training akin to emergent behavior, e.g., the agent learned to identify doors and exits without having this explicit input information. However, how to properly encourage semantic reasoning still an open problem. In Reinforcement Learning (RL) for E-AI, learning is conditioned by sparse rewards usually tied to collisions and progress towards the episode objective. The agent's strategy, i.e., the learned policy, converges towards efficiency by iterative exploration and exploitation of data. Sax et al. (2020) demonstrated that by using **visual priors** instead of purely raw RGB, such as depth estimation and semantic segmentation, the policy achieved better path planning performance.

3.4 MID-LEVEL VISUAL REPRESENTATIONS

One of the goals of computer vision is formulating useful visual priors about the world and developing methods for extracting them. Conventionally, this is done by defining a set of problems, e.g., object detection, depth estimation, and so on, and solving them independently of any ultimate downstream active task, e.g., navigation, manipulation (Anderson et al., 2018a). Sax et al. (2020), provided a study of how such standard vision objectives can be used within RL frameworks as mid-level visual representations (Peirce, 2015) in order to train effective visuomotor policies.

How much does having visual priors about the world (e.g., the fact that the world is 3D) assists in learning to perform downstream motor tasks (e.g., navigating a complex environment)? What are the consequences of not utilizing such visual priors in learning? Sax et al. (2020), studied these questions by integrating a generic perceptual skillset, such as a distance estimator or an edge detector within a reinforcement learning framework (see Figure 3.7). This skill set ('mid-level vision') provides the policy with a more processed state of the world compared to raw images. Their large-scale study demonstrated that using mid-level vision results in policies that learn faster, generalize better, and achieve higher final performance when compared to learning from scratch and/or using state-of-the-art visual and non-visual representation learning methods.

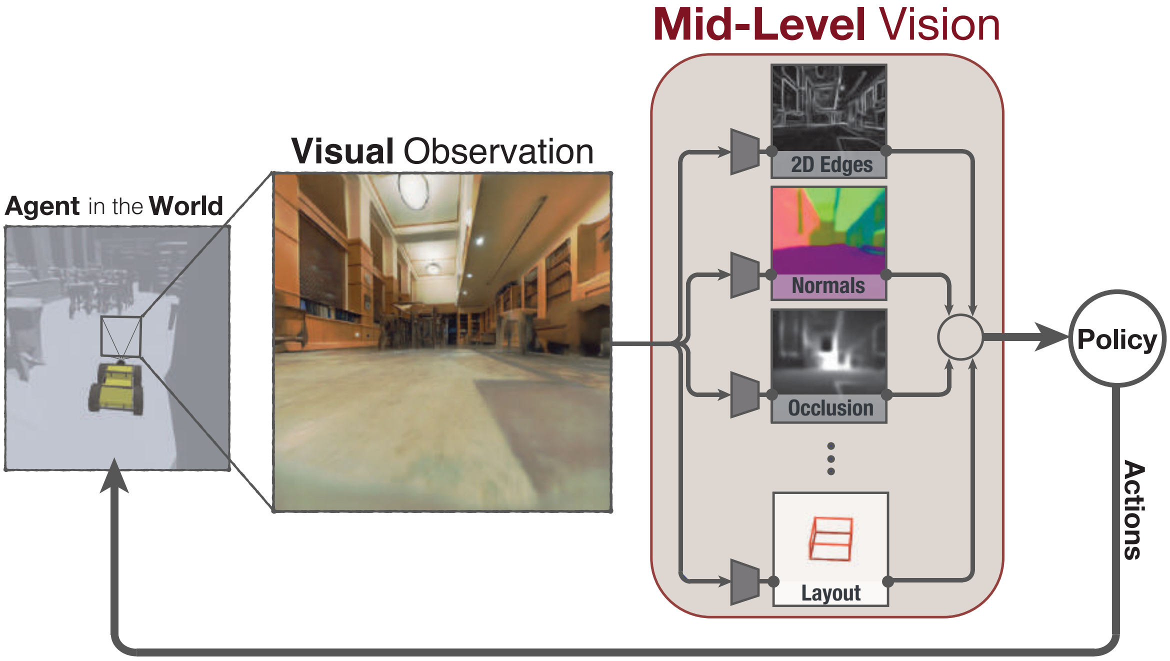


Figura 3.7: Mid-level vision in an end-to-end framework for learning active robotic tasks (Sax et al., 2020).

In their work Sax et al. (2020) investigated three E-AI tasks: Local Planning (Point-Based Navigation (PointNav)), Visual Exploration (exploration using egocentric RGB), Visual-Target Navigation (the agent must navigate and align its pose to match as much as possible the provided image input). Additionally, they showed that conventional computer vision objectives are particularly effective in this regard and can be conveniently integrated into reinforcement learning frameworks. An interesting insight was that no single visual representation was universally useful for all downstream tasks (Sax et al., 2020).

Sax et al. (2020), used mid-level visual representations encoded via an Encoder-Decoder architecture. Their work relies on the dictionary of features and their relationships proposed by Zamir et al. (2018), that studied the transferability of knowledge between computer vision tasks. Figure 3.8 illustrates how the dictionary of features was incorporated into the policy network.

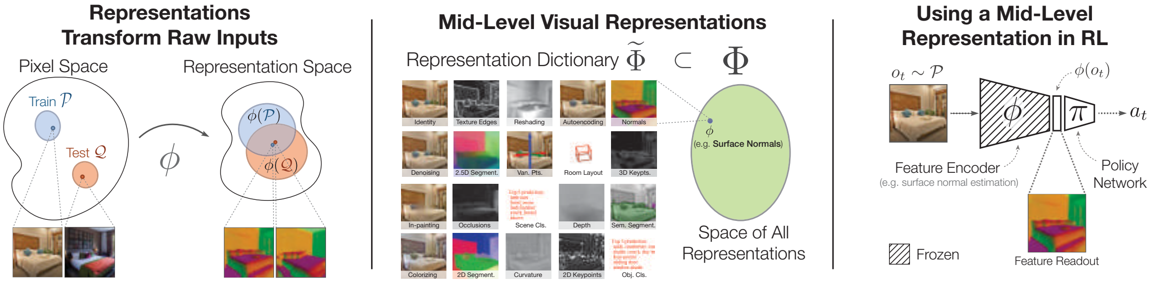


Figura 3.8: *Left:* The job of a feature is to warp the input distribution, potentially making the train and test distributions look more similar to the agent. *Middle:* Visualizations for 19 of 24 mid-level vision objectives contained in a dictionary $\widetilde{\Phi}$. The dictionary is a sample of all possible transforms (Φ), and the best function for a given task must have the proper 'invariance', i.e., ignore irrelevant parts of the input while retaining the information required for solving the downstream task. *Right:* a visual observation o_t belonging to a training set $\mathcal{P}$ is encoded to $\phi(o_t)$ using a pretrained feature encoder, this representation is then fed to a policy network π that infer a action a_t (Sax et al., 2020).

Shen et al. (2019) also employ mid-level visual representations, but they argue that multiple policies, where each policy learns from a single task, e.g., surface normals, weighted by the tasks' affinity (Zamir et al., 2018) can better generalize than a single policy that concatenates multiple representations.

3.5 CHAPTER SUMMARY

In this chapter, we reviewed several works of different areas, a necessity due to the multidisciplinary nature of robotics and how Object-Goal Navigation (ObjectNav) is a task composed of several sub-problems. Much effort has been employed to introduce Deep Learning methods to robotics. Nevertheless, there is still a lot of fragmented work, especially with E-AI where comparability is tied to simulators. E-AI tasks have a tendency towards Deep-RL instead of Imitation Learning since the former is oracle free. Additionally, we discussed mid-level visual representations, a way to reuse computer vision algorithms to train effective visuomotor policies.

4 DATASETS AND SIMULATORS

In this chapter, we discuss virtual environments, state-of-the-art simulators, and competitions with proper rules about embodiment, environment, tasks to perform, and how to evaluate agents on those tasks. Three different competitions for Embodied Artificial Intelligence (E-AI) held in 2020 were discussed Habitat Challenge 2020, Sim2Real Challenge with iGibson, and RoboTHOR Challenge 2020. Those competitions provided a fair comparison between agents in E-AI navigation tasks and set the current state-of-the-art performance on the area. A smaller emphasis was given to the RoboTHOR Challenge 2020 since we did use the AI2-THOR simulator for our experiments. We did not partake in any of those competitions, but a comparison between our proposed approach BEyond and the results reported at the Habitat Challenge 2020 leaderboard is presented at section 7.1.

The main requirements of E-AI navigation tasks are that the agent needs a physical embodiment being it simulated or real and that it can freely navigate within a finite environment. The embodiment used in this work is discussed in section 5.1. The episode sets and scenes used in this work's experiments are described in sections 6.2.1, and 6.3.

4.1 ENVIRONMENTS

The environment setting depends heavily on the simulator used. Experiments on the real robot could be reproduced by providing sufficient logging. However, following a recorded experiment with the real robot limits the agent to a predetermined path. In none of the steps during an episode, the agent can take a different action than the one in the recording. Meaning that while the experiment is reproducible, it is not comparable with others. Due to the sequential nature of episodes, an action taken in a step t in time directly impacts the following state. In the case of path planning, a suboptimal action does not cause immediate failure. An agent can take a series of suboptimal actions and still reach the goal with a longer path than the optimal. This characteristic is impossible to recreate with a pre-recorded single episode. Thus, the AI research community is undergoing a paradigm shift from tasks involving static datasets of images or pre-recorded experiment trials to tasks that rely on the deployment of an active, embodied agent in the world (Savva et al., 2019; Batra et al., 2020).

Gazebo (Koenig and Howard, 2004) is an older software used in education and research (Takaya et al., 2016; Vega and Cañas, 2019). Gazebo's development began in the fall of 2002 at the University of Southern California. Open Source Robotics Foundation (OSRF) continues Gazebo's development with support from a diverse and active community, with Gazebo 11.2.0 released in (2020-09-30). While Gazebo is a more mature simulator, it lacks photorealism and compatibility with the ObjectNav task research community's datasets. An important feature of this simulator is its integration with Robot Operating System (ROS). Figure 4.1 displays one of our preliminary tests with the VRIBot on the Gazebo simulator.

iGibson, Habitat, and AI2-THOR follow the OpenAI gym convention (Brockman et al., 2016) for setting up episodes, observation space, action space, rewards (in case of reinforcement learning), and episode termination.

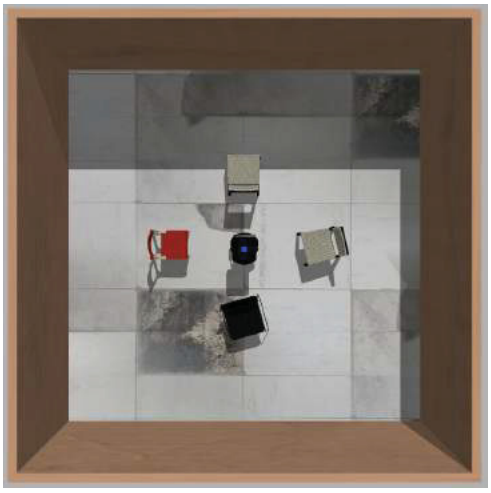


Figura 4.1: Example of a simulation in the Gazebo Simulator from a top view. The robot is at the center, and four different assets are present as obstacles. The scene is a single enclosed room with textured floor and walls with directional light and Phong shading.

4.1.1 iGibson

iGibson, the Interactive Gibson Environment (Xia et al., 2020), is a simulation environment providing fast visual rendering and physics simulation, based on Bullet (Coumans and Bai, 2019). iGibson allows to train and evaluate robotic agents that use RGB images and/or other visual sensors to solve interactive indoor navigation and manipulation tasks, such as opening doors, picking up and placing objects, or searching in cabinets. It is an improvement over the original GibsonEnv proposed by Xia et al. (2018). Figure 4.2 presents the architecture of iGibson Simulator.

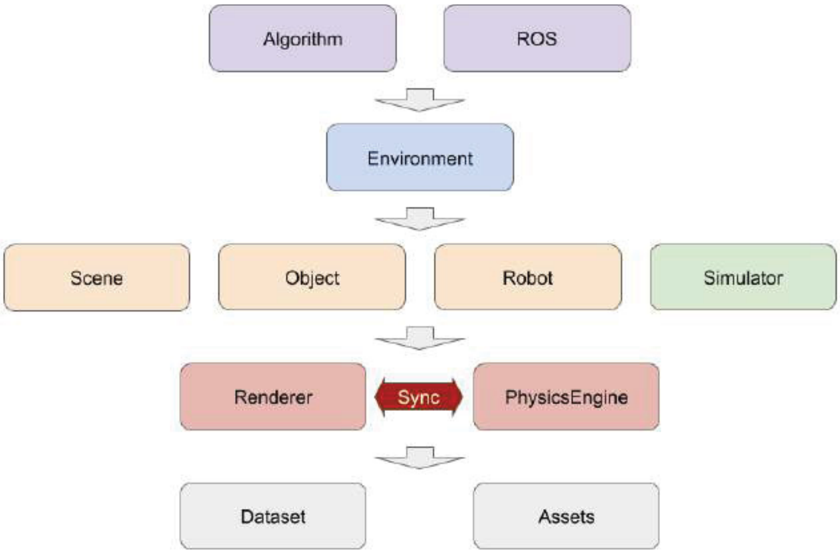


Figura 4.2: Architecture of iGibson, higher layers are built on top of abstractions/resources provided by lower layers (Xia et al., 2020).

At the top layer, lies the *Algorithm* and *ROS*. *Algorithm* can be any algorithm that accommodates OpenAI gym interface and conventions (Brockman et al., 2016). In this layer is the integration with ROS that allows for evaluation and visualization of software in the ROS pipeline, such as the ROS Navigation Stack.

In the next layer lies the *Environment*. Environment follows the OpenAI gym convention and provides an API interface for applications such as Algorithms and ROS. *Environment* usually defines a task for an agent to solve, which includes observation space, action space, reward, termination condition.

In the next layer, it is the *Scene*, *Object*, *Robot*, and *Simulator*. Scene loads 3D scene meshes from the supported datasets. *Object* loads interactable objects from a collection of assets. *Robot* loads robots from a pool that contains each robot description and their respective components. *Simulator* maintains an instance of *Renderer* and *PhysicsEngine* and provides APIs to import Scene, Object and Robot into both of them and keep them synchronized at all time.

In the next layer, is the *Renderer* and *PhysicsEngine*. These are the two pillars that ensure the visual and physics fidelity of iGibson. The developers created their own MeshRenderer that supports customizable camera configuration and various image modalities with fast rendering speed. The open-sourced PyBullet is the underlying physics engine. It can simulate rigid body collision and joint actuation for robots and articulated objects accurately and efficiently. At the bottom layer, lies the *Dataset* and *Assets*. *Dataset* contain 3D reconstructed real-world environments. Assets contain models of robots and objects. Figure 4.3 displays an example from iGibson Simulator and visual output modalities.

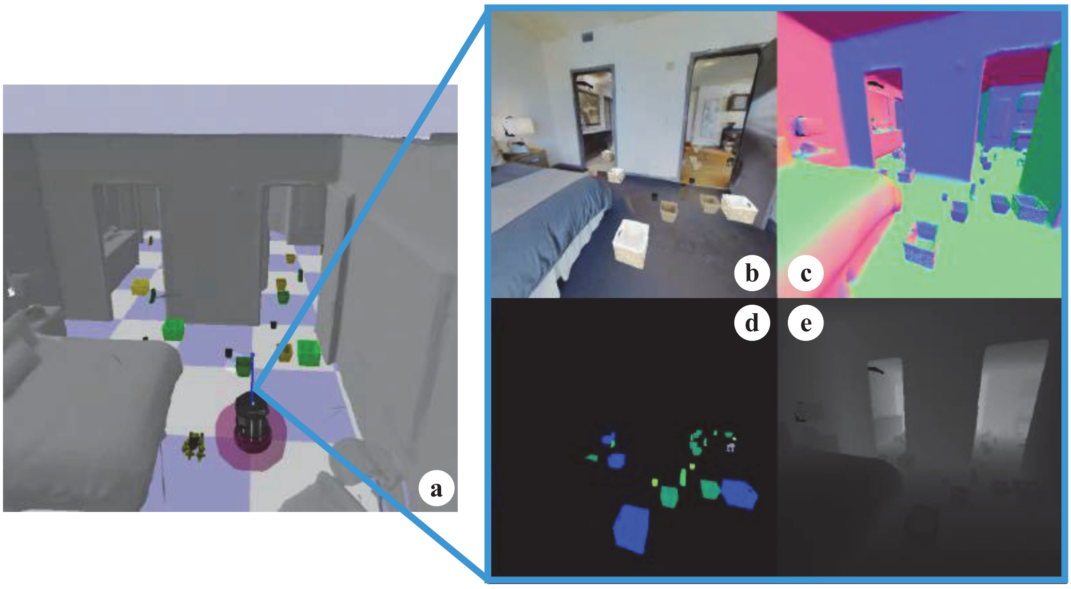


Figura 4.3: iGibson Simulator and output modalities. 3D view of the agent in the environment (a) and four of the visual streams provided by the Interactive Gibson Environment: RGB images (b), surface normals (c), semantic segmentation of interactable objects (d), and depth (e). Figure from Xia et al. (2020).

4.1.2 Habitat

Habitat enables the training of embodied AI agents in a photorealistic and efficient 3D simulator before transferring the learned skills to reality. The simulator endorses the paradigm shift from ‘internet AI’ based on static datasets, e.g., ImageNet (Deng et al., 2009), COCO (Lin et al., 2014), VQA (Antol et al., 2015), to embodied AI where agents act within realistic environments, by active perception, long-term planning, learning from interaction, while grounded in an environment (Savva et al., 2019).

The core API of Habitat is named Habitat-Sim, see Figure 4.4. The Habitat-Sim delegates management of all resources related to 3D environments to a *ResourceManager* that

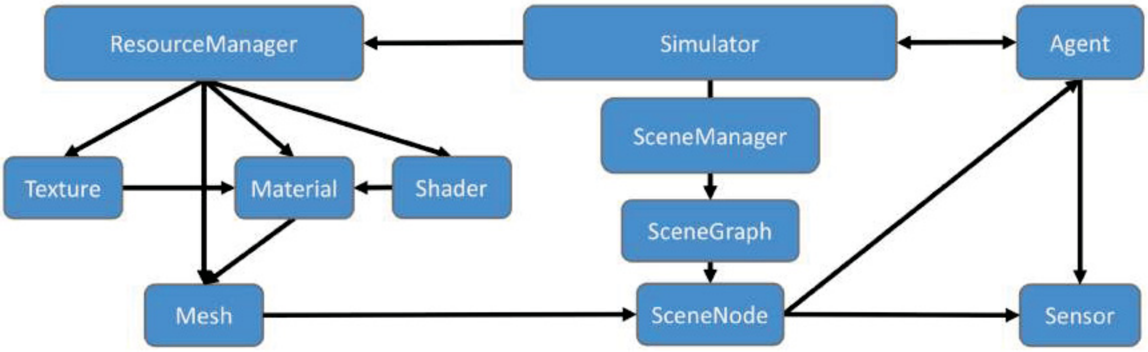


Figura 4.4: Architecture of Habitat-Sim main classes. The Simulator delegates management of all resources related to 3D environments to a ResourceManager. Agents and their Sensors are instantiated by being attached to SceneNodes in a particular SceneGraph (Savva et al., 2019).

is responsible for loading and caching 3D environment data from a variety of on-disk formats. These resources are used within *SceneGraphs* at the level of individual *SceneNodes* that represent distinct objects or regions in a particular Scene. Agents and their Sensors are instantiated by being attached to *SceneNodes* in a particular *SceneGraph* (Savva et al., 2019).

An auxiliary framework called Habitat-Lab facilitates the setup of a navigation task and its evaluation, see Figure 4.5. Some of the internal nomenclature used by Habitat-Lab and the components of Figure 4.5 are further described as follows:

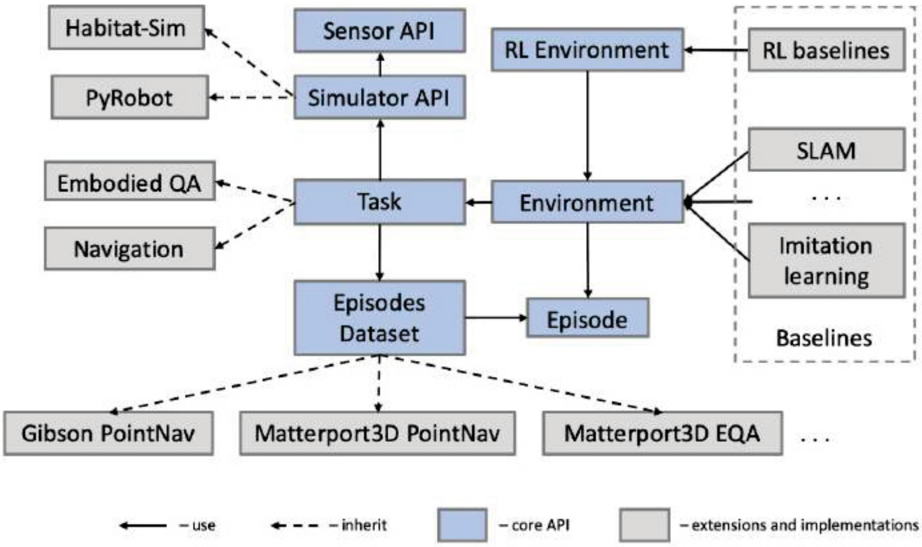


Figura 4.5: Architecture of Habitat-Lab. The core functionality defines fundamental building blocks such as the API for interacting with the simulator backend and receiving observations through Sensors. Concrete simulation backends, 3D datasets, and embodied agent baselines are implemented as extensions to the core API (Savva et al., 2019).

- **Env:** the fundamental environment concept for Habitat. All the information needed for working on embodied tasks with a simulator is abstracted inside an Env. This class acts as a base for other derived environment classes. Env consists of three major components: a Simulator, a Dataset (containing Episodes), and a Task, and it serves to connects all these three components together (Savva et al., 2019).

- **Episodes Dataset:** contains a list of task-specific episodes from a particular data split and additional dataset-wide information. Handles loading and saving of a dataset to disk, getting a list of scenes, and getting a list of episodes for a particular scene (Savva et al., 2019).
- **Episode:** a class for episode specification that includes the initial position and orientation of an Agent, a scene id, a goal position and optionally shortest paths to the goal. An episode is a description of one task instance for the agent (Savva et al., 2019).
- **Task:** this class builds on top of the simulator and dataset. The criteria of episode termination and measures of success are provided by the Task (Savva et al., 2019).
- **Sensor:** a generalization of the physical Sensor concept provided by a Simulator, with the capability to provide Task-specific Observation data in a specified format (Savva et al., 2019).
- **Observation:** data representing an observation from a Sensor. This can correspond to physical sensors on an Agent (e.g., RGB, depth, semantic segmentation masks, collision sensors) or more abstract sensors such as the current agent state (Savva et al., 2019).

Figure 4.6 displays a Habitat example of sensor observations for three sensors (color camera, depth sensor, semantic instance mask) in two different environment datasets.

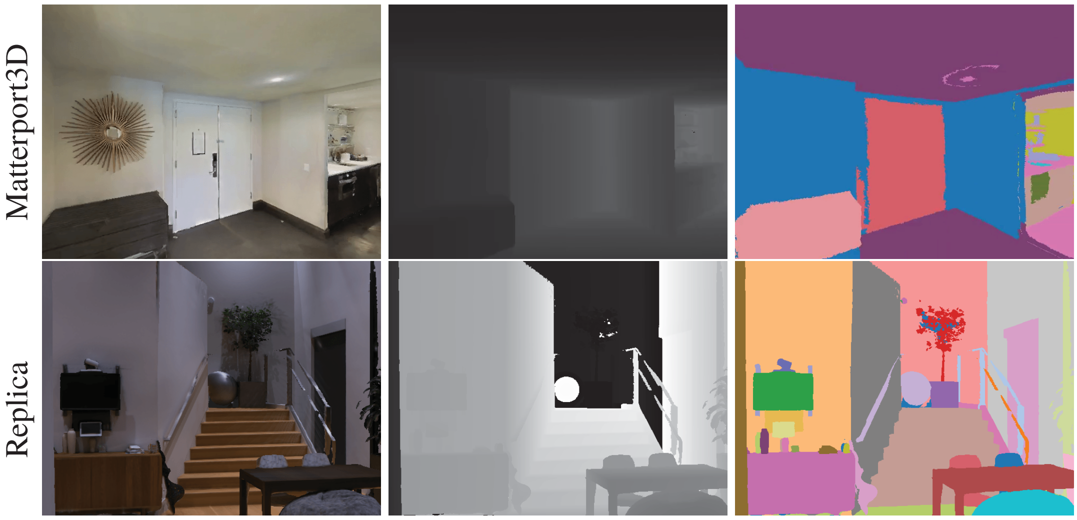


Figura 4.6: Habitat example of sensor observations for three sensors (color camera, depth sensor, semantic instance mask) in two different environment datasets. A Matterport3D (Chang et al., 2017) environment is in the top row, and a Replica (Straub et al., 2019) environment in the bottom row (Savva et al., 2019).

4.1.3 Environments - Datasets

The Gibson simulator is accompanied by two datasets, the original Gibson dataset with 572 reconstructed 3D environments (Xia et al., 2018) and the interactive dataset that contains ten environments with annotated instances of furniture (chairs, tables, desks, doors, sofas). Table 4.1 displays how the original Gibson dataset was divided.

Armeni et al. (2019) provided instance and semantic annotations for the original Gibson dataset. The semantic information for models in the tiny and medium Gibson splits was verified

Split Name	Train	Val	Test	Hole Filled	Total Size
Tiny	25	5	5	100%	8 GiB
Medium	100	20	20	100%	21 GiB
Full	360	70	70	100%	65 GiB
Full+	412	80	80	90.9%	89 GiB

Tabela 4.1: Gibson dataset splits (Xia et al., 2018).

via crowdsourcing. Object labels follow the COCO dataset (Lin et al., 2014) categorization and Figure 4.7 present the object distribution per class on the annotated scenes.

Matterport3D (Anderson et al., 2018b; Chang et al., 2017) is a large-scale RGB-D dataset containing 10,800 panoramic views from 194,400 RGB-D images of 90 building-scale scenes. The scenes have surface reconstructions, and labeled camera poses, and 2D and 3D semantic segmentations.

Both the Gibson Dataset and Matterport3D are supported by iGibson and Habitat. The RoboTHOR dataset was proposed with the RoboTHOR Challenge, and it is tied to the AI2-THOR simulator. Thus it is discussed in section 4.2.3.

4.2 EMBODIED AI NAVIGATION COMPETITIONS

In recent years, learning algorithms have been applied with impressive results to vision-based indoors autonomous navigation. However, most of this work has been trained and tested only in simulation because the learning algorithms are data-hungry, and running real robot tests requires specialized hardware and knowledge. These issues left the final judge of the algorithms' performance – the real world – out of the evaluation. Additionally, recent simulation environments with high fidelity in visuals and physics have reduced the gap between simulation and the real world so that models trained in simulation transfer seamlessly to reality. However, there has not yet been a proper and extensive evaluation on how successfully visuomotor navigation policies trained in these simulators transfer to the real world (Xia et al., 2020).

Thus, the scientific community has been organizing several competitions in an effort to compare the current state-of-the-art in Embodied AI navigation tasks in a simulation stage and a real-world stage. In this section, we briefly discuss three such competitions held in 2020 that employed the SPL metric: Habitat Challenge 2020, Sim2Real Challenge with iGibson, and RoboTHOR Challenge 2020. Table 4.2 presents some of the statistics for different environments that are used for ObjectNav challenges.

	# of scenes	# of target categories	# of instances per category	# of instances per scene per category	Sim/Real	min/max geodesic distance (m)	# of episodes
Habitat (Facebook AI Research, 2019)	90	21	~397	4	Sim	1/30	2 803K
RoboTHOR (Deitke et al., 2020)	89	12	~20	1	Both	0.71/16.8	34K

Tabela 4.2: Statistics for different environments that were used for the Habitat Challenge 2020 and RoboTHOR Challenge 2020. Sim2Real did not included a ObjectNav task so it was not included in this table (Batra et al., 2020).

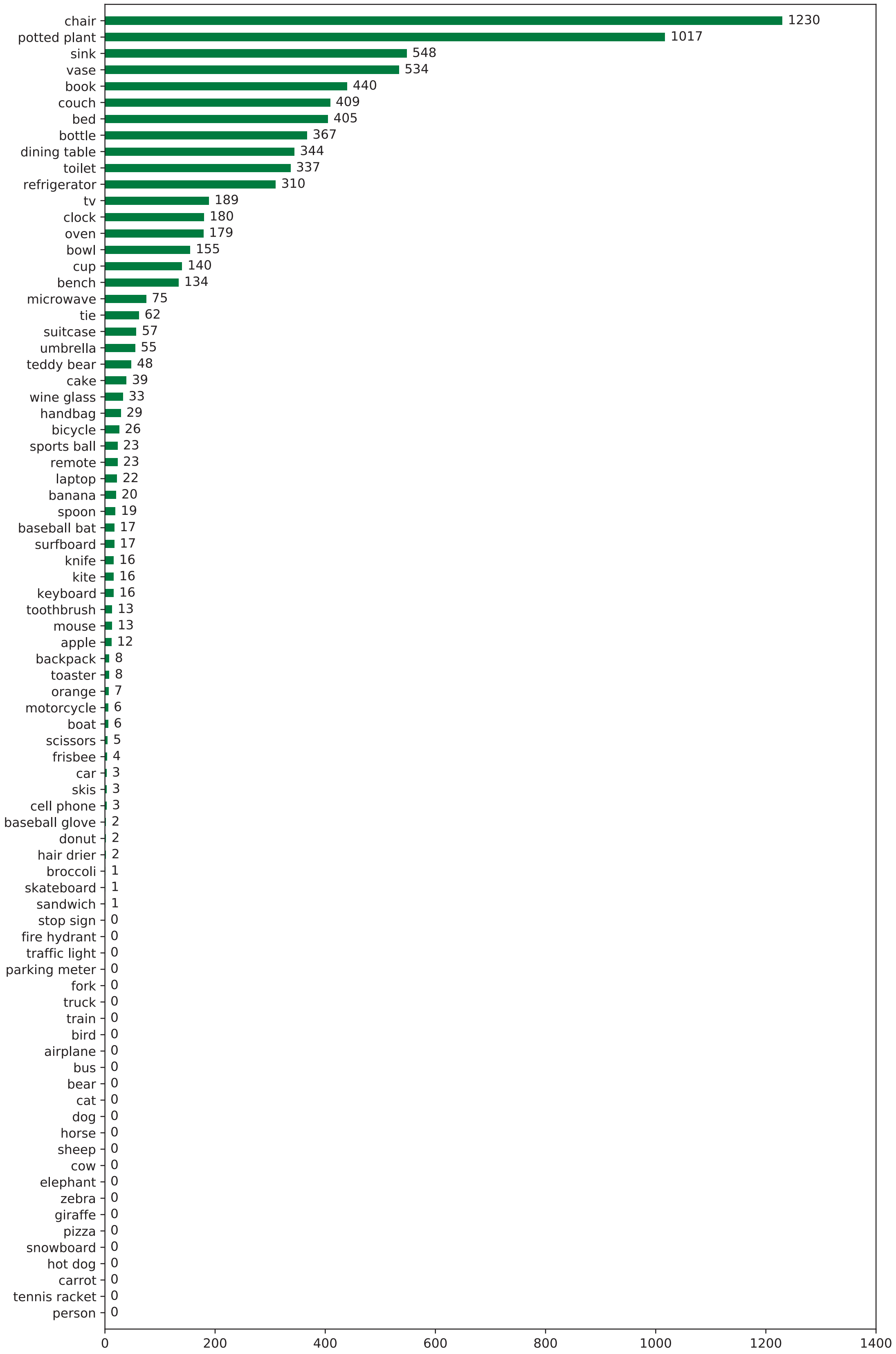


Figura 4.7: Object distribution of Armeni et al. (2019)'s semantic 105 annotated scenes from Gibson-Medium. The most frequent object is the chair, and the less common, but non-zero, objects are broccoli, skateboard and sandwich. Plot based on the work of (Armeni et al., 2019).

4.2.1 Habitat Challenge 2020

In 2020 the Facebook AI research organized the Habitat Challenge 2020, with two leaderboard categories: PointNav, with some differences from the Habitat Challenge 2019, and the new category ObjectNav. The best-submitted results were announced at Embodied AI 2020 Workshop organized together with Conference on Computer Vision and Pattern Recognition (CVPR) 2020. The leaderboards are displayed at tables 4.3 and 4.4, respectively. The PointNav category employed the Gibson 3D scenes dataset (Xia et al., 2018) and the ObjectNav the Matterport3D dataset (Chang et al., 2017).

Rank	**Team**	**SPL ↑**	**SoftSPL ↑**	**DISTANCE TO GOAL ↓**	**SUCCESS RATE ↑**
1	OccupancyAnticipation	0.21	0.50	2.29	0.28
2	ego-localization	0.15	0.60	1.82	0.19
3	DAN	0.13	0.24	4.00	0.25
4	Information Bottleneck	0.06	0.43	2.72	0.09
5	cogmodel_team	0.01	0.33	4.27	0.01
6	UCULab	0.001	0.11	5.97	0.002

Tabela 4.3: Habitat Challenge 2020: PointNav Leaderboard (FAIR A-STAR (Habitat)), 2020).

Rank	**Team**	**SPL ↑**	**SoftSPL ↑**	**DISTANCE TO GOAL ↓**	**SUCCESS RATE ↑**
1	Arnold (Chaplot et al., 2020)	0.10	0.18	6.33	0.25
2	SRCB-Robot-Sudoer	0.10	0.22	6.91	0.19
3	Active Exploration	0.05	0.17	7.34	0.13
4	Black Sheep	0.03	0.16	7.03	0.10
5	Blue Ox	0.02	0.14	7.23	0.07
6	UCULab	0.001	0.11	5.97	0.002

Tabela 4.4: Habitat Challenge 2020: ObjectNav Leaderboard (FAIR A-STAR (Habitat)), 2020).

4.2.2 Sim2Real Challenge with iGibson

Sim2Real Challenge with iGibson, was organized by the Stanford VL and Robotics at Google in 2020. It consisted of a simulation phase and a real-world phase, with three different tasks, here called scenarios. They were evaluated Scenario 1, PointNav scenario in clean environments; Scenario 2, a PointNav scenario with interactive objects; Scenario 3, a PointNav scenario among dynamic agents. An additional demonstration of the best entries was showcased live at the CVPR 2020.

The simulation phase employed the iGibson, the Interactive Gibson Environment, that includes multiple 3D scenes reconstructed from real-world apartments, and the state-of-the-art PyBullet physics engine. The participants got access to all Gibson 3D reconstructed scenes, 572 total, 72 high-quality ones, which the organization recommended for training, and an additional 3D reconstructed scene called Castro, named after the street of the apartment, that contains part of the real-world apartment that was used in the real world phase. The other part of the apartment, called CastroUnseen, was not available during the training and was used to perform the evaluation. The real-world phase employed a set of real navigating robots LoCoBots (Carnegie Mellon University, 2019). The participants got sessions of 30 minutes per day to perform the experiments on the real-world apartment. The first Gibson Sim2Real Challenge was composed of three navigation scenarios that represent important skills for autonomous visual navigation:

- **Scenario 1, PointNav scenario in clean environments:** the goal in this scenario is for an agent to successfully navigate to a given point location based on visual information (RGB-D images). In this scenario, the agent is not allowed to collide with the environment. This scenario evaluated the sim2real transference of the most basic capability of a navigating agent. The evaluated performance in this scenario used Success weighted by Path Length (SPL) (Anderson et al., 2018a).

- **Scenario 2, PointNav scenario with interactive objects:** in this scenario the agent is allowed (even encouraged) to collide and interact with the environment in order to push obstacles away. Some of the obstacles are not movable. This scenario evaluates agents in Interactive Navigation tasks (Xia et al., 2020), navigation problems that consider interactions with the environment. The Interactive Navigation Score (INS) (Xia et al., 2020) was used to evaluate the performance of agents in this scenario.

- **Scenario 3, PointNav scenario among dynamic agents:** the goal in this scenario is to navigate to the given point location, defined using global coordinates in 2D, avoiding collisions with a dynamic agent that follows unknown navigating patterns. Reasoning, predicting, and avoiding other moving agents is challenging, and this was used to measure how well existing solutions perform in these conditions. As with PointNav scenarios in clean environments, no collisions are allowed in this scenario. The SPL was also used to evaluate the performance of the controlled agent.

Tables 4.5 and 4.6 present the leaderboard in the simulated environments and the one set on the real world, respectively.

Rank	**Participant Team**	**Scenario 1** (SPL) ↑	**Scenario 2** (INS) ↑	**Scenario 3** (SPL) ↑	**Total Sum** ↑
1	inspir.ai.robotics	0.76	0.73	0.44	1.93
2	DAN	0.61	0.57	0.24	1.42
3	**Baseline - Soft Actor-Critic**	0.35	0.29	0.11	0.75
4	VGAI - TCS Research Kolkata	0.34	0.23	0.07	0.63
5	**Baseline - Soft Actor-Critic (Distributed)**	0.12	0.10	0.04	0.26
6	Joanne	0.09	0.08	0.04	0.20
7	cindynian	0.00	0.00	0.00	0.00

Tabela 4.5: Sim2Real 2020, Challenge in Simulation. SPL stands for success weighted by inverse path length, see section 2.17. INS stands for Interactive Navigation Score (Stanford VL and Robotics at Google, 2020).

Rank	**Participant team**	**Scenario 1** (SPL) ↑	**Scenario 2** (INS) ↑	**Scenario 3** (SPL) ↑	**Total Sum** ↑
1	DAN	0.44	0.33	0.11	0.89
2	inspir.ai.robotics	0.33	0.33	0.22	0.89
3	VGAI - TCS Research Kolkata	0.03	0.02	0.01	0.06
4	Joanne	0.00	0.00	0.00	0.00

Tabela 4.6: Sim2Real 2020, Challenge in Real World. SPL stands for success weighted by inverse path length, see section 2.17. INS stands for Interactive Navigation Score (Stanford VL and Robotics at Google, 2020).

4.2.3 RoboTHOR 2020 Challenge

The RoboTHOR 2020 challenge was organized by the PRIOR team at the Allen Institute for AI and dealt with the task of Visual Semantic Navigation/ObjectNav. An agent starts from a random location in an apartment and is expected to navigate towards an object that is specified by its name. The challenge restricted teams to using an ego-centric RGB-D camera mounted on the robot. Participants trained their models in simulation, and these models were evaluated by the challenge organizers using a real robot in physical apartments. The agents were evaluated for their object framing and navigation, and visibility was one of the success criteria.

Split	**Difficulty**	**Total**
Train	easy	8939
Train	medium	8939
Train	hard	9717
Val	easy	1974
Val	medium	1974
Val	hard	2168

Tabela 4.7: RoboTHOR Challenge dataset splits (Deitke et al., 2020).

Training and evaluation were performed within the RoboTHOR framework. RoboTHOR is composed of four parts: (I) 60 simulated apartments in Train, (II) 15 simulated apartments in Validation, (III) four simulated apartments with real counterparts in Test-Dev, (IV) and ten simulated apartments with real counterparts in Test-Challenge. The dataset used consists of 27595 training episodes and 6116 validation episodes, where each episode consists of an agent starting position/rotation and the target object. The following target object types exist in the dataset: Alarm Clock, Apple, Baseball Bat, Basketball, Bowl, Garbage Can, House Plant, Laptop, Mug, Spray Bottle, Television, Vase, a subset of the object classes used by the COCO dataset (Lin et al., 2014). Table 4.7 presents the RoboTHOR Challenge dataset splits.

Tables 4.8 and 4.9 presents the leaderboards for the simulated environments and the real one, respectively.

Rank	**Participant team**	**SPL ↑**	**Success Rate ↑**	**SPL-Proximity Only ↑**	**Success Rate-Proximity Only ↑**
1	Sequoia	0.14	0.31	0.14	0.32
2	tinyTangent	0.04	0.06	0.05	0.08
3	FuturexGO	0.01	0.01	0.01	0.01
4	Siryus	0.01	0.01	0.02	0.05
5	RiveSunder	0.00	0.00	0.01	0.02

Tabela 4.8: RoboTHOR 2020, Phase: Test Dev RGB-D. SPL stands for success weighted by inverse path length, see section 2.17. (Allen Institute for AI, 2020).

Rank	**Participant team**	**SPL ↑**	**Success Rate ↑**	**SPL-Proximity Only ↑**	**Success Rate-Proximity Only ↑**
1	Sequoia	0.16	0.32	0.16	0.34
2	tinyTangent	0.04	0.06	0.06	0.09
3	Siryus	0.01	0.01	0.02	0.05
4	RiveSunder	0.00	0.00	0.01	0.02

Tabela 4.9: RoboTHOR 2020, Phase: Test Challenge RGB-D. SPL stands for success weighted by inverse path length, see section 2.17. (Allen Institute for AI, 2020).

4.3 CHAPTER SUMMARY

In this chapter, we introduced four simulators giving a more in-depth discussion of iGibson and Habitat. Several datasets were introduced as well, such as Matterport3D, Gibson, Replica, and RoboTHOR. Finally, three different competitions for E-AI held in 2020 were discussed Habitat Challenge 2020, Sim2Real Challenge with iGibson, and RoboTHOR Challenge 2020. The tasks for each competition and their respective leaderboards were also presented. The environments presented are the testbeds for our proposed pipelines.

5 PROPOSED APPROACH

Our primary goal in this work is to introduce semantic information into navigation pipelines and exploit it so that mobile robots can progress from traditional point-based navigation to the Embodied Artificial Intelligence (E-AI) task of Visual Semantic Navigation/ObjectNav.

In this work we proposed and evaluated three complete pipelines composed of hierarchical algorithms, here referred to as the building blocks. Each complete pipeline was considered a single entity and evaluated as a single algorithm. Our pipelines are: EXchangeable, AUTOcrat, and BEyond

Our VRIBot definition, see section 5.1, is an embodiment that is modular and can be easily assembled with Commercial Off-The-Shelf (COTS) components, resulting in a robot similar to popular ones, such as LoCoBot (Carnegie Mellon University, 2019) and TurtleBot (Wise and Foote, 2010). Additionally, the embodiment can be simulated in simulators compatible with Unified Robot Description Format (URDF) (Open Robotics, 2019) and Robot Operating System (ROS) (Open Robotics, 2018), and the agent's internal logic can be used in real and simulated environments. Our Goal prediction network contained within our BEyond pipeline can bridge dedicated modules of semantic segmentation and motion planning, see section 5.5.2.

The AUTOcrat pipeline, while not wholly novel, it is composed of elements from previous works (ResNet-50, Mid-level semantic encoder, Long Short-Term Memory (LSTM)) with a new Multilayer Perceptron (MLP) head for action classification. It was adapted to be trained with Imitation Learning instead of Deep Reinforcement Learning (Deep-RL).

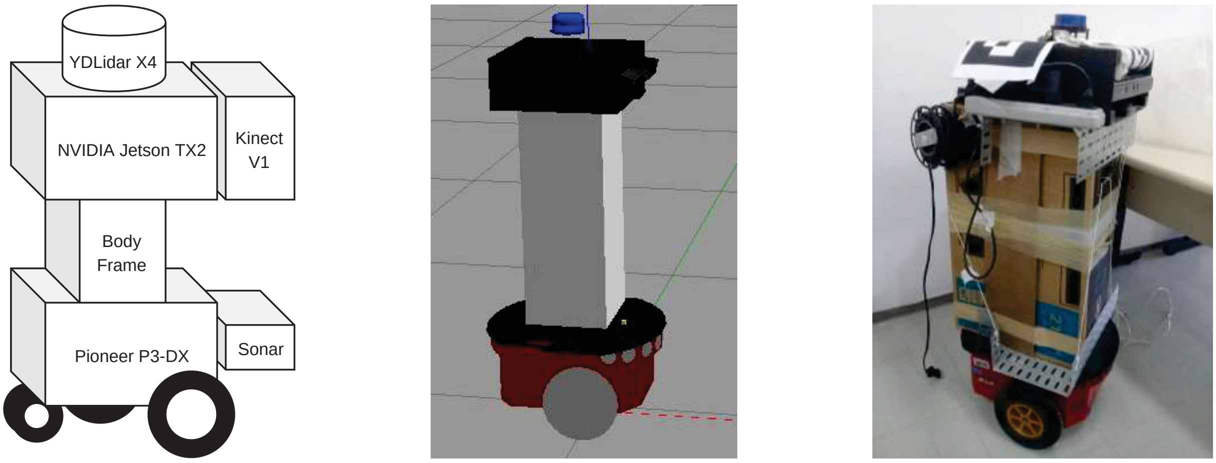
To the best of our knowledge, this is the first time YOLACT++ (Bolya et al., 2020) was incorporated into a pipeline for Object-Goal Navigation (ObjectNav). Mask-RCNN (He et al., 2017) is more commonly employed for this. YOLACT++ was included on our BEyond pipeline, see section 5.5. We decided for the YOLACT++ due to its improved Frames Per Second (FPS) performance while retaining a high accuracy, as reported by (Bolya et al., 2020) YOLACT++ runs above 30 FPS while Mask-RCNN stays below 10 FPS.

For our experiments, we propose a methodology to scrutinize the relation between episode settings and the agent's performance. In the literature, it is common to only report the average metrics for the entire dataset, which contains several completely different scenes. Sometimes the performance per class is also reported (Mousavian et al., 2019; Shen et al., 2019; Fang et al., 2019). We believe that this traditional way of running and reporting experiments for ObjectNav overlook several nuances, such as how the episode distance interferes with the performance. We investigate if the episode's distance is a property similar to how the size of the object interferes in object detection.

For our BEyond pipeline we propose a goal prediction network that can bridge dedicated modules of semantic segmentation and motion planning (see section 5.5.2). We hypothesize that by separating semantics from the motion planning, we could achieve better performance with BEyond (see section 5.5) than AUTOcrat (see section 5.4), for the experiments, see chapter 6 and the discussion on chapter 7. By introducing a dedicated module for semantic reasoning, a modular approach such as our BEyond pipeline can be an alternative to end-to-end policies that attempt to map observations to actions directly.

5.1 EMBODIMENT

As a proof-of-concept, a differential drive wheeled robot, named VRIBot, was designed. This robot is an ensemble of different off-the-shelf products such as the commercial robot Pioneer P3-DX from the company Mobile Robots, see Figure 5.1. The robot has a two-wheeled drive system with independent actuators for each wheel and a non-driven wheel, which is a caster wheel. This allows the use of wheel encoders for odometry and navigation limited to a 2D plane, constraining movement to three degrees of freedom (x, y, yaw).



(a) Diagram

(b) 3D Rendering

(c) Real Hardware

Figura 5.1: Illustrative depictions of VRIBot. Subfigure (a) shows a component diagram. Subfigure (b) shows the Phong shaded 3D rendering of the model in the simulation. Lastly, Subfigure (c) shows the prototype used for the experiments.

An embedded power-efficient AI computing device named NVIDIA Jetson TX2 was employed, and it is responsible for the computation (NVIDIA, 2015). The visual sensors added were the RGB-D sensor Microsoft Kinect V1 (Wasenmüller and Stricker, 2016), and a 360 degrees 2D-LiDAR the YDLidar X4 (Shenzhen EAI Technology Co., 2017). Combining an RGB-D camera (been the Microsoft Kinect V1 one of the first RGB-D commercial off-the-shelf used for research) and a 2D LiDAR (range finder sensors has been applied extensively for SLAM methods) is an alternative to the much more expensive 3D LiDAR (Hess et al., 2016; Catapang and Ramos, 2016). The Pioneer P3-DX has an onboard computing device responsible for controlling an 8-array sonar and the wheels' two electric motors. Its communication is done via a serial port. A body frame was added to increase the robot height to approximately one meter high. This increase in height brings an interesting property since it is closer to a typical humanoid height. The robot design resembles the LoCoBot (Carnegie Mellon University, 2019) used on navigation competitions, such as RoboTHOR Challenge 2020 and Sim2Real Challenge with iGibson.

The embodiment in the simulation varies on the simulator used. The iGibson (Xia et al., 2020) and Gazebo (Koenig and Howard, 2004) are directly compatible with ROS Melodic Morenia (Open Robotics, 2018) and URDF (Open Robotics, 2019), which is an XML format for representing a robot model. URDF allows the robot's description in geometric form; the definition of physical properties like mass and a simulation of the actual sensors, including the motors, can be tuned with noise models to provide more realistic measurements and behavior, meaning that any robot modeled in URDF could be used directly on the simulation.

In the Habitat (Facebook AI Research, 2020) however, there is no direct support for ROS. What is offered is the integration with the framework called PyRobot (Murali et al., 2019). According to Murali et al. (2019), PyRobot is a light-weight high-level interface on top of ROS that provides a consistent set of hardware-independent mid-level APIs to control different robots. PyRobot abstracts away details about low-level controllers and inter-process communication and allows non-robotics researchers, such as Machine Learning (ML), Computer Vision (CV) researchers, to focus on building high-level AI applications. Thus, that we can describe our visual sensors parameters (pose, Field of View (FOV), resolution, depth range in the RGB-D case, and noise), but not the actual robot as in iGibson and Gazebo.

AI2-THOR (Deitke et al., 2020) is similar to Habitat in this regard, where there is an abstract layer to communicate to a real robot, but there is no direct integration with ROS or a form to run a custom robot described by URDF.

In the case of Habitat, the navigation task ObjectNav is simplified to use a set of discrete actions of $A = \{$ 'FORWARD' (0.25m ahead), 'TURN LEFT' ($+30°$ in-place rotation), 'TURN RIGHT' ($-30°$ in-place rotation)$\}$. iGibson can use discrete and continuous action space. Additionally, while the Kinect V1 has a limited degree of freedom on its pitch axis, we decided not to model actions for camera movement control on the evaluation (target framing).

5.2 BUILDING BLOCKS

We employ a modular and incremental approach for our solution, and in the following, we briefly discuss the building blocks used in this work.

5.2.1 Actuation Control

The most basic operation we are interested in is the planar movement with three degrees of freedom (x, y, yaw). Our agent must be capable of changing its pose in relation to the world to reach its goal. Since our agent is a differential drive wheeled robot, this is achieved by controlling the two independently driven wheels on the Pioneer P3DX mobile base, i.e., the actuation control. Since low-level motor control can be easily abstracted on the simulators, here we briefly comment on how it is done on the real robot.

The very low-level control over the motors is handled by the microcontroller's firmware in the Pioneer P3DX. This includes maintaining the platform's drive speed, heading, and acquiring sonar readings. Interface with the microcontroller happens in a client-server model, and it is provided by the Advanced Robotics Interface for Applications (ARIA), a C++-based open-source software. Communication with ARIA is wrapped on the ROSARIA software, a ROS node wrapper that handles messages with the microcontroller via the serial port. The serial communication thus happens in a client-server environment via a cable between the Pioneer P3DX and the NVIDIA Jetson TX2, where the latter runs the ROS node. In practice, this provides us with a convenient node that exposes to the ROS environment a way to ask for sonar readings, battery status, and the agent's linear and angular velocity, and a way to set such velocities.

Therefore, we can then abstract the planar movement to a series of linear and angular velocities throughout time. This abstraction holds true both if the agent is in the real world or in a simulation. Additionally, in a deterministic fashion, if the initial pose is known, the agent's trajectory can be estimated with a series of velocities along time. In reality, actuators and sensors have noise and external factors that are better modeled as stochastic rather than deterministic. To emulate this phenomenon, simulators usually allow modeled noise to be inserted on data, such as the RGB and Depth noise models proposed by Handa et al. (2014).

Hence, if we are using a continuous action space, our motion planner should set the linear and angular velocities at each predetermined time interval. Furthermore, if we are using a discrete action space, an intermediary module must convert actions such as ‘FORWARD,’ ‘ROTATE LEFT’ to a series of intermediary poses so that the motion planner can deal with those poses similarly to the continuous action space case.

5.2.2 Inter-process communication

Here we comment on Inter-Process Communication (IPC) and why ROS (Open Robotics, 2018) was chosen for this necessity. In a simulator, all the modules, such as computer graphics management, physics simulation, and so on, can run in a single centralized process. However, in reality, the mobile base, digital camera, LiDAR, and Single Board Computer (SBC) all have independent internal computing devices that must cooperate, making IPC a necessity for our agent. Instead of implementing our own IPC approach, we opted to use ROS, since it was designed with this particular necessity in mind and it has great popularity in academic research (Mur-Artal and Tardós, 2017; Takaya et al., 2016; Vega and Cañas, 2019).

In essence, ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms (Open Robotics, 2018). ROS offers a message passing interface that provides inter-process communication and is commonly referred to as middleware. The ROS middleware provides these facilities:

- publish/subscribe anonymous message passing, handling synchronization.
- recording and playback of messages.
- request/response remote procedure calls.
- distributed parameter system.

By incorporating ROS in our project, we can abstract IPC and message synchronization and build and run individual programs that each handle a different task, such as drivers, image processing, Simultaneous Localization and Mapping (SLAM) and so on, that can be executed in different devices.

5.3 EXCHANGEABLE - OUR FIRST PIPELINE

With the building blocks of actuation control, IPC, SLAM, and motion planning, we built our first pipeline named EXchangeable. The name was inspired by its decoupled modular nature, where each module can be exchanged without much effort. This approach follows the ROS Navigation Stack structure, and it is composed of classical model-based algorithms. The global motion planner was the A* (Hart et al., 1968) and the local motion planner used for EXchangeable was the Dynamic Window Approach (DWA) (Fox et al., 1997).

This pipeline was the first step towards our goal of performing ObjectNav. In this stage, which still lacks some of the modules for solving ObjectNav, i.e., there is no dedicated module for identifying objects, but we can perform Point-Based Navigation (PointNav). A flowchart of our first pipeline is presented in Figure 5.2

During navigation, if the agent perceives itself as stuck, unable to continue traversing the path, a series of recovery behaviors were set. Those are some simple heuristics to resume navigation. First, obstacles outside of a custom specified region will be cleared from the agent’s

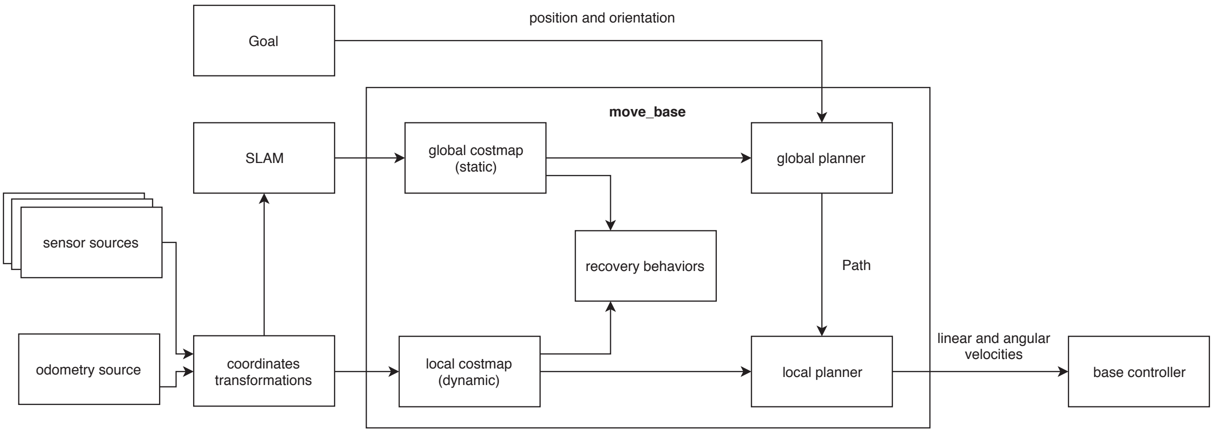


Figura 5.2: EXchangeable, our first pipeline. Given the input from sensors such as RGB-D camera, LiDAR (sensor sources), wheel encoder (odometry source), coordinates transformations are performed. After transformation, the input is forward to the SLAM algorithm, such as Real-Time Appearance-Based Mapping (RTAB-Map). Another input is the goal, a pose that the robot must reach. Given a goal, the global planner computes an unobstructed path, and then the local planner computes the local correction to remain in that path and the pertinent linear and angular velocities. While the robot plans and navigates to the goal, it performs SLAM using the transformed sensors and odometry data. The global and local cost map derives from the SLAM's occupancy map and a cost function that weights regions as navigable and non-navigable.

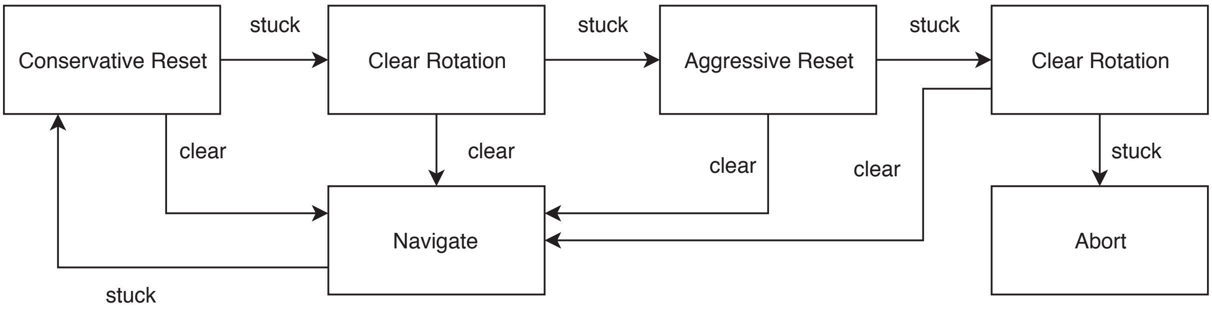


Figura 5.3: Recovery behaviors diagram. During navigation, the agent can perceive that it is "stuck" then trigger a series of operations starting from the conservative reset to recover personal space and continue navigating. In case all these actions fail, it will abort navigation.

cost map, resulting in a forced refresh of far off areas. Next, the agent will attempt an in-place rotation. This operation could free it from a contacting obstacle. If this action also fails, the agent will more aggressively clear from its current map everything within its sensoring range, removing all perceived obstacles inside of a rectangular region in which it can rotate in place, resulting in a forced refresh of the agent's "personal space". Another in-place rotation will follow this. If all this fails, the agent will consider its goal infeasible and abort navigation. Figure 5.3 illustrates this process.

EXchangeable was intended as a functional intermediary solution and was employed in some basic experiments to assert if the proof-of-concept robot VRIBot could perform autonomous navigation towards an objective. Our early experiments and results are in section 6.1.

5.4 AUTOCRAT - OUR SECOND PIPELINE

Inspired by the work of Sax et al. (2020), we proposed our second pipeline named AUTOcrat. The name was inspired by the fact that a single module does most of the work, see Figure 5.4, constituted by a single end-to-end approach, a policy network, instead of specialized modules cooperating. Full SLAM was replaced by simple odometry estimation, and the path planner is a discrete local planner. The outputs are no longer velocities but a discrete action towards the episode's objective. This action represents a pose within the agent's sensoring range.

AUTOcrat's Policy Net was modeled as a partially observable Markov decision process (POMDP) and employ a LSTM a type of Recurrent Neural Network (RNN) to handle the hidden state, similarly to the approach ofMousavian et al. (2019); Wijmans et al. (2019). In some capacity the LSTM replaces the memory manager employed by RTAB-Map. LSTMs are discussed in more detail in section 2.13.

Our policy network was trained using Immitation Learning similarly to Mousavian et al. (2019); Shen et al. (2019) instead of Reinforcement Learning (RL). The policy does not learn by sparse rewards; instead, it learns by imitating a greedy path planner oracle for each step. However, our learning method has a fundamental difference from those works. In their work, due to holonomic movement, different feasible translations are possible in a step (e.g., forward, backward, left, right), so their policy learns a regression cost for each action, i.e., a regression problem using root-mean-square error loss. Due to a different embodiment, our translation is limited to a single direction since our agent is nonholonomic, so we opted to model the problem as classification instead of regression. We use a cross-entropy loss, where the discrete action provided by the oracle is the only correct and must be followed. For a better generalization, our sampling works similarly to DAgger (Ross et al., 2011), where some actions follow the oracle, the master, and some actions follow the learned policy, the student, this unsure that the policy learns how to recover from a previous mistake instead of only learning the consequence of correct actions, as in a cloning behavior approach.

Instead of only using multiple mid-level representations, we opted for a Residual Network (ResNet)-50 (He et al., 2016) for a more generic feature representation and a single pre-trained mid-level encoder for semantic segmentation. ResNet-50 will learn during training a lower-dimensional data representation of the RGB-D data according to this task, taking advantage of residual connections instead of multiple encoders pre-trained in different problems. Mousavian et al. (2019) also used concatenated multi-modal visual information on their approach.

A proximity sensor was used to model the agent's sonar and contains a single measurement within the [0,1] range. The simulator's episodic position and the yaw orientation are given as separate inputs and compose the agent's current pose. The egocentric RGB-D is also provided by the simulator in a separate matrix for RGB and one for Depth and was configured to match our embodiment. Each object class is represented by an integer ID matching the class name. Thus, a constant target object class ID is fed to the agent for every step based on the episode's settings, see Table 6.4. The previous prediction is also used as a feedback input to keep a recent memory of previous decisions taken. Every input is encoded using its own embedding procedure, as can be seen in figure 5.4. After the embedding step, the encoded representations are concatenated into a single 1D array of size 1152 and fed to the LSTM cells along with the current hidden state. The output of the LSTM is a 1D array with a size of 512 and can be seen as the resulting sensor fusion. This is the last data representation before the prediction. This array enters the final fully connected layer. Thus, the output is an array with a score for every possible discrete action. In this case there are 4 discrete actions: ['STOP', 'MOVE FORWARD', 'ROTATE CLOCKWISE', 'ROTATE ANTI-CLOCKWISE']. Finally, the prediction with the highest score value is chosen

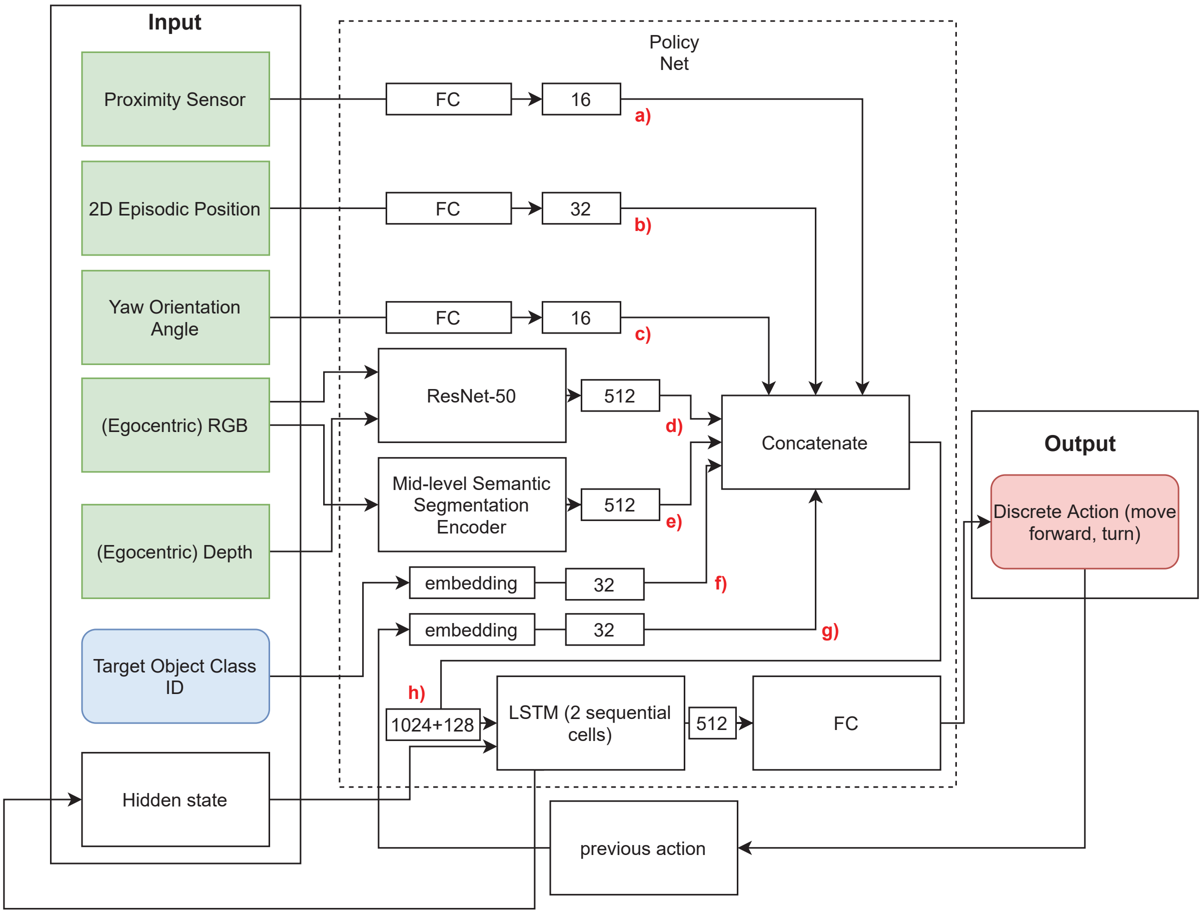


Figura 5.4: AUTOcrat, our second pipeline. In green are the agent's observations at time t. In blue is the target object class ID on the current episode. This ID is constant for every step of the episode. In red, the output is the predicted discrete action. FC stands for Fully Connected layer. In a) is the 1D array of size 16 resulting from the Fully Connected layer of the proximity sensor. In b) is the 1D array of size 32 resulting from the Fully Connected layer of current 2D Cartesian position using the episodic coordinate system. In c) is the 1D array of size 16 resulting from the Fully Connected layer of the agent's current yaw angle in radians. In d) is the 1D array of size 512 resulting from the ResNet-50 encoding of the RGB and Depth observations. In e) is the 1D array of size 512 produced by the Mid-level Semantic Segmentation Encoder using the RGB observation. In f), the class ID is embedded using a lookup table to a 1D array of size 32. In g) the previous prediction is fed again to the model to keep a recent memory as input. This representation is initialized with 0 and is encoded using a lookup table to a 1D array of size 32. The modules a,b,c,d,e,f,g, are concatenated into a single 1D array of size 1152 subsequently fed to the LSTM cells together with the current hidden state. The output of the LSTM is a 1D array of size 512 that is fed to the final Fully Connected layer to predict the action. This pipeline was inspired by the works of Wijmans et al. (2019); Sax et al. (2020)

as the agent's discrete action in this simulation's step. Forward motion is made in steps of 0.25 meters and rotation of angles of 30 degrees.

Every layer is updated during training with the exception of those belonging to the mid-level semantic segmentation encoder where we used frozen weights from its pre-train similarly to Shen et al. (2019).

5.5 BEYOND - OUR THIRD PIPELINE

Our third pipeline in this work was named BEyond. This pipeline was intended to go beyond any other of our previous solutions. In this pipeline, we explore a hybrid solution of the first (EXchangable) and second pipeline (AUTOcrat). A hybrid approach was inspired by the work of (Chaplot et al., 2020). The BEyond pipeline was developed using the iterative and incremental design pattern leading to three variants. BEyond with Oracle (section 5.5.3) the most simple, BEyond Intermediary (section 5.5.4) with semantic segmentation and local planner, BEyond Complete (section 5.5.5), with SLAM, semantic segmentation and local planner.

The main concept explored here is semantic mapping and a goal prediction module. Instead of an end-to-end policy, we fallback to the modular approach. We introduce a goal prediction module that uses an episodic top-down semantic map, the trajectory, and the target object class ID to predict a point goal, see section 5.5.2. This allows that solutions for PointNav can be reused for ObjectNav. By employing the more mature PointNav solutions for navigation, we intend to mitigate problems like collisions, a major issue with the AUTOcrat solution, see section 6.2.2.

Semantic segmentation received its own separate module, a dedicated neural network for this purpose. In this case, YOLACT++ (Bolya et al., 2020) is a state-of-the-art approach that explores concepts from the YOLOv3 achitecture (Redmon and Farhadi, 2018) to speed up segmentation while preserving quality. This approach provides a much more accurate segmentation than the encoder-decoder of the second pipeline. Figure 5.6 presents the qualitative difference between the semantic segmentation using YOLACT++ and the one using the mid-level encoder-decoder in our research lab physical room [1]. Another major difference is that the egocentric semantic segmentation prediction is projected using the depth image into a top-down semantic map, see section 5.5.1. Figure 5.5 presents our BEyond pipeline on its Complete variant, for further details see section 5.5.5.

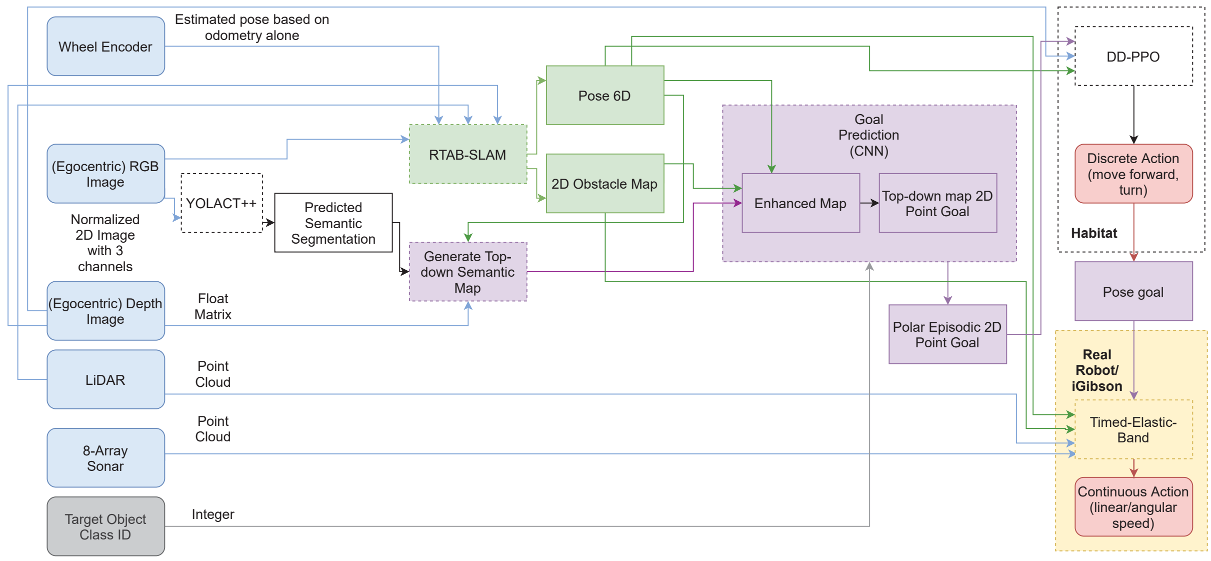
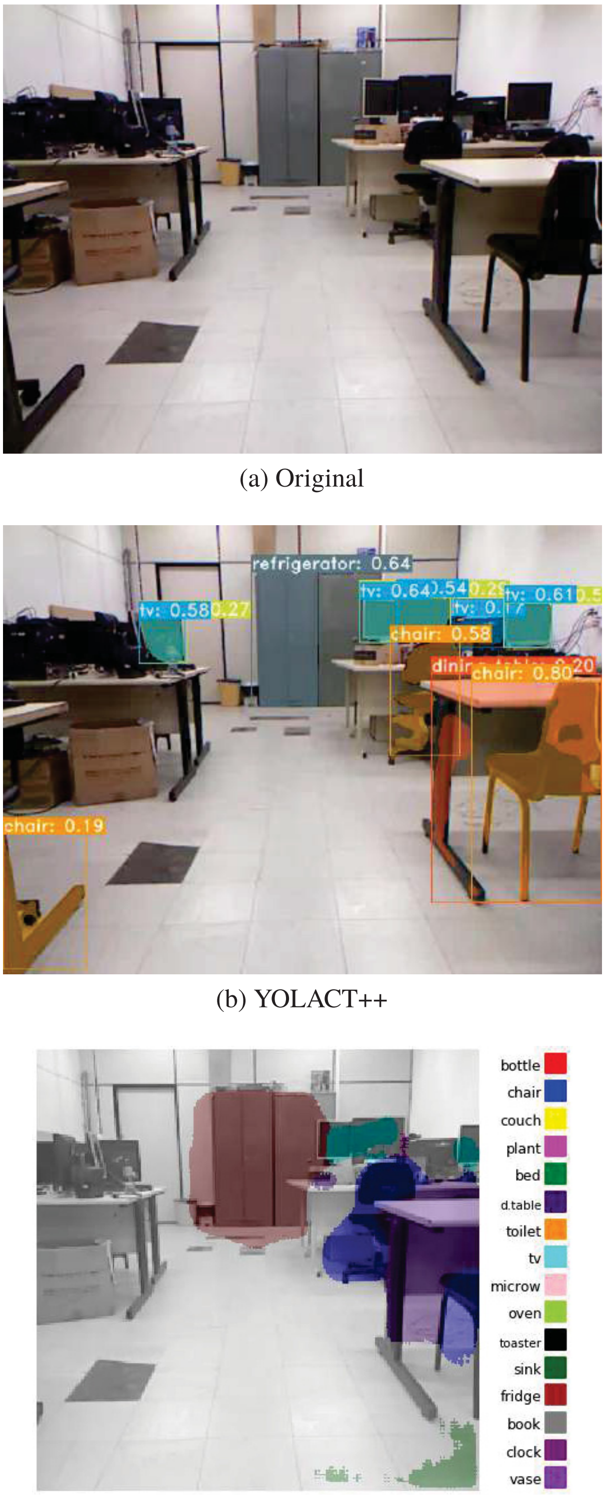


Figura 5.5: BEyond flowchart. Boxes with dashed outlines represent different algorithms, and data are represented with a solid outline. In blue is the input provided by the sensors at time t in gray the target class ID provided by the episode's settings. Green represents the RTAB-SLAM module and its outputs. Purple represents our BEyond modules and pertinent data. Red represent predicted output data. BEyond can operate with a discrete action space such as in the Habitat Simulator using the Decentralized Distributed Proximal Policy Optimization (DD-PPO) algorithm and a continuous action space such as necessary for the real robot or the iGibson Simulator using the motion planner Timed-Elastic-Band. The module in yellow is not necessary when dealing only with a discrete action space.

Cadena et al. (2016) classified semantic map models into three main categories: (I) SLAM helps Semantics; (II) Semantics helps SLAM; (III) Joint SLAM and Semantics inference.

[1] Video is available online at: `https://github.com/VRI-UFPR/BeyondSight`



(a) Original

(b) YOLACT++

(c) Encoder-Decoder

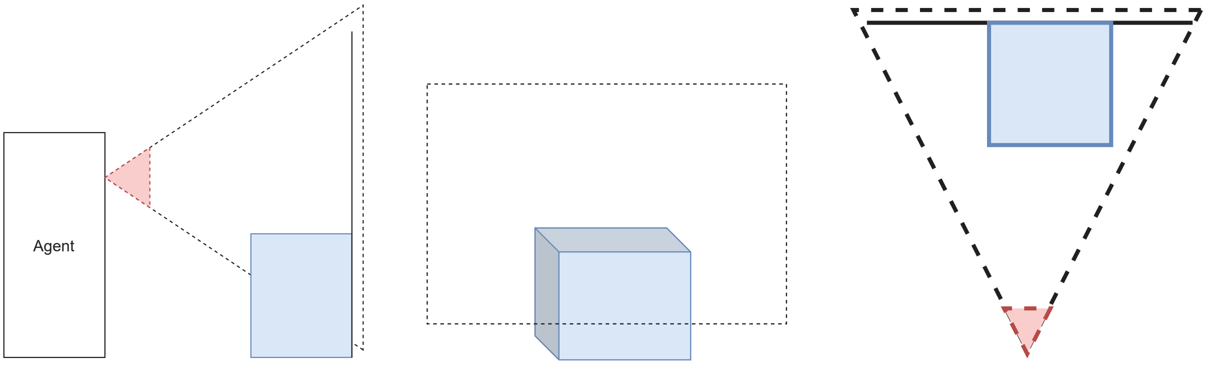
Figura 5.6: Qualitative comparison between YOLACT++ and the mid-level Encoder-Decoder for Semantic Segmentation. The image frame is from the video recording of an episode inside our lab—the same episode used for the early qualitative mapping experiment. The colored bounding boxes show the instance's class and confidence. Coloring is per class, not per instance. In subfig (b), the monitors have a high classification intersection of tv (blue) and laptop (green). The metal cabinet was misclassified to the refrigerator/fridge mainly because there is no cabinet class on the COCO classes.

Our approach falls into SLAM helps Semantics, where the localization provided by the SLAM is the basis for the correctness of our depth projection to create the top-down semantic map. Initially, we thought of coupling SLAM with semantic segmentation into a single module, which would be more accurate, e.g., the top-down semantic map would have loop closure, and SLAM would have landmarks derived from the segmentation. However, we decided to maintain the modules decoupled where SLAM is performed first, then the estimated agent's pose is used to produce the top-down semantic map. This decision was to preserve modularity, making it easy to exchange the SLAM or/and the semantic segmentation algorithms.

5.5.1 Top-down semantic map

This map is calculated using the agent's current pose in the episodic coordinate system, the egocentric depth image from the RGB-D sensor, and the RGB image's semantic segmentation. The map is squared and represent 51.2 meters on each axis. Each cell size represents 0.10 meters, resulting in 512x512 cells on the map. The semantic map has 32 channels, 30 to encode semantics and 2 to encode the episode's trajectory. The channels were chosen based on the objects' sampling within the scenes, see 4.7, where the class with most samples is the chair with 1230 instances, and the 30th class in descending order is laptop has 22 instances. The shape of the map is [32, 512, 512].

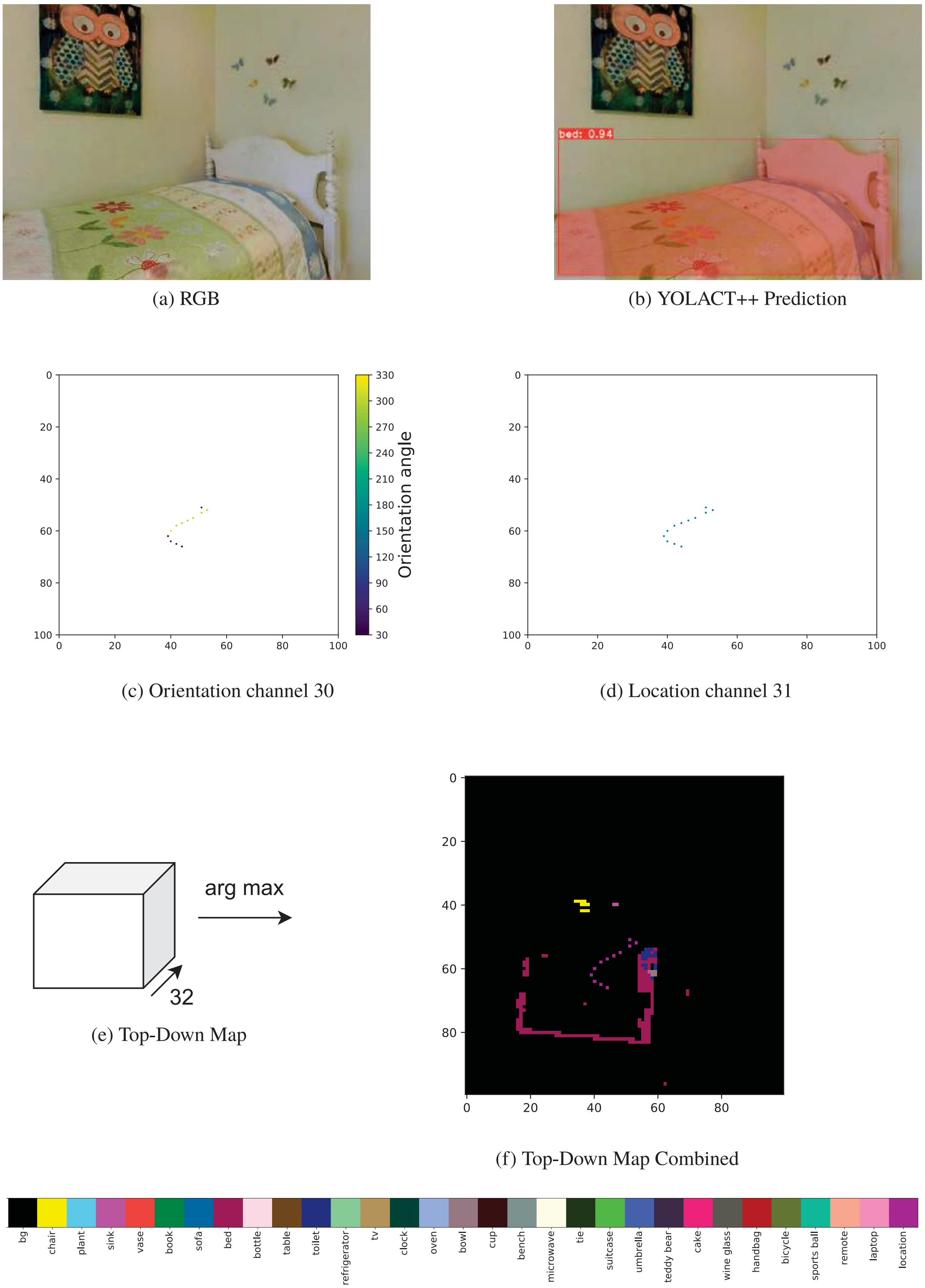
Figure 5.7 illustrates the camera projection to achieve a top-down view. Figure 5.8 illustrates the top-down semantic map during execution.



(a) Side View (b) Front View (c) Top View

Figura 5.7: Multiple viewpoints to illustrate how the perceived object is projected per step to a top-down view. The red triangle represents the near clip region of the frustum, meaning that anything within that range is discarded.

The map is initialized in such a way that the agent always starts at the center cell for each episode, i.e., coordinate [255,255] maps the episode's origin. Each semantic pixel is mapped according to its depth to a cell in the map, this is done by using projection since the RGB-D sensor's depth range, and field of view are known. Each cell holds the summed score of the mapped semantic pixels, e.g., for standard occupancy, each pixel would count as one, but here we use the confidence per pixel of the prediction that is between [0,1]. The map is normalized so that each cell value falls between [0,1]. Channel 30 encodes the agent's orientation at each cell of the discrete trajectory, and channel 31 the location in binary. At each step t the first 30 channels of the first dimension are updated using $max(map_{t-1}, map_t)$, at time step 0 the map is initialized with zeros. The last two channels that encode orientation and location update a single cell per step, the cell that maps the agent's current position. The entire map is reset to zeros when an episode ends.



(a) RGB

(b) YOLACT++ Prediction

(c) Orientation channel 30

(d) Location channel 31

(e) Top-Down Map

(f) Top-Down Map Combined

Figura 5.8: Top-down semantic map diagram. Step 32nd of an episode. For ease of visualization, the 512x512 map was center cropped to 100x100. For ease of visualization, the 32 separate channels were exemplified by the 30th in subfigure (c) and the 31st in subfigure (d). Subfigure (f) is the image resulting from stacking channels using the arg max function. The cells were colored-coded by the semantic class with the highest summed value of projected scores within that cell coordinate.

5.5.2 Goal Prediction Network

We developed this CNN architecture based on the Neural SLAM (Zhang et al., 2017). Unlike the Neural SLAM, which predicts a local occupancy map and the agent current pose to generate an episodic top-down occupancy map, our network predicts a 2D point goal towards a semantic instance of the input class ID. Our architecture uses Depthwise Separable Convolutions (Sandler et al., 2018), which reduce the number of necessary trainable parameters within the model. The use of ReLU activation before the convolutions was inspired by Szegedy et al. (2016).

Module	Type	Kernel Size	Stride	Padding	Output
Visual Embedding	ReLU	-	-	-	32x512x512
	Conv	1x1	1	0	64x512x512
	Conv	3x3	2	1	64x256x256
	BatchNorm	-	-	-	64x256x256
	ReLU	-	-	-	64x256x256
	Conv	1x1	1	0	128x256x256
	Conv	3x3	2	1	128x128x128
	BatchNorm	-	-	-	128x128x128
	ReLU	-	-	-	128x128x128
	Conv	1x1	1	0	256x128x128
	Conv	3x3	2	1	256x64x64
	BatchNorm	-	-	-	256x64x64
	ReLU	-	-	-	256x64x64
	Conv	1x1	1	0	128x64x64
	Conv	3x3	2	1	128x32x32
	BatchNorm	-	-	-	128x32x32
	ReLU	-	-	-	128x32x32
	Conv	1x1	1	0	64x32x32
	Conv	3x3	2	1	64x16x16
	BatchNorm	-	-	-	64x16x16
	ReLU	-	-	-	64x16x16
	Conv	1x1	1	0	32x16x16
	Conv	3x3	2	1	32x8x8
	BatchNorm	-	-	-	32x8x8
	Flatten	-	-	-	2048
Orientation Embedding	FC	-	-	-	8
	ReLU	-	-	-	8
Object Goal Embedding	Lookup Table	-	-	-	8
Final	Concat	-	-	-	2048+8+8=2064
	FC	-	-	-	512
	ReLU	-	-	-	512
	FC	-	-	-	2
	Sigmoid	-	-	-	**2**

Tabela 5.1: Our Goal Prediction Network. This module receives the top-down semantic map, the episode's object goal ID, and the agent's starting orientation. This module's outputs, denoted with the color **magenta**, are (x,y) map cell coordinates normalized within [0,1].

This module receives the top-down semantic map, the episode's object goal ID, and the agent's starting orientation. Table 5.1 displays each layer and the operations of our Convolutional Neural Network (CNN). This module's outputs are map cell coordinates normalized within [0,1] that subsequently are converted to discrete cell coordinates. The map coordinates are transformed into coordinates in the episodic coordinate system and are used as the PointNav objective for the sub-sequential modules.

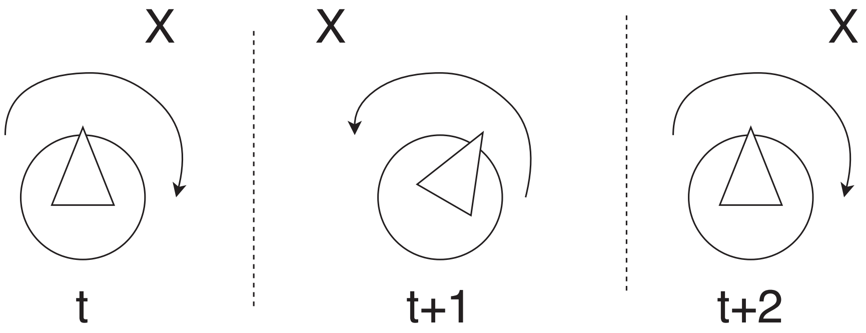


Figura 5.9: Illustration of the infinite loop caused by an abrupt change on the point goal. X represents the point goal. The circle represents the agent. The triangle represents orientation. The arrow represents the action to be taken at each step.

For temporal consistency the current goal is updated at $t \ mod \ 8 = 0$ or $euclidean\_distance(current\_goal, agent\_position) < success\_distance$ with the mean of all previous predictions on the episode. This heuristic for temporal consistency was introduced to avoid a case where the agent got stuck in an infinity loop of repeated rotations because the point goal changed abruptly to a different direction in relation to the agent. After the agent's rotation, the prediction returned to the previous location, see Figure 5.9. Evaluating a temporal encoder like the LSTM within the goal prediction module was left for future works, and it is briefly discussed at section 8.1.

5.5.3 BEyond with Oracle - First BEyond Variant

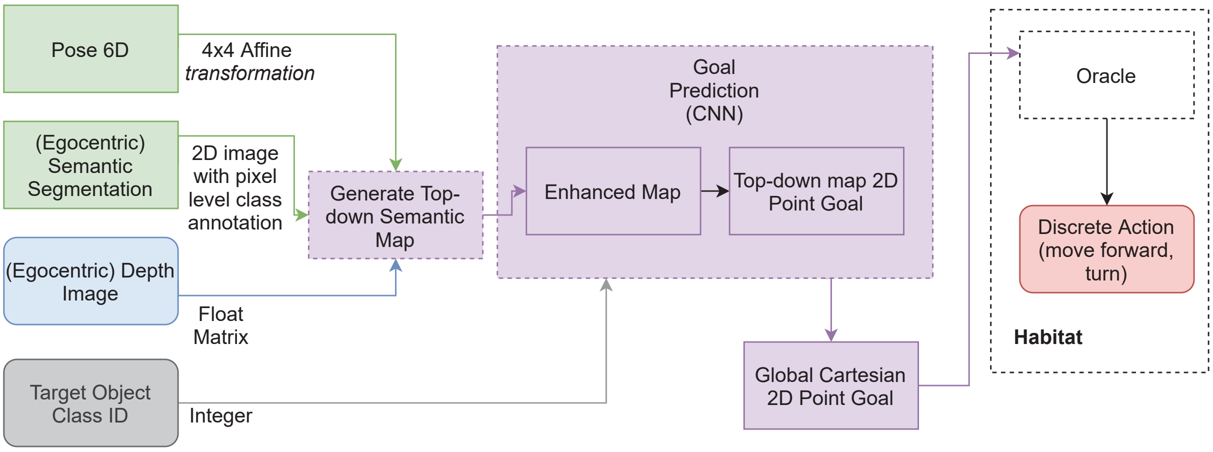

Pose 6D
4x4 Affine transformation
(Egocentric) Semantic Segmentation
2D image with pixel level class annotation
(Egocentric) Depth Image
Float Matrix
Target Object Class ID
Integer
Generate Top-down Semantic Map
Goal Prediction (CNN)
Enhanced Map
Top-down map 2D Point Goal
Global Cartesian 2D Point Goal
Oracle
Discrete Action (move forward, turn)
Habitat

Figura 5.10: BEyond With Oracle flowchart. In green are the simulation ground truth data. In blue is the depth image observed at time t. In gray is the target class ID provided by the episode's settings. In red is the output containing the predicted action.

To evaluate the goal prediction module in a disentangled way, we first evaluate it using ground truth pose, RGB-D data and semantic segmentation provided by the simulator, and an oracle for the path navigation. The oracle is a greedy variation of the A^* algorithm implemented in the Habitat simulator with perfect geometric knowledge of the scene but does not have any access to the semantics. We named this variant BEyond With Oracle, see Figure 5.10. This variant aims to evaluate the ability to identify objects and deal with semantics, decoupled from traditional navigation processes such as motion planning and object avoidance.

5.5.4 BEyond Intermediary - Second BEyond Variant

Unlike the BEyond with Oracle, this variant uses the YOLACT++, pre-trained with the COCO dataset, to predict the segmentation and a DD-PPO (Wijmans et al., 2019) module pre-trained for PointNav on the Gibson dataset, see figure 5.11. The DD-PPO was trained to use an episodic Polar coordinate system instead of a Cartesian one, so the point goal predicted by the goal prediction module is converted accordingly. The DD-PPO module was chosen due to its performative results reported at (Wijmans et al., 2019) and because it was used as the baseline for the Habitat-Challenge 2020. This is similar to the Habitat-Challenge ObjectNav conditions, where only egocentric RGB-D and the agent's pose are provided by the simulator.

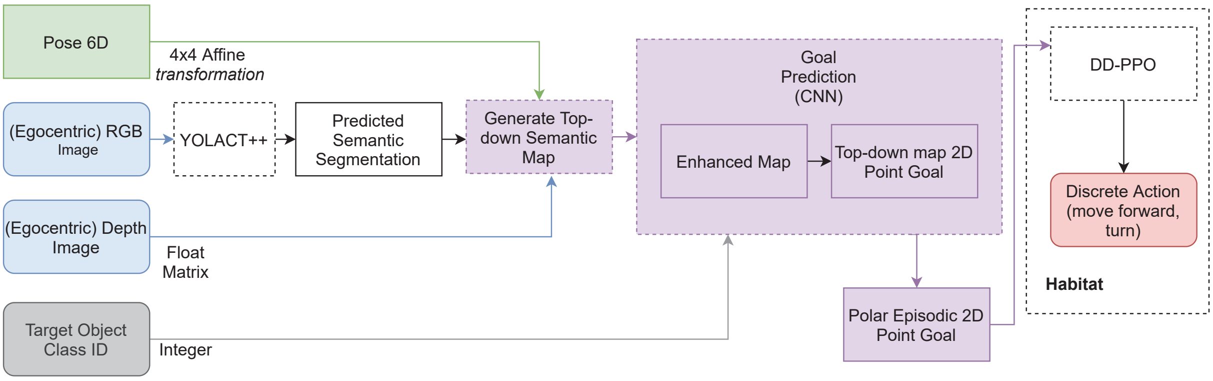


Figura 5.11: BEyond Intermediary flowchart. In green are the simulation ground truth data. In blue are the RGB and depth images observed at time t. In gray the target class ID provided by the episode's settings. In red is the output containing the predicted action.

5.5.5 BEyond Complete - Third BEyond Variant

BEyond Complete is the version to be used with ROS on our agent, see figure 5.5. It is intended as a proof-of-concept of how the BEyond pipeline operates on a realistic scenario, i.e., without any privileged information of a simulator, only the agent's own sensors. It uses SLAM to estimate the agent's pose and occupancy map and the YOLACT++ module to perform semantic segmentation. Motion planning uses DD-PPO for discrete action as in the BEyond Intermediary, and Timed-Elastic Band (TEB) (Rösmann et al., 2012) for continuous actions, unlike any of the previous variants.

The map generated by SLAM is only reused by the TEB local planner. A discrete action predicted by the DD-PPO module can be performed using the TEB planner by transforming the action into a pose goal. Coupling between the SLAM map and the top-down semantic map was left for future works. LiDAR and Sonar data are used within the SLAM and TEB modules.

5.6 CHAPTER SUMMARY

In this chapter, we introduce our proof-of-concept robot VRIBot, which follows unicycle kinematics with three wheels, where electric motors control two, and one is a passive caster wheel. Our three proposed pipelines were also presented. EXchangeable our early attempt, using model-based algorithms to perform integration of multi-sensor data fusion, SLAM, path planning, and motion planning to our proof-of-concept robot. AUTOcrat our first attempt to perform ObjectNav. This pipeline employs an end-to-end policy network to predict discrete actions directly from monocular egocentric RGB-D observations. Finally, our most robust approach, BEyond, was introduced. Due to the highly modular design philosophy, the pipeline has three variants, BEyond with oracle, BEyond Intermediary, and BEyond Complete, each with more components than the previous one. Table 5.2 presents a summary of the three approaches. In the following chapter, we present the experiments to evaluate each of our proposed pipelines.

Pipeline	**Tasks**	**Environment**	**Main Modules**
EXchangeable	PointNav Only	Gazebo Real World	RTAB-Map DWA
AUTOcrat	ObjectNav Only	Habitat	Mid-level Semantic encoder ResNet-50 LSTM ***Policy Network***
BEyond	PointNav ObjectNav	Habitat iGibson	RTAB-Map DD-PPO TEB YOLACT++ ***Top-down Semantic Map*** ***Goal Prediction Network***

Tabela 5.2: Table summarizing the differences between the three proposed approaches. Modules in italic and bold letters are our proposed algorithms.

6 EXPERIMENTS

In this chapter, we present the experiments performed with the models proposed in this work and their results. The EXchangeable pipeline's experiments are the basis for the entire system, where we experimented with how traditional model-based algorithms could be used with our proposed VRIBot embodiment to perform point-based navigation and obstacle avoidance in indoor environments. Experiments were performed both in simulated environments and the real world. The AUTOcrat experiments were intended to verify the performance of our policy in Object-Goal Navigation (ObjectNav). In the BEyond pipeline, the most robust of our proposed approaches were evaluated against the baseline and our other proposed methods.

6.1 EARLY EXPERIMENTS

We refer to the experiments performed with the EXchangeable pipeline as the early experiments. We divided our early experiments into two types: the ones performed in the Gazebo simulator, and those in the real world, performed on our research lab Vision, Robotics and Image (VRI)[1] at the Federal University of Paraná.

6.1.1 Early Experiments - Simulated World Zero

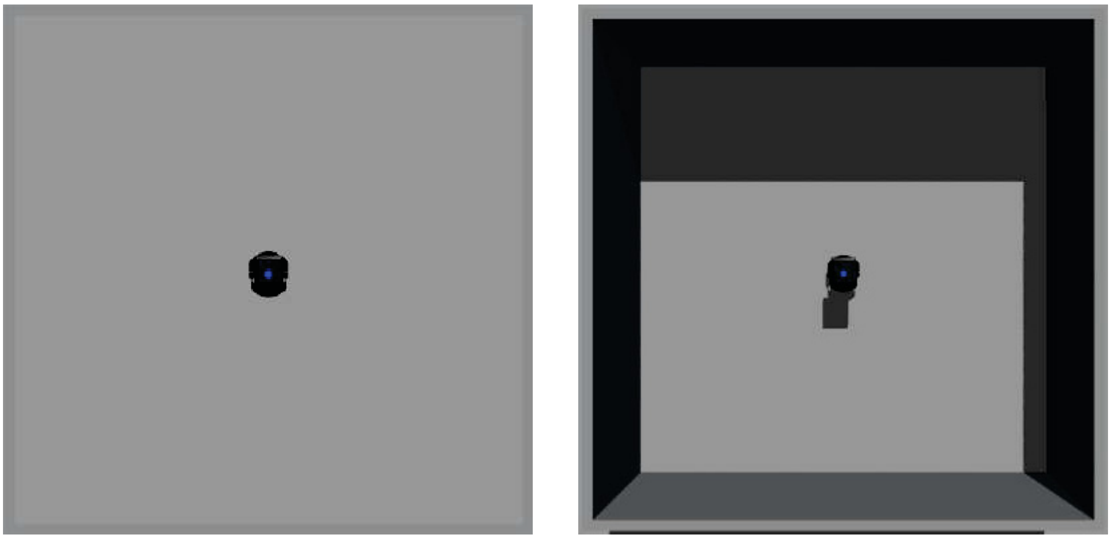
The first experiment was performed in the simple world, here named Simulated World Zero. It was designed as a scenario where the agent could perform SLAM and autonomous navigation. This world contains an untextured gray ground plane at 0 height with 6x6 meters. The world contains a single room with four perfectly untextured gray smooth walls, each with 5.1x0.1x2.5 (length, thickness, height) in meters. The agent was spawned in the center of the room at the ground truth world coordinate system origin. Thus, the episodic coordinate system origin matches the ground truth world coordinate system. Figure 6.1 illustrates the initial state of this episode.

There are no obstacles except the four walls. Note that the world is symmetrical on axis x and y. The scene contains a directional light source to project shadows in the world, and it is located at the x, y origin and 10 m in z. The light is oriented by the vector $[-0.5, 0.1, -0.9]^T$. The Gazebo simulator performs simple Phong Shading.

This world prevents Real-Time Appearance-Based Mapping (RTAB-Map) from finding appearance-based loop closures due to the lack of distinctively visual features. The trajectory was defined by a set of goal poses in the episodic coordinate system. First, an in-place 360° rotation, then four times a 2x2 square route around the origin, see figure 6.2.

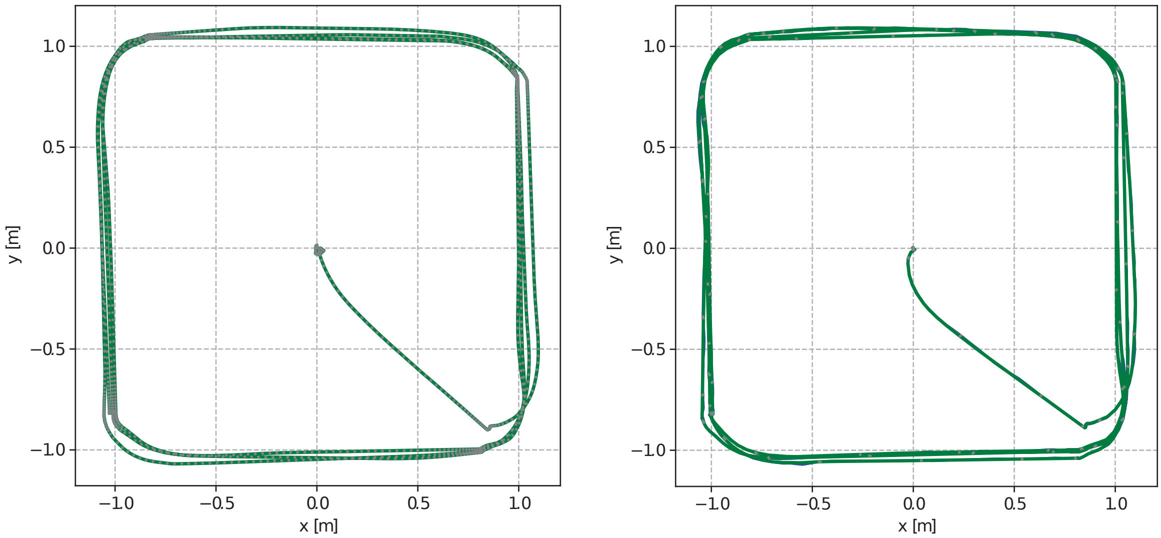
The same set of goals were successfully reached by the agent using two different types of odometry sources in two separate episodes. In episode 0, the agent had the RGB-D, LiDAR, sonar, and wheel encoders as the observation set. In episode 1, the agent had only the RGB-D, sonar, and LiDAR as the observation set, where the LiDAR was used for mapping and as the odometry source. The reached poses were within the error margins of 0.1 meters for the position and 0.5 radians for orientation. Note that odometry using the LiDAR as the source would not perform well in a large empty space, where the measurements are too sparse.

[1] `https://web.inf.ufpr.br/vri/`



(a) Orthographic View (b) Perspective View

Figura 6.1: Simulated world zero's initial state from a top view.



(a) Episode 0, estimated trajectory using odometry from wheels' encoders

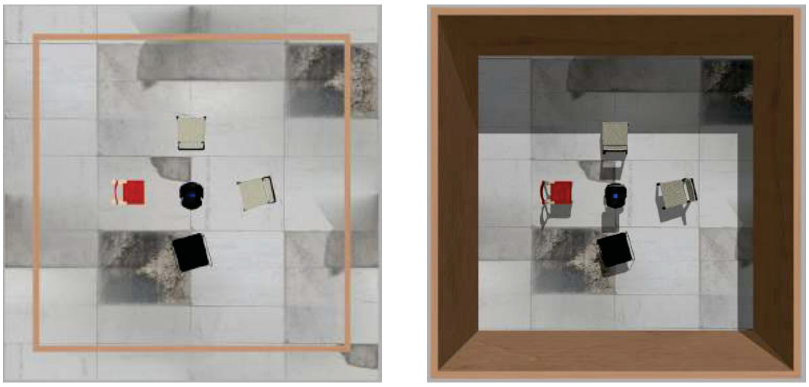
(b) Episode 1, estimated trajectory using odometry from LiDAR

Figura 6.2: Top view of the episodes' estimated trajectory in simulated world zero.

The experiment demonstrated that the EXchangeable pipeline could perform a series of PointNav tasks successfully within a simple simulated environment with large visually uniform areas using either the wheel encoder or the LiDAR as the odometry source.

6.1.2 Early Experiments - Simulated World One

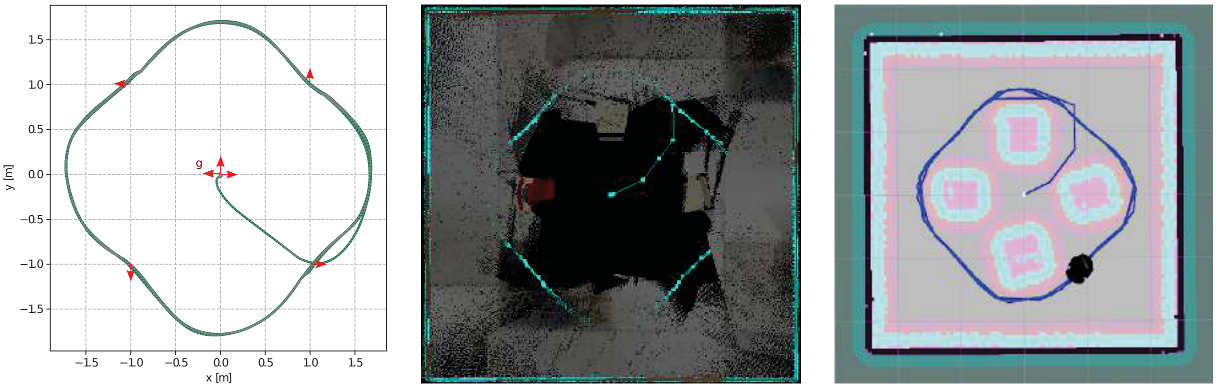
In this experiment, we tested if the agent was capable of obstacle avoidance in a simple scene. Here, the ground plane is textured with floor tiles, and the walls are covered with a cardboard-like texture. Four chairs were added as obstacles; three of them possess different polygonal meshes and material colors, and the fourth is a duplicate with a different pose. All of them are lower than the LiDAR height in the robot embodiment, meaning that only the RGB-D camera and the sonar can perceive them. The robot must reach the same goals as in the previous experiment and avoid collisions. The same light source as the previous experiment was used here, see figure 6.3. The resulting mapping and the trajectory performed are shown in Figure 6.4.



(a) Orthographic View (b) Perspective View

Figura 6.3: Simulated world one's initial state from a top view.

The agent reached the goals successfully without collisions during the episode. Figure 6.4 displays the episode's estimated trajectory, the point cloud computed by the Simultaneous Localization and Mapping (SLAM) module and a 2D top view plot with the occupancy map, where black points are occupied, and gray is navigable, the cost map, where the obstacles are inflated by a cost function and offer an 'unsafe zone' to approach for the path planners is represented in purple and cyan. The cost function rates a trajectory based on the perceived obstacles. It either returns negative costs if the trajectory passes through an obstacle or zero else.



(a) Episode's estimated trajectory (b) Point Cloud obtained at the episode's final state, top view (c) 2D Plot at the episode's final state

Figura 6.4: Episode's trajectory estimated using odometry from the wheel's encoders in simulated world one. Subfigure (a) displays the performed trajectory in green, and the red arrows represent the set of goals. Subfigure (b) shows the point cloud of the world at a top orthographic view. The pose graph is highlighted in cyan, also in cyan is the LiDAR data, here limited to the walls because the chairs are lower than the LiDAR. Subfigure (c) shows the agent at the episode's final state. The pose graph is seen in blue, in purple/cyan, the global cost is computed by the navigation stack.

6.1.3 Early Experiments - Real World

To assert the trajectory quality in the real world, we needed a ground truth external to the agent for comparison. For this purpose, we used a ceiling-mounted camera, more specifically a Sony PlayStation 3 Eye camera, and used its recordings to performed visual odometry estimation. The videos were recorded with a 640x480 resolution and in 60 frames per second. Camera calibration was performed to extract the intrinsic camera parameters. Instead of the common checker grid pattern, we used a CharUco board, a combination of chessboard pattern, and arUco, a type of square-based fiducial marker (Munoz-Salinas, 2012; Garrido-Jurado et al., 2016; Romero-Ramirez et al., 2018). The main advantage of the CharUco board over the traditional checker pattern is that since each fiducial is uniquely identified, it is less affected by partial occlusion or rotations. The board is shown in figure 6.5.

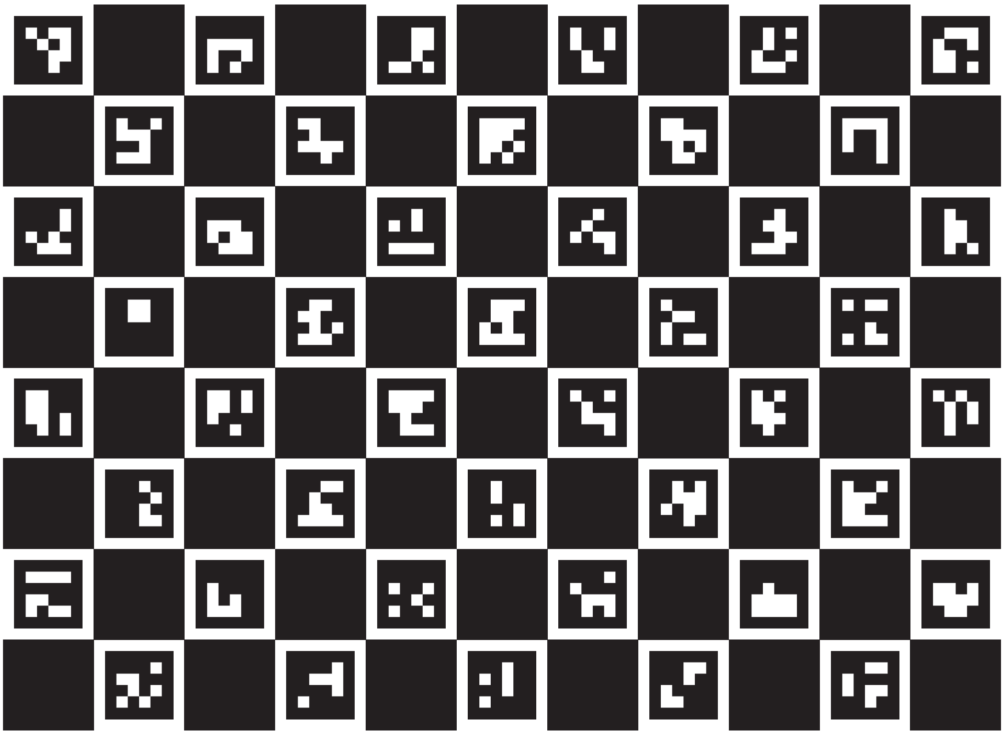


Figura 6.5: CharUco board used for camera calibration, a.k.a estimation of intrinsic camera parameters. Each checker length is 15mm, and the dictionary used was the 4x4. The board has 200x150 mm in an 11x8 layout.

The visual odometry was computed using the pose estimation of three arUco markers of 15x15 cm on top of the robot. The markers used the 4x4_100 dictionary (OpenCV, 2015), and their IDs were 0, 1, 2. Their placement on top of the robot was known and co-planar. An issue for the comparison between the visual odometry produced by the ceiling-mounted camera and the wheel odometry was time synchronization. The error estimation is time-sensitive, and each odometry operates in a different frequency. The camera sampling rate is 60 Hz, while the wheel has a mean sample frequency of 10 Hz.

6.1.4 Early Experiments - Real World Zero

In this experiment, the robot was enclosed by a rudimentary setup of cardboards and paper hold together by pipes with the dimensions 4x3.5x1.5 (x width, y depth, z height). The agent had to rotate in place 360° then perform a rectangle of 2x1 meters. Figure 6.6 illustrates the episode's estimated trajectory.

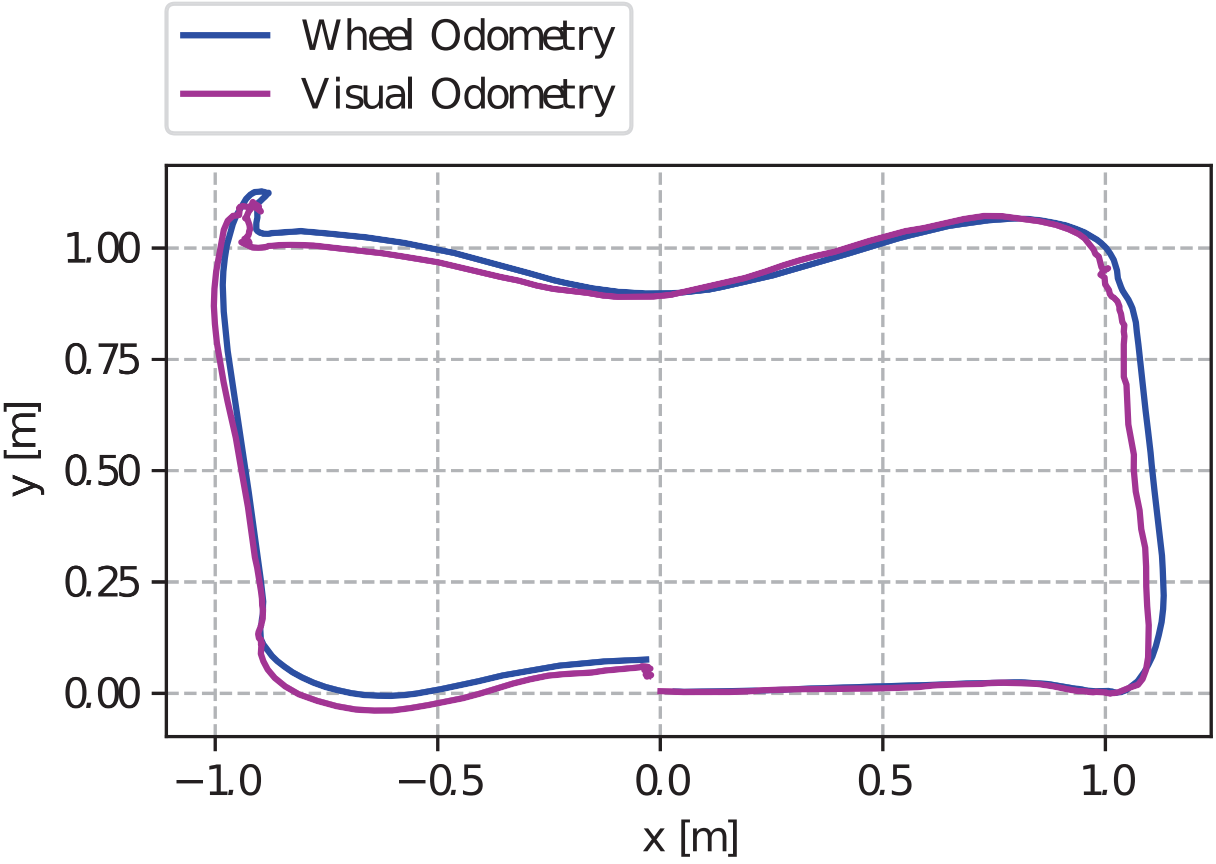
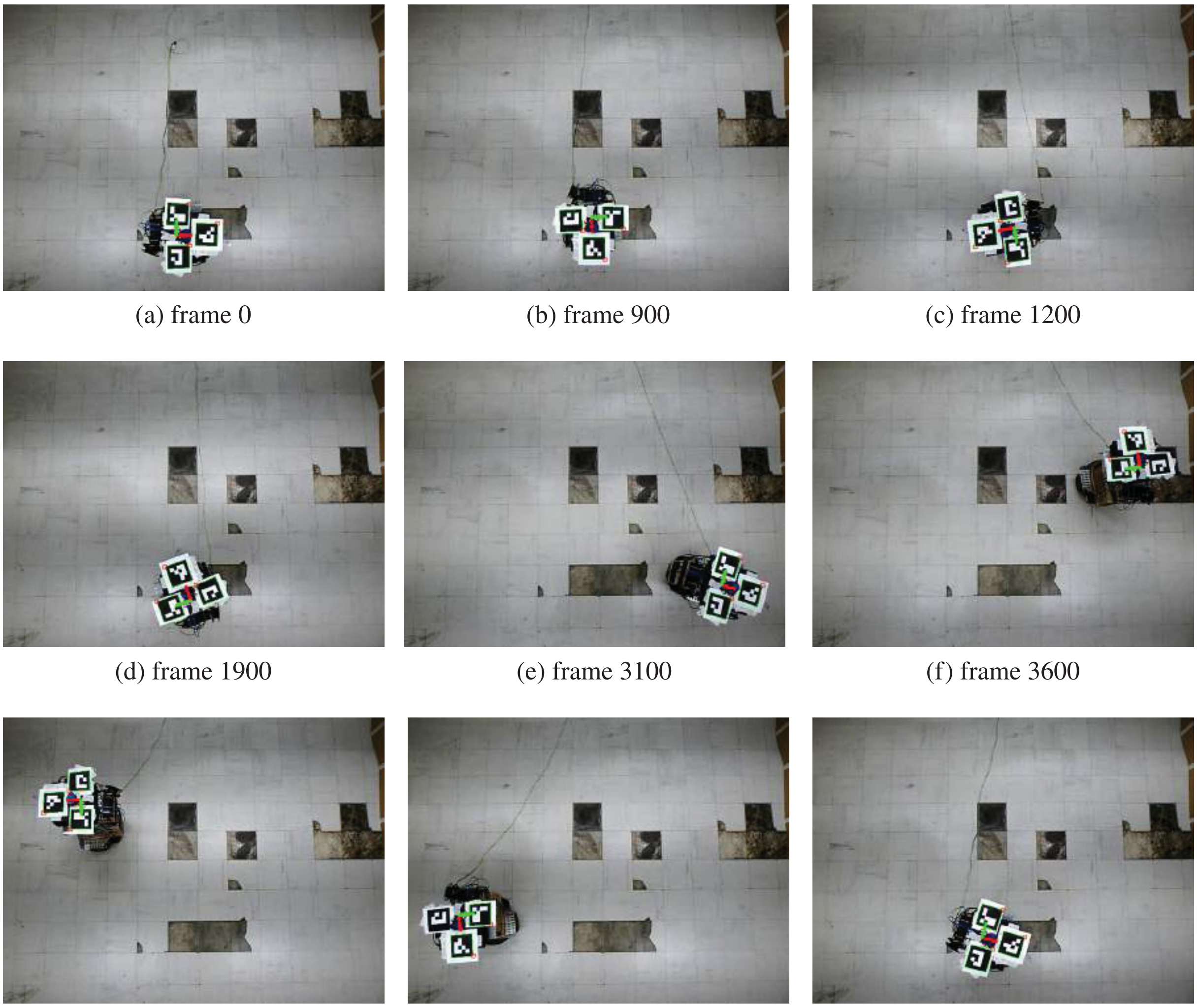


Figura 6.6: Episode's estimated trajectory in real-world zero. Visual Odometry was computed using a ceiling-mounted camera. Wheel Odometry is derived from the agent's wheels' encoders.

The agent successfully reached the goals but performed a more curved trajectory than in the previous simulated experiments. Figure 6.7 displays some of the captured frames by the ceiling-mounted camera during the experiment with the pose estimation superimposed [2]. Note that both the wheel odometry and visual odometry had some degree of imprecision within the margin of a few centimeters, even so, the visual odometry was used as the ground truth data because it relied on a camera independent of the agent on a fixed location on the ceiling.

[2] Video is available online at:`https://github.com/VRI-UFPR/BeyondSight`



(a) frame 0

(b) frame 900

(c) frame 1200

(d) frame 1900

(e) frame 3100

(f) frame 3600

(g) frame 4000

(h) frame 5100

(i) frame 6400

Figura 6.7: Frames captured by the ceiling-mounted camera from the trajectory of Real World Zero. Pose estimation was superimposed, the small red square represents the orientation of each marker, the green quadrilateral represents the detected maker edges, and the green and red axis represent the position and orientation of the estimated agent's pose.

6.1.5 Early Experiments - Real World One

A similar experiment as described in simulated world one was performed in the real world with the real robot. A few things are different. Instead of four chairs, two were used, and both of them were black. They were placed in different poses, and the initial robot location was different. The agent was not at the room center but in an arbitrary location that was used as the episodic origin. The set of goals were different. The robot again should rotate in place 360°, then perform a rectangle of 2x1.5 meters while avoiding the chairs. The Figure 6.8 illustrates the trajectory performed by the robot with those goals [3]. Figure 6.9 shows some of the frames captured during the experiment.

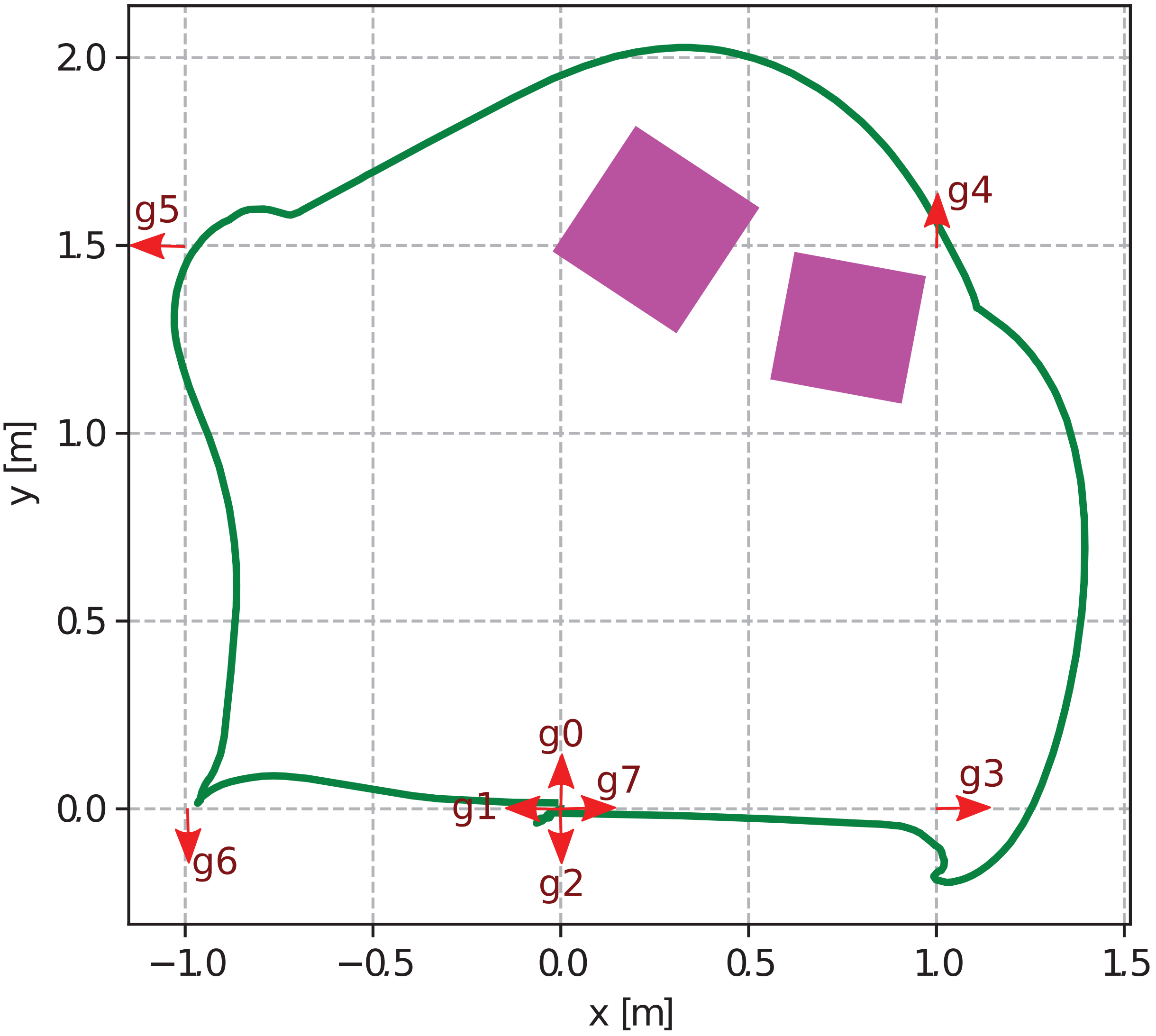
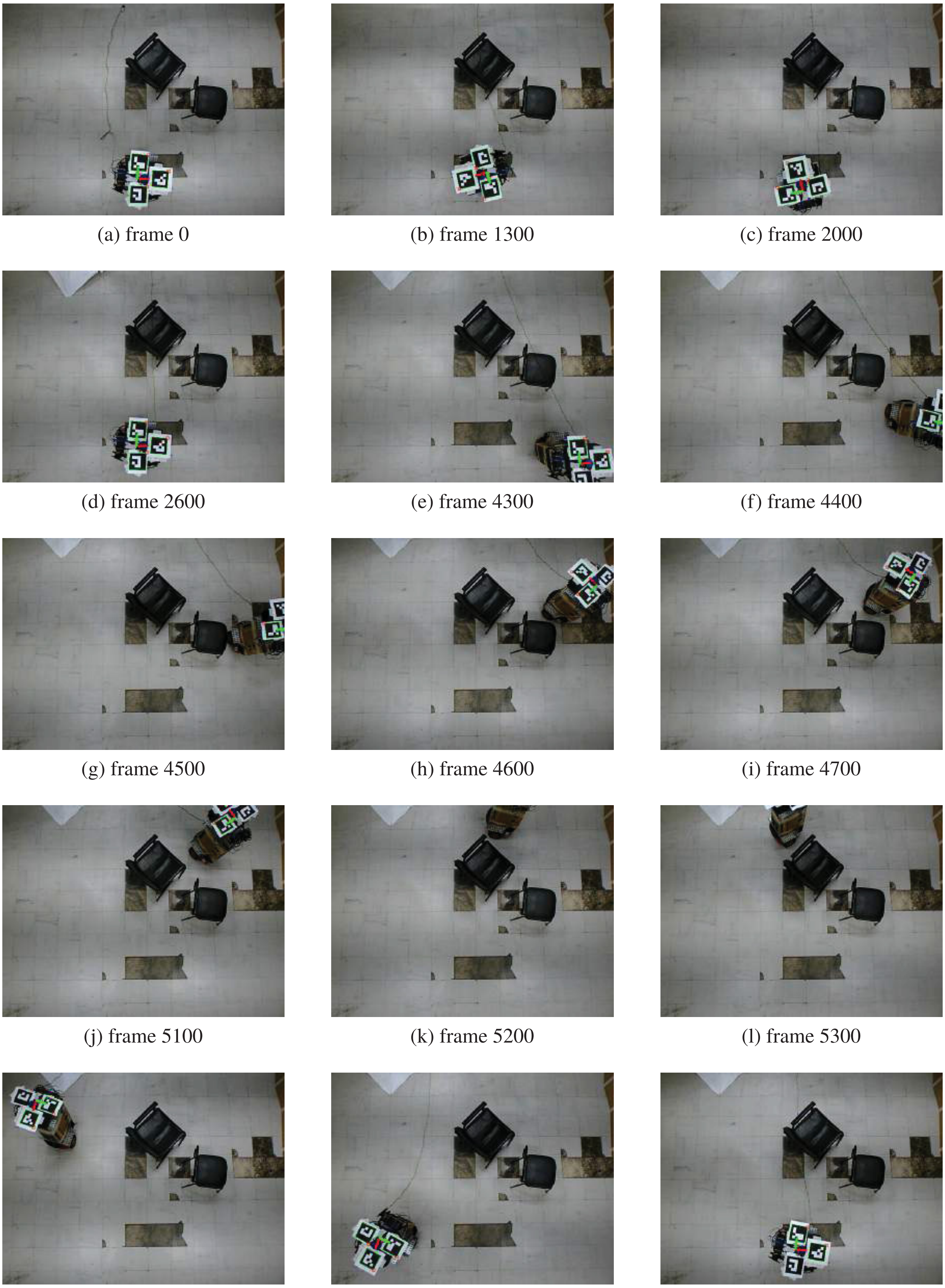


Figura 6.8: Episode's estimated trajectory using wheel odometry set in the Real World One together with the pose goals. The trajectory was represented in green. The red arrows, g0 to g7, represent the set of goals. The magenta squares represent an approximation of the obstacles.

In this experiment, during the trajectory, the robot was occluded from the view of the ceiling-mounted camera for several frames, on the trajectory between goal $g4$ and $g5$, preventing a reliable comparison between the wheel odometry and the visual one for the complete trajectory. Figure 6.9 displays some of the captured frames by the ceiling-mounted camera during the experiment with the pose estimation superimposed.

[3] Video is available online at: `https://github.com/VRI-UFPR/BeyondSight`



(a) frame 0

(b) frame 1300

(c) frame 2000

(d) frame 2600

(e) frame 4300

(f) frame 4400

(g) frame 4500

(h) frame 4600

(i) frame 4700

(j) frame 5100

(k) frame 5200

(l) frame 5300

(m) frame 5900

(n) frame 6200

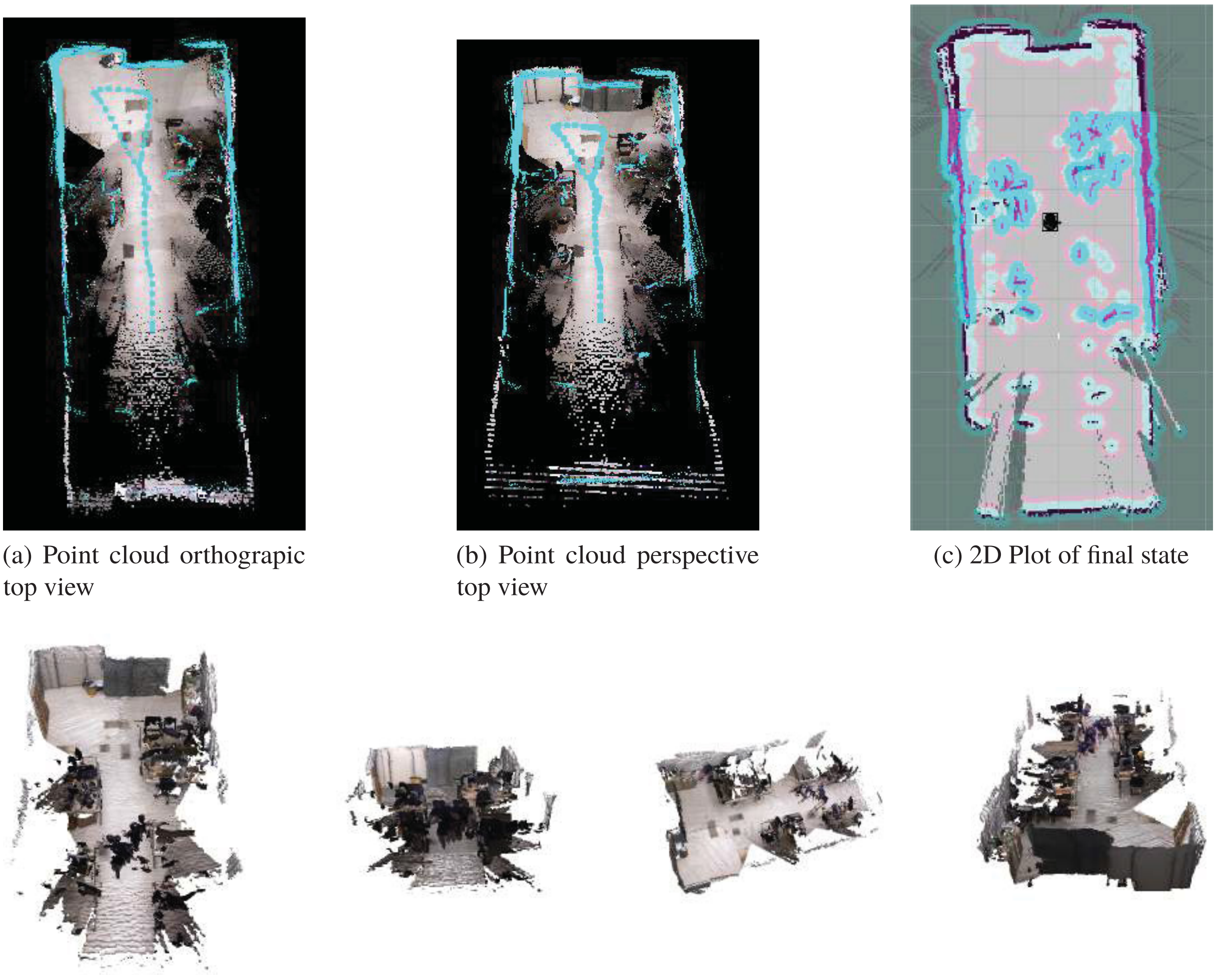
(o) frame 7700

Figura 6.9: Frames captured by the ceiling-mounted camera from the trajectory of Real World One. Pose estimation was superimposed, the small red square represents the orientation of each marker, the green quadrilateral represents the detected maker edges, and the green and red axis represent the position and orientation of the estimated agent's pose.

6.1.6 Early Experiments - Qualitative results from real indoor mapping

One experiment was performed to evaluate mapping in a real indoor room. The room is at the Federal University of Paraná. Figure 6.10 illustrates the results obtained [4]. The agent's mission was to traverse the corridor in one direction, briefly explore the region close to the door by performing a triangular path, then return to the corridor traversing it in the opposite direction.

The reconstructed mesh contains a person in the room during mapping, at the main corridor. We do not have a dedicated module for avoiding collisions with dynamic agents because dealing with dynamic agents is beyond this work scope. Thus, a person is considered a distinct static obstacle in each observed state, forming a cluster of noise on the reconstruction. This could lead to a path/region completely blocked by the 'ghost' obstacle. A naive solution for this is to reset/refresh the map periodically or erase regions without reobservations.



(a) Point cloud orthograpic top view

(b) Point cloud perspective top view

(c) 2D Plot of final state

(d) Mesh Reconstruction

(e) Mesh Reconstruction

(f) Mesh Reconstruction

(g) Mesh Reconstruction

Figura 6.10: Lab Indoor mapping with multiple view points. The pose graph is shown in cyan in Subfigures (a) and (b). Subfigure (c) shows the final state projected in a 2D plane. Subfigures (d) to (g) show multiple views of resulting mesh reconstruction.

[4] Video is available online at: `https://github.com/VRI-UFPR/BeyondSight`

6.2 AUTOCRAT - EXPERIMENTS

To ensure comparability in solving ObjectNav we migrated from Gazebo to Habitat. Gazebo lacks photorealism and support for the datasets commonly used with ObjectNav. Habitat supports the Matterport3D and Gibson datasets, which were created from reconstructed real environments, instead of purely synthetic assets like those of SUNCG (Song et al., 2017) and the ones we were using in the early experiments in Gazebo.

6.2.1 AUTOcrat episode sets and classes

In this section, we define which datasets, scenes, and episode sets were used with AUTOcrat, complementing information from chapter 4. The Matterport3D and Gibson datasets were used for the experiments. Both datasets are divided by scenes, where each scene is a different reconstructed building that exists in the real world. However, Matterport3D has its own semantic classes, a total of 40 different classes. Because of this there is not a perfect one-to-one correspondence between the 80 COCO classes (Lin et al., 2014) used to annotate Gibson by Armeni et al. (2019). Additionally, not all COCO classes are sampled on the Gibson scenes as displayed on figure 4.7, e.g., no persons or animals. This forced us to choose a subset of classes similarly to Chaplot et al. (2020), which used the subset = ['chair', 'couch', 'potted plant', 'bed', 'toilet', 'tv']. For training the AUTOcrat's policy network, we used the subset of five classes with equivalents in both datasets ['chair', 'table', 'sofa', 'bed', 'plant']. We chose this particular set of five classes based on the number of objects within the scenes and equivalence between the datasets before Chaplot et al. (2020)'s work was publicly available. This means that each scene is still the same, but the set of episodes is a subset containing as targets only the five classes. Figure 6.1 shows the equivalences between the two notations.

Gibson (COCO Classes)	Matterport3D Habitat-Challenge ObjectNav
chair	chair
potted plant	plant
couch	sofa
bed	bed
dining table	table
toilet	toilet
tv	tv_monitor
refrigerator	cabinet

Tabela 6.1: Equivalences between classes names. The 8 classes chosen as episodes targets for evaluation.

A comparison of metrics between the Gibson, SUNCG, and Matterport3D dataset is presented in table 6.2. The Specific Surface Area (SSA) and Navigation Complexity metrics are described as follows:

- **Specific Surface Area (SSA)**: the ratio of inner mesh surface and volume of the convex hull of the mesh. This is a measure of clutter in the models.

- **Navigation Complexity**: Longest A^* navigation distance between two randomly placed points (s_i, s_j) divided by the straight line distance between them (d_{l2}). The computed highest navigation complexity $\max_{s_i,s_j} \frac{d_{A^*}(s_i,s_j)}{d_{l2}(s_i,s_j)}$ for every model.

Dataset	Gibson	SUNCG	Matterport3D
Number of Scenes	572	45622	90
Avg. # of floors	2.53	1.09	2.61
Total Coverage m^2	211k	5.8M	46.6K
SSA	1.38	0.74	0.92
Nav. Complexity	5.98	2.29	7.80

Tabela 6.2: **Benchmarking Space Databases:** Comparison of Gibson database with SUNCG (Song et al., 2017) (hand designed synthetic), and Matterport3D (Chang et al., 2017). Adapted from Xia et al. (2018).

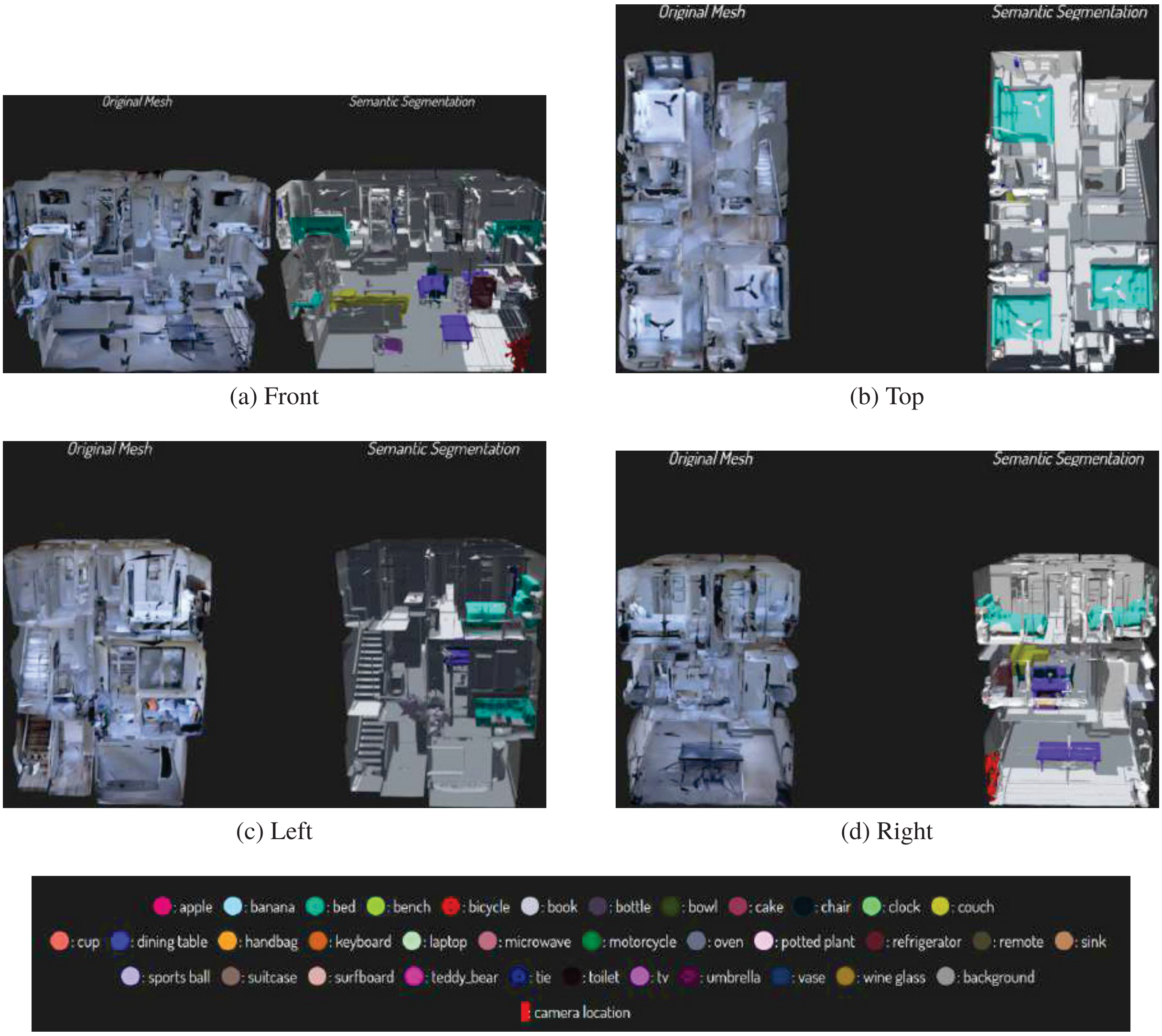
The training was performed with a subset of episodes from the training set provided for the Habitat Challenge 2020 ObjectNav on the Matterport3D dataset. The testing was performed with four scenes of Gibson that have semantic annotation: Churchton, Emmaus, Gravelly, and Micanopy. Those particular scenes were chosen due to their high number of objects, see table 6.3. Figure 6.11 illustrate the Churchton scene with multiple viewpoints from the textured polygonal mesh and the same viewpoints colorized with semantic annotations. Figures 6.12, 6.13, 6.14, 6.15 present a panoramic view inside these scenes from multiple arbitrary locations. The agent's camera sensor does not provide panoramas, but an image with a specific field of view.

The training was performed from scratch using Imitation Learning using the optimizer Stochastic Gradient Descent with Online Learning (SGDOL) (Zhuang et al., 2019). The learning rate started with 10^{-3}. The gradient was clipped to 0.2 to avoid an exploding gradient during learning. The training was performed for half a million steps and took around two weeks using a machine with 32GB of RAM, Intel(R)Xeon(R) CPU E5-2620 with 2.00GHz, and a GeForce TITAN Xp GPU with 12GB of VRAM.

Scene	Churchton	Emmaus	Gravelly	Micanopy
# Rooms	17	25	27	29
# Floors	3	3	3	3
Total Area m^2	275.265	871.08	530.688	374.441
SSA	1.187	1.189	1.156	0.498
Nav. Complexity	1.346	6.55	5.877	2.006
Total # Objs.	63	123	132	98
# Chairs	4	28	9	17
# Plants	13	16	32	3
# Couches	2	5	4	6
# Beds	4	5	7	8
# Tables	7	10	1	3
# Toilets	3	8	4	4
# TVs	2	4	7	6
# Refrig.	1	3	1	5

Tabela 6.3: Metrics of Gibson scenes used to evaluate AUTOcrat.

An important reminder is that ObjectNav does not encourage a vast exploration prior to navigating to the target object. An extensive pre-exploration will result in a low SPL, SoftSPL, and low absolute number of steps taken within the episode. ObjectNav was designed as a way to evaluate generalization to unseen and unconstrained scenarios and as a building block to even more complex tasks as previously discussed in chapter 2. Each episode usually starts in the same



(a) Front

(b) Top

(c) Left

(d) Right

Figura 6.11: Multiple viewpoints of Churchton scene. On the left side of each subfigure is the textured polygonal mesh and on the right side the colorized semantic annotation. Adapted from Armeni et al. (2019).

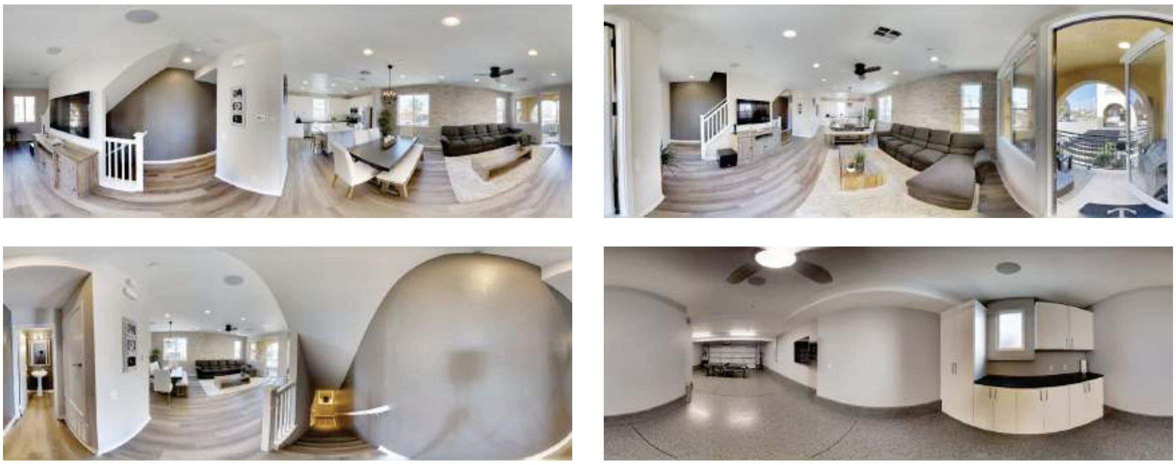


Figura 6.12: Four different Churchton panoramas from arbitrary locations in the scene.

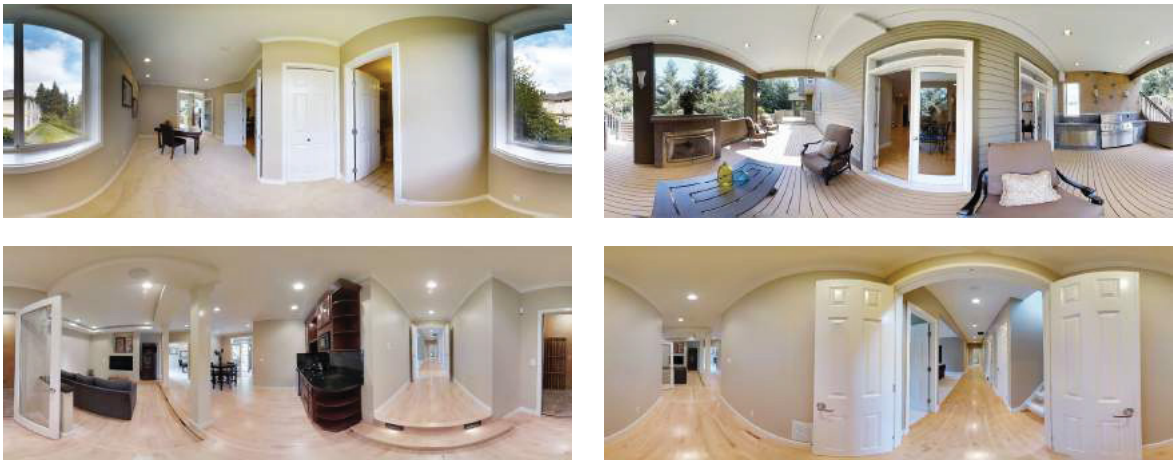


Figura 6.13: Four different Emmaus panoramas from arbitrary locations in the scene.

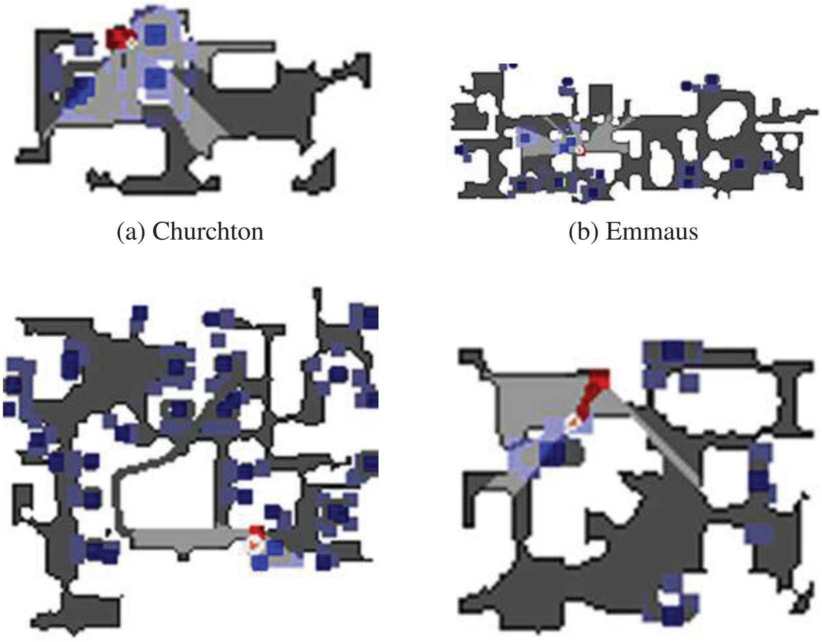
Figura 6.14: Four different Gravelly panoramas from arbitrary locations in the scene.

Figura 6.15: Four different Micanopy panoramas from arbitrary locations in the scene.

room of the target or at least the same floor, resulting in relatively short episodes compared to the mature exploration and navigation of PointNav that does not depend on egocentric images. Meaning that the task has a relatively short horizon when compared to other tasks.

Since there were no publicly available pre-generated episodes for ObjectNav on the Gibson dataset, we generated our own set of episodes using an implementation similar to the one provided for PointNav. We created four different sets of episodes without intersection. All four sets have eight possible targets, where each episode only has one semantic target. We expanded the initial five classes equivalence to eight as displayed in table 6.1, where the refrigerator is the most imprecise match, i.e., it was matched with the cabinet class. Due to its high sampling within Gibson's scenes, the refrigerator class was chosen, see figure 4.7. The reason to extend the initially proposed five classes to eight is to work with a power of two and with a broader set of episodes. Each particular class is tied to the match between notations and the number of objects within the scenes. Sets 0, 1, 2, 3 were created to evaluate the agent in increasing difficulty.

Set 0 contains only episodes within the minimal geodesic distance, computed by an oracle path planner, to any suitable target within the range of [1,2] meters. The agent starts in a random navigable point within these constraints and with a random yaw orientation. In Set 1, the distance is within [2,4] meters, and an additional rule is imposed, the ratio between the shortest Geodesic distance and Euclidean distance must be over 1.1. If the ratio is nearly 1, it indicates a few obstacles, and the episode is easy; if the ratio is larger than 1, the episode is difficult because strategic navigation is required. So in Set 1 and 2, we enforce a ratio of over 1.1 to ensure more complex episodes than in Set 0. Set 2, the range is within [4,8] meters. Sets 0, 1, 2 have 2^{13} episodes each, 8 target classes, each class has 2^{10} episodes. Set 3 is more generic and has episodes within [0.5, 20] meters where the agent cannot succeed by just using the STOP action as the first action on the episode.



(a) Churchton

(b) Emmaus

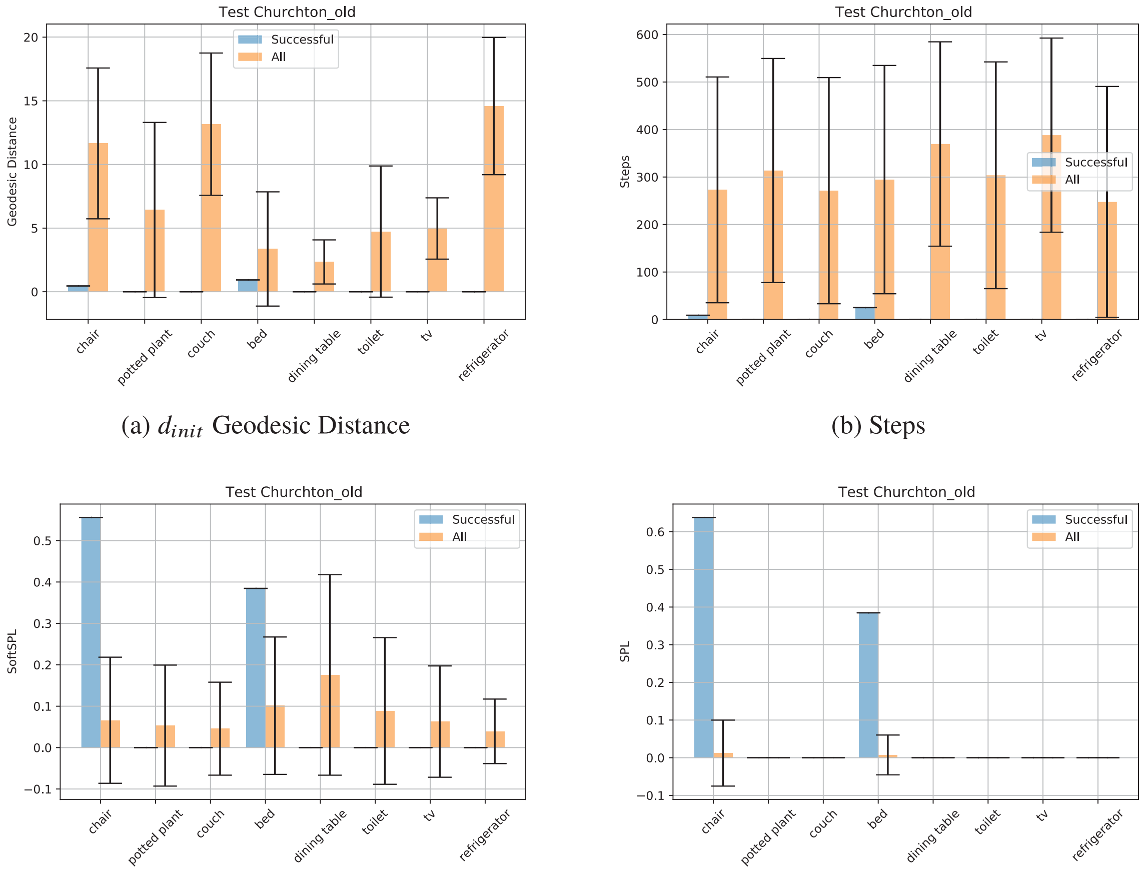
(c) Gravelly

(d) Micatopy

Figura 6.16: Top-down navigation. Episodes performed with AUTOcrat, the second pipeline. The agent is represented by a white circle with an orange arrow within. Agent's path is in red. Blue squares represent the target objects' origin. Purple regions are the regions considered viewpoints of the object. Red square is the episode's origin. Black represents obstacles. The agent's path is represented in darker red; it is better observed in subfigure (d). In other cases, the agent mostly performed rotations that are hard to represent without overlapping other graphical elements on the plot. Gray represents navigable space, and light gray is the observed space, dark gray represents unobserved space. In subfigure (d), the agent stopped outside the goal's range, making the episode unsuccessful.

6.2.2 AUTOcrat's results

Figure 6.16 provides illustrations of the agent's navigation on each of the scenes. The episodes displayed are those with the best-achieved SoftSPL per scene. The results in figures 6.17, 6.18, 6.19, 6.20 were evaluated on Set 3, an additional test was made on Set 0, and it is shown on tables 6.10, 6.11, 6.12, 6.13 in section 6.5. Sets 0, 1, 2 were evaluated with a different pipeline than AUTOcrat, see chapter 7. As a restriction, if the agent did not emit the STOP action within 500 steps, the episode was ended with a failure. This ensured that the episode ended even if the agent was stuck in an infinite loop.



(a) d_{init} Geodesic Distance

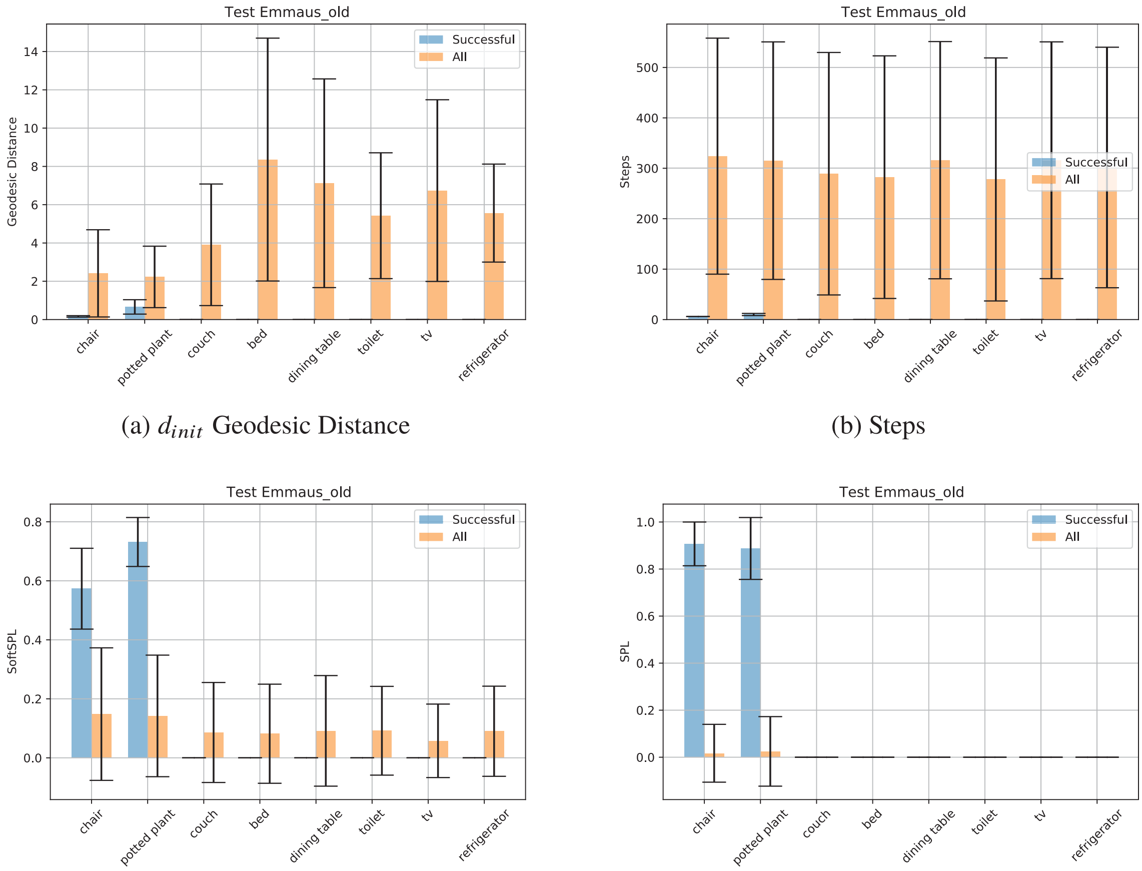
(b) Steps

(c) SoftSPL

(d) SPL

Figura 6.17: AUTOcrat results off Set 3 on the scene Churchton. The agent only succeeded in episodes with chair or bed classes out of the 8 target classes.

Figures 6.17, 6.18, 6.19, 6.20 displays four different metrics of evaluation in boxplots. Each bar's top edge represents the average value and the edges of the line in black the standard deviation. d_{init} Geodesic Distance measure the shortest distance from the starting agent's position to any object's viewpoint belonging to the target class on the respective episode. A large d_{init} Geodesic Distance means a longer horizon for the task and thus a more complex one. Steps represent the number of discrete steps taken by the agent during the episode. SoftSPL and SPL were previously described in section 2.17. In blue, we have the subset metrics from episodes where the agent succeeds. This subset is within the total set displayed in orange. This distinction lets us analyze the general performance of the agent and which are the cases it succeeds. AUTOcrat only succeeded on the easiest episodes where the agent starts close to the target, as can be observed at Subfigure 6.17 (a). It succeeded in at most two target classes per scene from the eight on the testing set.



(a) d_{init} Geodesic Distance

(b) Steps

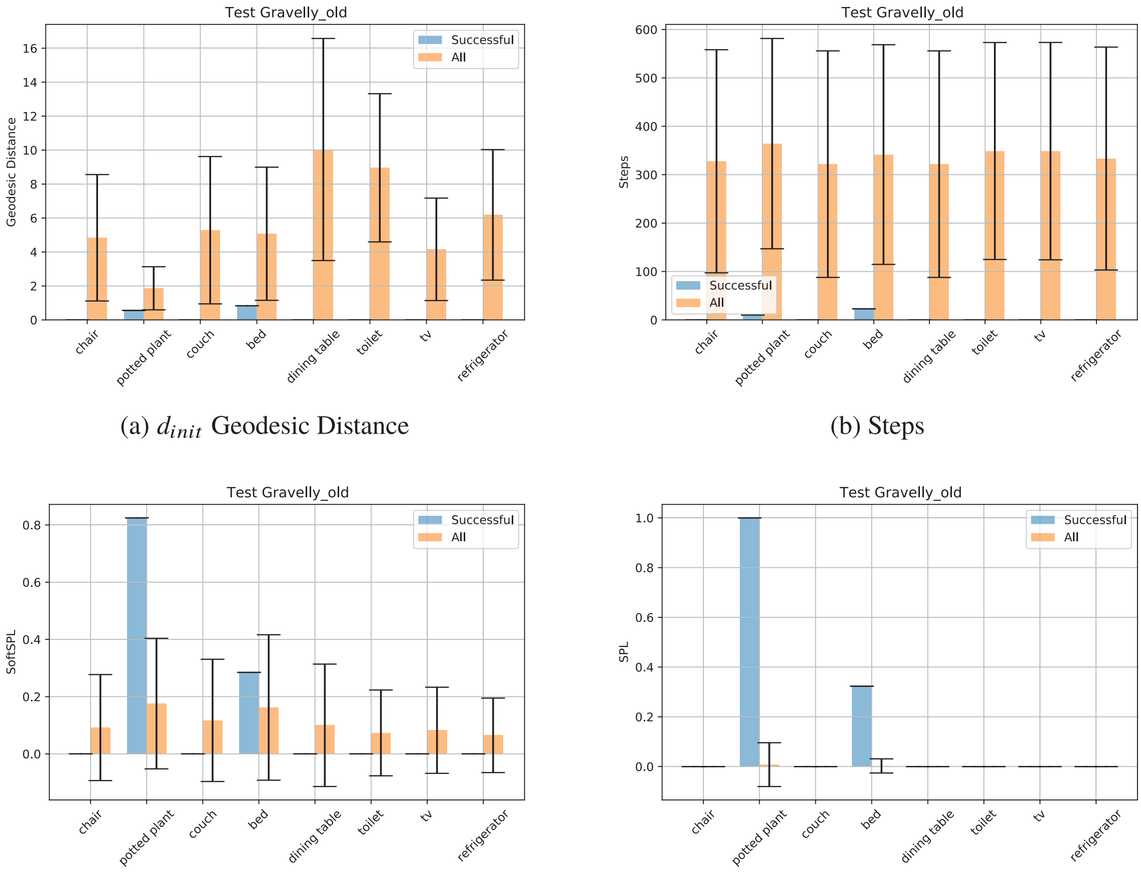
(c) SoftSPL

(d) SPL

Figura 6.18: AUTOcrat results off Set 3 on the scene Emmaus. The agent only succeeded on episodes with the chair or potted plant classes out of the 8 target classes.

As observed during testing, the agent learned to exploit a behavior where it rotates in-place several times and then takes a step forward. This led to a large number of steps per episode. Since SoftSPL only measures the steps taken forward, it was not affected as much as the absolute number of steps. A high number of collisions was also observed, and in cases where the agent did not recover from the collision, it ended failing the episode.

This differs from the oracle behavior, where the oracle performs the shortest path without unnecessary rotations and avoid collisions, meaning that the policy network mimicked poorly the oracle, perhaps a consequence of limited training or even the architecture chosen, since it was inspired by the baseline approach that also had poor performance as displayed on tables 6.10, 6.11, 6.12, 6.13 in section 6.5. A limitation of using Imitation Learning is that the student will not surpass the master's efficiency nether develop strategies/behaviors that are not present on the master. This is a trade-off of having a smaller search space and faster convergence towards efficiency. Our master was a greedy path planner based on A^* that have perfect knowledge about the scene, and the agent state, so it does not take decisions based on semantics, e.g., if a human were in an unknown room and had to find a TV it would search for it close to a sofa, chair, bed then after first seeing it, they would walk towards it. This is not the case with the oracle since it already knows from the start where the target is, resulting in learning that does not properly encourage semantic reasoning.



(a) d_{init} Geodesic Distance

(b) Steps

(c) SoftSPL

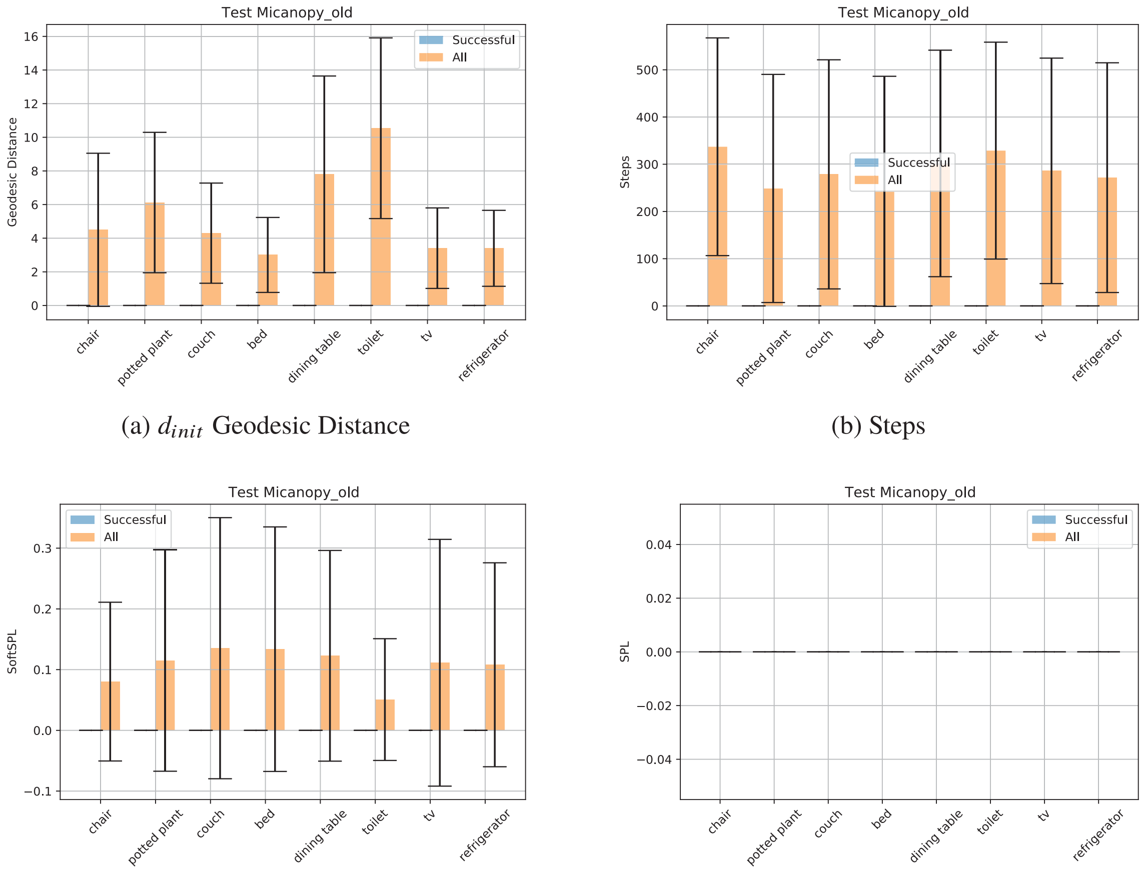
(d) SPL

Figura 6.19: AUTOcrat results off Set 3 on the scene Gravelly. The agent only succeeded on episodes with potted plant or bed classes out of the 8 target classes.

Figure 6.20 displays the results in the Micanopy scene. Even though none of the episodes succeeded, the SoftSPL metric achieved a similar result to other scenes, implying that were cases where the agent stopped just outside the success range, as exemplified by subfigure (d) of figure 6.16. As previously discussed in section 2.17.1 the success rate and the SPL metric are set to zero in case of failure, even if the agent stopped just outside the success range, meaning that they can be insufficient to draw conclusions about the performance, and the SoftSPL should be used to analyze nuances on the path performed.

AUTOcrat experiments were evaluated on Set 3. As expected, the classes unavailable during training [’tv’, ’refrigerator’, ’toilet’] resulted in a success rate of zero. However, SoftSPL was not so different for these classes than for the ones that had successes.

This poor overall performance led us to propose our third pipeline named BEyond, see section 5.5.



(a) d_{init} Geodesic Distance

(b) Steps

(c) SoftSPL

(d) SPL

Figura 6.20: AUTOcrat results off Set 3 on the scene Micanopy. The agent did not succeed on any of the episodes.

6.3 BEYOND - EXPERIMENTS

The experiments conducted for the BEyond pipeline were the evaluate the performance of its three variants: BEyond with Oracle, BEyond Intermediary, and BEyond Complete and how they compare with other algorithms. The first two were evaluated on the Habitat simulator and the latter on the iGibson simulator.

To ensure comparability with other algorithms used on the Habitat simulator, we focus on 8 object classes that are equivalent between the Gibson dataset with COCO classes and Matterport3D own classes, see table 6.4. The training was performed in two different scenes of Gibson that have semantic annotation: Neshkoro and Byers. The testing was performed with four scenes of Gibson that have semantic annotation: Churchton, Emmaus, Gravelly, and Micanopy. Table 6.5 presents some specific statistics in these scenes.

Gibson (COCO Classes)	Matterport3D Habitat-Challenge ObjectNav
chair	chair
potted plant	plant
couch	sofa
bed	bed
dining table	table
toilet	toilet
tv	tv_monitor
refrigerator	cabinet

Tabela 6.4: Equivalences between classes names. The 8 classes chosen as episodes targets for evaluation. This is the same table as table 6.1.

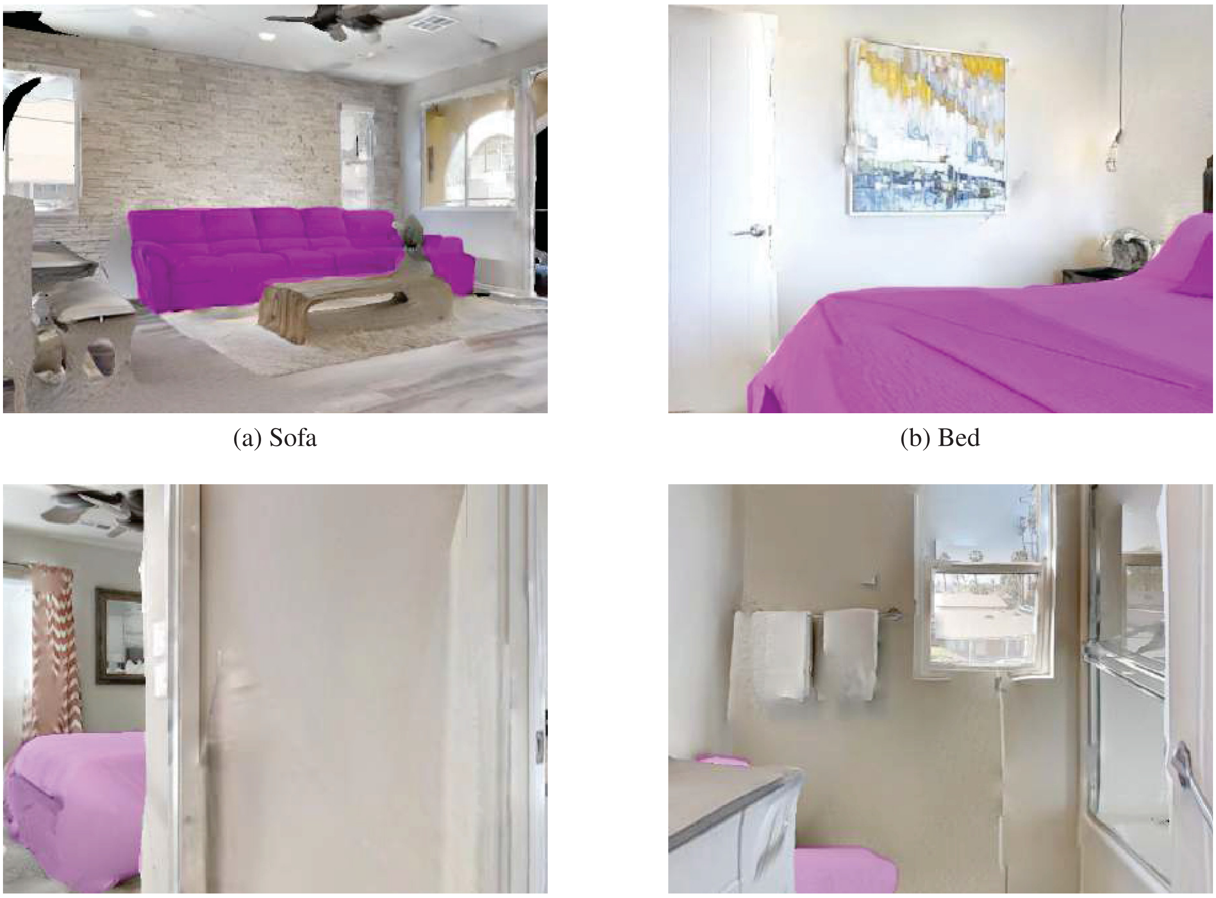
Scene	Neshkoro	Byers	Churchton	Emmaus	Gravelly	Micanopy
# Rooms	26	17	17	25	27	29
# Floors	3	3	3	3	3	3
Total Area m^2	789.459	217.218	275.265	871.08	530.688	374.441
SSA	1.247	1.203	1.187	1.189	1.156	0.498
Nav. Complexity	4.976	1.693	1.346	6.55	5.877	2.006
Total # Objs.	194	56	63	123	132	98
# Chairs	22	9	4	28	9	17
# Plants	39	4	13	16	32	3
# Couches	10	2	2	5	4	6
# Beds	5	1	4	5	7	8
# Tables	4	3	7	10	1	3
# Toilets	5	4	3	8	4	4
# TVs	3	5	2	4	7	6
# Refrig.	1	2	1	3	1	5

Tabela 6.5: Metrics off the Training and Testing Gibson scenes used to evaluate BEyond.

The training was performed from scratch using Imitation Learning with a greedy A* oracle path planner and the two scenes before mentioned for training and the optimizer Stochastic Gradient Descent (SGD). The learning rate started with $2.5 \cdot 10^{-3}$. The training was done on batches of four, where each instance of the batch ran in parallel and was set on a different simulated environment. The training was performed for half a million steps and took around two weeks using a machine with 32GB of RAM, Intel(R)Xeon(R) CPU E5-2620 with 2.00GHz, and a GeForce TITAN Xp GPU with 12GB of VRAM.

Our first objective with BEyond was to evaluate the proposed goal prediction module in a disentangled manner. This is particularly significant because it allows to isolate the module's performance, i.e., how well it can perform with perfect location estimation, perfect semantic segmentation input, and perfect local planning. We hypothesized that BEyond's performance would deteriorate with below optimal information. This was confirmed with our results, see tables 6.10, 6.11, 6.12, 6.13.

To properly evaluate our goal prediction module, we used the BEyond with Oracle variant with the Sets 0, 1, 2 previously explained in section 6.2.1. An additional metric called First Target Observation, which we propose in this work, was employed here to monitor the first step within the episode that the agent observed at least one pixel of the semantic target class, i.e., the agent's visibility of the target. This was used to evaluate how many steps after seeing the object the agent still takes until using the action STOP to end the episode. The relation between Steps and First Target Observation let us evaluate if the agent sees the object then navigate to it. First Target Observation is initialized with -1. Figure 6.21 presents some examples of the RGB image seen by the agent at the first target observation in different episodes.



(a) Sofa

(b) Bed

(c) Bed

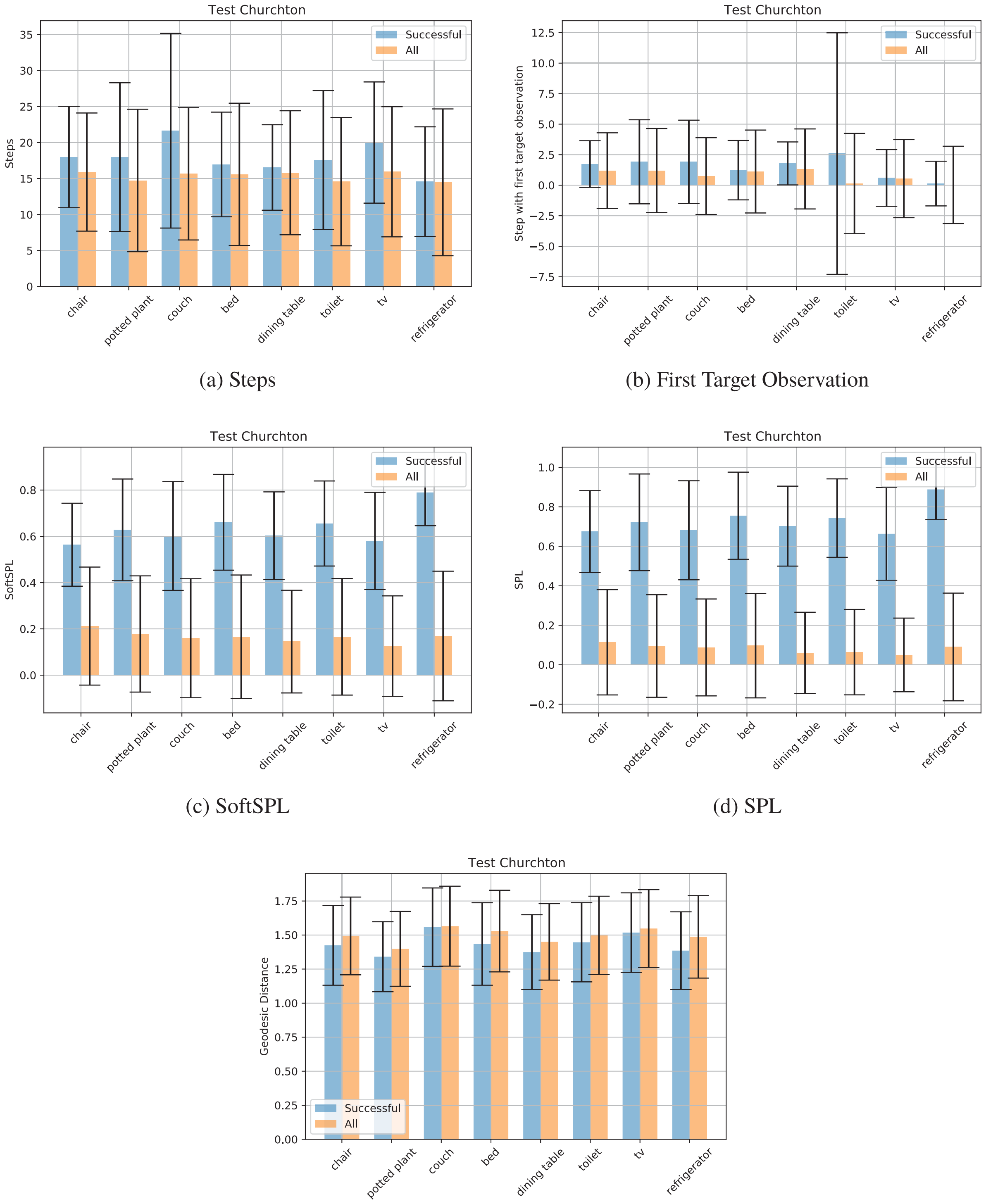
(d) Toilet

Figura 6.21: Examples of the first observation of an instance belonging to the episode target as perceived by the agent with the RGB sensor on the Churchton scene. Purple was used to highlight the semantic target.

6.3.1 BEyond with Oracle on SET 0

Figures 6.22, 6.23, 6.24, 6.25 present the results on the experiment BEyond with Oracle on Set 0, on the scenes Churchton, Emmaus, Gravelly, and Micanopy. Similarly to section 6.2.2, the bar plots are divided into two sets: Successful and All. Successful is contained within All. The class with the highest SPL was refrigerator in Churchton, Emmaus, and Gravelly. In Micanopy, the highest SPL was the TV class. Most of the episodes took between 15 and 20 steps. Forward steps happened in 0.25 meters. It was also observed that most of the steps were rotations, which happens at 30 degrees, meaning that the agent had to change its orientation multiple times and did not perform long straight lines. In Set 0, at least one instance of the target object class lies within the range of [1, 2] meters from the agent's starting point. The first target observation occurred before the 10th step, meaning that it happened within the first half of the episode.

The successful episodes were all above 0.5 SoftSPL, meaning adherence to the ground truth path. The number of steps on the All set was smaller in most cases to the Successful ones, meaning that the agent failed by issuing an early stop. However, the early stops were in their majority far from the success zone since the average SoftSPL was low. Set 0 has 2^{13} episodes per scene on the set. The agent had a success rate between 11% and 13% per scene. This can be seen at tables 6.6, 6.7, 6.8, 6.9 in section 6.5. We hypothesize that since the refrigerator class was the one with fewer samples per scene and it is usually a single one per room, after predicting the location of the object the subsequent predictions, stood on the vicinity of the object. Chairs have several instances close together that might have caused the goal prediction to change abruptly between instances at each step. This assumption is based on the higher number of steps and smaller SoftSPL of the chair's episodes than of the refrigerator's, while the minimal distance between the agent's starting position and its goal is nearly the same.



(a) Steps

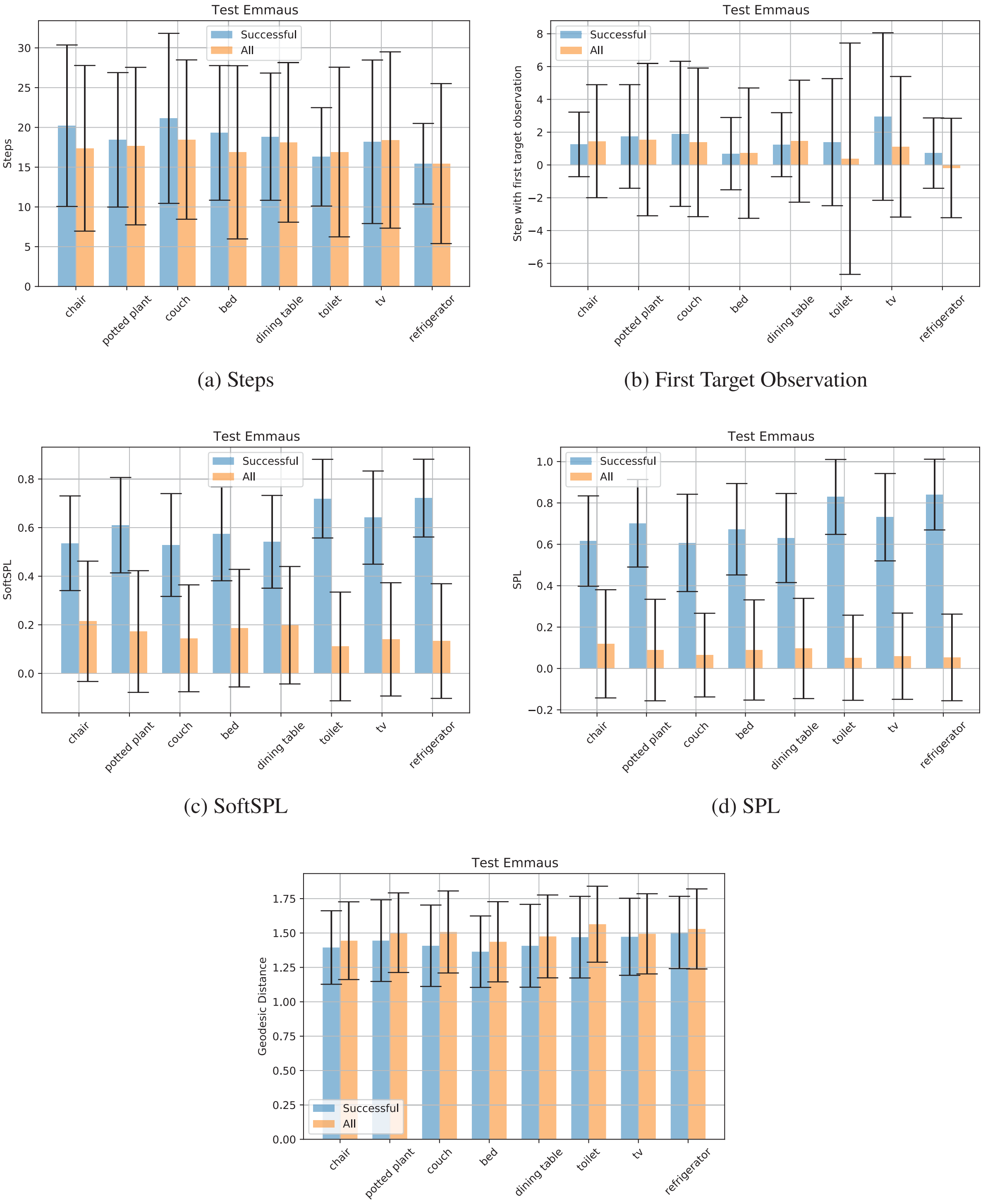
(b) First Target Observation

(c) SoftSPL

(d) SPL

(e) d_{init} Geodesic Distance

Figura 6.22: BEyond with Oracle's results on Churchton scene using Set 0. Note how the BEyond approach had successful episodes for every target class evaluated, and the successful ones had a SoftSPL over 0.5.



(a) Steps

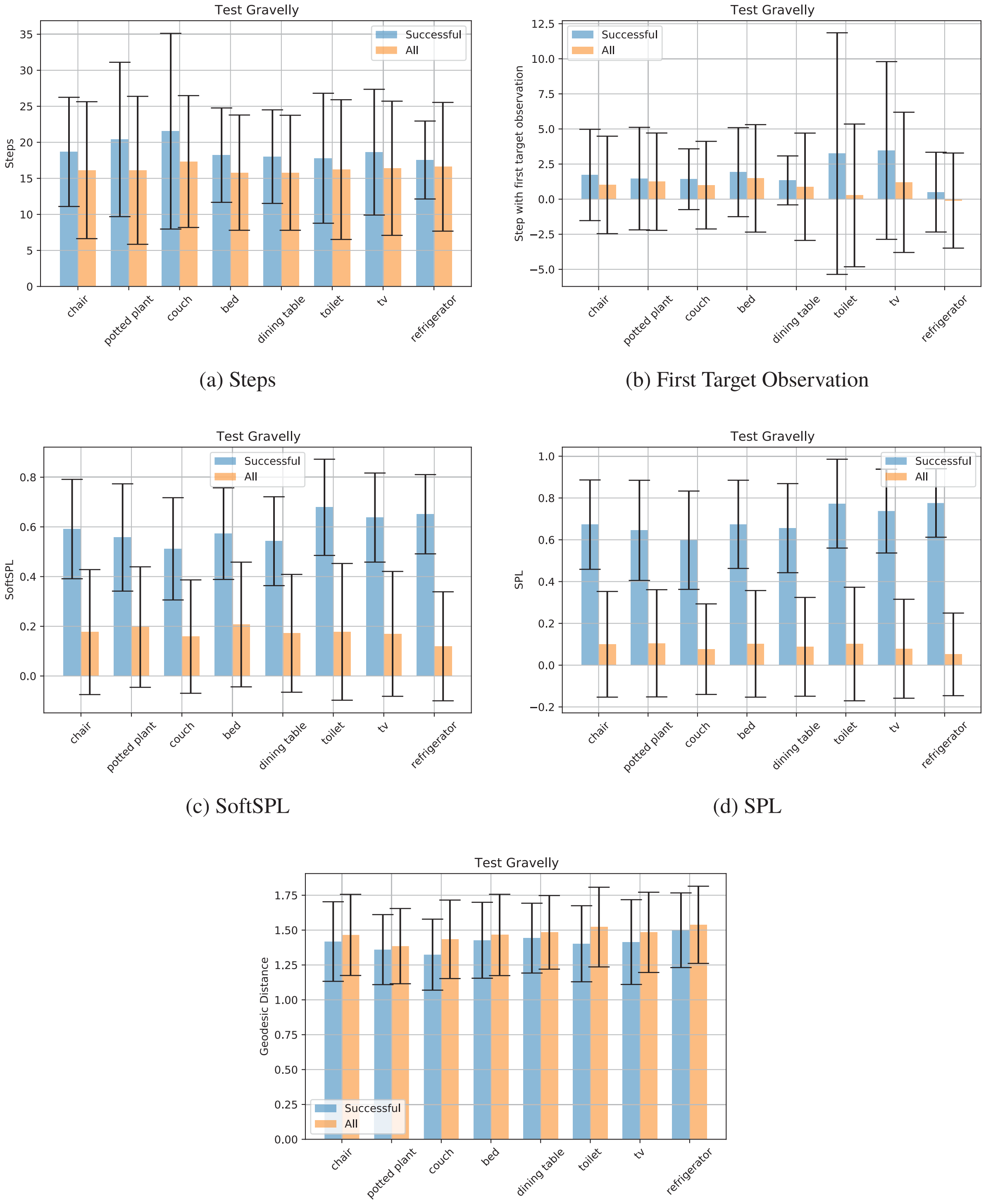
(b) First Target Observation

(c) SoftSPL

(d) SPL

(e) d_{init} Geodesic Distance

Figura 6.23: BEyond with Oracle's results on Emmaus scene using Set 0. Note how the BEyond approach had successful episodes for every target class evaluated, and the successful ones had a SoftSPL over 0.5



(a) Steps

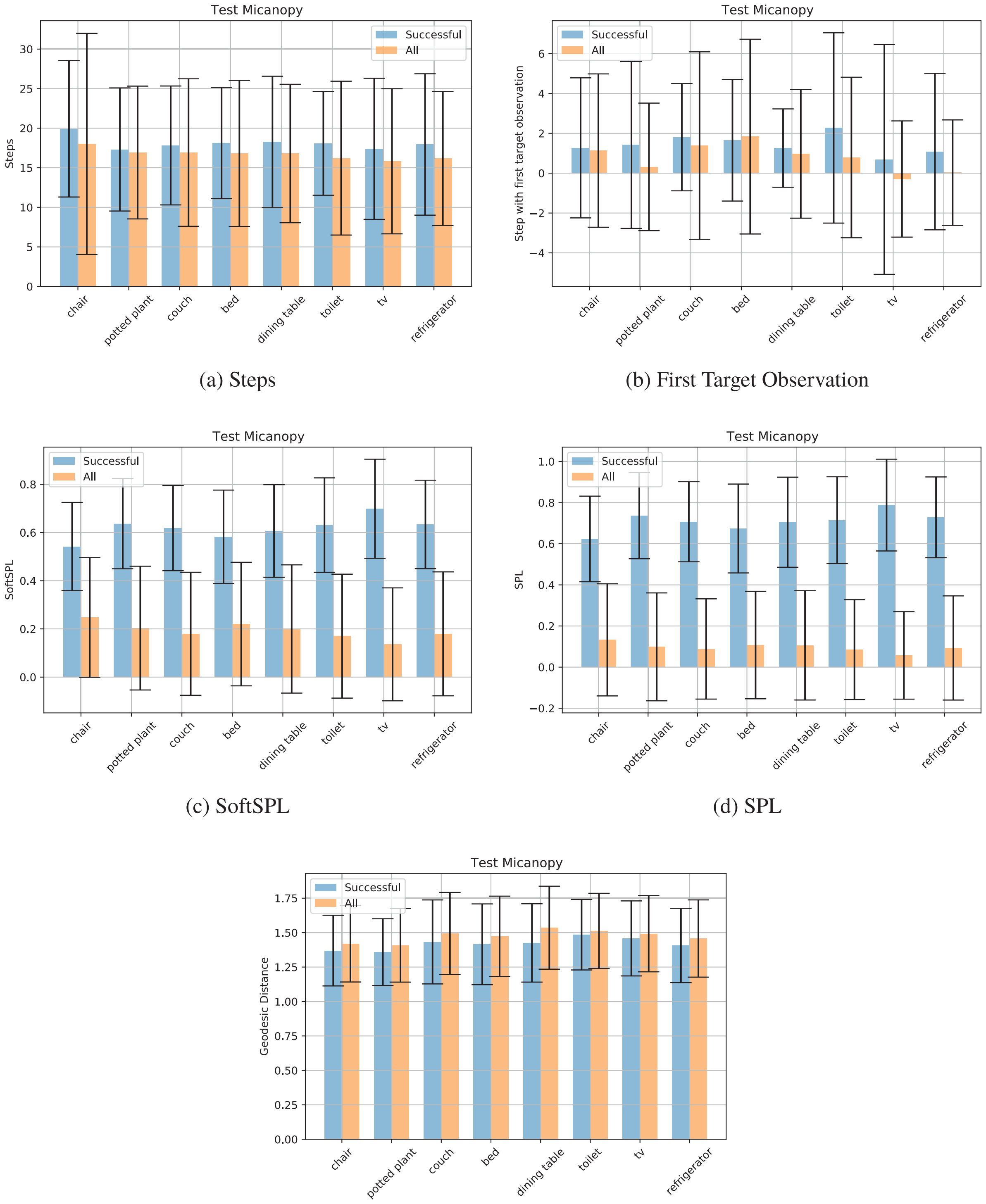
(b) First Target Observation

(c) SoftSPL

(d) SPL

(e) d_{init} Geodesic Distance

Figura 6.24: BEyond with Oracle's results on Gravelly scene using Set 0. Note how the BEyond approach had successful episodes for every target class evaluated, and the successful ones had a SoftSPL over 0.5



(a) Steps

(b) First Target Observation

(c) SoftSPL

(d) SPL

(e) Geodesic Distance

Figura 6.25: BEyond with Oracle's results on Micanopy scene using Set 0. Note how the BEyond approach had successful episodes for every target class evaluated, and the successful ones had a SoftSPL over 0.5

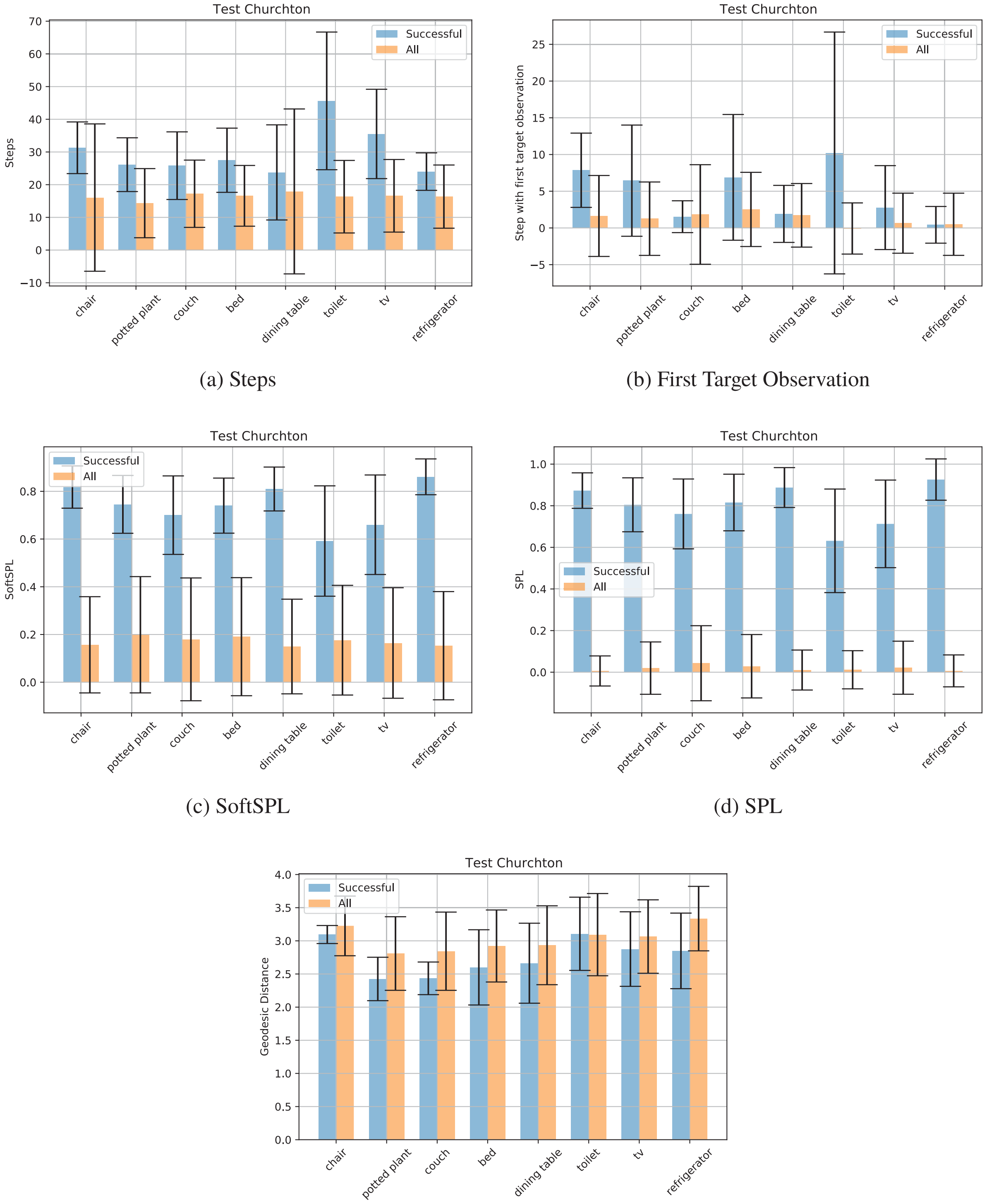
6.3.2 BEyond with Oracle on SET 1

Figures 6.26, 6.27, 6.28, 6.29 present the results on the experiment BEyond with Oracle results on Set 1, on the scenes Churchton, Emmaus, Gravelly, and Micanopy. As expected, due to the increase in distance, the number of steps also increased. If the number of steps were the same, the episodes would result in early stops and failures.

This increase in the number of steps is especially noticeable in the toilet class on Churchton, with an average of over 40 steps on the successful episodes, a major difference to the other classes within the same scene, that most were smaller than 30. This noticeable difference was also observed in the refrigerator class on the Gravelly scene. This difference implies that the agent had more trouble maneuvering to those objects than to the others since the shortest path to these objects is in the same range of [2,4] meters as the other classes. Each of the four scenes has at least 63 instances of labeled objects, see table 6.5. A longer distance means more possible obstacles between the agent's starting position and its target. Since this variant uses the oracle planner, the number of collisions is expected to be near zero, meaning that the additional steps were necessary to avoid collisions. This is confirmed by the enforced ratio of over 1.1 between the minimal Euclidean distance and minimal geodesic distance, one of the rules used to create Set 1.

The distance caused a major drop in the success rate from 13% on Set 0 to 2% on Set 1 on the best case. By doubling the episode's minimal distance, the agent's success rate dropped 6.5 times. Some decay was already expected since longer distances mean longer horizons and more branching decisions, but it happened more sharply than expected. Even though the success rate dropped, the SoftSPL rose, meaning that the fewer successful episodes had even higher adherence to the ground truth path.

Even though the distance was doubled, the first target observation was obtained before the 10th step. Meaning the agent saw the object early but had to take a longer path to it. This is likely due to the fact that the RGB sensor can perceive objects several meters away if there is no occlusion, but the forward step length remains at 0.25 meters.



(a) Steps

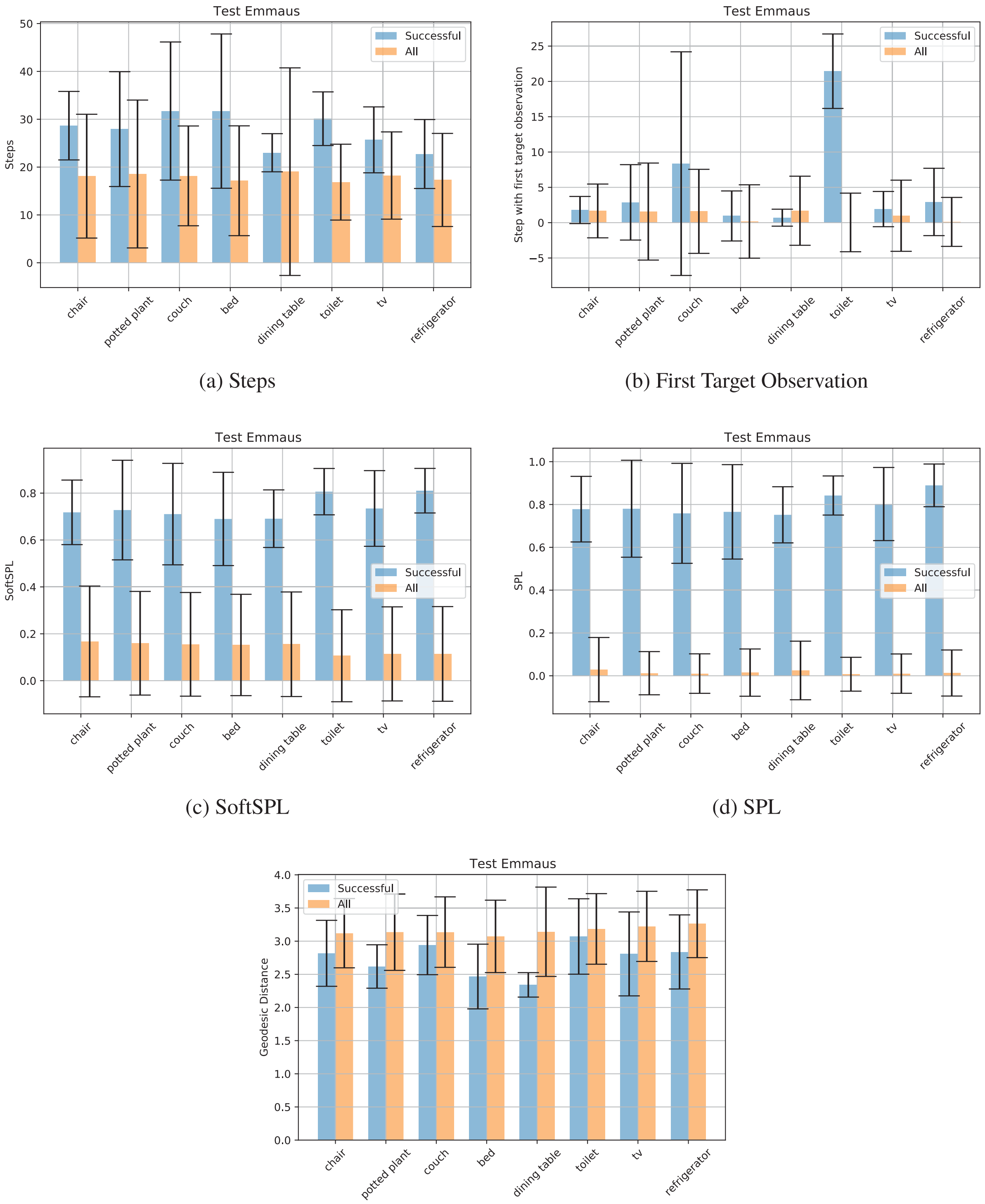
(b) First Target Observation

(c) SoftSPL

(d) SPL

(e) d_{init} Geodesic Distance

Figura 6.26: BEyond with Oracle's results on Churchton scene using Set 1. Note that even though the d_{init} Geodesic distance increased in comparison to the Set 0 (figure 6.22), the SoftSPL rose, especially in the chair class, from around 0.6 to around 0.9.



(a) Steps

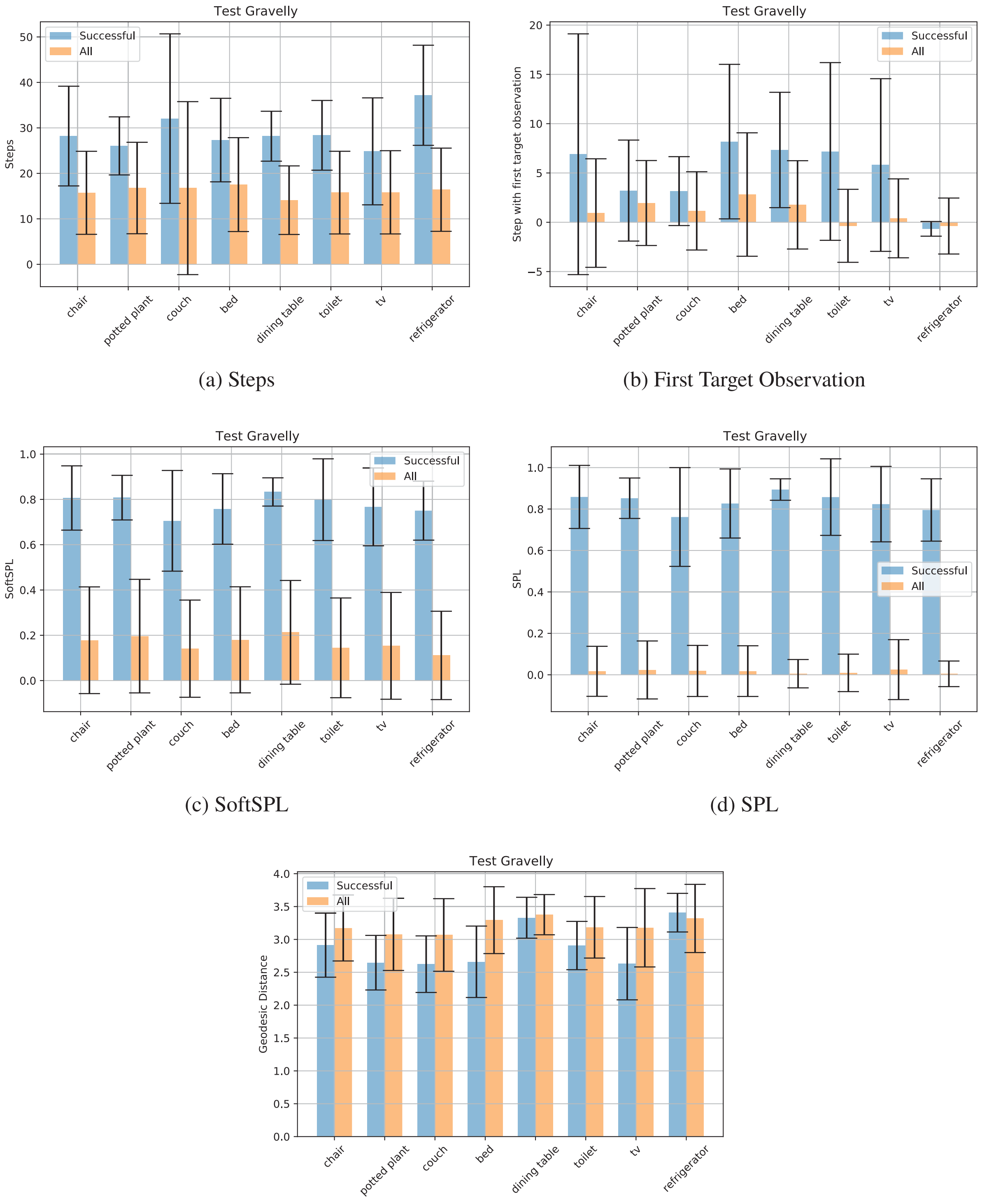
(b) First Target Observation

(c) SoftSPL

(d) SPL

(e) d_{init} Geodesic Distance

Figura 6.27: BEyond with Oracle's results on Emmaus scene using Set 1.



(a) Steps

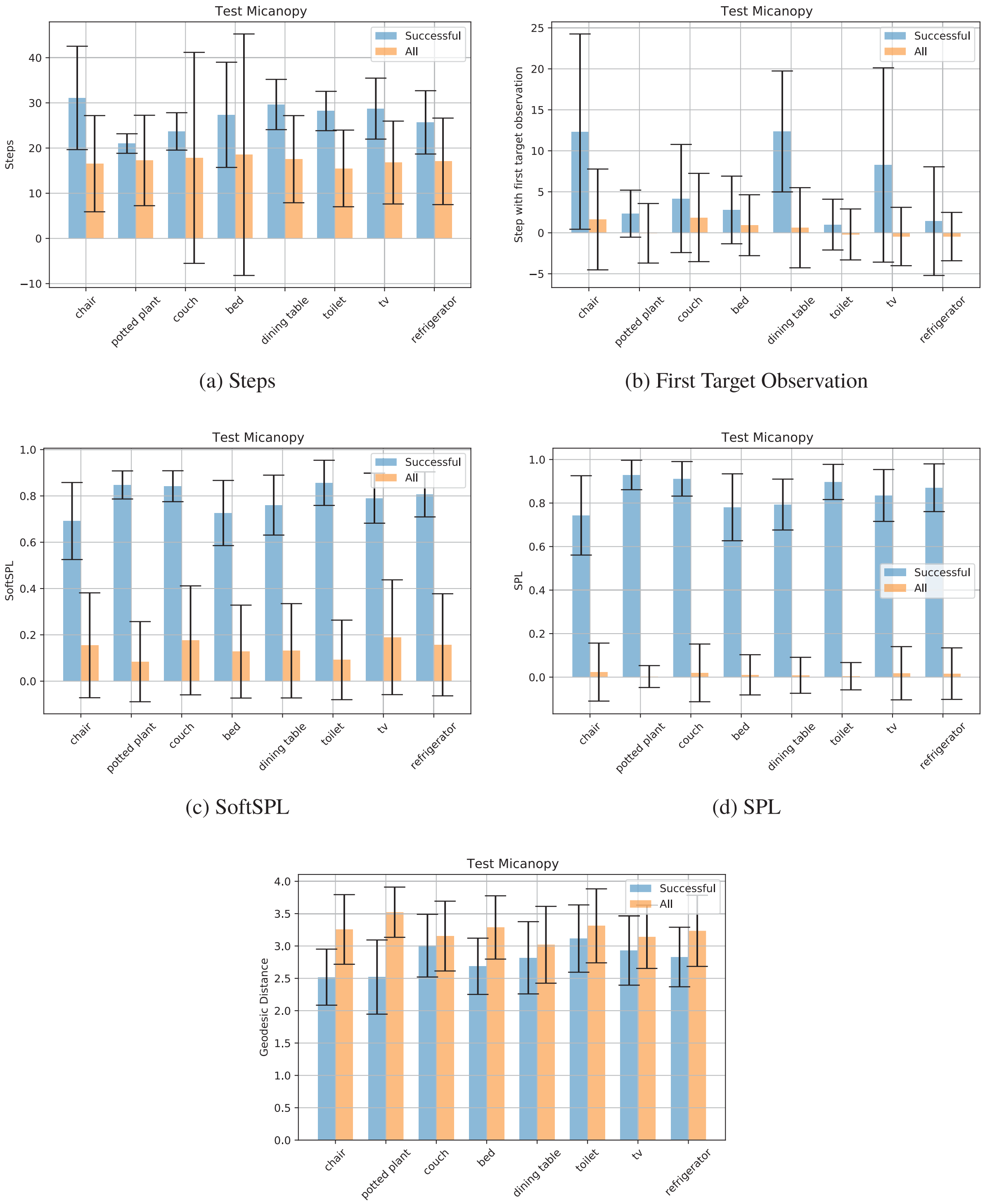
(b) First Target Observation

(c) SoftSPL

(d) SPL

(e) d_{init} Geodesic Distance

Figura 6.28: BEyond with Oracle's results on Gravelly scene using Set 1.



(a) Steps

(b) First Target Observation

(c) SoftSPL

(d) SPL

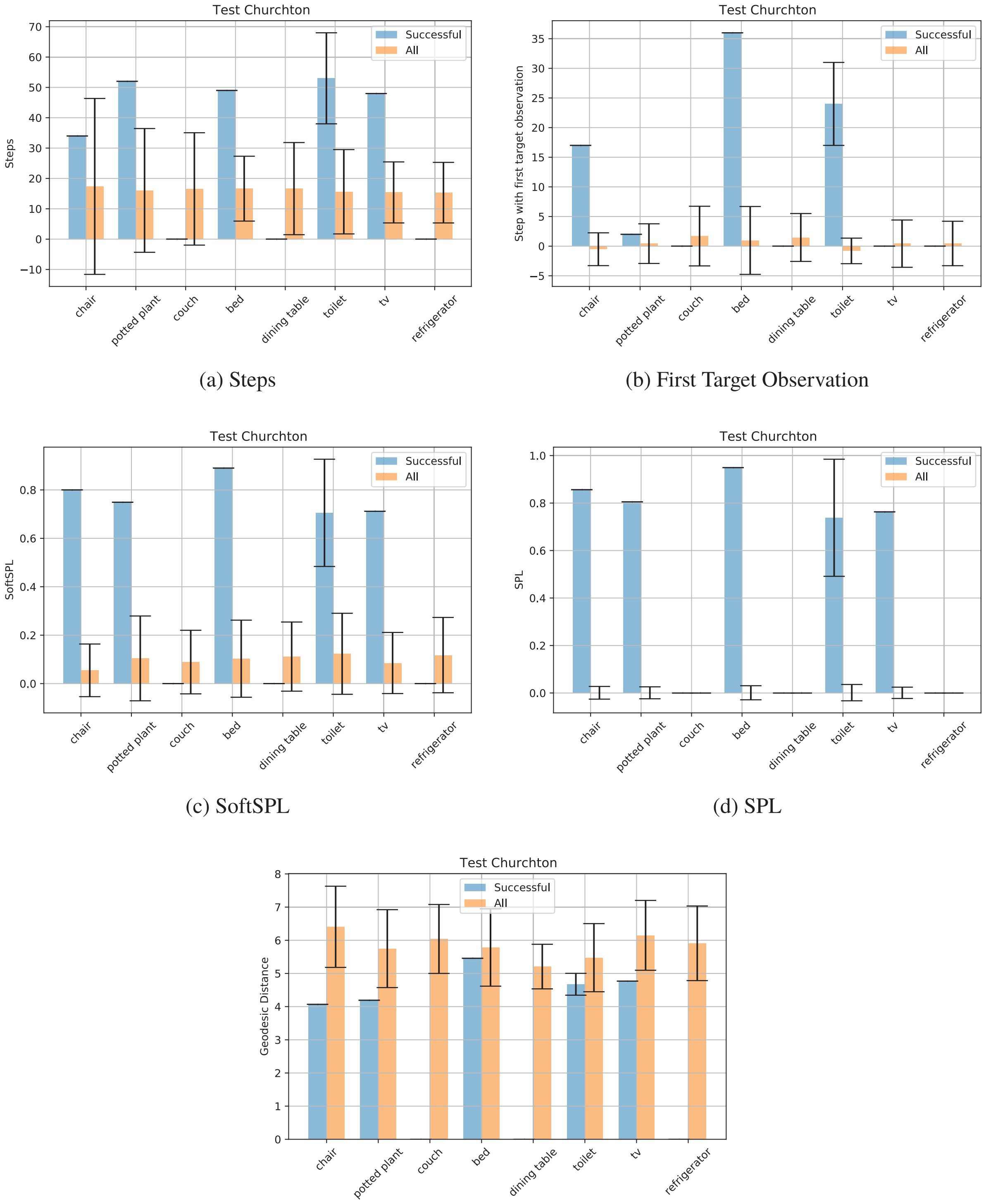
(e) Geodesic Distance

Figura 6.29: BEyond with Oracle's results on Micanopy scene using Set 1.

6.3.3 BEyond with Oracle on SET 2

Figures 6.30, 6.31, 6.32, 6.33 present the results on the experiment BEyond with Oracle results on Set 2, on the scenes Churchton, Emmaus, Gravelly, and Micanopy. A major difference seen in this set is the total failure to reach some of the classes. On Set 0 and Set 1, each class was reached successfully at least once. Here on the worst case, only 2 classes were reached, and in the best case, 5 of the 8 classes. This closely related with the absolute number of successful episodes here, in Churchton 6 out 8192, in Emmaus 7 out of 8192, in Gravelly 5 out of 8192 and in Micanopy only 2 out of 8192, see tables 6.6, 6.7, 6.8, 6.9 in section 6.5. The SoftSPL difference between Set 0 and Set 1 was smaller than between Set 1 and Set 2. Even though the number of successful episodes was meager, the few successful episodes had the SoftSPL over 0.7 on its majority.

By doubling the distance between Set 1 and Set 2, the success rate decreased 36.8 times. An even more sudden drop than between Set 0 and Set 1. This suggests that as the distance increase linearly, the success rate decays exponentially. As previously stated, a longer episode means more possible branching decisions, more necessary steps taken means more necessary goal predictions where a gross error in prediction could lead to several steps along a wrong path. The enforced ratio of over 1.1 between the minimal Euclidean distance and minimal geodesic distance used to create Set 2 ensures that the shortest path between the starting pose to the episode's target is not a straight line. The implication of this is that not only is the path length longer, but it can contain more obstacles. Additionally, due to the indoor nature of the scenes, longer episodes could mean traversing from a room to another, which could be filled with clutter and without a large margin for maneuverability, for example, the agent could start inside a bedroom and need to find a toilet, to do it needs to traverse the small bedroom with a large bed and some furniture, then go through a narrow door opening and traverse the tight bathroom to reach the success zone of the toilet.



(a) Steps

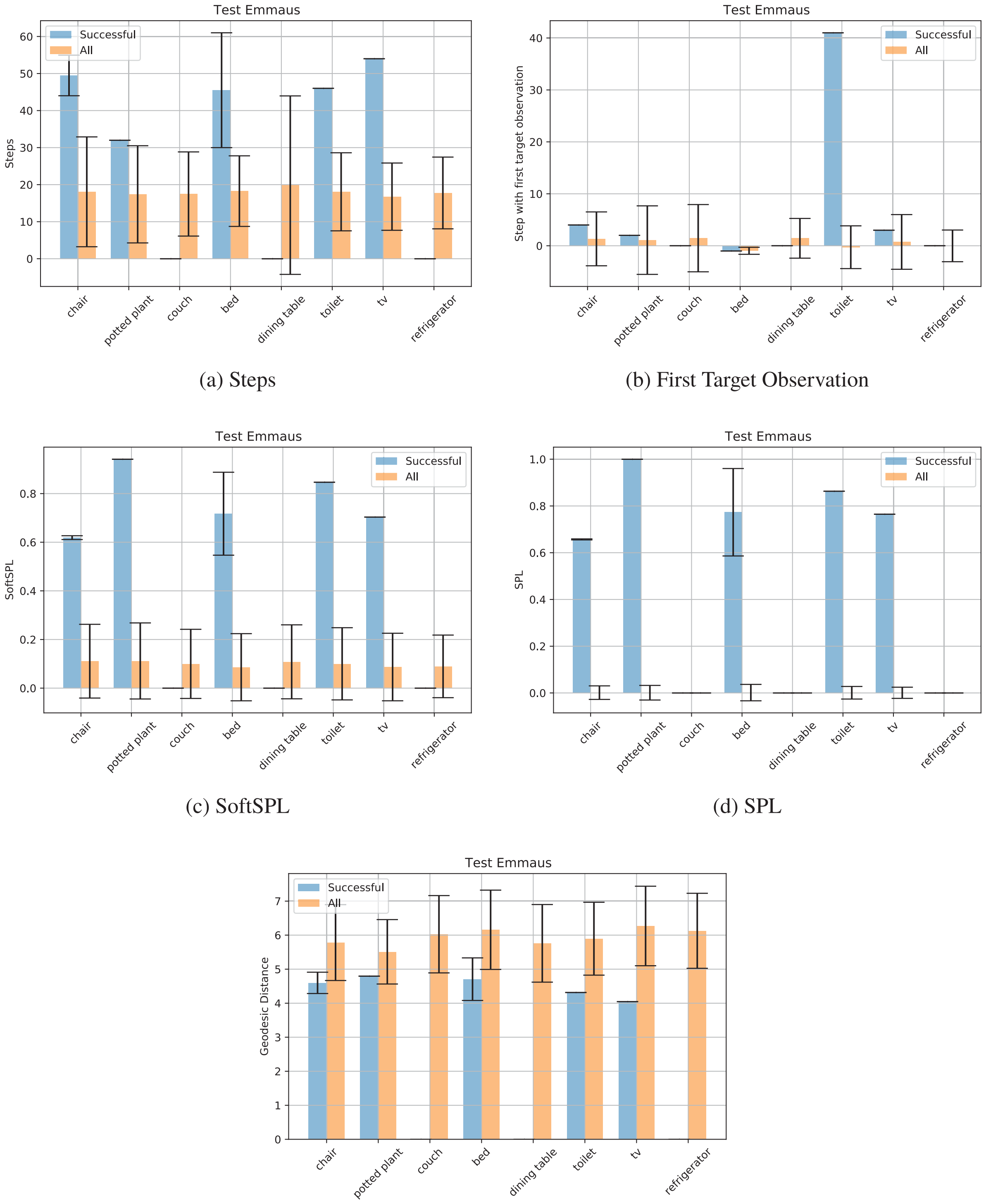
(b) First Target Observation

(c) SoftSPL

(d) SPL

(e) d_{init} Geodesic Distance

Figura 6.30: BEyond with Oracle's results on Churchton scene using Set 2. The agent succeed only on 5 classes out of 8.



(a) Steps

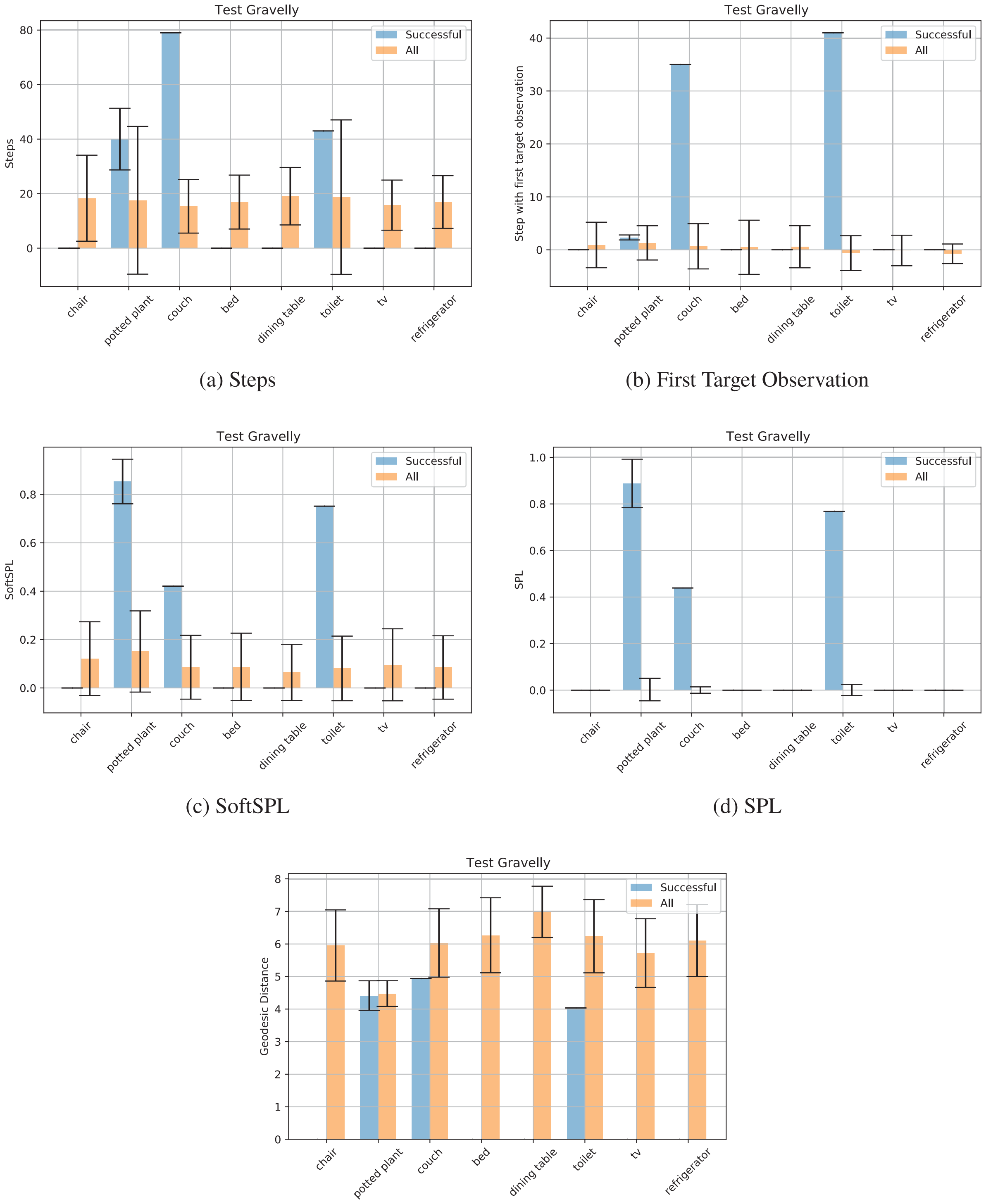
(b) First Target Observation

(c) SoftSPL

(d) SPL

(e) d_{init} Geodesic Distance

Figura 6.31: BEyond with Oracle's results on Emmaus scene using Set 2. The agent succeed only on 5 classes out of 8. The same 5 classes that succeed in the Churchton scene.



(a) Steps

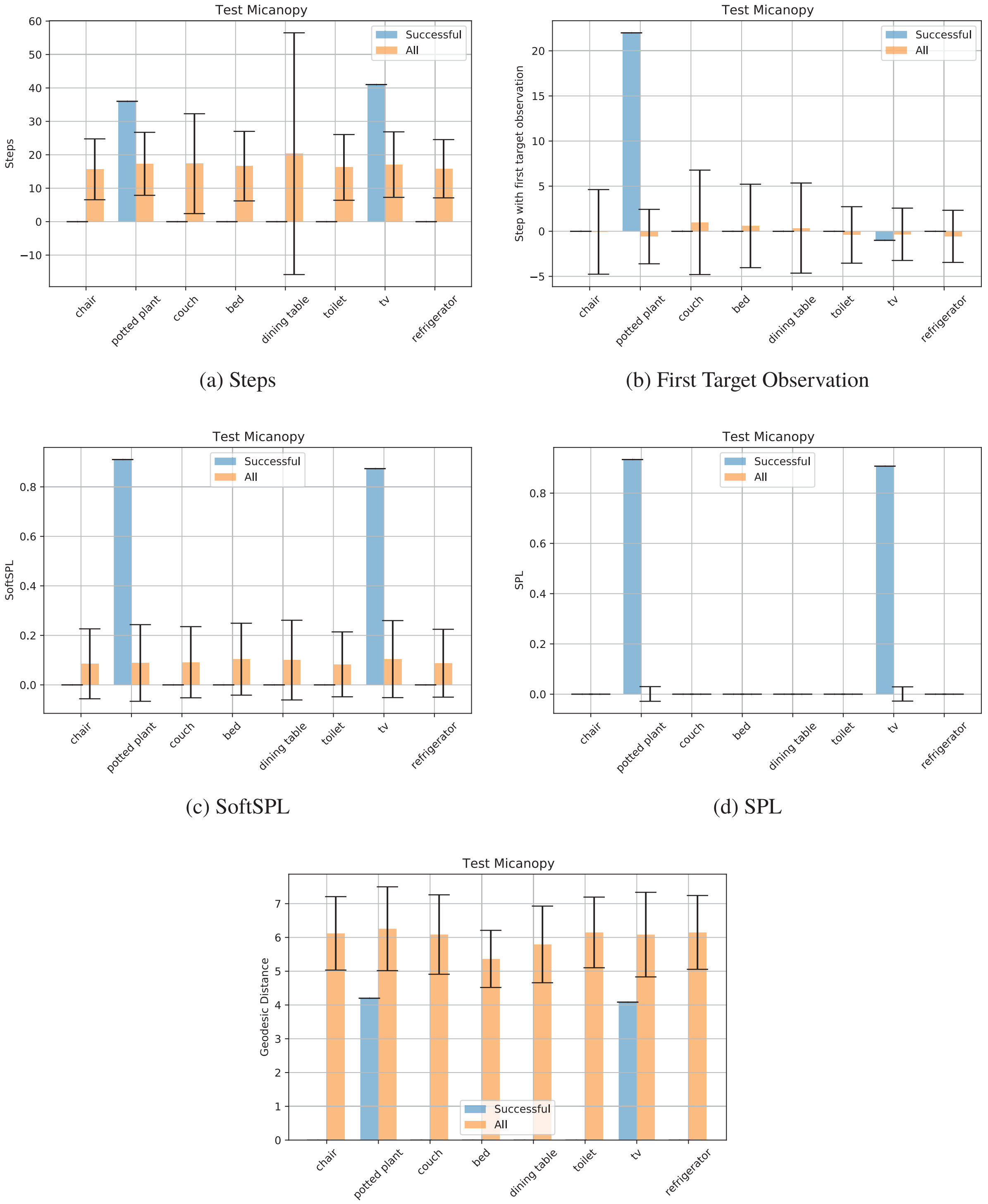
(b) First Target Observation

(c) SoftSPL

(d) SPL

(e) d_{init} Geodesic Distance

Figura 6.32: BEyond with Oracle's results on Gravelly scene using Set 2. The agent succeed only on 3 classes out of 8.



(a) Steps

(b) First Target Observation

(c) SoftSPL

(d) SPL

(e) Geodesic Distance

Figura 6.33: BEyond with Oracle's results on Micanopy scene using Set 2. The agent succeed only on 2 classes out of 8.

6.4 QUALITATIVE RESULTS OF BEYOND COMPLETE

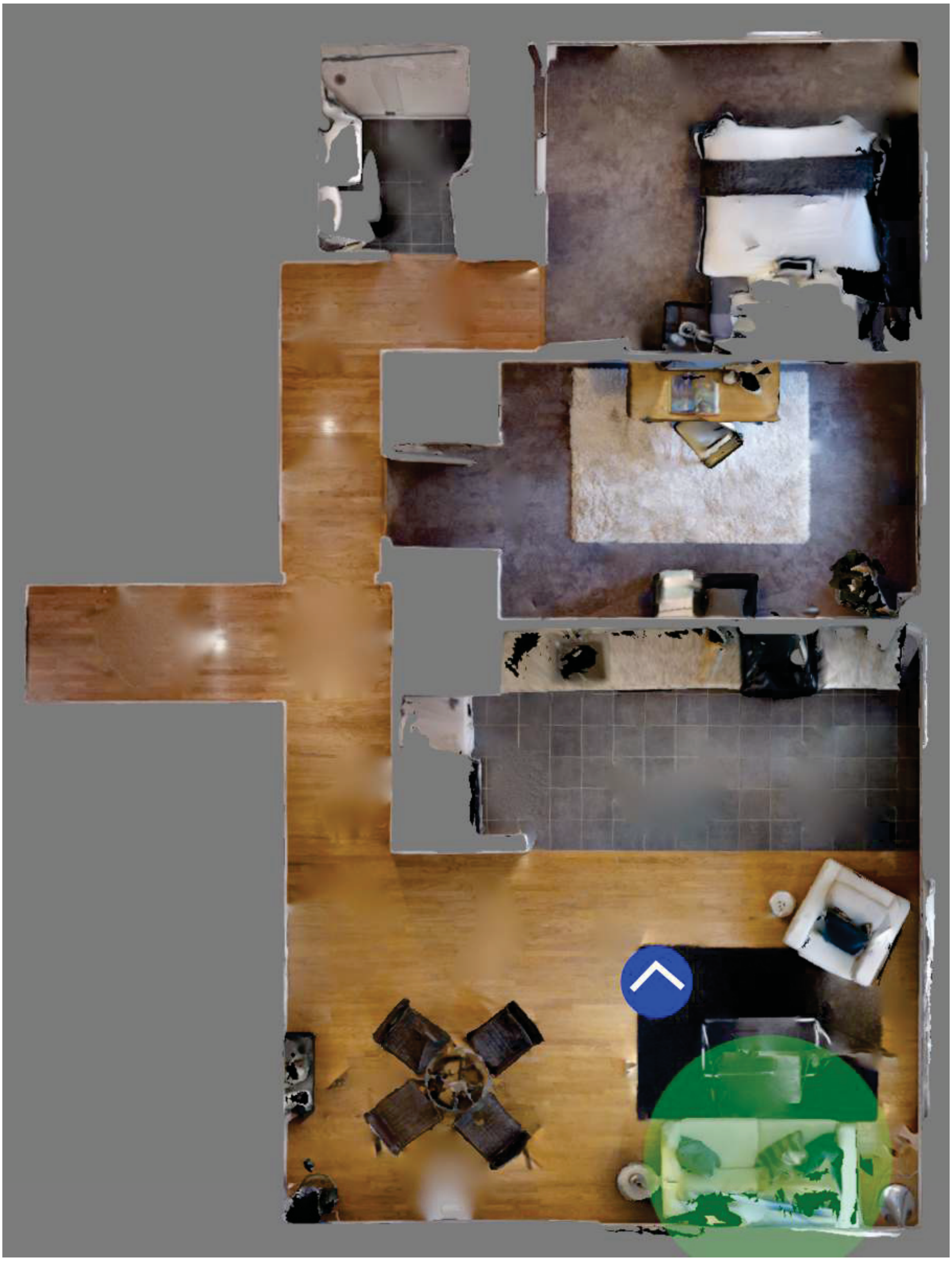
As said in section 5.5.5, the BEyond Complete is a proof-of-concept of how our BEyond pipeline can operate in a realistic scenario. Here we present some of the qualitative results obtained in the iGibson simulator. A quantitative analysis was left for future work, and it is related to the sim-to-real evaluation discussed in section 8.1.

To properly demonstrate BEyond Complete, we used the iGibson simulator, version 1.0.1, that has integration with Robot Operating System (ROS). We used the ROS Noetic that is Python 3 compatible. The implementation used the NVIDIA library CUDA 11.0, Pytorch 1.7.1 library, the YOLACT++ v1.2, and Decentralized Distributed Proximal Policy Optimization (DD-PPO) trained for point-based navigation using polar coordinates (for discrete actions). The DD-PPO used here is the one contained in the habitat baselines package v0.1.5. The predicted discrete actions were transformed into pose goals and fed to the Timed-Elastic Band (TEB), a continuous local planner running with ROS. Our VRIBot model was included in the simulator. It is worth mentioning that we simulated a Kinect v1 camera with 640x480 of RGB-D resolution and 43 degrees of vertical Field of View (FoV) and 58 degrees of horizontal FoV. In reality, the depth resolution is different from the RGB, but since the drivers already properly scale it to match the RGB, so in simulation, they both have the same resolution and field of view.

The experiment was fairly simple. The agent was spawned in the Placida scene, one of the Gibson scenes that was updated on the Interactive Gibson dataset (Xia et al., 2020), where objects were separated from the environment mesh into discrete objects with physical simulation, i.e., they react to collisions. The agent was spawned in a living room facing the kitchen and had to perform ObjectNav to a sofa. The sofa is in the opposite direction from the starting agent's pose and within the 4m radius of the starting position. In the living room, there are several other objects such as chairs, armchairs, and a table, see Figure 6.34. Note that there is no object on the carpet, as the top view seems to indicate. Since currently iGibson lack proper implementation of the ObjectNav task, we adapted the point-based task similarly to how it is done on Habitat. A point goal was set at the sofa center, and a radius of 1.15 meters, which was enough to encompass its perimeter, was used. If the agent reaches the success area (the area inside the circle), the episode is considered successful.

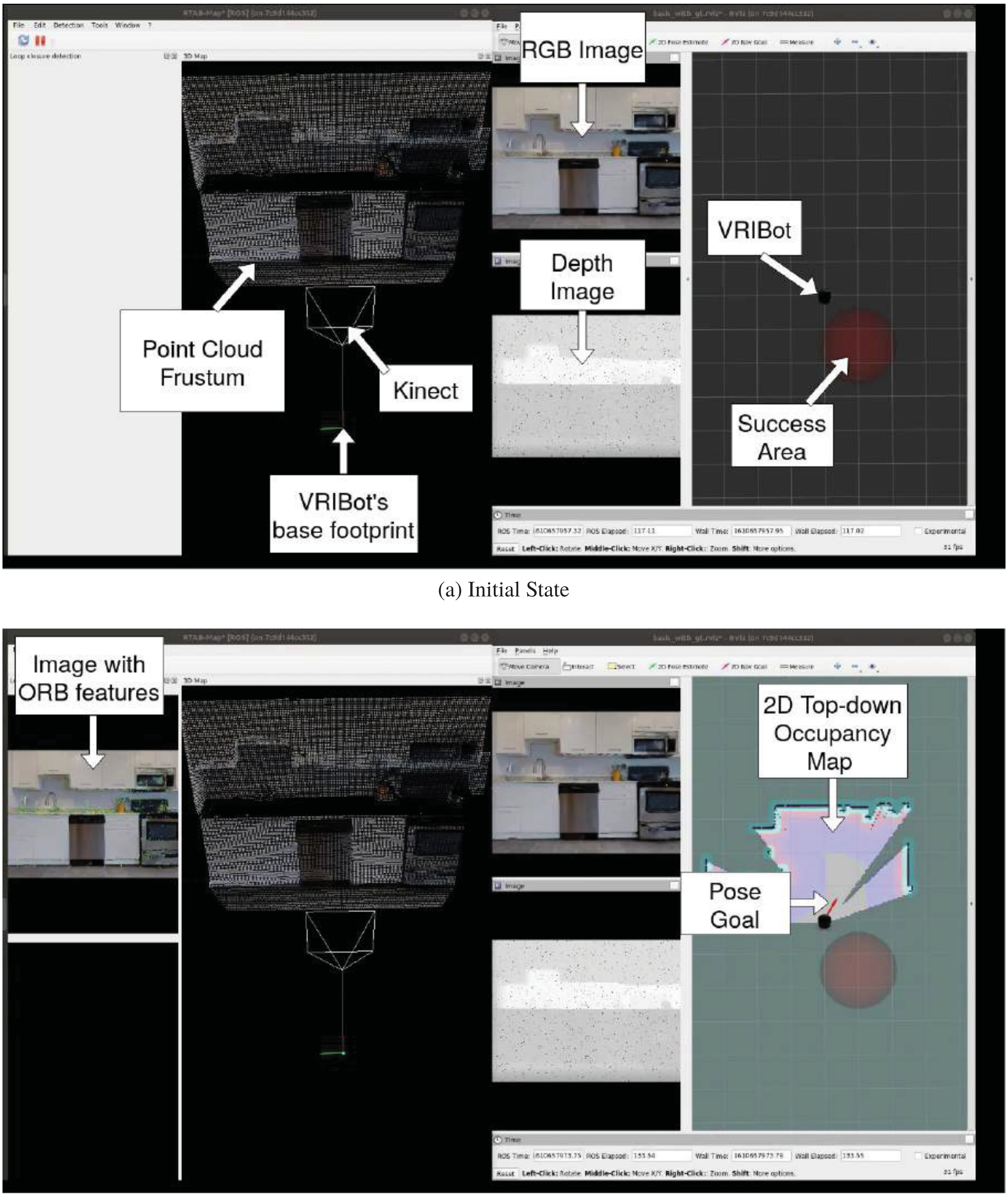
The simulation used a 30Hz sampling frequency to simulate real-time. The agent completed the episode successfully with a SPL of 1.0, see figures 6.35 to 6.41 [5]. 12 actions were predicted, and the path was almost optimal. Without any collisions and the agent has dislocated 1.16 meters. There were 4 predicted forward actions meaning that deterministicly, the agent would have moved only 1m, since each forward step is 0.25 and each turning movement is 30° degrees. The episode took 668 simulated ticks, meaning that the simulated path took 22.26 seconds.

[5] Video is available online at: `https://github.com/VRI-UFPR/BeyondSight`



(a) Initial State

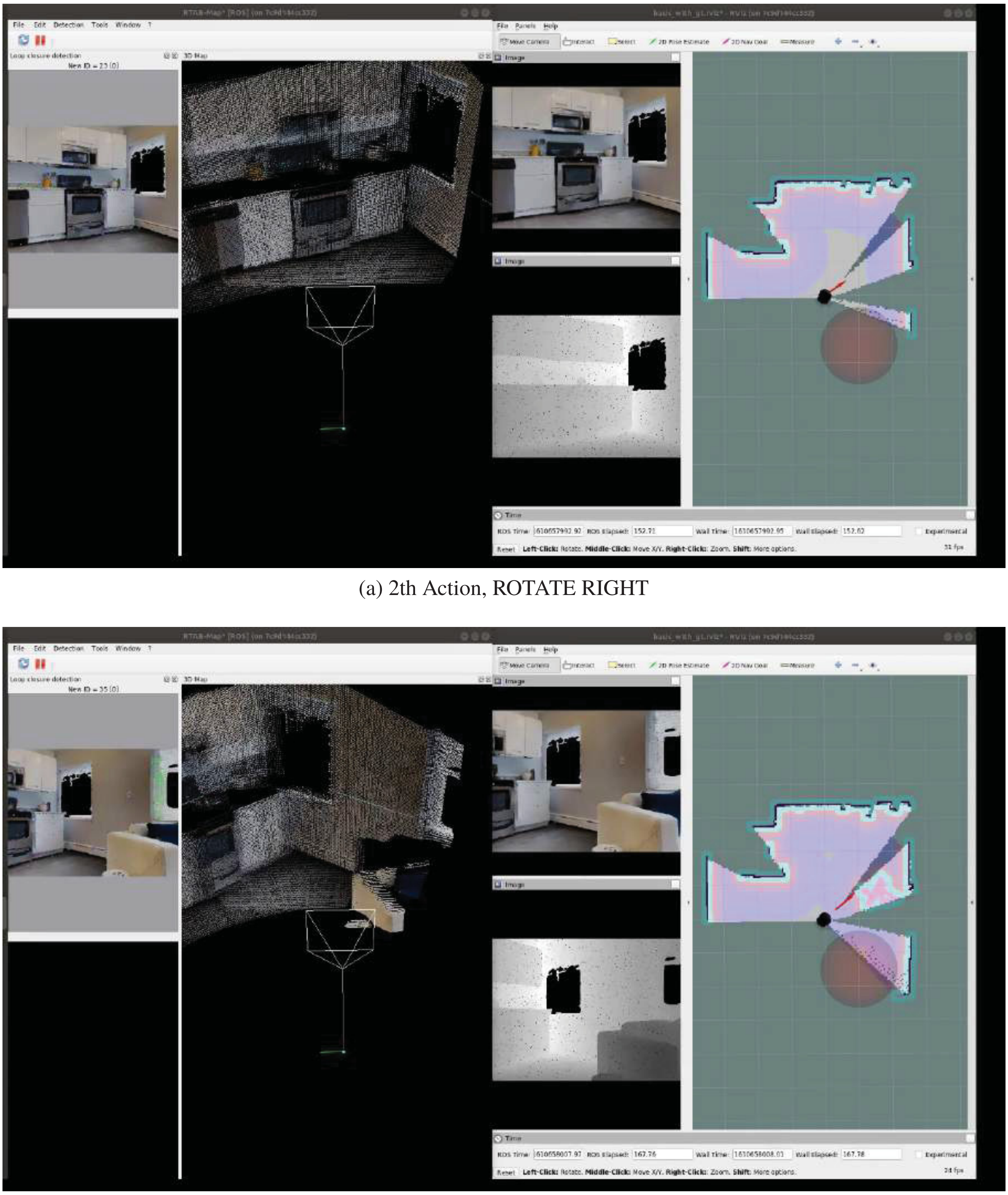
Figura 6.34: Top-down view of the Placida first floor. In blue is the illustration of the robot's initial pose, the white arrow inside denotes its orientation, and in green, the success area. Note that there is no object on the carpet, as the top view seems to indicate. The plotted agent's pose and success area are merely illustrative and not in proper scale.



(a) Initial State

(b) First Action, ROTATE RIGHT

Figura 6.35: BEyond Complete qualitative results. At each observation, an RGB point cloud is estimated by the RTAB-Map, sequential observations are aligned using the graph created by the method. The 3D Map is projected into a top-down 2D plane and broadcast to the path planners via ROS messages. The success area denoted by a red circle is not known by the agent. ORB features are used by RTAB-Map in the Bayes filter to detect loop closures using a bag-of-words, see section 2.5. Black regions on RGB-D images are holes in the mesh, an artifact of imperfect 3D reconstruction.



ROS Time: 1610657992.92 ROS Elapsed: 152.71 Wall Time: 1610657992.95 Wall Elapsed: 152.62

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.

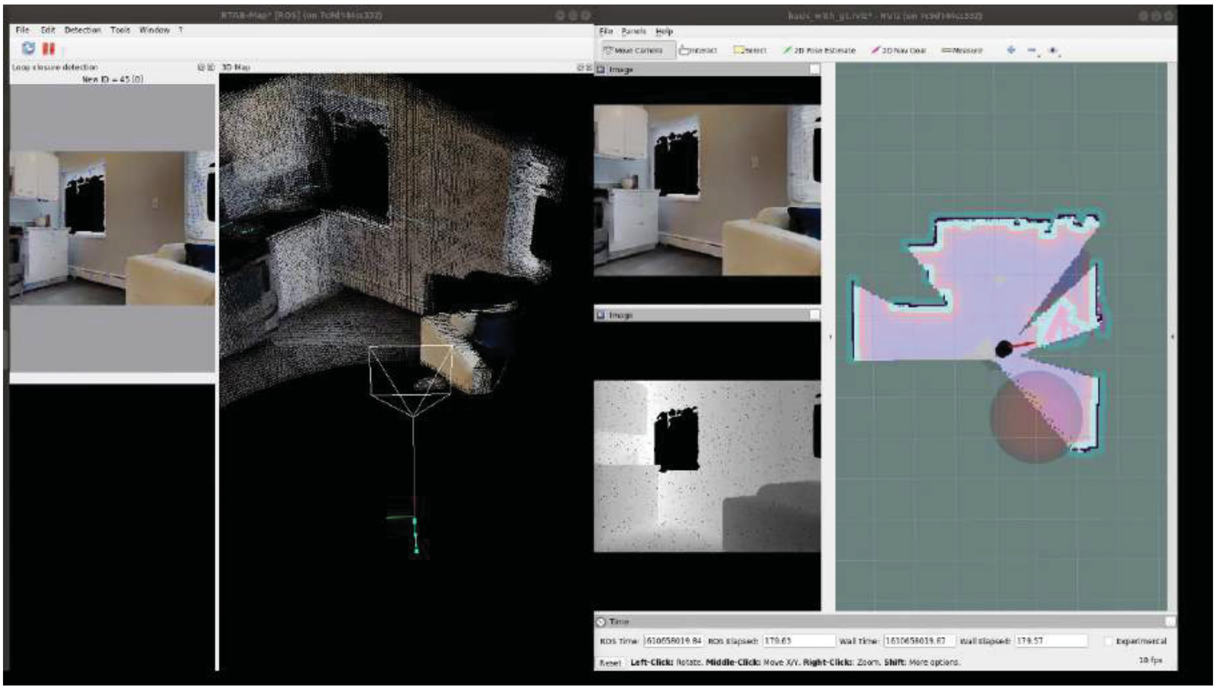
(a) 2th Action, ROTATE RIGHT

ROS Time: 1610658007.97 ROS Elapsed: 167.76 Wall Time: 1610658008.01 Wall Elapsed: 167.78

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.

(b) 3th Action, FORWARD

Figura 6.36: BEyond Complete qualitative results. Each subfigure displays the frame when the *n*-th pose goal is proposed.



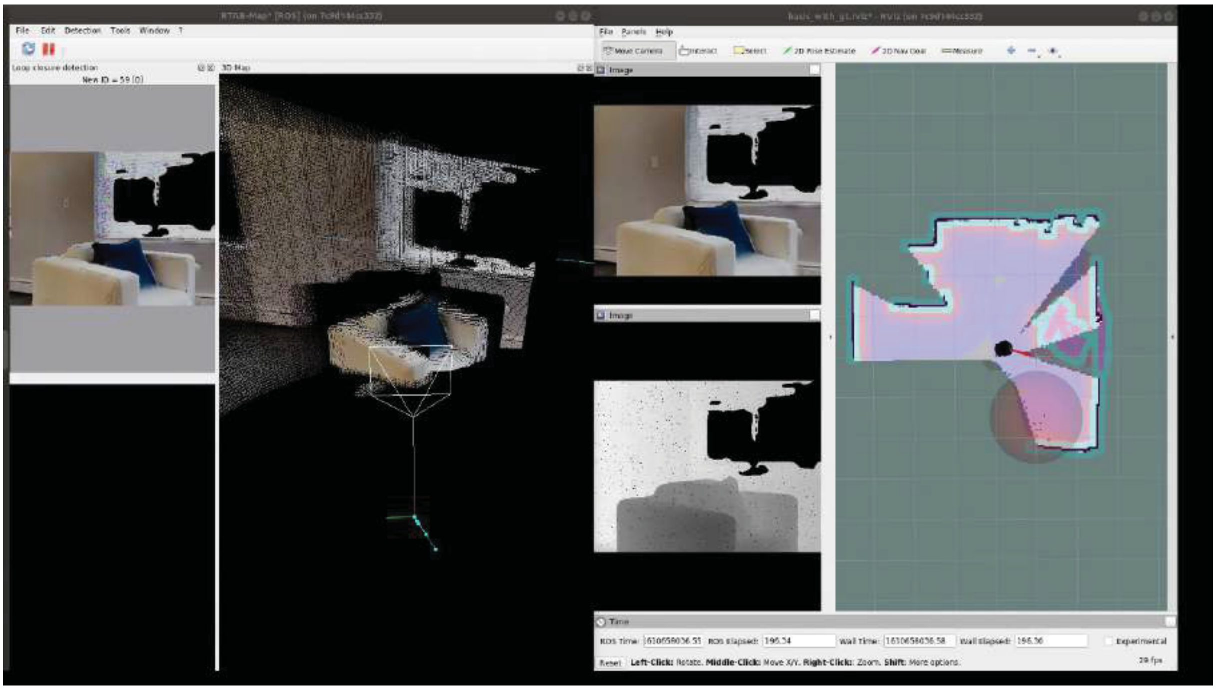
New ID = 45 (0)

ROS Time: 1610658019.84 ROS Elapsed: 179.63 Wall Time: 1610658019.87 Wall Elapsed: 179.57

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:** Zoom. **Shift:** More options.

10 fps

(a) 4th Action, ROTATE RIGHT



New ID = 59 (0)

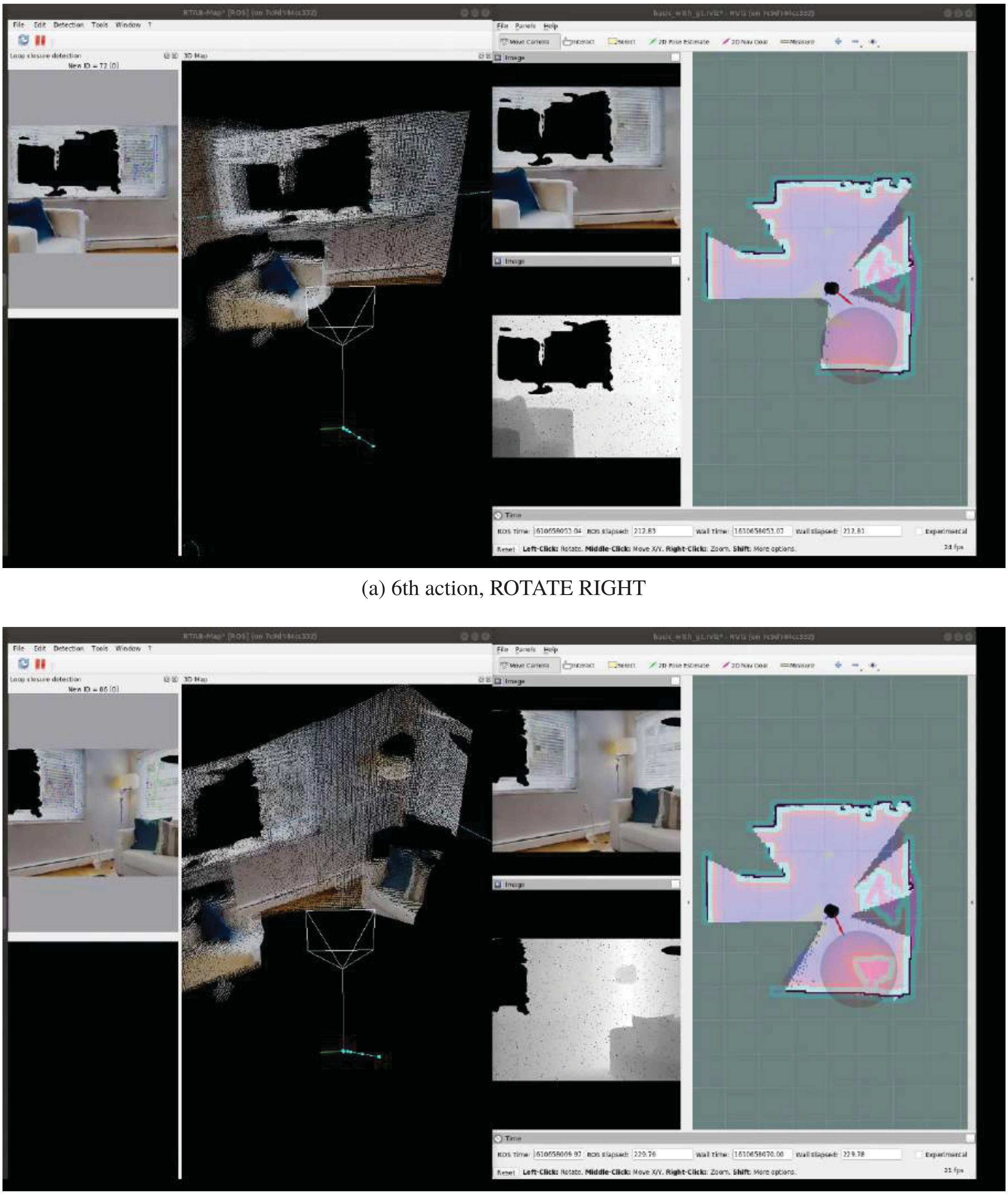
ROS Time: 1610658036.53 ROS Elapsed: 196.54 Wall Time: 1610658036.58 Wall Elapsed: 196.56

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:** Zoom. **Shift:** More options.

29 fps

(b) 5th Action, ROTATE RIGHT

Figura 6.37: BEyond Complete qualitative results. Each subfigure displays the frame when the *n*-th pose goal is proposed.



File Edit Detection Tools Window ?
Loop closure detection
New ID = 72 (0)
3D Map
File Panels Help
Move Camera Interact Select 2D Pose Estimate 2D Nav Goal Measure
Image
Image
Time
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.
24 fps

ROS Time: 1610658053.04 ROS Elapsed: 212.83 Wall Time: 1610658053.07 Wall Elapsed: 212.81 □ Experimental

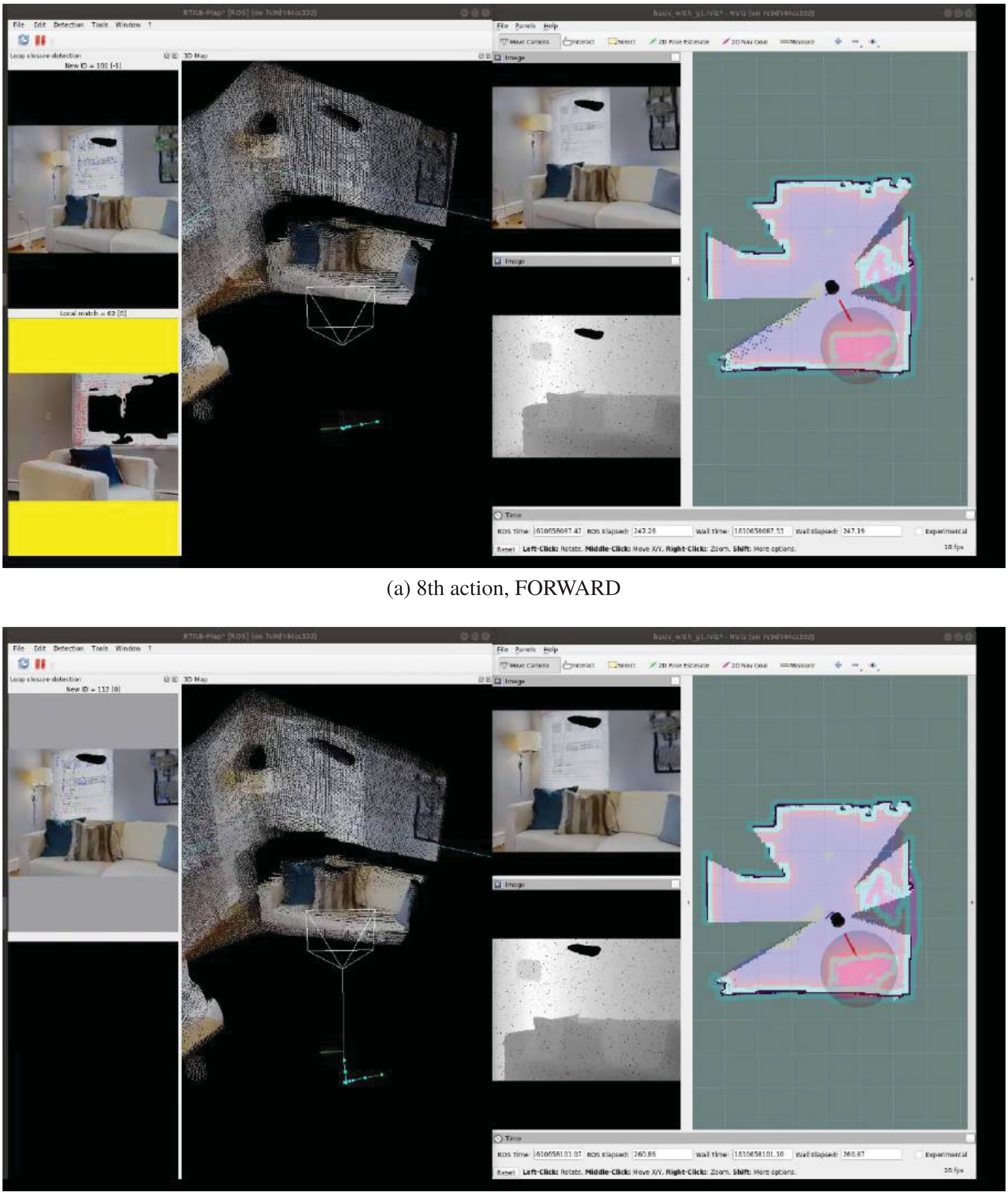
(a) 6th action, ROTATE RIGHT

File Edit Detection Tools Window ?
Loop closure detection
New ID = 86 (0)
3D Map
File Panels Help
Move Camera Interact Select 2D Pose Estimate 2D Nav Goal Measure
Image
Image
Time
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.
21 fps

ROS Time: 1610658069.97 ROS Elapsed: 229.76 Wall Time: 1610658070.00 Wall Elapsed: 229.78 □ Experimental

(b) 7th action, ROTATE RIGHT

Figura 6.38: BEyond Complete qualitative results. Each subfigure displays the frame when the *n*-th pose goal is proposed.



New ID = 101 [-1]

Local match = 62 [0]

ROS Time: 1610658087.47 ROS Elapsed: 247.26 Wall Time: 1610658087.53 Wall Elapsed: 247.19

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.

18 fps

(a) 8th action, FORWARD

New ID = 112 [0]

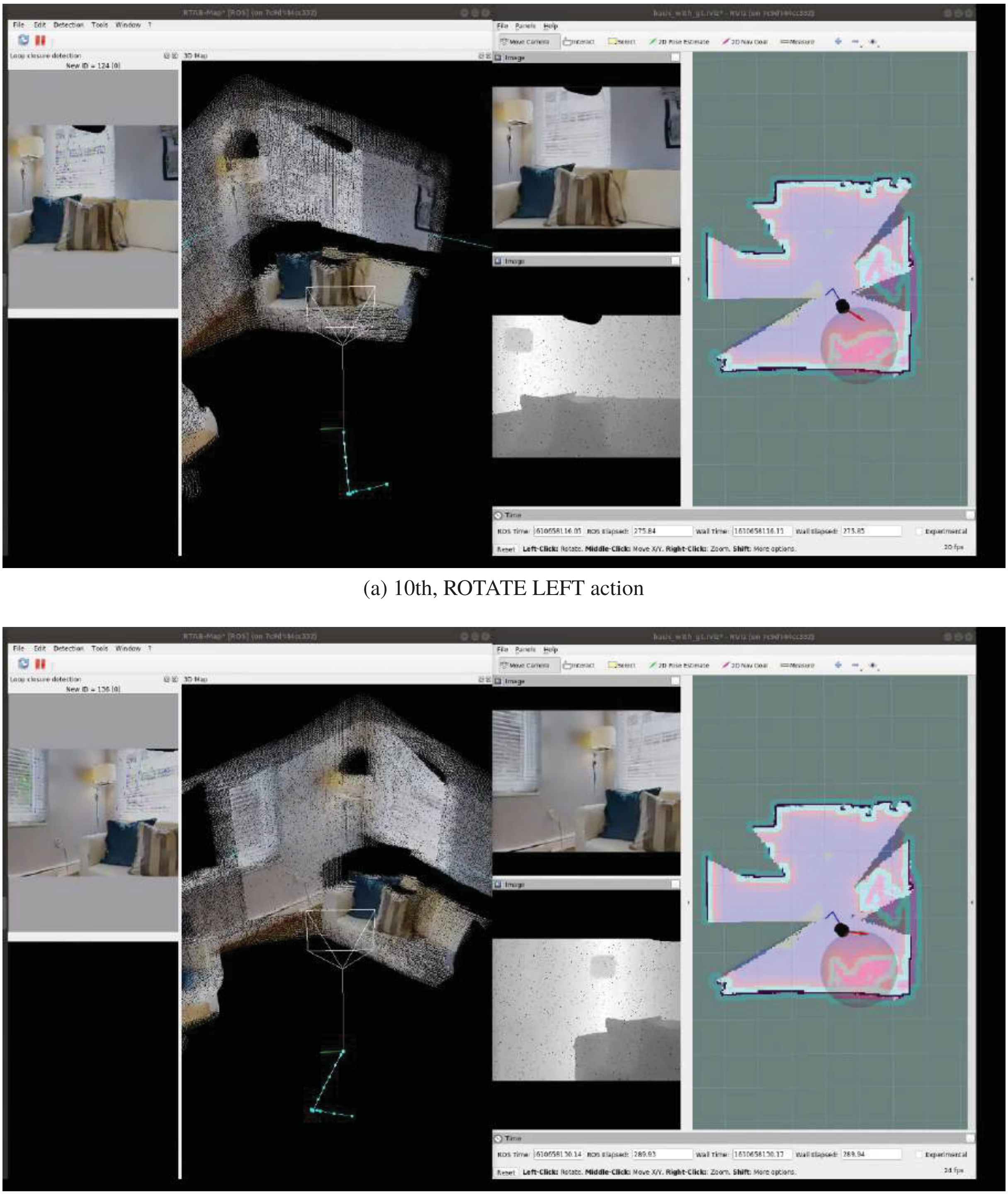
ROS Time: 1610658101.07 ROS Elapsed: 260.86 Wall Time: 1610658101.16 Wall Elapsed: 260.87

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.

26 fps

(b) 9th action, FORWARD

Figura 6.39: BEyond Complete qualitative results. Each subfigure displays the frame when the *n*-th pose goal is proposed.



New ID = 124 (0)

ROS Time: 1610658116.05 ROS Elapsed: 275.84 Wall Time: 1610658116.11 Wall Elapsed: 275.85

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.

20 fps

(a) 10th, ROTATE LEFT action

New ID = 136 (0)

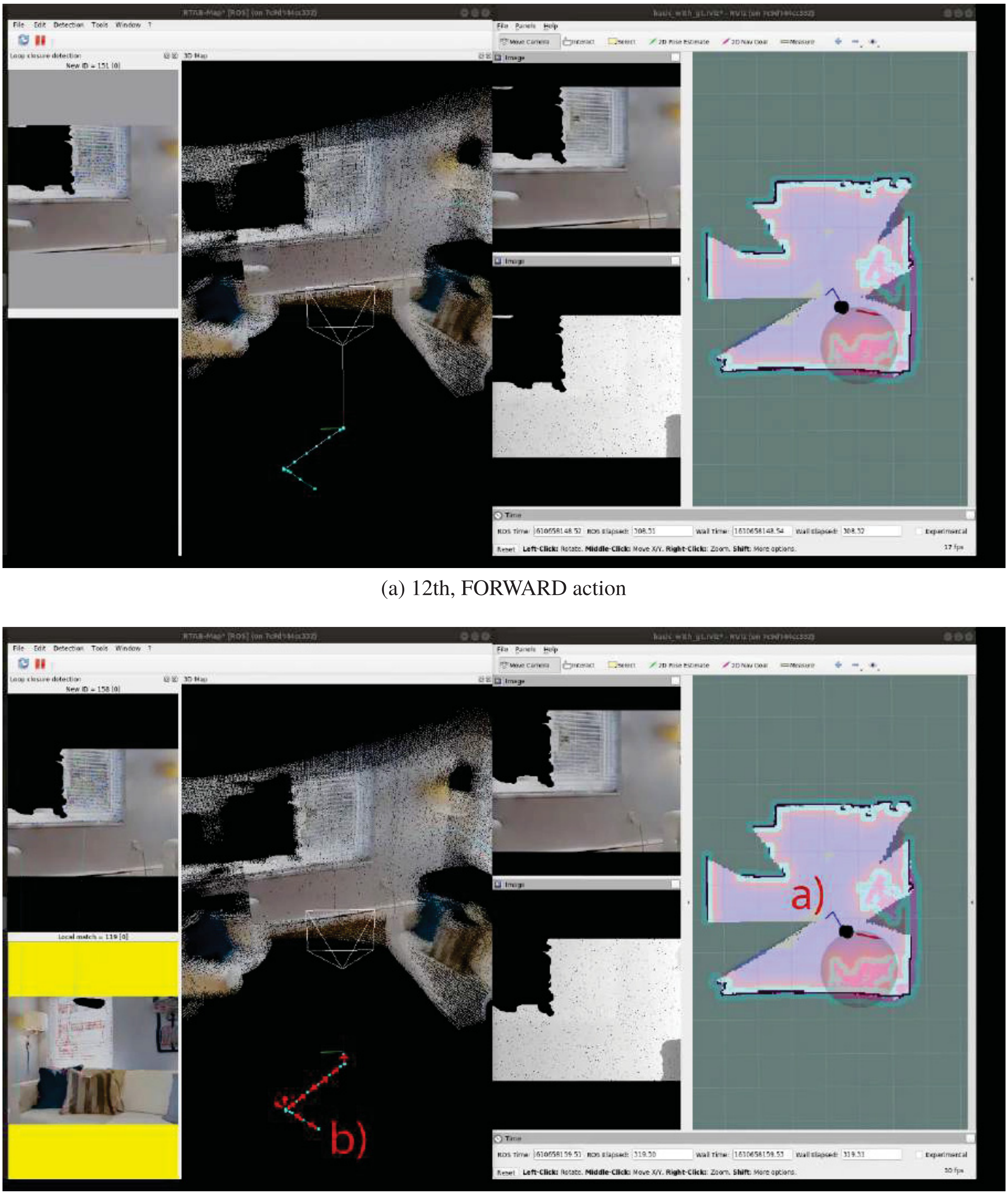
ROS Time: 1610658130.14 ROS Elapsed: 289.93 Wall Time: 1610658130.17 Wall Elapsed: 289.94

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.

24 fps

(b) 11th, ROTATE LEFT action

Figura 6.40: BEyond Complete qualitative results. Each subfigure displays the frame when the *n*-th pose goal is proposed.



(a) 12th, FORWARD action

(b) Final State

Figura 6.41: BEyond Complete qualitative results. On the right, denoted by a), the dark blue represents the trajectory as registered by the RTAB-Map the same is seen, denoted by b), in cyan on the left, where points representing the vertices can be observed as long with the edges, and the red arrows illustrate the orientation.

6.5 RESULTS OVERVIEW

As previously commented, the BEyond Complete had only qualitative results at the time of writing this text, but it was demonstrated that the BEyond pipeline retains the performance even on a more realistic scenario, with noises on the sensors and real-time interprocess communication.

Tables 6.6, 6.7, 6.8, 6.9 present a summarized version of the previous plots. Furthermore, present data related to the BEyond with Oracle variant. Distance range refers to the distance between the agent's starting position and the closest target. Distance to the goal measures the distance between the agent and the closest target at the STOP action.

Distance Range	Distance to Goal ↓	Success Rate ↑	Success # Abs ↑	SPL ↑	SoftSPL ↑
Set 0 [1,2] m	1.7117	0.1132	928	0.0822	0.1651
Set 1 [2,4] m	2.8790	0.0238	195	0.0184	**0.1711**
Set 2 [4,8] m	5.7487	0.0007	6	0.0005	0.0985

Tabela 6.6: Metrics resulting of the BEyond with Oracle Churchton scene evaluation. As the distance increases linearly the success rate falls exponentially, however the SoftSPL in Set 2 actually increased in relation to Set 1, and decreased in approximately half when comparing Set 2 and Set 3.

Distance Range	Distance to Goal ↓	Success Rate ↑	Success # Abs ↑	SPL ↑	SoftSPL ↑
Set 0 [1,2] m	1.7795	0.1149	942	0.0778	0.1624
Set 1 [2,4] m	3.2600	0.0194	159	0.0152	0.1406
Set 2 [4,8] m	5.8307	0.0008	7	0.0006	0.0991

Tabela 6.7: Metrics resulting of the BEyond with Oracle Emmaus scene evaluation. The same relation between distance to goal and sucess rate is observed in this scene. The SoftSPL on Set 2 did not surpassed Set 1 but suffered a drop of only 0.0218.

Distance Range	Distance to Goal ↓	Success Rate ↑	Success # Abs ↑	SPL ↑	SoftSPL ↑
Set 0 [1,2] m	1.7115	0.1284	1052	0.0877	0.1722
Set 1 [2,4] m	3.1064	0.0181	149	0.0150	0.1646
Set 2 [4,8] m	5.9838	0.0006	5	0.0004	0.0962

Tabela 6.8: Metrics resulting of the BEyond with Oracle Gravelly scene evaluation. The same relation between distance to goal and sucess rate is observed in this scene. The SoftSPL on Set 2 did not surpassed Set 1 but suffered a drop of only 0.0076.

Distance Range	Distance to Goal ↓	Success Rate ↑	Success # Abs ↑	SPL ↑	SoftSPL ↑
Set 0 [1,2] m	1.5708	0.1375	1127	0.0960	0.1919
Set 1 [2,4] m	3.3464	0.0156	128	0.0128	0.1390
Set 2 [4,8] m	5.9667	0.0002	2	0.0002	0.0929

Tabela 6.9: Metrics resulting of the BEyond with Oracle Micanopy scene evaluation.

The success rate is measured between [0,1], where one means that all episodes ended in success. Success # abs stand for the absolute number of episodes out of the total that was successful. Sets 0, 1, 2 have 2^{13} episodes per scene. SPL, which is the inverted path weighted by success, and SoftSPL, which measures the navigation performance without being influenced by the success rate, were previously described in section 2.17. These results are discussed in chapter 7.

To properly compare the performance between different approaches we run 4 different pipelines on the same 4 testing scenes and the results are presented on tables 6.10, 6.11, 6.12, 6.13. Baseline DD-PPO ObjectNav was the pre-trained baseline publicly available that was used in the Habitat-Challenge 2020. It was trained using scenes from the Matterport3D dataset. We used the class ID mapping, see table 6.4, to ensure that classes matched between datasets. These results are discussed in chapter 7.

Algorithm	Churchton Set 0			
	Distance to Goal ↓	Success Rate ↑	SPL ↑	SoftSPL ↑
Baseline DD-PPO ObjectNav	2.4787	0.0024	0.0003	0.0248
AUTOcrat	1.7053	0.0031	0.0025	0.1277
BEyond with Oracle	1.7117	**0.1132**	**0.0822**	**0.1651**
BEyond Intermediary	**1.6109**	0.0211	0.0141	0.1474

Tabela 6.10: Mean values for the 2^{13} episodes in each scene. Distance to goal refers to the remaining distance in the euclidean distance to the object when the agent stopped, i.e., 0 is the best value. Best results are highlighted in bold letters.

Algorithm	Emmaus Set 0			
	Distance to Goal ↓	Success Rate ↑	SPL ↑	SoftSPL ↑
Baseline DD-PPO ObjectNav	2.7373	0.0109	0.0029	0.0356
AUTOcrat	1.7844	0.0037	0.0024	0.1332
BEyond with Oracle	1.7795	**0.1149**	**0.0778**	**0.1624**
BEyond Intermediary	**1.5749**	0.0213	0.0135	0.1484

Tabela 6.11: Mean values for the 2^{13} episodes in each scene. Distance to goal refers to the remaining distance in the euclidean distance to the object when the agent stopped, i.e., 0 is the best value. Best results are highlighted in bold letters.

Algorithm	Gravelly Set 0			
	Distance to Goal ↓	Success Rate ↑	SPL ↑	SoftSPL ↑
Baseline DD-PPO ObjectNav	2.8731	0.0212	0.0066	0.0547
AUTOcrat	1.9913	0.0047	0.0033	0.1397
BEyond with Oracle	1.7115	**0.1284**	**0.0877**	**0.1722**
BEyond Intermediary	**1.5452**	0.0146	0.0102	0.1486

Tabela 6.12: Mean values for the 2^{13} episodes in each scene. Distance to goal refers to the remaining distance in the euclidean distance to the object when the agent stopped, i.e., 0 is the best value. Best results are highlighted in bold letters.

Algorithm	Micanopy Set 0			
	Distance to Goal ↓	Success Rate ↑	SPL ↑	SoftSPL ↑
Baseline DD-PPO ObjectNav	2.4348	0.0135	0.0035	0.0400
AUTOcrat	1.5509	0.0064	0.0043	0.1598
BEyond with Oracle	1.5708	**0.1375**	**0.0960**	**0.1919**
BEyond Intermediary	**1.4741**	0.0200	0.0135	0.1544

Tabela 6.13: Mean values for the 2^{13} episodes in each scene. Distance to goal refers to the remaining distance in the euclidean distance to the object when the agent stopped, i.e., 0 is the best value. Best results are highlighted in bold letters.

6.6 CHAPTER SUMMARY

In this chapter, we described and presented the results of our experiments for our three proposed pipelines.

EXchangeable's experiments were divided into the simulation, performed on the Gazebo simulator, and real-world performed with our real robot inside Federal University of Paraná/*Universidade Federal do Paraná* (UFPR). The agent could perform point-based navigation and avoid static obstacles.

AUTOcrat was evaluated using the Habitat simulator, using scenes from Matterpord3D to train and 4 scenes from Gibson to test the agent's performance. The scenes were Churchton, Emmaus, Gravelly, and Micanopy. The 8 classes of objects used were chair, plant, sofa, bed, table, toilet, tv, refrigerator. The metrics used were the number of discrete steps, the SoftSPL, SPL, and Geodesic Distance. The 3 distinct sets of episodes generated for evaluation were also presented. Although AUTOcrat could perform successfully ObjectNav, it had issues with collisions and some of the target classes, reaching success in at most two of the eight target classes on the four scenes.

The experiments performed with BEyond were quantitative and qualitative. The focus was given to the BEyond with Oracle to properly evaluate our Goal prediction network in different sets of episodes in increasing complexity in a disentangled performance from motion planning. The metric First Target Observation was introduced and demonstrated. To evaluate the BEyond quantitatively in an entangled manner, the BEyond Intermediary was used. Moreover, finally, to demonstrate qualitatively, the proof-of-concept BEyond Complete was used in an experiment using one episode on the Placida scene belonging to the Gibson dataset.

7 DISCUSSION

In this chapter, we discuss the results obtained by our proposed pipelines and how they compare with other state-of-the-art methods.

The success rate fall drastically as the episode minimal geodesic distance increase, see tables 6.6, 6.7, 6.8, 6.9. We observed that the longer the episode the higher the chance of one or more obstacles blocking a straight-line path to the goal. Also, complex maneuvers mean more steps and more sequential goal predictions. Due to compound probability, the performance of the agent quickly decays.

As previously discussed in 2.17.1, not all failures are equal. Even if the agent stops just before the success zone, it is a failure. SoftSPL (Datta et al., 2020) was proposed to mitigate this issue. Unlike the success rate and the SPL that are majorly impacted by if the agent stopped inside the success zone or not, the SoftSPL measures the agent's trajectory adherence to the oracle's shortest path ground truth. A high SoftSPL with a low SPL is indicative that the agent ran close to the shortest path but chose an early stop. The opposite means that the agent took a sub-optimal path, perhaps exploring the environment but ultimately reached the goal.

The ideal agent will stop as close as possible to the target. Thus, we want to minimize early stops. An agent that maps an egocentric observation directly to an action is sensitive to changes on the observation that should issue the STOP action. This sensitivity is tied to the model's robustness against underfitting. Since the STOP action is the class with the fewest examples, only one per episode, it can lead to underfitting, which means that novel poses can cause wrong predictions if the model is not extensively trained or there are no additional modules within the model to account for that. We hypothesize that by separating semantics from the motion planning, we could achieve better performance with BEyond than AUTOcrat. Our results confirmed this, see tables 6.10, 6.11, 6.12, 6.13.

Our initial hypothesis was that the BEyond with Oracle would perform better than the other later variants of BEyond because it had access to ground truth semantic segmentation and an oracle motion planner that would likely never collide and plan the shortest path to the predicted point goal. This was confirmed with our results, see tables 6.10, 6.11, 6.12, 6.13. Although BEyond with Oracle had a better overall performance than BEyond with Intermediary as hypothesized, BEyond with Intermediary surpassed all other approaches on the distance to goal.

The distance to goal is instrumental with this set of episodes. Set 0 was generated with episodes where the starting location is guaranteed to be at least 1 meter away from any target objects and at the max 2 meters as measured by geodesic distance. Meaning that if the distance to the goal, which measures the distance between the agent and the closest target at the stop action, is higher than 2, the agent stopped more distant from the target than when it started instead of closer. The baseline was the only case where the agent ended diverging from the target on average.

The BEyond Intermediary had the best distance to goal on all scenes even though the SoftSPL was smaller than BEyond with Oracle. This is likely because the DD-PPO motion planner took a sub-optimal path but still got closer to the goal before issuing a STOP action. If the predicted point goal is not navigable, the Oracle will immediately issue a STOP action causing an early stop. The implication is that out of the four algorithms evaluated, BEyond Intermediary was the least prone to early stops.

AUTOcrat surpassed the baseline DD-PPO on the SoftSPL and distance to goal on all four scenes and only surpassed the success rate on the Churchton scene. Both BEyond variants surpassed both the baseline and AUTOcrat in all metrics and all four scenes, meaning that the modular approach utilized in BEyond is more suited for this task than the end-to-end design we used in AUTOcrat.

The results obtained with BEyond Complete are merely for the sake of demonstration and present qualitative results since it is a concept not fully explored. A more extensive evaluation of this proposed pipeline was left for future works, and it is related to the sim-to-real discussion in section 8.1.

7.1 TRANSFER LEARNING TO MATTERPORT3D CLASSES

To present a comparison to other methods submitted to Habitat-Challenge 2020 ObjectNav, we did a transfer learning to the 21 Matterport3D classes used in the competition. Due to limited hardware resources, we used only a subset of the original training set available to the participants to a balanced subset with around 2^{12} episodes per class divided between scenes, see table 7.1. We believe that the use of a balanced subset could provide better training than an unbalanced one. An evaluation of using only a subset against the full set of episodes was left for future works.

Class	Original Training Set	Our Subset
chair	545,293	4,125
sofa	78,947	4,128
picture	135,163	4,140
table	337,131	4,144
plant	116,657	4,100
stool	82,165	4,100
counter	69,853	4,140
cabinet	213,887	4,108
fireplace	23,862	4,108
cushion	363,213	4,136
bed	97,150	4,116
chest_of_drawers	104,915	4,140
sink	115,331	4,116
bathtub	29,019	4,107
towel	83,044	4,116
toilet	66,693	4,140
seating	67,205	4,037
shower	57,437	4,116
gym_equipment	7,610	3,614
tv_monitor	20,677	4,100
clothes	17,170	3,972

Tabela 7.1: Difference between the original training set and our subset. Our subset has almost the same number of episodes per object, reducing the bias towards certain classes.

A major issue in this work is that we did not have access to any other methods implementations or the testing set used during the competition to ensure a completely fair comparison. We present in table 7.2 a comparison between our method and the results reported on the online leaderboard of the competition on the val_mini set, a sanity set, a small set created to ensure that the local metrics obtained by the participants and the remote ones were equal. This set is particularly biased towards some classes, do not have an episode for each of the classes, and it contains only 30 episodes, meaning that the result only serves to provide some notion on which ranking our method would fit. This is far from the ideal comparison since there are major differences in performance and datasets between the val_mini, test_dev, and test_challenge sets used in the competition. Nevertheless, after 180,224 steps of transfer learning, our method achieved the 5th rank between the 12 algorithms on the leaderboard.

Rank	Participant team	SPL ↑	SoftSPL ↑	Distance to goal ↓	Success ↑
1	Arnold (SemExp) (Chaplot et al., 2020)	0.24587	0.28115	3.33397	0.46667
2	SRCB-robot-sudoer	0.12439	0.20050	4.84811	0.23333
3	Active Exploration (Pre-explore)	0.10840	0.21828	5.07862	0.16667
4	Blue Ox	0.08304	0.18793	4.25381	0.13333
5	**BEyond_with_DDPPO_180224**	**0.00817**	**0.08793**	**5.96305**	**0.03333**
6	Casa	0.00000	0.04773	6.53870	0.00000
7	Objects In Mirror Are Closer Than They Appear	0.00000	0.00791	6.32285	0.00000
8	Forward_Only	0.00000	0.01880	6.61044	0.00000
9	Clever Tortoise	0.00000	0.02601	6.28284	0.00000
10	PPO RGBD	0.00000	0.02994	6.05539	0.00000
11	Black Sheep	0.00000	0.02739	6.35476	0.00000
12	Habitat Team (RandomAgent)	0.00000	0.00513	6.37901	0.00000

Tabela 7.2: Comparison with Habitat Challenge 2020 ObjectNav val_mini set. The sanity set is a small set with 30 episodes in total, which is publicly available and used to verify if the participants' local metrics and the remote submissions are equal. Our method achieved the 5th rank.

8 CONCLUSIONS

In this work, we provided a discussion about the Embodied Artificial Intelligence (E-AI) task called Semantic Navigation, a problem still regarded as in its infancy. We discussed the competitions held to evaluate current state-of-the-art, the metrics used and their limitations, the simulators, and the environments used, and our agent's embodiment, the VRIBot, our physical robot built with by Commercial Off-The-Shelf (COTS) components. We proposed and evaluated three pipelines and their variants for navigation: EXchangable, AUTOcrat, and BEyond.

EXchangable allowed us to experiment with traditional point-based navigation using multiple sensors, LiDAR, RGB-D, wheel encoders, and sonar with our own VRIBot. This was fundamental as a proof-of-concept for our agent's embodiment, and we had successful results in our early experiments. Experimenting early on with a real robot shed some light on the limitations of model-based approaches and the sensors measurements' stochastic nature.

The AUTOcrat was our first attempt towards a module that would handle the semantics, a way to perform semantic navigation instead of traditional point-based navigation. This approach allowed us to understand the challenges of connecting semantics with navigation.

In BEyond, we proposed our own architecture for a goal prediction neural network that uses a fusion of top-down semantic map, the agent's trajectory, and a target object class ID to predict a point goal towards an instance of the target object class. It was designed as a bridge between semantics and point-based navigation. The proposed net was evaluated using the BEyond pipeline and different reconstructed scenes from real physical buildings.

Our main contributions are:

- An overview of the state-of-the-art of E-AI Semantic Navigation, see sections 2.15, 3, 4.
- VRIBot, an embodiment that is modular and can be easily assembled with COTS components to achieve a similar robot to popular ones such as LoCoBot (Carnegie Mellon University, 2019) and TurtleBot (Wise and Foote, 2010). Additionally, the embodiment can be simulated in simulators compatible with Unified Robot Description Format (URDF) (Open Robotics, 2019) and Robot Operating System (ROS) (Open Robotics, 2018), and the agent's internal logic can be used in real and simulated environments, see section 5.1
- A methodology to scrutinize the relation between episode settings and the agent's performance, see section 6.2.1. The proposal and demonstration of the First Target Observation metric which is useful to determine which steps of an episode contain exploration and which contain steering towards the target, see section 6.3.
- Our goal prediction network that can bridge dedicated modules of semantic segmentation and motion planning, see section 5.5.2.
- Our promising approach named BEyond reached 5th rank out of 12 on the val_mini set of the Habitat-Challenge 2020 Object-Goal Navigation (ObjectNav) when compared to other reported results on the leaderboard, see section 7.1.
- The code related with this work is publicly available at `https://github.com/VRI-UFPR/BeyondSight`

The goal prediction net could encode the episode's state to a latent representation of the semantic map and the agent's trajectory and use it to predict point goals as intended. However, a weakness in our approach was that the goal prediction net had no way to keep track of previous predictions. If sequential point goal predictions vary too much, they undermine progress instead of encouraging it. Our attempt to mitigate this issue with a heuristic was described in section 5.5.2. Nevertheless, this was not as effective, as can be seen by the low success rate of the episodes. Perhaps a more effective solution would be to predict heatmaps and use the closest centroid of clusters within the prediction than regressing a single point per step. In section 8.1 we briefly discuss how exploiting attention could also improve our architecture.

Imitation Learning was more effective in BEyond than in AUTOcrat. In AUTOcrat, the agent was instructed in every step without being encouraged to learn to explore semantic data usage since the oracle providing the ground truth for training does not need to explore semantic data due to its perfect knowledge. In BEyond, the agent was encouraged to correlate the top-down semantic map input with the point goals offered as training data by the oracle. We believe that employing Deep Reinforcement Learning (Deep-RL) using positive rewards for setting point goals in the vicinity of the target objects would be a better training procedure since it would remove the necessity and limitations of an oracle, which was left for future works.

In all variants of BEyond, only the goal prediction network's model parameters were updated with training. In future works, we want to explore a training procedure and loss function that optimizes all trainable models within the pipeline in a single pass, akin to how the modules in Generative adversarial networks (GANs) (Goodfellow et al., 2014) compete and improve together.

Long-horizon tasks have ever been complex to solve. The longer the task, the more data needs to be handled, and more possibilities for branching decisions exist. In a non-deterministic and unconstrained scenario, it can quickly become intractable.

There still a large room for improvement for E-AI tasks, especially ObjectNav, even the winner of the Habitat-Challenge 2020 ObjectNav reached a success rate of only 25%. In RoboTHOR 2020, the winner achieved a success rate of 32%. The egocentric PointNav without GPS or Compass category of Habitat-Challenge 2020 was not so far off, where the winner had a success rate of 28%.

This particular task was fairly recently proposed, and there are still some open issues to address. How long should be the episodes, how many object classes should be used, how big should be the navigable area, which are the best metrics to evaluate the navigation, should obstacle avoidance and target object visibility one of the cornerstones of the task, should it be limited to indoor environments? Semantic navigation is seen as an important step towards generalization and autonomy for robots, a way to use robots in more unconstrained environments. It is a highly complex task and involves solving a series of sub-problems that usually tackle individually, i.e., object detection, motion planning, semantic reasoning, image framing, and so on.

Ultimately, none of our methods reached the same performance as the top 3 methods on the leaderboard for the Habitat-Challenge 2020 ObjectNav competition, but BEyond did surpass several others on the val_mini set, meaning that it is a promising approach with a large room for improvements.

8.1 FUTURE WORKS

Sequential tasks like navigation usually employ RNNs, particularly the LSTM. However, LSTMs have three main issues: they are not sensitive to transfer learning, they can cause exploding gradients during training, and current implementations on famous frameworks such as Tensorflow and PyTorch are not particularly efficient, variations of the LSTM try to mitigate that such as the IndRNN (Li et al., 2018), causing a tendency to replace LSTMs by the Transformer architecture (Vaswani et al., 2017).

The Transformer architecture (Vaswani et al., 2017), has been widely adopted within the Natural Language Processing (NLP) community (Devlin et al., 2019; Lee et al., 2019; Brown et al., 2020; Zhang et al., 2020) due to its self-attention mechanism to compute representations of its input and output without using sequence-aligned RNNs or convolution. Generative Pre-trained Transformer 3 (GTP-3) (Brown et al., 2020) achieved impressive results with a massive model of 175 billion parameters. Their approach was a single model that learned an internal understanding of language and can be fine-tuned with zero, one, and few-shot learning and solve different tasks. With this, GTP-3 surpassed state-of-the-art models that were extremely fine-tuned to particular tasks.

The idea of one-shot and few-shot learning is that the new task is described with a single or a few examples. After that, the model should be able to solve the task. A different approach to a standard supervised learning where the model needs massive amounts of data before it can solve the task (Estevam et al., 2021). Bruce et al. (2017) use one-shot learning as a way to fine-tune robot navigation to measurements obtained in a real embodiment.

This same concept can be adapted to actions within an environment, which inspired the work of Fang et al. (2019), a Scene Memory Transformer for Embodied Agents in Long-Horizon Tasks. Thus, we are interested in investigating transformers as memory encoders for ObjectNav in future works.

Also inspired by attention is the Salient Object Detection (SOD) field that deals with visual attention, known as visual salience. The area includes dealing with segmentation as seen in SOD (Krinski et al., 2019). In our published work (Ruiz et al., 2019) a novel data augmentation for SOD was proposed and improved in (Ruiz et al., 2020). Ruiz et al. (2019) was selected as one of the best papers presented on the ICAR 2019 – International Conference on Advanced Robotics. The coupling and evaluation of SOD within ObjectNav were left for future works.

Progress in NLP has encouraged the Vision-and-Language Navigation (VLN) field, a type of E-AI task that is a converging field between semantic navigation, NLP and and Human-Robot Interaction (HRI). Progress in ObjectNav would greatly benefit other more complex tasks such as VLN.

Being able to train agents in a simulated environment is safer, and it can be done faster than in real-time. Zhao et al. (2020) provided a survey on sim-to-real transfer for Deep-RL; according to them, the main areas are domain randomization, domain adaptation, imitation learning, meta-learning, and knowledge distillation.

The gap between simulation and reality is also an important issue. It has inspired several works on domain adaptation like one of our publications (Ruiz et al., 2020), which received the award of honorable mention on the full paper category at the COTB 2020 conference where it was presented. Where we evaluated quantitatively and qualitatively the transformation of objects along with their labeled mask, i.e., an RGB image and a binary mask, from one class to another and vice-versa that have different scales, textures, and morphology, while preserving background information, pose lighting conditions, and other features. An important distinction between the method evaluated and traditional morphing is that intermediary solutions should generate images that make sense on their own and should produce pertinent segmentation mask along with the transformed RGB image.

This raises the questions: does an evaluation in simulation predict real-world performance? If one method outperforms another in simulation, how likely is that trend to hold in reality on a robot? Crucial questions addressed by Kadian et al. (2020). In which the authors investigated the sim-to-real predictivity of Habitat-Sim for PointGoal navigation. They used a 3D-scan of physical lab space to create a virtualized replica and ran parallel tests of 9 different models in reality and simulation. The authors propose a new metric called Sim-vs-Real Correlation Coefficient (SRCC).

If SRCC is high (close to 1), the progress made in the simulation will translate to 'real' progress because the improvements in a simulation will generalize to real robotic testbeds. If SRCC is low (close to 0), the results achieved in the simulation will not hold up in reality. They present three main findings: parameterization on the simulator can make a huge difference on the SRCC; good performance could be caused by 'cheating', where the agent 'slide' along walls on collision. Essentially, the virtual robot is capable of cutting corners by sliding around obstacles, leading to unrealistic shortcuts through parts of non-navigable space and 'better than optimal' paths. Naturally, such exploits do not work in the real world where the robot stops in contact with walls; the ranking between models reported at the competition leaderboard was not preserved on their experiments with a real robot, suggesting that the results/winners may not be the same if the challenge were performed with a real agent in a real environment.

Anderson et al. (2020) evaluated sim-to-real transfer for VLN in a reconstructed building named Coda. A $325m^2$ collaborative shared space in a commercial office building. They found that sim-to-real transfer to an environment not seen in training is successful if an occupancy map and navigation graph can be collected and annotated in advance (success rate of 46.8% vs. 55.9% in the sim), but much more challenging in the hardest setting with no prior mapping at all (success rate of 22.5%). Evaluation of the sim-to-real transfer of our proposed BEyond Complete was left for future works.

REFERENCES

Allen Institute for AI (2020). RoboTHOR Challenge 2020. `https://eval.ai/web/challenges/challenge-page/558/`. Accessed: 2020-10-30.

Alvarez, J. C., Shkel, A., and Lumelsky, V. (1998). Accounting for mobile robot dynamics in sensor-based motion planning: experimental results. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, volume 3, pages 2205–2210 vol.3.

Alves, J. H. and d. Oliveira, L. F. (2020). Optimizing neural architecture search using limited GPU time in a dynamic search space: A gene expression programming approach. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8.

Alves, J. H., Neto, P. M. M., and Oliveira, L. F. (2018). Extracting lungs from CT images using fully convolutional networks. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Anderson, P., Chang, A., Chaplot, D. S., Dosovitskiy, A., Gupta, S., Koltun, V., Kosecka, J., Malik, J., Mottaghi, R., Savva, M., et al. (2018a). On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*.

Anderson, P., Shrivastava, A., Truong, J., Majumdar, A., Parikh, D., Batra, D., and Lee, S. (2020). Sim-to-real transfer for vision-and-language navigation. *arXiv preprint arXiv:2011.03807*.

Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., Reid, I., Gould, S., and van den Hengel, A. (2018b). Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., and Parikh, D. (2015). VQA: Visual Question Answering. In *International Conference on Computer Vision (ICCV)*.

Armeni, I., He, Z.-Y., Gwak, J., Zamir, A. R., Fischer, M., Malik, J., and Savarese, S. (2019). 3d scene graph: A structure for unified semantics, 3d space, and camera. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5664–5673.

Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495.

Bailey, T. and Durrant-Whyte, H. (2006). Simultaneous localization and mapping (slam): Part ii. *IEEE robotics & automation magazine*, 13(3):108–117.

Batra, D., Gokaslan, A., Kembhavi, A., Maksymets, O., Mottaghi, R., Savva, M., Toshev, A., and Wijmans, E. (2020). Objectnav revisited: On evaluation of embodied agents navigating to objects. *arXiv preprint arXiv:2006.13171*.

Bolya, D., Zhou, C., Xiao, F., and Lee, Y. J. (2020). Yolact++: Better real-time instance segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–1.

Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Bruce, J., Suenderhauf, N., Mirowski, P., Hadsell, R., and Milford, M. (2017). One-shot reinforcement learning for robot navigation with interactive replay. In *Conf. on Neural Information Processing Systems (NIPS), Workshop on Acting and Interacting in the Real World: Challenges in Robot Learning. PDF*, Long Beach, USA.

Bujanca, M., Gafton, P., Saeedi, S., Nisbet, A., Bodin, B., Michael F.P., O., Davison, A. J., Paul H.J., K., Riley, G., Lennox, B., Luján, M., and Furber, S. (2019). Slambench 3.0: Systematic automated reproducible evaluation of slam systems for robot vision challenges and scene understanding. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6351–6358.

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. (2016). Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332.

Caminal, I., Casas, J. R., and Royo, S. (2018). Slam-based 3d outdoor reconstructions from lidar data. In *2018 International Conference on 3D Immersion (IC3D)*, pages 1–8.

Carnegie Mellon University (2019). Locobot, an open source low cost robot. `http://www.locobot.org/`. Accessed: 2020-10-29.

Catapang, A. N. and Ramos, M. (2016). Obstacle detection using a 2d lidar system for an autonomous vehicle. In *2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pages 441–445.

Chang, A., Dai, A., Funkhouser, T., Halber, M., Niessner, M., Savva, M., Song, S., Zeng, A., and Zhang, Y. (2017). Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*.

Chaplot, D. S., Gandhi, D. P., Gupta, A., and Salakhutdinov, R. R. (2020). Object goal navigation using goal-oriented semantic exploration. *Advances in Neural Information Processing Systems*, 33.

Chen, C., Wang, B., Lu, C. X., Trigoni, N., and Markham, A. (2020). A survey on deep learning for localization and mapping: Towards the age of spatial machine intelligence. *arXiv preprint arXiv:2006.12567*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Coumans, E. and Bai, Y. (2016–2019). Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`. Accessed: 2020-10-25.

Crespo, J., Castillo, J. C., Mozos, O. M., and Barber, R. (2020). Semantic information for robot navigation: A survey. *Applied Sciences*, 10(2).

Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 303–312, New York, NY, USA. Association for Computing Machinery.

Dai, A., Niessner, M., Zollhöfer, M., Izadi, S., and Theobalt, C. (2017). Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. *ACM Trans. Graph.*, 36(4).

Dai, A. and Nießner, M. (2018). 3dmv: Joint 3d-multi-view prediction for 3d semantic scene segmentation. *Lecture Notes in Computer Science*, page 458–474.

Das, A., Datta, S., Gkioxari, G., Lee, S., Parikh, D., and Batra, D. (2018). Embodied question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–10.

Datta, S., Maksymets, O., Hoffman, J., Lee, S., Batra, D., and Parikh, D. (2020). Integrating egocentric localization for more realistic point-goal navigation agents. *arXiv preprint arXiv:2009.03231*.

Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (2007). Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067.

Deitke, M., Han, W., Herrasti, A., Kembhavi, A., Kolve, E., Mottaghi, R., Salvador, J., Schwenk, D., VanderBilt, E., Wallingford, M., et al. (2020). Robothor: An open simulation-to-real embodied ai platform. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3164–3174.

Dellaert, F. (2012). Factor graphs and gtsam: A hands-on introduction. Technical report, Georgia Institute of Technology.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271.

Dissanayake, G., Huang, S., Wang, Z., and Ranasinghe, R. (2011). A review of recent developments in simultaneous localization and mapping. In *2011 6th International Conference on Industrial and Information Systems*, pages 477–482.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16.

Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110.

Durrant-Whyte, H. and Henderson, T. C. (2008). *Multisensor Data Fusion*, pages 585–610. Springer Berlin Heidelberg, Berlin, Heidelberg.

Engel, J., Schöps, T., and Cremers, D. (2014). Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer.

Estevam, V., Pedrini, H., and Menotti, D. (2021). Zero-shot action recognition in videos: A survey. *Neurocomputing*.

Facebook AI Research (2019). Habitat Embodied Agents Workshop. CVPR 2019. `https://aihabitat.org/challenge/2019/`. Accessed: 2020-10-20.

Facebook AI Research (2020). CVPR 2020 Embodied AI Workshop. `https://embodied-ai.org/`. Accessed: 2020-10-20.

FAIR A-STAR (Habitat)) (2020). Habitat challenge 2020. `https://eval.ai/web/challenges/challenge-page/580/`. Accessed: 2020-10-31.

Fang, K., Toshev, A., Fei-Fei, L., and Savarese, S. (2019). Scene memory transformer for embodied agents in long-horizon tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Finman, R., Whelan, T., Kaess, M., and Leonard, J. J. (2013). Toward lifelong object segmentation from change detection in dense rgb-d maps. In *2013 European Conference on Mobile Robots*, pages 178–185.

Forster, C., Pizzoli, M., and Scaramuzza, D. (2014). Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22.

Fox, D., Burgard, W., and Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33.

Garcia-Garcia, A., Orts-Escolano, S., Oprea, S., Villena-Martinez, V., and Garcia-Rodriguez, J. (2017). A review on deep learning techniques applied to semantic segmentation. *arXiv preprint arXiv:1704.06857*.

Garrido, S., Moreno, L., Abderrahim, M., and Martin, F. (2006). Path planning for mobile robot navigation using voronoi diagram and fast marching. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2376–2381.

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., and Medina-Carnicer, R. (2016). Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51:481 – 491.

Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. (2013). Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27:2672–2680.

Gordon, D., Kembhavi, A., Rastegari, M., Redmon, J., Fox, D., and Farhadi, A. (2018). Iqa: Visual question answering in interactive environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4089–4098.

Greene, W. N. and Roy, N. (2017). Flame: Fast lightweight mesh estimation using variational smoothing on delaunay graphs. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4696–4704. IEEE.

Grinvald, M., Furrer, F., Novkovic, T., Chung, J. J., Cadena, C., Siegwart, R., and Nieto, J. (2019). Volumetric instance-aware semantic mapping and 3d object discovery. *IEEE Robotics and Automation Letters*, 4(3):3037–3044.

Grisetti, G., Kummerle, R., Stachniss, C., and Burgard, W. (2010). A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43.

Grisetti, G., Stachniss, C., and Burgard, W. (2009). Nonlinear constraint network optimization for efficient map learning. *IEEE Transactions on Intelligent Transportation Systems*, 10(3):428–439.

Handa, A., Whelan, T., McDonald, J., and Davison, A. J. (2014). A benchmark for rgb-d visual odometry, 3d reconstruction and slam. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1524–1531.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Herbst, E., Henry, P., and Fox, D. (2014). Toward online 3-d object segmentation and mapping. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3193–3200.

Hermans, A., Floros, G., and Leibe, B. (2014). Dense 3d semantic mapping of indoor scenes from rgb-d images. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2631–2638.

Hess, W., Kohler, D., Rapp, H., and Andor, D. (2016). Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hoy, M., Matveev, A. S., and Savkin, A. V. (2015). Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey. *Robotica*, 33(3):463–497.

Jain, V., Magalhaes, G., Ku, A., Vaswani, A., Ie, E., and Baldridge, J. (2019). Stay on the path: Instruction fidelity in vision-and-language navigation. *arXiv preprint arXiv:1905.12255*.

Julier, S. J. and Uhlmann, J. K. (1997). New extension of the Kalman filter to nonlinear systems. In Kadar, I., editor, *Signal Processing, Sensor Fusion, and Target Recognition VI*, volume 3068, pages 182 – 193. International Society for Optics and Photonics, SPIE.

Kadian, A., Truong, J., Gokaslan, A., Clegg, A., Wijmans, E., Lee, S., Savva, M., Chernova, S., and Batra, D. (2020). Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters*, 5(4):6670–6677.

Kähler, O., Prisacariu, V. A., Ren, C. Y., Sun, X., Torr, P., and Murray, D. (2015). Very high frame rate volumetric integration of depth images on mobile devices. *IEEE transactions on visualization and computer graphics*, 21(11):1241–1250.

Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small ar workspaces. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–10. IEEE Computer Society.

Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE.

Koenig, S. and Likhachev, M. (2005). Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3):354–363.

Krähenbühl, P. and Koltun, V. (2011). Efficient inference in fully connected crfs with gaussian edge potentials. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 109–117, Red Hook, NY, USA. Curran Associates Inc.

Krinski, B. A., Ruiz, D. V., Machado, G. Z., and Todt, E. (2019). Masking salient object detection, a mask region-based convolutional neural network analysis for segmentation of salient objects. In *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*, pages 55–60.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NIPS*.

Kähler, O., Adrian Prisacariu, V., Yuheng Ren, C., Sun, X., Torr, P., and Murray, D. (2015). Very high frame rate volumetric integration of depth images on mobile devices. *IEEE Transactions on Visualization and Computer Graphics*, 21(11):1241–1250.

Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613.

Labbé, M. and Michaud, F. (2019). Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446.

Laroca, R., Araujo, A. B., Zanlorensi, L. A., de Almeida, E. C., and Menotti, D. (2020). Towards image-based automatic meter reading in unconstrained scenarios: A robust and efficient approach. *arXiv preprint*, arXiv:2009.10181:1–14.

Laroca, R., Barroso, V., Diniz, M. A., Gonçalves, G. R., Schwartz, W. R., and Menotti, D. (2019). Convolutional neural networks for automatic meter reading. *Journal of Electronic Imaging*, 28(1):013023.

Laroca, R., Severo, E., Zanlorensi, L. A., Oliveira, L. S., Gonçalves, G. R., Schwartz, W. R., and Menotti, D. (2018). A robust real-time automatic license plate recognition based on the YOLO detector. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–10.

Laroca, R., Zanlorensi, L. A., Gonçalves, G. R., Todt, E., Schwartz, W. R., and Menotti, D. (2021). An efficient and layout-independent automatic license plate recognition system based on the YOLO detector. *IET Intelligent Transport Systems*, pages 1–21.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., and Kang, J. (2019). BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240.

Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., and Furgale, P. (2015). Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334.

Li, S., Li, W., Cook, C., Zhu, C., and Gao, Y. (2018). Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5457–5466.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

Lo, K., Wang, L. L., Neumann, M., Kinney, R., and Weld, D. (2020). S2ORC: The semantic scholar open research corpus. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4969–4983, Online. Association for Computational Linguistics.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.

Lucio, D. R., Laroca, R., Zanlorensi, L. A., Moreira, G., and Menotti, D. (2019). Simultaneous iris and periocular region detection using coarse annotations. In *Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 178–185.

Lumelsky, V. J. and Shkel, A. M. (1995). Incorporating body dynamics into the sensor-based motion planning paradigm. the maximum turn strategy. In *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, volume 2, pages 1637–1642 vol.2.

Magrin, C. E., Brito, R. C., and Todt, E. (2019). A systematic mapping study on multi-sensor fusion in wheeled mobile robot self-localization. In *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*, pages 132–137.

Martinović, A., Knopp, J., Riemenschneider, H., and Van Gool, L. (2015). 3d all the way: Semantic segmentation of urban scenes from start to end in 3d. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4456–4465.

McCormac, J., Handa, A., Davison, A., and Leutenegger, S. (2017). Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In *2017 IEEE International Conference on Robotics and automation (ICRA)*, pages 4628–4635. IEEE.

Meng, T., Jing, X., Yan, Z., and Pedrycz, W. (2020). A survey on machine learning for data fusion. *Information Fusion*, 57:115 – 129.

Milioto, A. and Stachniss, C. (2019). Bonnet: An open-source training and deployment framework for semantic segmentation in robotics using cnns. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7094–7100.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

Mousavian, A., Toshev, A., Fišer, M., Košecká, J., Wahid, A., and Davidson, J. (2019). Visual representations for semantic target driven navigation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8846–8852.

Munoz-Salinas, R. (2012). Aruco: a minimal library for augmented reality applications based on opencv. *Universidad de Córdoba*.

Mur-Artal, R., Montiel, J. M. M., and Tardos, J. D. (2015). Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163.

Mur-Artal, R. and Tardós, J. D. (2017). ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262.

Murali, A., Chen, T., Alwala, K. V., Gandhi, D., Pinto, L., Gupta, S., and Gupta, A. (2019). Pyrobot: An open-source robotics framework for research and benchmarking. *arXiv preprint arXiv:1906.08236*.

Nekrasov, V., Dharmasiri, T., Spek, A., Drummond, T., Shen, C., and Reid, I. (2019). Real-time joint semantic segmentation and depth estimation using asymmetric annotations. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7101–7107.

Newcombe, R. A., Fox, D., and Seitz, S. M. (2015). Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 343–352.

Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohi, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011). Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136.

Nguyen, D. T., Hua, B.-S., Yu, L.-F., and Yeung, S.-K. (2018). A robust 3d-2d interactive tool for scene segmentation and annotation. *IEEE Transactions on Visualization and Computer Graphics*, 24(12):3005–3018.

Nießner, M., Zollhöfer, M., Izadi, S., and Stamminger, M. (2013). Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)*, 32(6):169.

Noh, H., Hong, S., and Han, B. (2015). Learning deconvolution network for semantic segmentation. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, page 1520–1528, USA. IEEE Computer Society.

NVIDIA (2015). Jetson tx2 module. `https://developer.nvidia.com/EMBEDDED/jetson-tx2`. Accessed: 2020-11-01.

Olah, C. (2015). Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

Open Robotics (2018). Robotic operating system. `https://www.ros.org`. Accessed: 2019-05-23.

Open Robotics (2019). Urdf. `http://wiki.ros.org/urdf`. Accessed: 2019-12-1.

OpenCV (2015). Detection of aruco markers. `https://docs.opencv.org/3.4/d5/dae/tutorial_aruco_detection.html`. Accessed: 2020-12-02.

Patel, N., Khorrami, F., Krishnamurthy, P., and Tzes, A. (2019). Tightly coupled semantic rgb-d inertial odometry for accurate long-term localization and mapping. In *International Conference on Advanced Robotics (ICAR)*, pages 1–6.

Pawlak, Z. (1992). *Rough Sets: Theoretical Aspects of Reasoning about Data*. Kluwer Academic Publishers, USA.

Peirce, J. W. (2015). Understanding mid-level representations in visual processing. *Journal of Vision*, 15(7):5–5.

Perumal, L. (2011). Quaternion and its application in rotation using sets of regions. *International Journal of Engineering and Technology Innovation*, 1(1):35.

Redmon, J. and Farhadi, A. (2017). Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271.

Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Reis, D. H. D., Welfer, D., Cuadros, M. A. D. S. L., and Gamarra, D. F. T. (2019). Mobile robot navigation using an object recognition software with rgbd images and the yolo algorithm. *Applied Artificial Intelligence*, 33(14):1290–1305.

Romero-Ramirez, F. J., Muñoz-Salinas, R., and Medina-Carnicer, R. (2018). Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76:38 – 47.

Rösmann, C., Feiten, W., Wösch, T., Hoffmann, F., and Bertram, T. (2012). Trajectory modification considering dynamic constraints of autonomous robots. In *ROBOTIK 2012; 7th German Conference on Robotics*, pages 1–6. VDE.

Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635.

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). ORB: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571.

Ruiz, D. V., Krinski, B. A., and Todt, E. (2019). ANDA: A novel data augmentation technique applied to salient object detection. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 487–492.

Ruiz, D. V., Krinski, B. A., and Todt, E. (2020). IDA: Improved data augmentation applied to salient object detection. In *2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 210–217.

Ruiz, D. V., Salomon, G., and Todt, E. (2020). Can giraffes become birds? an evaluation of image-to-image translation for data generation. *Anais do Computer on the Beach*, 11(1):176–182. Available: `https://siaiap32.univali.br/seer/index.php/acotb/article/view/16766`.

Saeedi, S., Bodin, B., Wagstaff, H., Nisbet, A., Nardi, L., Mawer, J., Melot, N., Palomar, O., Vespa, E., Spink, T., et al. (2018). Navigating the landscape for real-time localization and mapping for robotics and virtual and augmented reality. *Proceedings of the IEEE*, 106(11):2020–2039.

Salomon, G., Britto, A., Vareto, R. H., Schwartz, W. R., and Menotti, D. (2020a). Open-set face recognition for small galleries using siamese networks. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 161–166.

Salomon, G., Laroca, R., and Menotti, D. (2020b). Deep learning for image-based automatic dial meter reading: Dataset and baselines. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.

Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., and Batra, D. (2019). Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Sax, A., Zhang, J. O., Emi, B., Zamir, A., Savarese, S., Guibas, L., and Malik, J. (2020). Learning to navigate using mid-level visual priors. In Kaelbling, L. P., Kragic, D., and Sugiura, K., editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 791–812. PMLR.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Seder, M., Macek, K., and Petrovic, I. (2005). An integrated approach to real-time mobile robot control in partially known indoor environments. In *31st Annual Conference of IEEE Industrial Electronics Society, 2005. IECON 2005.*, pages 6 pp.–.

Shafer, G. (1976). *A mathematical theory of evidence*, volume 42. Princeton university press.

Shen, W. B., Xu, D., Zhu, Y., Guibas, L. J., Fei-Fei, L., and Savarese, S. (2019). Situational fusion of visual representation for visual navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Shenzhen EAI Technology Co. (2017). Ydlidar x4. `https://www.ydlidar.com/products/view/5.html`. Accessed: 2020-11-01.

Sheridan, F. K. J. (1991). A survey of techniques for inference under uncertainty. *Artificial Intelligence Review*, 5(1):89–119.

Silberman, N., Hoiem, D., Kohli, P., and Fergus, R. (2012). Indoor segmentation and support inference from rgbd images. In Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., and Schmid, C., editors, *Computer Vision – ECCV 2012*, pages 746–760, Berlin, Heidelberg. Springer Berlin Heidelberg.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Song, S., Yu, F., Zeng, A., Chang, A. X., Savva, M., and Funkhouser, T. (2017). Semantic scene completion from a single depth image. *Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition*.

Srivastava, N., Mansimov, E., and Salakhudinov, R. (2015). Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852.

Stanford VL and Robotics at Google (2020). Sim2Real Challenge with Gibson. `https://eval.ai/web/challenges/challenge-page/537/`. Accessed: 2020-10-30.

Straub, J., Whelan, T., Ma, L., Chen, Y., Wijmans, E., Green, S., Engel, J. J., Mur-Artal, R., Ren, C., Verma, S., Clarkson, A., Yan, M., Budge, B., Yan, Y., Pan, X., Yon, J., Zou, Y., Leon, K., Carter, N., Briales, J., Gillingham, T., Mueggler, E., Pesqueira, L., Savva, M., Batra, D., Strasdat, H. M., Nardi, R. D., Goesele, M., Lovegrove, S., and Newcombe, R. (2019). The Replica dataset: A digital replica of indoor spaces. *arXiv:1906.05797*.

Sun, W., Venkatraman, A., Gordon, G. J., Boots, B., and Bagnell, J. A. (2017). Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *International Conference on Machine Learning*, pages 3309–3318. PMLR.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*.

Takaya, K., Asai, T., Kroumov, V., and Smarandache, F. (2016). Simulation environment for mobile robots testing using ros and gazebo. In *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 96–101. IEEE.

Tateno, K., Tombari, F., and Navab, N. (2015). Real-time and scalable incremental segmentation on dense slam. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4465–4472.

Thoma, M. (2016). A survey of semantic segmentation. *arXiv preprint arXiv:1602.06541*.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic robotics*. MIT press.

Tripathi, A. (2017). Practical machine learning cookbook.

Valentin, J., Vineet, V., Cheng, M.-M., Kim, D., Shotton, J., Kohli, P., Nieundefinedner, M., Criminisi, A., Izadi, S., and Torr, P. (2015). Semanticpaint: Interactive 3d labeling and learning at your fingertips. *ACM Trans. Graph.*, 34(5).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Vega, J. and Cañas, J. M. (2019). Pybokids: An innovative python-based educational framework using real and simulated arduino robots. *Electronics*, 8(8).

Vineet, V., Miksik, O., Lidegaard, M., Nießner, M., Golodetz, S., Prisacariu, V. A., Kähler, O., Murray, D. W., Izadi, S., Pérez, P., et al. (2015). Incremental dense semantic stereo fusion for large-scale semantic scene reconstruction. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 75–82. IEEE.

Wasenmüller, O. and Stricker, D. (2016). Comparison of kinect v1 and v2 depth images in terms of accuracy and precision.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Watt, J., Borhani, R., and Katsaggelos, A. (2020). *Machine learning refined: foundations, algorithms, and applications*. Cambridge University Press.

Weihs, L., Salvador, J., Kotar, K., Jain, U., Zeng, K.-H., Mottaghi, R., and Kembhavi, A. (2020). Allenact: A framework for embodied ai research. *arXiv*.

Welch, G. and Bishop, G. (1995). An introduction to the kalman filter. Technical report, University of North Carolina at Chapel Hill, USA.

Whelan, T., Leutenegger, S., Salas-Moreno, R. F., Glocker, B., and Davison, A. J. (2015). Elasticfusion: Dense slam without a pose graph. In *Robotics: Science and Systems*.

Whelan, T., McDonald, J., Kaess, M., Fallon, M., Johannsson, H., and Leonard, J. J. (2012). Kintinuous: Spatially extended kinectfusion. In *Proceedings of RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*.

Wijmans, E., Kadian, A., Morcos, A., Lee, S., Essa, I., Parikh, D., Savva, M., and Batra, D. (2019). Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. *arXiv preprint arXiv:1911.00357*.

Wise, M. and Foote, T. (2010). Turtlebot. `https://www.turtlebot.com/`. Accessed: 2020-10-29.

Wu, B., Zhou, X., Zhao, S., Yue, X., and Keutzer, K. (2019). Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4376–4382.

Xia, F., R. Zamir, A., He, Z.-Y., Sax, A., Malik, J., and Savarese, S. (2018). Gibson env: real-world perception for embodied agents. In *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*. IEEE.

Xia, F., Shen, W. B., Li, C., Kasimbeg, P., Tchapmi, M. E., Toshev, A., Martín-Martín, R., and Savarese, S. (2020). Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments. *IEEE Robotics and Automation Letters*, 5(2):713–720.

Yang, W., Wang, X., Farhadi, A., Gupta, A., and Mottaghi, R. (2018). Visual semantic navigation using scene priors. *arXiv preprint arXiv:1810.06543*.

Zadeh, L. (1965). Fuzzy sets. *Information and Control*, 8(3):338 – 353.

Zadeh, L. (1978). Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1(1):3 – 28.

Zamir, A. R., Sax, A., Shen, W. B., Guibas, L. J., Malik, J., and Savarese, S. (2018). Taskonomy: Disentangling task transfer learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

Zanlorensi, L. A., Luz, E., Laroca, R., Britto Jr., A. S., Oliveira, L. S., and Menotti, D. (2018). The impact of preprocessing on deep representations for iris recognition on unconstrained environments. In *Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 289–296.

Zhang, J., Tai, L., Boedecker, J., Burgard, W., and Liu, M. (2017). Neural slam: Learning to explore with external memory. *arXiv preprint arXiv:1706.09520*.

Zhang, J., Zhao, Y., Saleh, M., and Liu, P. (2020). PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 11328–11339, Virtual. PMLR.

Zhao, W., Queralta, J. P., and Westerlund, T. (2020). Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744. IEEE.

Zhuang, Z., Cutkosky, A., and Orabona, F. (2019). Surrogate losses for online learning of stepsizes in stochastic non-convex optimization. In *ICML*.