Master’s Thesis

Mental Simulation for Autonomous Learning and Planning Using an Ontology-Based Modeling Framework

Yuri Goncalves Rocha

Department of Electrical and Computer Engineering

The Graduate School

Sungkyunkwan University

Master’s Thesis

Mental Simulation for Autonomous Learning and Planning Using an Ontology-Based Modeling Framework

Yuri Goncalves Rocha

Department of Electrical and Computer Engineering

The Graduate School

Sungkyunkwan University

The author of this thesis is a student who is sponsored by the Global Korea Scholarship program.

Mental Simulation for Autonomous Learning and Planning Using an Ontology-Based Modeling Framework

Yuri Goncalves Rocha

A Master's Thesis Submitted to the Department of Electrical and Computer Engineering and the Graduate School of Sungkyunkwan University in partial fulfillment of the requirements for the degree of Master of Science in Engineering

April 2020

Supervised by

Tae-Yong Kuc

Major Advisor

This certifies that the Master's Thesis
of Yuri Goncalves Rocha is approved.

Committee Chairman:

Committee Member:

Major Advisor:

The Graduate School
Sungkyunkwan University
June 2020

Table of Contents

List of Tables

List of Figures

Abstract

Mental Simulation for Autonomous Learning and Planning Using an Ontology-Based Modeling Framework

Cognitive science findings have shown that humans have an outstanding ability to create simulated worlds on their minds. Those simulated environments can be used to prospect into the future, re-act memories from the past, simulate different outcomes for a known situation, and, ultimately, learn and test planned actions without the need to act on the physical world. Such cognitive skills could similarly enhance current robotic systems, allowing them to predict the outcomes of their planned actions before executing them. It also provides a platform that can be run autonomously to perform reinforcement learning algorithms, gradually improving the robot's skills. Hence, the environment and the robot itself should be modeled using an information-rich descriptive framework.

In this work, we used and expanded the Triplet Ontological Semantic Model to represent the robot and its surroundings. This modeled data was used to autonomously generate a complete simulated world, without the need for human aid. This simulated world was used to train a reinforcement learning policy for autonomous navigation. We also proposed a hybrid navigation method that combines classical planners with the trained policy, improving its long-term navigation capabilities.

Experiment results show that the mental simulation can be used to both train

the policy and verify a plan feasibility. The proposed method was able to perform long-distance navigation tasks using only sparse range data and taking 7ms to execute a control command.

Keywords: Mental Simulation, Autonomous Navigation, Reinforcement Learning, Path Planning, Probabilistic Roadmaps

1.Introduction

Early studies on cognitive science [1] proposed that mental simulation is one of the fundamental cognitive skills. Humans (and perhaps other animals as well [2]) can predict and anticipate outcomes by recalling past experiences and re-simulating them in their minds. Such ability is considered one of the foundations on episodic memory [3] and paramount for task planning during navigation [4]. It can be subsumed as building a simulated world inside one's mind, which is capable of working on its own (with varying levels of complexity) and using this world to review, infer and predict outcomes, which finally shapes one's behavior on the real world.

Mental simulation theory starts from the presumption that behaviors can be simulated, perception systems can also be simulated, and outcomes can be anticipated by combining simulated behaviors and perception [5]. Early researches on cognitive science and neuropsychology showed that such a mechanism is also used to predict one's counterpart's thoughts and behaviors [6].

During the early stages of artificial intelligence, Alan Turing proposed that, to think and speak like a human, a robot might need a human-like body, capable of mimicking sensorial and motor capabilities, and that the development of its cognitive skills should be as simple as teaching a child [7]. Some more recent works suggested that cognitive capabilities, such as mental simulation, should not only be integrated into learning and planning algorithms, but also be their central architectural layer [8]. To do so, it is paramount that the robot can semantically understand its surrounding environment. Thus, adding another

layer to the robot knowledge, allowing a better understanding of the relations and intrinsic concepts of the world, which are naturally inferred by humans. Semantic data has been explored on several fields of robotics and artificial intelligence, such as navigation [9], [10], knowledge representation [11]–[14] and computer vision [15]–[17], however just a small fraction of them use it to perform mental simulations [18]–[20].

Mental simulation is important not only for remembering past actions and planning the future ones but also "learning from imagination" [21]. Initially, humans learn by experience. Indeed, our first learning experiences as a toddler come from interacting with the environment and with ourselves and observing the outcome. After acquiring enough experience, we can simulate such experiences in our mind, and learn by thinking about those experiences, reducing the need to always have a propitious environment, thus accelerating the learning process. Such a skill can and should also be incorporated into robotic systems. In robotics, one of the fields on Artificial Intelligence (AI) is called Reinforcement Learning (RL). RL was based on the way humans learn, exploring the environment and receiving rewards or penalties depending on how well the robot executed the given task. A sub-division of RL, known as Deep Reinforcement Learning(deep-RL), combines the concepts of RL with deep neural networks and has seen a rise in popularity on several robot applications, such as manipulation [22], [23] and autonomous navigation [24]–[26].

This work uses semantic modeling to represent the environment and robots themselves, enabling them to automatically generate a complete mental simulation environment without human assistance. This simulated environment was used to validate planned actions before executing them in the real world. It was also used when the robot is idle (e.g., charging, or in-between missions)

to run RL algorithms, taking robots one step closer to full autonomy. Our major contributions are:

- An extension of the Triplet Ontological Semantic Modeling framework to allow representing a robot;
- A mental simulation generating algorithm that creates a simulated environment automatically using only the information on the robot's database;
- A hybrid navigation method that combines a sampling-based planner with a reinforcement learning navigation policy, enabling real-time long-range navigation.

The rest of the paper is organized as follows. Section 2 contains an overview of mental simulation, reinforcement learning, and planning solutions. Then, the environment and robot modeling and the architecture used to generate an automatic mental simulation are presented in Section 3. Furthermore, Section 4 describes two different applications for the mental simulation: training a reinforcement learning policy and validating a plan. Section 5 presents our simulation results, where we compare them with the current state-of-the-art algorithms. Finally, Section 6 concludes the paper and presents research directions for future works.

2.Related Work

2.1 Knowledge and Semantic Modeling

A large portion of our current surrounding environments was designed by humans, for humans. Throughout our history, we shaped our world according to our needs and convenience. Robots are now starting to work on those dynamic environments and, to successfully act on our world, they should be able to understand it in a similar fashion as we do. In other words, robots need knowledge about their environment, task, and its design and capabilities. Such knowledge should be structured in a way machines can understand effectively while representing the intrinsic relationship between different individuals on a large scale.

Several works proposed ways of incorporating knowledge into computers. Some gathered a large amount of encyclopedic knowledge into their databases [12], [13], [27], [28]. Also known as general knowledge graph databases, they tried to subsume a vast spectrum of domain-agnostic knowledge. CYC [12] tried to formalize commonsense knowledge, mostly handcrafted by knowledge experts. As of today, it is composed of more than 10 thousand predicates and more than 25 million assertions, mostly composed of proprietary data owned by Cycorp, the creator of CYC. The Suggested Upper Merged Ontology (SUMO) [13] was created by merging already publicly available knowledge bases into a single standardized structure. At the time of its publication, SUMO had 654 terms and 2351 assertions, distributed as an open-source knowledge base. WordNet [27] is a database of English words, mostly nouns, verbs, and

adjectives, connected by a small set of semantic relations. Its second version subsumed around 200000 words. ConceptNet [28] was automatically generated from English sentences of the Open Mind Common Sense (OMCS) corpus. It combined simple semi-structured English sentences connecting them using twenty specified semantic relations. The whole network had more than 300000 connected nodes.

The main drawback of general knowledge graphs is that it tries to englobe a large variety of data from different domains, which is good for textual tasks and contextual commonsense reasoning but lacks accuracy and granularity when used on specific domains, such as robotics. The robotics domain needs more detailed knowledge about the physical world, its occupants, objects, tasks and behaviors, and the relationship between them. The Open Mind Indoor Common Sense (OMICS) [14] knowledge base is an object-centric framework that contains knowledge needed for robots to execute several tasks in an indoor home or office environment. Its structure was based on the OMCS sentence templates. The RoboEarth [11] project tried to create the World Wide Web for robots, a distributed cloud-based platform where robots would be able to autonomously share and reuse knowledge. The cloud database would store sensorial and modeled data from objects (such as CAD models, point clouds, and images) and human-readable action receipts. Each robot would have a skill abstraction layer that would receive those action sequences and convert to actions that can be executed by the robot hardware. OpenEASE [19] went on a similar path by also offering a cloud solution for knowledge storage, but it also allowed full manipulation task episodes to be stored, which could be later queried, visualized, and analyzed. This work proposes a new semantic knowledge framework, which is able to represent not only the environment but

also the robot itself. It focuses on representing knowledge in a general way so that it can be used in a large span of applications, but also with enough information to allow robots to simulate the environment based only on their internal knowledge.

2.2 Mental Simulation on Humans

The mental simulation paradigm has shown constant interest from cognitive science and neuropsychology fields [1], [6], [29], [30]. One of the pioneering works on this field was done by Kahneman and Tversky [1], by suggesting there were some situations when finding an answer for a problem, humans would run a mechanism which resembled a simulation. They also affirmed that such simulations would yield various outcomes, giving us a bias of which result was more likely to happen. They affirmed that these simulations could be used for predicting future events, asserting a probability of its occurrence and reasoning about causality (i.e., whether event A led to event B). This causality principle was then shown to affect how we do counterfactual reasoning, easily described as "what if" scenarios, which we use to imagine different outcomes to a series of events that have already happened. In an earlier study [31], Tversky and Kahneman also showed that the ease of the subject to determine an outcome of an event was greatly influenced by the availability of similar events in their memory. Gordon [6] showed the usage of mental simulation when predicting someone else's behavior, intents, and beliefs, a concept also known as 'Theory of Mind' [29].

Later works were more focused on which areas of the brain are activated when performing mental simulations [30] or on which specific applications they

are used. Kappes et al. [32] suggested that we use it as a substitute for physical experiences due to its easier availability. Lombrozo [21] showed that those virtual experiences could be used as a learning tool. Hegarty [33] and Bates et al. [34] demonstrated that mental simulation is also used when dealing with complex physics concepts, such as mechanical reasoning (e.g., predicting which way a gear would turn when connected with several other gears) and predicting liquid dynamics (e.g., reasoning that water is more likely to spill when being poured than a high viscosity liquid). Burgess [4] showed that mental simulation and episodic memory are paramount when performing navigation. Finally, Bergen et al. [35] showed the influence of grammatical structures when mentally simulating a sequence of actions, while Speed et al. [36] argued that odor has little to no influence on mental simulation, whereas visual and auditory information plays a significantly bigger role.

2.3 Mental Simulation on Robots

Despite being thoroughly studied by cognitive science researchers, the mental simulation concept is yet to be fully explored in Artificial Intelligence and Robotics fields. Indeed, it only started to be applied to computational systems a few decades ago. The first works on the field focused on the "putting yourself on someone else's shoes" application, where an agent would simulate itself on another actor's perceived state in order to infer about its feelings and intentions. Laird [37] created a Quake bot that would try to predict its opponent's next action by simulating itself on the opponent's pose and then running its own algorithm. Leonardo [4] was created to assist a human operator on its task. To do so, Leonardo would simulate itself on the human's perceived

state and guess its intentions by using his own internal module. Leonardo would then try to assist with the predicted action. Buchsbaum et al. [38] developed a simulated character that would try to infer its counterpart's action and imitate it based on its own internal kinematic model.

Cassimatis et al. [39] proposed a planning architecture that combined several different reasoning and inference algorithms by using a common knowledge representation so a robot would be able to perform logical simulations to track another agent. ORPHEUS [40] is a system created to aid a hunter to catch prey. It used mental simulation to run several different plans and select the most suitable one to be applied to the real situation. The mental simulation was automatically generated based on the data extracted from the hunter perception. Buche et al. [41] extended this work by adding multiple agents to the environment and allowing online adaptation of the planned path. They also used the same approach to let a virtual juggler predict the ball path while juggling either alone or collaborating with other agents, being them simulated or humans. Vicente et al. [42] used mental simulation combined with proprioceptive data to simulate on which pose a robot hand would appear on its image and then match this simulated result with the real camera image using Bayesian techniques. This combined data was used to do an online update to the robot's internal kinematic model. KnowRob 2.0 [20] is a knowledge processing framework that is able to perform logical reasoning using both a semantic knowledge database and mental simulation (there called "The mind's eye") capable of replaying the robot's past experiences to explore new outcomes. This work, however, focused mainly on manipulation tasks. Vanderelst et al. [43] developed an ethical robot that used mental simulation to validate its several planned behavioral alternatives according to Asimov's laws of robotics. The robot would

then choose the action which complied with those laws. Except for [20], [40], [41], most of the cited works focused on a specific application, where the simulation environment was tailored by domain experts. We reckon such cognitive skills should be an intrinsic skill for robotic systems, so they can achieve full autonomy. This work proposes a mental simulation automatic generation algorithm, using only the robot's internal knowledge without the need for human aid. This simulation was then used to perform reinforcement learning and to validate motion plans.

2.4 Reinforcement Learning

New evidence from neuroscience and psychology studies suggest that animals might replay navigation sequences when sleeping, discovering new routes, and also consolidating the memorized ones [44], [45]. We envision that the next generation of robots should be able to use their idle time to constantly learn new skills. One of the ways of achieving such autonomous learning capabilities is by using Reinforcement Learning approaches. Reinforcement Learning (RL) algorithms try to map a set of sensory inputs to actions by formulating this task as a Partially Observable Markov Decision Process (POMDP) [46]. Deep RL uses deep neural networks (DNN) to approximate such mapping.

Mnih et al. [47] created a DNN named Deep-Q Network (DQN) to approximate the Q-value estimation for a value-based RL approach. Shah et al. [25] developed a novel end-to-end architecture, which learned how to interpret natural language instructions and a semantic segmented RGB-D image to navigate in places without any map or current position information. They used

a combination of Convolution Neural Networks (CNN), a bi-directional Gated Recurrent Unit (GRU) weighted by an attention mechanism, and a DQN, which learned a policy that converts the input data into discrete move left front and right control actions. Due to only being able to output discrete actions, DQN is not suitable for several robotic applications, which are inherently continuous. Lillicrap et al. [48] proposed an actor-critic RL approach called Deep Deterministic Policy Gradients (DDPG), which used separated DNNs for action generation and Q-value approximation. This architecture was able to perform continuous actions. Tai et al. [24] proposed an end-to-end architecture, which takes as input sparse 10-dimensional laser range readings and a relative goal coordinate, and converts it to a continuous velocity output. They extended the DDPG algorithm to an asynchronous version, which was shown to collect four times more samples than the original synchronous version. DDPG, however, suffered from value over-estimation and unstable learning. Fujimoto et al. [49] proposed the Twin Delayed Deep Deterministic Policy Gradients (TD3), which learned two critic networks but only used the smallest value between them when predicting future rewards. They also delayed the actor training step to improve overall learning stability. Kahn et al. [26] developed a novel model, called Generalized Computation Graphs (GCG) for reinforcement learning, to generate a hybrid model-free, model-based algorithms for robot navigation. The key aspect of this GCG was the use of a multiplicative integration Long Short Term Memory (LSTM), which would encode the next H actions and its predicted rewards. By carefully choosing the output values of the LTSM (i.e., the expected reward data shape), the GCG could either behave as a value-based model-free algorithm or as a model-based algorithm. Long et al. [50] applied RL paradigms to the multi-robot field. They used a centralized learning,

decentralized execution approach, where each robot would execute its next action based on individual readings, however, a single shared policy would be trained by the experience collected by every robot simultaneously.

2.5 Motion Planning

Motion planning can be defined as finding a valid motion path given a goal point, a sensorial input, and a list of constraints, such as not colliding with obstacles. Planners are usually compared based on their computational efficiency and scalability and on their ability to find an optimal solution in finite time [51]. Motion planners can be roughly divided as cell decomposition methods (CDM), potential field methods (PFM), and sampling-based methods (SBM).

CDMs divide the environment's free space into small cells and find a sequence of adjacent cells that connect the current position to the goal while avoiding occupied spaces. They suffer, however, from several issues such as limited granularity, combinatorial explosion, and generating infeasible solutions [52]. In PFMs, every goal is modeled as an attractive force, while obstacles are repulsive ones. To get to the goal, the robot follows the gradient direction generated by the combination of those fields. Nevertheless, they are susceptible to converge into local minima, which would trap the robot midway [52]. SBMs, on the other hand, have attracted considerable attention due to its scalability and probabilistic completeness, namely, if the number of samples converges to infinity, the probability of finding the optimal path converges to 1. Probabilistic roadmaps (PRM) and rapidly-exploring random trees (RRT), two of the most notable PRMs [53], sample points in a similar way, but differ largely on how

they connect those points. PRM [54] maintains several graph expansions simultaneously, having a good performance on high−dimensional spaces. RRT [55], on the other hand, rapidly explores a single graph, being more suitable to smaller environments. Despite being probabilistic complete algorithms, Karaman et al. [56] showed that most of the time, both algorithms returned non−optimal solutions. The authors then introduced the asymptotically optimal versions of each method, PRM* and RRT*, respectively. Nonetheless, PRM* and RRT* provided no theoretical guarantees of their optimality [57] and had a slower convergence compared to their original forms. Faust et al. [58] tried to solve the shortcomings of SBMs and RL algorithms by combining both into a hybrid approach. A similar method was used in this work.

3. Automatic Mental Simulation

In this section, we describe in detail the environment modeling and the mental simulation generation algorithm.

3.1 Triplet Ontologic Semantic Modeling

Researches on the cognitive sciences and neuroscience fields [4], [59] showed that the human brain has an outstanding ability to generate, maintain and update spatial maps of known environments, also called the brain "GPS." Humans rely heavily on relational information instead of precise measurements, allowing our brain to efficiently map even large environments. This scalability and data efficiency remains unparalleled when compared to the latest technologies. Robots still need to be fed with memory inefficient but high-resolution metric data to localize themselves and navigate through known environments. Several different map definitions have been used by the robotics community, such as shown in Figure 1. All those maps, however, can share some common data between them, wasting memory space and hindering the robot's long-term capabilities.

The Triplet Ontologic Semantic Modeling (TOSM) [60] was created to change the way data is stored and used by robots. TOSM can be divided into three main pillars, as shown in Figure 1. The explicit model is used to describe any data that can be perceived by the robot sensors, such as shape, color, pose, and size. Most of the data used by current robotic systems can be englobed on this model. The implicit model, on the other hand, contains any intrinsic

information that cant be obtained directly from sensors and should be inferred using the robot's current knowledge. It can range from physical properties such as mass and friction coefficients, to relational semantic data, that describes objects/places relative position. It can also store encyclopedic knowledge, such as "an automatic door opens if there is a subject in front of it." Finally, the symbolic model contains a human-like definition for the elements, namely symbols, names, descriptions, and identification numbers.

In order to store the TOSM-encoded data using a computer-readable format, both the robot and the environment data were stored into separated Ontology Web Language (OWL) files. OWL was chosen due to being a well-defined language vastly used by the community and with several tools and applications openly available. We used the Protégé framework [61] to manipulate and visualize OWL files.

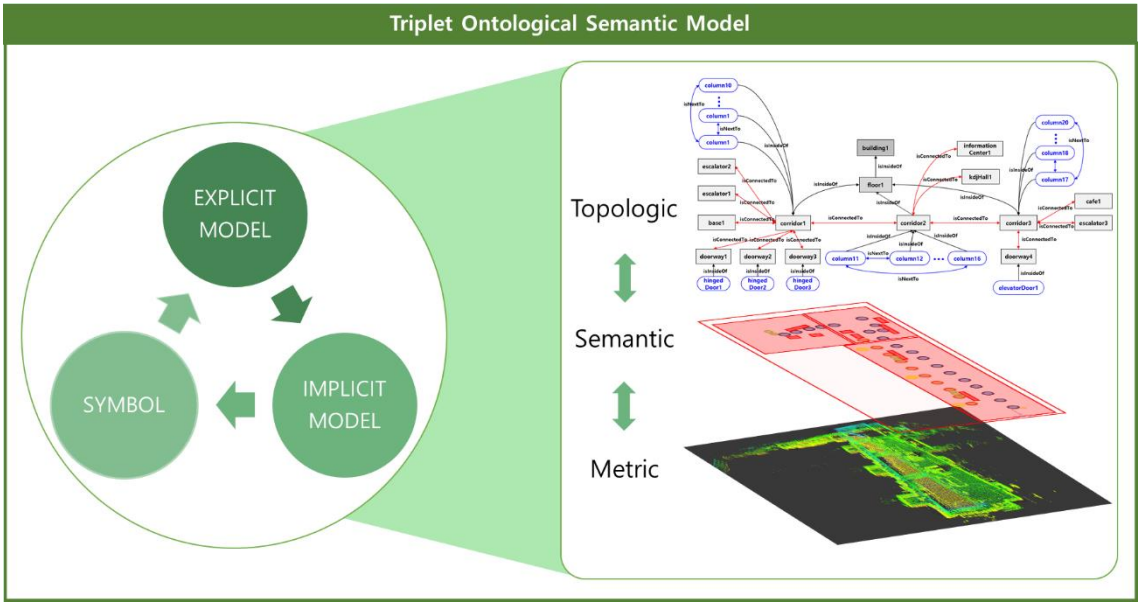

Figure 1 – TOSM description

3.2 Environment Description

The environment can be divided into places and objects. Places can be anything from a building to a single floor or even a small room. Their explicit model contains their boundary points, while the implicit model subsumes the relational data, i.e., which other places it is connected to and which places it is inside of. The implicit data also stores the *complexity* of the place, a number between 0 and 1, which represents the proportion of the area which is occupied.

Regarding objects, the explicit model contains the object's size, pose, color, and shape. The implicit model stores the object's mass and material, and the relational spatial data, such as " in front of," "next to," and "inside of." The symbolic data is the same for places and for objects, containing the place/object name and an identification number.

In this work, we extended the TOSM framework to also encode a complete robot definition.

3.3 Robot Description

In this work, we modeled the differential robot shown in Figure 2. The robot is equipped with a stereo camera and a laser range finder. The modeling proposed in the section, though, can be applied to several different types of robots.

A modular approach was used to describe robots. A robot can be divided into four main components: structural parts, joints, wheels, and sensors. A robot can contain none, one, or multiple instances of any of those parts. Each part was modeled using TOSM encoding (i.e. each part has an explicit, implicit, and

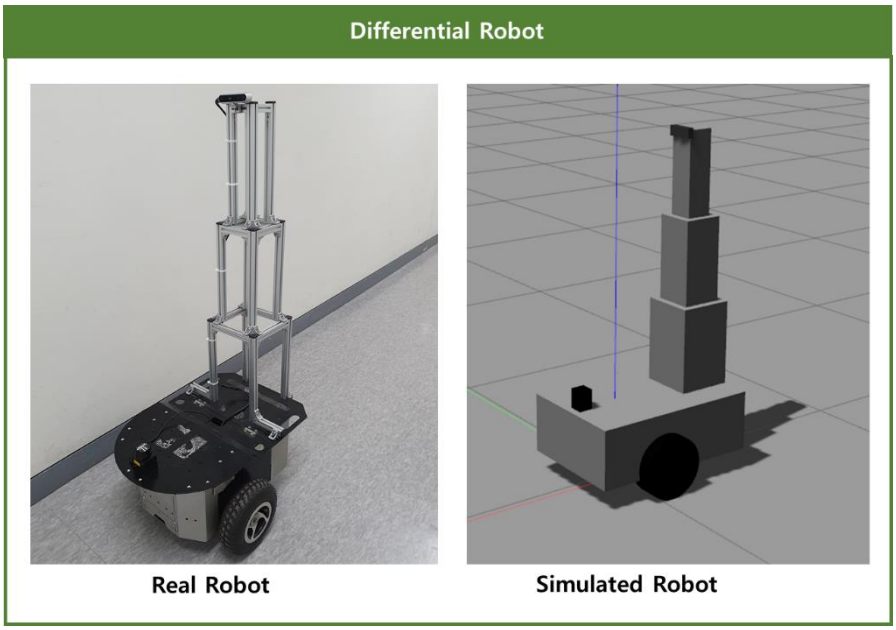

Figure 2 – Differential robot used in this work

symbolic model). The explicit data is virtually the same across all robot parts, while the symbol data contains the name and an identification number. The implicit model, however, is unique for each category. Structural parts need to encode their weight and material information. The wheel representation extends it to also store whether it is an active or a passive wheel. Joints, on the other hand, store which two parts they are connected to. Finally, sensors have different implicit data depending on their type. Cameras can be represented by their resolution, frames per second, field of view, and, in the case of RGB-D cameras, by their minimum and maximum ranges. A laser range finder is described by its range, view angle, number of samples, and resolution. The OWL datagraph of the aforementioned differential robot can be seen in Figure 3, and the simulated robot generated through this data is shown in Figure 2.

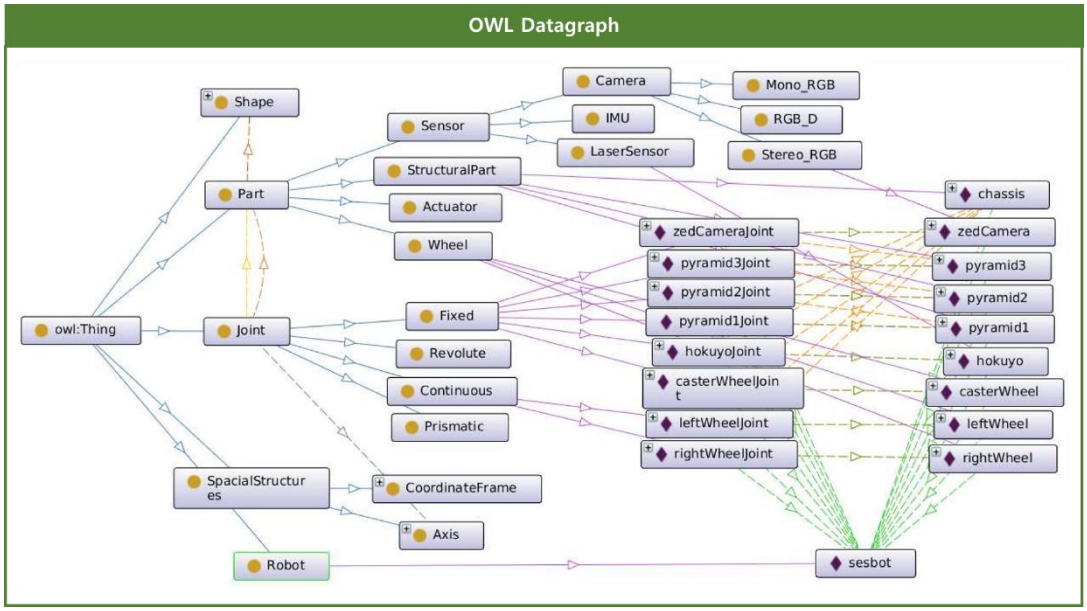

Figure 3 – Robot OWL datagraph

3.4 Generating the Mental Simulation

One of the key issues when implementing mental simulation algorithms is the need of domain experts to model the simulation according to the robot's real environment. This task becomes unfeasible for robots operating in large dynamic environments. In order to improve the robot's autonomous capabilities and to remove the need for domain experts, we developed a middle-ware, named simulation parser, capable of generating a simulation environment using only the data stored in the robot's database.

Whenever the robot receives a mission, it performs a query on the complete database, generating a subset containing only the environment data relevant to the current mission. This newly generated subset is called the *on-demand database*. Through this system, we can reduce the memory requirements on the

robot by storing the complete database in the cloud and providing to the robot only the data relevant to the current mission. The simulation parser then consumes the data in the on-demand database and generates the needed files to generate a corresponding simulated environment. The data-flow can be seen in Figure 4. Two different files are generated by the parser. The first one is a Simulation Description Format (SDF) file, which is consumed by the Gazebo Simulator to generate the simulated environment. The second file is a Universal Robot Description File (URDF), which is used by both the Gazebo Simulator and the Robot Operating System (ROS) to generate and control the simulated robot. Whenever the robot needs to use the mental simulation, both files are re-generated, assuring that the simulation will be constantly updated whenever the robot perceives a change in the real environment and updates its database.

To ensure the simulation similarity and consistency with the real world, we used a library containing 3D models for several common objects, such as doors, tables, and chairs. Nonetheless, when exploring new environments, the robot is

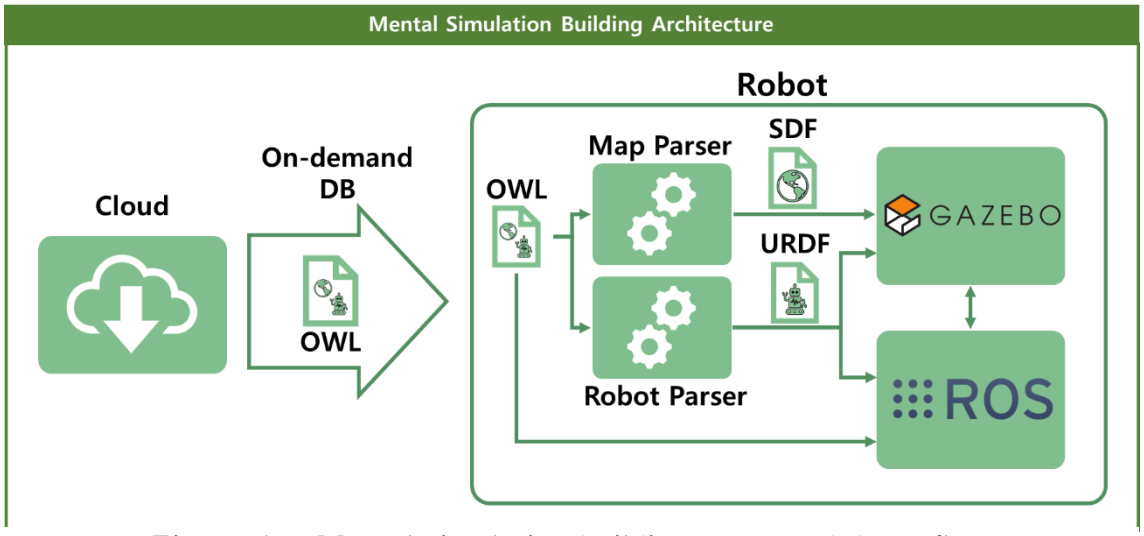

Figure 4 – Mental simulation building steps and data-flow

bound to encounter objects whose there is no corresponding 3D model. Hence, to improve the algorithm robustness, whenever the robot needs to simulate an object of which it has no 3D model available, a placeholder is generated instead, using the color and shape information stored in the on-demand database. A comparison between an object of which there is a 3D mesh available and an object of which a placeholder was generated be seen in Figure 5. The 3D model database contains a mesh for a steel door, which was used in the simulation. On the other hand, there is no available 3D model for a beverage vending machine, hence the mental simulation building algorithm automatically generates a red block placeholder, based on the color and shape perceived by the robot.

Beetz et al. [20] used a semantic modeling framework combined with mental

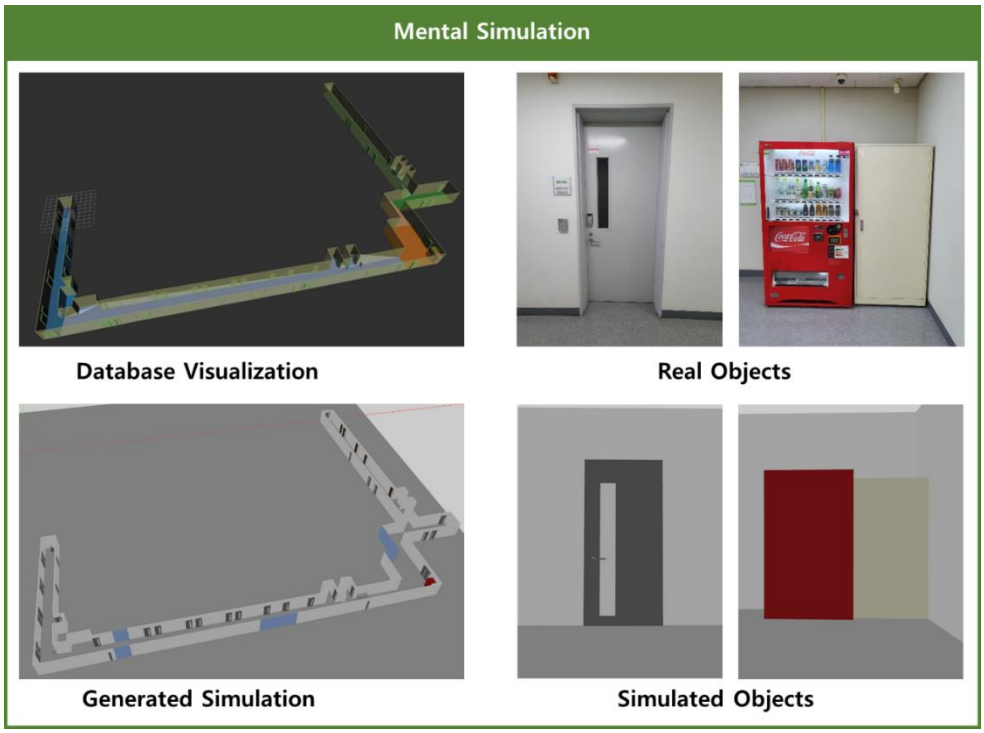

Figure 5 – Environment generated by the mental simulation building algorithm

simulation to validate planned actions. However, their focus was on manipulation tasks, building only small simulations with dynamic objects. The number of available objects was limited to household items, of which there were 3D meshes available. Our proposed approach allows the robot to simulate previously unknown objects by combining the perceived shape, size, and color information into a placeholder object. Hence, allowing us to perform large scale simulations on unknown environments, while Beetz et al. approach was limited to small indoor environments.

4.Mental Simulation Application

In this section, we introduce two different mental simulation applications, namely, reinforcement learning and motion planning.

4.1 Reinforcement Learning

Reinforcement learning (RL) is inspired by one of the first learning methods used by humans, namely learning by experience. RL works by letting an agent explore a given environment and giving or removing rewards based on how well the agent performed the desired task. RL has shown promising results in board games [62] and computer games [63], mostly due to their controlled environment and clear objectives. Robots, on the other hand, act on the physical world, making it difficult to generate a large variety of experiences without putting either the robot or their surroundings at risk. One solution for this issue is to train a simulated robot and then transfer its knowledge to the real robot [24]. In this work, we used the mental simulation as a training environment for an autonomous navigation reinforcement learning algorithm.

A navigation problem can be formulated as a Partially Observable Markov Decision Process (POMDP) [46] ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{O}$, $\mathrm{R(s,a)}$, $\mathrm{T(s'|s,a)}$, $\mathrm{P(o|s)}$). $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{O}$ are the state, action and observation spaces, respectively, while $\mathrm{R(s,a)}$ is a function which receives the current state and action and returns a corresponding reward. Finally, $\mathrm{T(s'|s,a)}$ and $\mathrm{P(o|s)}$ are the transition and observation probabilities, respectively. RL involves finding a policy $\pi(o) \rightarrow a, o \in \mathcal{O}, a \in \mathcal{A}$, which maps the current observation into an action that maximizes the sum of

the expected future rewards. To simplify this notation, we will refer to this policy as $\pi(a|s)$, i.e., a function that maps a state to an action. RL methods can be divided into model-based and model-free value-based approaches. Model-based approaches use a predictive function that receives the current state and a sequence of actions and outputs the sum of the expected rewards. The policy then selects the action sequence that maximizes the expected rewards based on the predicted states. To generate such outputs, model-based algorithms need to understand its environment and learn both the reward function R and the transition probability function T. Model-free algorithms, on the other hand, directly learn either a policy function or a value function (or both in the case of actor-critic networks). A value function receives the current state and a given action and outputs the sum of expected rewards. Generally, model-based approaches are sample-efficient, while the model-free ones are better at learning high-dimensional, complex tasks.

4.1.1 Problem Definition

Our navigation task consists of a robot that receives a laser scan and a goal relative position and outputs a velocity command. Similarly to [24], instead of using the raw laser scan data, we used 23 equally spaced discrete laser readings. Using incomplete data forces the robot to better generalize to faulty input information, increasing its robustness when encountering real-world scenarios [24]. We also added Gaussian errors to input and output data to further mimic real-world conditions. The goal relative position was encoded on the (r, θ) format. Moreover, the robot should output two distinct commands: the desired linear and angular velocities.

The environment reward is one of the key hyperparameters when defining RL algorithms. It is used to evaluate a given action and to generate gradients that steer the policy into pursuing better rewards. The navigation problem described in this work used the following rewards:

$$r(s,a) = \begin{cases} r_{completion} - t * r_{time}, & \text{when arriving at the goal at time t,} \\ r_{closer}, & \text{if getting closer to the goal,} \\ -r_{collision}, & \text{if too close to the obstacle,} \end{cases} \quad \text{Eq. 1}$$

where $r_{completion}$, r_{time}, r_{closer} and $r_{collision}$ are positive real numbers.

4.1.2 Q-Learning

A value function $V(s)$ is a function that gives the expected return when starting from the state s. The value function is defined using a self-consistency equation called the Bellman equation [64]. The Bellman equation states that the value of an input can be modeled as the immediate reward of this initial choice, plus a weighted sum of the resulting state value. The on-policy and optimal value functions can be defined as

$$V^{\pi}(s) = \underset{a\sim\pi}{E}[r(s,a) + \gamma V^{\pi}(s')], \quad \text{Eq. 2}$$

$$V^{*}(s) = \max_{a} r(s,a) + \gamma V^{*}(s'), \quad \text{Eq. 3}$$

where π is the policy, a is an action, s is the current state, s' is the next state, $r(s,a)$ is the reward function and $\gamma \sim [0,1]$. Similarly, we can define the action-value function, also known as Q function. The Q function $Q(s,a)$ returns the

expected value of being on the state s and performing the action a. The on−policy and the optimal action−value functions can also be modeled by the Bellman equation as follows:

$$Q^{\pi}(s,a) = r(s,a) + \gamma \underset{a' \sim \pi}{E}[Q^{\pi}(s',a')], \quad \text{Eq. 4}$$

$$Q^{*}(s,a) = r(s,a) + \gamma \max_{a'} Q^{*}(s',a'). \quad \text{Eq. 5}$$

Q−Learning approaches try to generate a $Q(s,a)$ function that approximates the optimal action−value function shown on Eq. 5, thus called the target function. An example of Q−Learning algorithm is the DQN [47] which uses a DNN to approximate $Q^{*}(s,a)$.

4.1.3 Policy Gradients

Policy gradient algorithms try to directly optimize the policy $\pi_{\theta}(a|s)$, where θ are some learnable parameters of π. To do so, an objective function $J(\pi_{\theta})$ is used to quantify the policy's performance. The policy can then be optimized through gradient ascent, as follows:

$$\theta_{t+1} = \theta_{t} + \alpha \nabla_{\theta} J(\pi_{\theta})|_{\theta_t} \quad \text{Eq. 6}$$

where α is the learning rate and $\nabla_{\theta} J(\pi_{\theta})$ is called the policy gradient. Due to explicitly learning the desired function, i.e., the policy, policy gradient approaches tend to be more stable when compared to Q−Learning algorithms,

which indirectly improves the policy by learning an action−value function. Q−Learning methods, though, can be performed off−policy, which means that they can efficiently reuse past data, increasing their sample−efficiency.

4.1.4 Actor−Critics

Actor−critic methods are a combination of policy and value learning, that aims to achieve both policy gradient algorithms stability and Q−learning sample efficiency. The main idea is to use two different networks: one to generate an action based on a state, and another one to evaluate this action by approximating a state value or action−state value function.

The actor receives the observation as input and tries to generate the best action. In other words, it learns how to approximate the optimal policy to control the agent's behavior. Its training is done using the policy gradient method by carefully choosing an objective function that is related to the critic's evaluation.

The critic evaluates the taken action by computing a value function. This value function learns how to predict either the state or the action−state value function by using either Eq. 3 or Eq. 5 as a target function and minimizing the minimum square error by performing the gradient descent algorithm.

Using this method, the critic learns to better evaluate how good or how bad is to take a given action on a given state. In contrast, the actor uses this evaluation to improve its action generation aiming to maximize future rewards. The result is that each network learns how to perform a task more efficiently when compared to when trained separately.

4.1.5 Deep Deterministic Policy Gradients

The deep deterministic policy gradients (DDPG) was originally proposed by Lillicrap et al. [48] and used one DNN as a policy approximator (actor) and another as an action−state value function estimator (critic). Its original motivation was to extend DQN capabilities to environments with continuous action spaces. When taking an optimal action, a DQN agent would compute an action $a = \max_a Q^*(s,a)$ which is trivial on discrete action spaces where there are a finite number of possible actions. However, on continuous spaces, directly finding $\max_a Q^*(s,a)$ becomes a non−trivial optimization problem, which would need to exhaustively search actions on a continuous space on each iteration. If the policy is deterministic, however, $\max_a Q^*(s,a)$ can be approximated to $\max_a Q^*(s,a) \approx Q^*(s,\pi_\theta(s))$ allowing us to adapt the target from Eq. 5 to

$$Q^*(s,a) = r(s,a) + \gamma Q^*\big(s', \pi_\theta(s)\big). \qquad \text{Eq. 7}$$

Then, we can sample a set $\mathcal{D}$ of independent experience tuples (s,a,r,s',d) where $d \in [0,1]$ is a boolean which states whether or not it was a terminal experience. The loss function for the critic network then becomes the mean−squared Bellman error (MSBE)

$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{E}\left[\left(Q_\phi(s,a) - (r + \gamma(1-d)Q_{\phi_{targ}}(s', \pi_{\theta_{targ}}(s'))\right)^2\right], \qquad \text{Eq. 8}$$

which we use to perform gradient descent

$$\phi \leftarrow \phi - \alpha \nabla_{\phi} L(\phi, \mathcal{D}) \quad \text{Eq. 9}$$

where $Q_{\phi_{targ}}$ and $\pi_{\theta_{targ}}$ are the target critic and target policy networks, and α is the learning rate hyperparameter. When minimizing the MSBE, we are trying to make the action−state value function as close as possible to the target. If this target uses the same network as the one being learned, the MSBE becomes unstable, as we would be using a moving target for learning. To solve this issue, Lillicrap et al. proposed a soft−updated target network, which would lag behind the trained networks to provide a more stable target. The target weight updates should happen every iteration using the Polyak averaging

$$\phi_{targ} \leftarrow \tau \phi_{targ} + (1 - \tau)\phi, \quad \text{Eq. 10}$$

$$\theta_{targ} \leftarrow \tau \theta_{targ} + (1 - \tau)\theta, \quad \text{Eq. 11}$$

where $\tau \sim [0,1]$.

To train the policy network $\pi_{\theta}(s)$ we need to define an objective function $J(\pi_{\theta})$. The authors of [48] chose to use a function that tries to maximize $Q_{\phi}(s,a)$. Hence, we can define the objective function as

$$J(\pi_{\theta}) = \max_{\theta} \underset{s \sim \mathcal{D}}{E} [Q_{\phi}(s, \pi_{theta}(s))], \quad \text{Eq. 12}$$

which we use to perform gradient ascent

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\pi_{\theta}). \quad \text{Eq. 13}$$

Finally, in order to encourage exploration, every action taken during training has a random normal error $\epsilon \sim \mathcal{N}(0, \sigma)$ added to it.

4.1.6 Twin Delayed DDPG

Despite its overall good performance, DDPG has a common failure point of overestimating the Q−values, which is then exploited by the policy network. This usually leads to unstable learning or the agent overfitting to a sub−optimal policy. Fujimoto et al. [49] proposed the twin delayed DDPG (TD3) to address those issues. TD3 differs from the original DDPG on three main points: target policy smoothing, clipped double Q−learning, and delayed policy updates.

Target policy smoothing adds a clipped random normal error to the target policy

$$a' = clip\left(\pi_{\theta_{targ}}(s') + clip(\epsilon, -\epsilon_{max}, \epsilon_{max}), a_{min}, a_{max}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma), \qquad \text{Eq. 14}$$

which works as a regularizer, avoiding the target function mistakenly exploiting sub−optimal actions. Clipped double Q−learning use two different DNNs to approximate $Q_{\phi 1}$ and $Q_{\phi 2}$. Both functions use the same target on their loss functions by choosing whichever network outputs the smallest values

$$y(r, s', d) = r + \gamma(1 - d)\min_{i=1,2} Q_{\phi i_{targ}}(s', a'), \qquad \text{Eq. 15}$$

$$L(\phi_1, \mathcal{D}) = \underset{(s,a,r,s',d)\sim \mathcal{D}}{E}\left[\left(Q_{\phi 1}(s, a) - y(r, s', d)\right)^2\right], \qquad \text{Eq. 16}$$

$$L(\phi_2, \mathcal{D}) = \underset{(s,a,r,s',d)\sim \mathcal{D}}{E}\left[\left(Q_{\phi 2}(s,a) - y(r,s',d)\right)^2\right]. \quad \text{Eq. 17}$$

This upper bounds the target, avoiding its overestimation thus increasing the learning stability. Finally, the policy objective function was then adapted to maximize $Q_{\phi 1}$, but updated only every n critic updates, allowing a more stable critic value to be used as an objective function. Our work modified the policy learning by adding a custom objective focused on navigation tasks

$$J(\pi_\theta) = \underset{\theta}{max}\, \underset{s\sim\mathcal{D}}{E}\left[Q_{\phi 1}\left(s, \pi_\theta(s)\right)\right] - \beta_1(\pi_\theta(s)[0] - 1)^2 - \beta_2(\pi_\theta(s)[1])^2, \quad \text{Eq. 18}$$

where β_1 and β_2 are hyperparameters and $\pi_\theta(s)[0]$ and $\pi_\theta(s)[1]$ are the linear and angular velocities, respectively. By trying to maximize this objective function, we encourage the robot to increase its linear velocity and decrease the angular one, which may lead to a more smooth behavior. Our TD3−based architecture is shown in Figure 6, and the training algorithm shown in Algorithm 1. We used a sigmoid function to encode the linear speed between $[0,1]$ and the hyperbolic tangent function to limit the angular speed between $[-1,1]$. Both values are multiplied by constant weights, $v = 0.8m/s$, $\omega = 0.5\, rad/s$, before being sent to the robot as a velocity command.

Algorithm 1 TD3 training

Create $\pi_\theta(s), Q_{\phi 1}(s, \pi_\theta(s))$ *and* $Q_{\phi 2}(s, \pi_\theta(s))$

Copy initial weights to the targets $\theta_{targ} \leftarrow \theta, \phi_{1_{targ}} \leftarrow \phi_1, \phi_{2_{targ}} \leftarrow \phi_2$

loop *until converges*

  Get observation s and compute $a = clip(\pi_\theta(s) + \epsilon, a_{min}, a_{max}), \epsilon \sim \mathcal{N}(0, \sigma_1)$

  Execute a and get the next state s', *the reward* r *and the done signal* d

  Store (s, a, r, s', d) *into the experience buffer* $\mathcal{E}_\mathcal{B}$

  If d *is equal to* 1, ***then*** *reset the environment*

  Sample a random batch $\mathcal{D} = \{(s, a, r, s', d)\}_n$ *from* $\mathcal{E}_\mathcal{B}$

  Calculate the target actions

$$a' = clip\left(\pi_{\theta_{targ}}(s') + clip(\epsilon, -\epsilon_{max}, \epsilon_{max}), a_{min}, a_{max}\right), \ \epsilon \sim \mathcal{N}(0, \sigma_2)$$

  Get the minimal target

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi i_{targ}}(s', a')$$

  Calculate the critic loss

$$L(\phi_i, \mathcal{D}) = \frac{1}{n} \sum_{(s,a,r,s',d) \sim \mathcal{D}} \left(Q_{\phi i}(s, a) - y(r, s', d)\right)^2, \ i = 1,2$$

  Perform gradient descent

$$\phi_i \leftarrow \phi_i - \alpha \nabla_{\phi i} L(\phi_i, \mathcal{D}), \ i = 1,2$$

  If *policyUpdate*, ***then***

    Calculate the objective function

$$J(\pi_\theta) = \frac{1}{n} \sum_{s \sim \mathcal{D}} Q_{\phi 1}(s, \pi_\theta(s)) - \beta_1(\pi_\theta(s)[0] - 1) - \beta_2 \pi_\theta(s)[1]$$

    Perform gradient ascent

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$$

    Update the target networks

$$\theta_{targ} \leftarrow \tau \theta_{targ} + (1 - \tau)\theta$$

$$\phi_{i_{targ}} \leftarrow \tau \phi_{i_{targ}} + (1 - \tau)\phi_i, \ i = 1,2$$

  End if

End loop

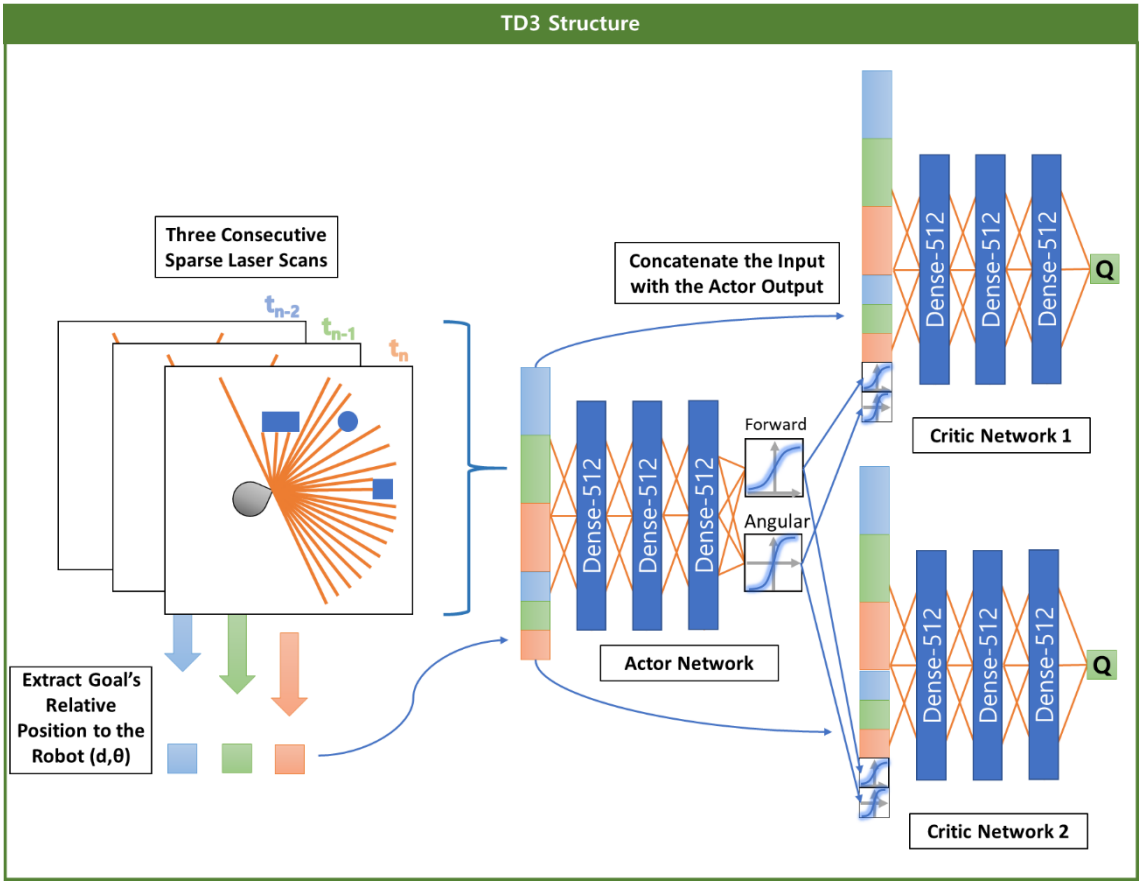

Figure 6 – TD3 architecture for autonomous navigation

4.2 Motion Planning

Despite the constant development of new RL technologies, they still have issues when dealing with long horizon rewards. The γ hyperparameter on Eq. 15 weights how important future rewards are when compared to immediate ones. The work done by Berner et al. [63] was able to use a 12 minutes reward horizon ($\gamma = 0.9998$), but further increasing of this horizon gave diminishing

returns. Navigation tasks on large-scale environments can take from several minutes to hours, which increases the difficulty of learning how advantageous an action was when its result comes several minutes in the future. Hence, in this work, we used a hybrid structure that combines a sampling-based planner with an RL policy, an approach which has already shown good results on long-term navigation [58]. We use the mental simulation environment to validate the feasibility of a motion plan before its execution in the real world. Such ability allows robots to test their plans on a simulation generated using its current knowledge, allowing them to verify a plan safety and feasibility without the risk of damaging itself or its surrounding environment.

4.2.1 Waypoint Generator

The waypoint generator receives a sequence of goal points in a large-scale environment and the robot's current position. It then queries the on-demand database for the local metric map. We adapted the PRM algorithm proposed by [54] to the mobile robot 2D navigation domain. It starts by generating n random sample points $\mathcal{N}_i$ on the metric map re-sampling any point outside of the map free space $\mathcal{C}_{free}$ until n samples are successfully generated. Then, for each sample $\mathcal{N}_i$, k valid edges are generating connecting $\mathcal{N}_i$ to its nearest neighbors. An edge $\mathcal{E}(\mathcal{N}_i, \mathcal{N}_j)$ is considered valid if $\mathcal{N}_i$ and $\mathcal{N}_j$ can be connected by a straight line completely inside $\mathcal{C}_{free}$ while also being smaller than the maximum distance threshold d_{max}. After generating all the edges, any sample that is not connected to any other node is re-sampled, and its edges are generated. This whole process depends solely on the 2D metric map, and can be performed off-line (when downloading the on-demand database) or whenever the robot has

available processing power. After receiving a goal point $\mathcal{N}_g$ and an initial position $\mathcal{N}_s$, the algorithm adds both to the PRM graph, connecting each to their k nearest neighbors. Finally, it finds the list of waypoints that are part of the shortest path between $\mathcal{N}_s$ and $\mathcal{N}_g$ using the Dijkstra's Algorithm [65]. This list of waypoints is then sent to the RL policy one-by-one, by sending the next waypoint whenever the robot is close enough to the current goal.

4.2.2 Hybrid PRM-RL Planner

This hybrid PRM-RL approach can be seen as a Global-Local planning scheme, where the PRM algorithm is responsible for generating a sparse set of goals based on the global view, while the RL algorithm should use the local data to react to dynamic changes in the environment while also following the goals set by the PRM block. The overall planning architecture can be seen in Figure 7.

After receiving a goal point, the hybrid planner sends the current robot position and the goal point to the PRM module. The PRM module generates a sequence of valid waypoints and returns it to the planner. The waypoint handler sub-module loop through this sequence, sending the waypoints one by one to the navigation policy. Every time a new waypoint is sent, a 15 seconds timer is started. If the navigation policy cannot reach the waypoint within this time, the hybrid planner performs replannning using the robot's new current position. The navigation policy uses only the actor-network when performing online actions, speeding up the process. After arriving at the waypoint, the navigation policy

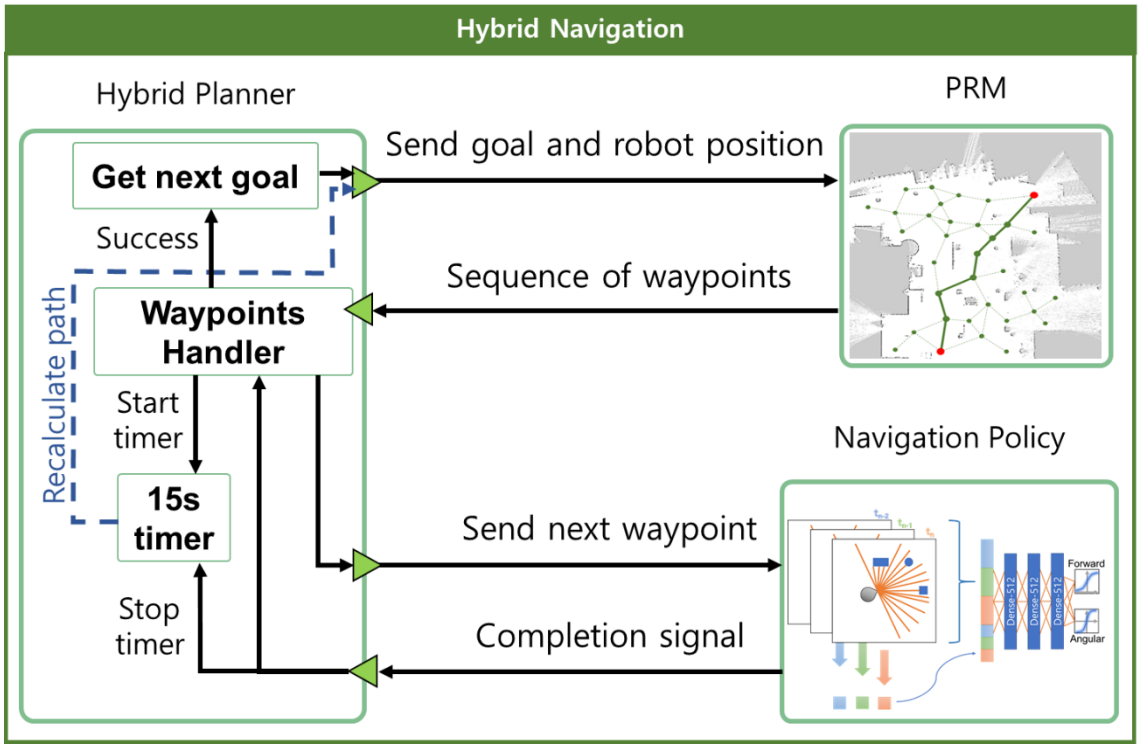

Figure 7 – Hybrid Planner Structure

sends a completion signal back to the hybrid planner, stopping the timer and requesting the next waypoint. After the robot arrives at the final goal, the hybrid planner emits a mission completed signal.

5.Experiments and Discussion

In this section, we evaluate the performance of the proposed hybrid navigation method.

5.1 Number of Samples

The number of samples n is one of the key parameters of the PRM algorithm. Too few and you might get a disconnected graph, which would fail to find a valid path requiring the graph to be re-generated. Too many and the algorithm's exponential nature makes the process time-inefficient. We conducted experiments to investigate the influence of the number of samples on the algorithm's performance. We used nine different metric maps from two different environments: the 1^{st} floor of the KDJ Convention Center and the 7^{th} floor of the Corporation Collaboration Center from Sungkyunkwan University as shown in Figure 8. Table 1 shows each map size information. We used each place's grid map width and height to calculate the overall map size. To calculate the free space ratio, we divided the number of free pixels by the total number of pixels in the grid map image. Finally, to generate the free space area, we multiplied the map's total size by the free space ratio.

Table 1 – Map Information

Place Name	Place Total Area (m^2)	Free Space Ratio	Free Space Area (m^2)
KDJ Convention Center			
1st floor	39759.4	0.123	4890.41
corridor 1	13632.0	0.258	3517.06
corridor 2	4672.0	0.391	1826.75
corridor 3	3124.0	0.271	846.60
Corporate Collaboration Center 7th floor			
corridor 1	350.0	0.160	56.01
corridor 2	528.0	0.277	146.42
corridor 3	157.25	0.513	80.70
corridor 4	200.0	0.275	55.09
corridor 5	423.0	0.212	89.75

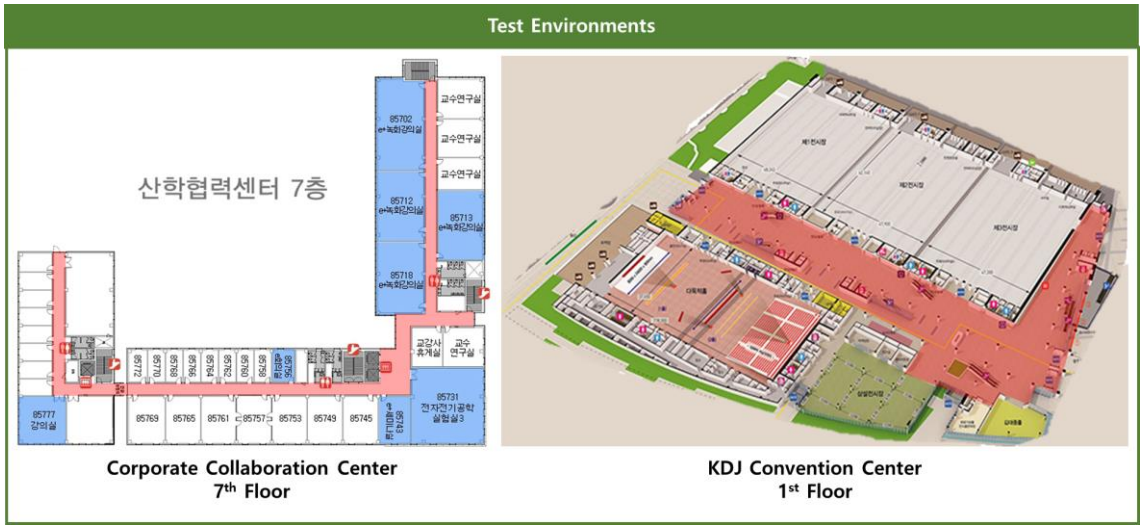

Figure 8 – Mapped Environments. The mapped area is shown in red.

We then heuristically chose a set of the number of samples $n = \{50, 100, 200, 300, 400, 500\}$, and ran the graph generation and pathfinding algorithm 50 times for each value of n on each map. Figure 9 shows the influence of the number of samples to four performance measurements: the time taken to build the PRM graph; the time taken to find a valid path from $\mathcal{N}_s$ to $\mathcal{N}_g$; the length of such path; and whether or not the path could be found. The values shown in Figure 9 are averaged across the 50 different tries. We can easily notice that while the PRM building time grew quadratically with the number of samples, the path search time grew linearly. Because in our approach we can generate those graphs off-line when downloading the map, the effect of the

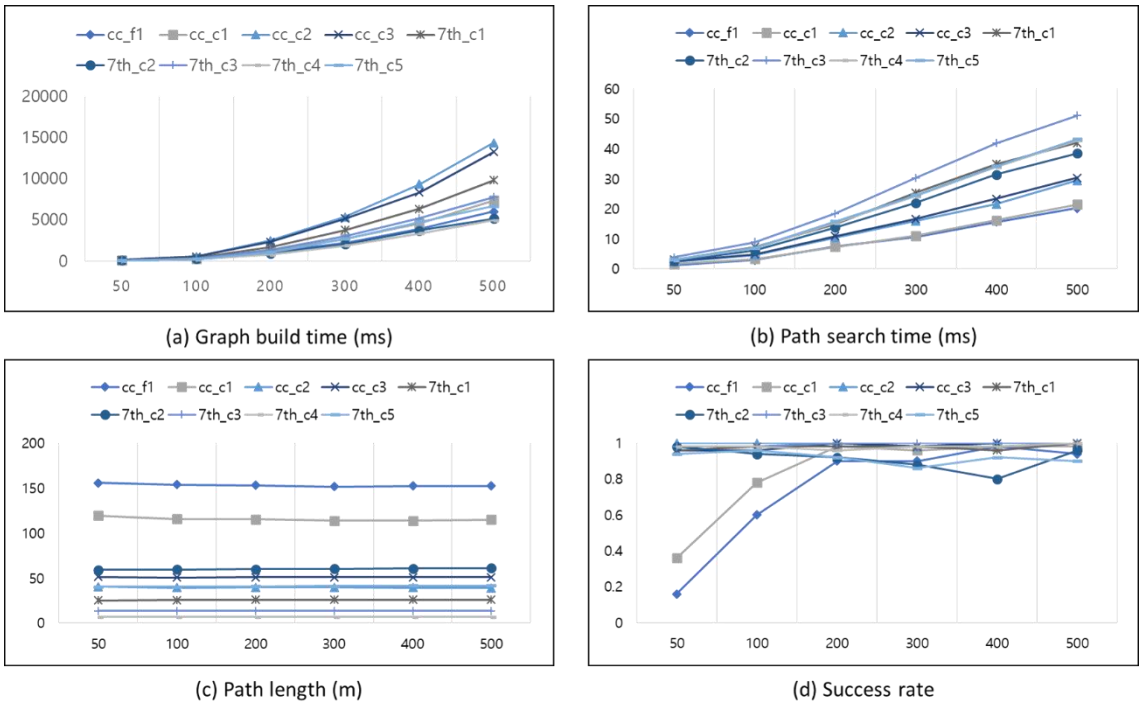

Figure 9 – PRM performance based on the number of samples (x-axis). In the graph labels, "cc" stands for Convention Center, "7th" for the 7th floor of the Corporate Collaboration Center and "c_i" for the corridor_i.

graph build time on the robot's performance can be mitigated. Interestingly, smaller places have shown a higher path search time for the same number of samples when compared to larger places. This is due to the nature of the Dijkstra's algorithm, where if many points are close to each other, the number of valid paths to the goal rises, increasing the path searching time. Moreover, for each map, there is a minimum number of nodes so that the path can be successfully found. In a real application, whenever a path cannot be found, the graph should be re-generated, leading to slowdowns or even deadlocks. This is not a desirable condition, showing that an optimal number of samples is necessary. Finally, the number of nodes did not seem to interfere with the overall path length.

These findings motivated us to find optimal n values based on the environment free space size. Thus, we performed a linear regression while optimizing two constraints: the path search time should be no longer than 15ms, and the success rate should be above 90%. Such optimization yield the following equation:

$$Number\ of\ Samples = ceil(0.05 * Free\ Space\ Area + 84), \quad \text{Eq. 19}$$

where $ceil$ is the rounding up operation. We then used Eq. 19 to calculate the optimal number of samples n^* for each map. After re-running the tests 50 more times using the corresponding n^* values, we obtained the results shown in Table 2. Using the proposed n^* values, both constraints were satisfied on every map.

Faust et al. [58] used fixed densities of 0.1x, 0.2x, and 0.4x, while we used

the regressed formula shown in Eq. *19*. For smaller maps, our approach had a higher number of nodes on average but a way lower amount of nodes on bigger maps. Faust et al. had success rates of 32%, 36%, and 50%, respectively, while our approach had a 90% success rate on executing the generated paths. It should be noted that Faust et al. maps were more complex than the ones used in our experiments. Faust et al. also did not perform edge collision checks to allow faster PRM graph generation. Instead, they validated the edges using a probability distribution of whether the RL algorithm would be able to follow such edge. This approach had a trade-off between lower success rates and lower graph construction times. We generated the PRM graph only when receiving the map, allowing us to check edge collision and generate safer paths, while keeping the real-timeness of the hybrid planner.

5.2 Loss Function Regularizer

To evaluate the influence of the regularization terms β_1 and β_2 from Eq. 18, we trained the robot using three different values for both regularizers: $\beta_1 = \beta_2 = \{0, 1.0, 5.0\}$. The average rewards obtained during each training are shown in Figure 10. The training was done in the mental simulation of the 7^{th} floor of the Corporate Collaboration Center, shown in Figure 8. All models were trained from scratch, using an Nvidia RTX 2080TI GPU to perform the batch optimizations. Each training session ended after 30000 episodes (approximately 1 million experiences and 116 hours of training). All the hyper-parameters were the same across the three models apart from the regularization term

Table 2 – Optimized Number of Samples

Place Name	Number of Samples	Graph Build Time (ms)	Path Search Time (ms)	Path Length (m)	Success Rate
KDJ Convention Center					
1st floor	329	2664.3	12.06	152.58	92%
corridor 1	260	1996.5	10.18	114.43	94%
corridor 2	176	1502.3	9.15	39.41	100%
corridor 3	127	759.68	7.86	50.68	98%
Corporate Collaboration Center 7th floor					
corridor 1	87	247.98	5.22	25.60	92%
corridor 2	92	183.0	4.68	59.48	94%
corridor 3	89	195.37	6.56	13.41	92%
corridor 4	87	119.20	4.99	6.71	98%
corridor 5	89	169.88	5.33	40.57	96%

Figure 10 shows that by adding a small regularization (green line) term to encourage the robot to move forward speeds up the learning when compared with the training without regularization (blue line). When there was no regularization, the robot would initially learn that simply not moving is a good strategy to avoid collisions. This behavior slows down the training and the variety of the experiences collected. The regularization term is a way to induce the robot to move, therefore improving the exploration of initial steps. The un-regularized policy eventually achieves similar rewards as the regularized version, which shows that most of the regularization benefits show on the exploration phase. In other words, the regularized model achieves similar performance while requiring a smaller amount of training. However, if the

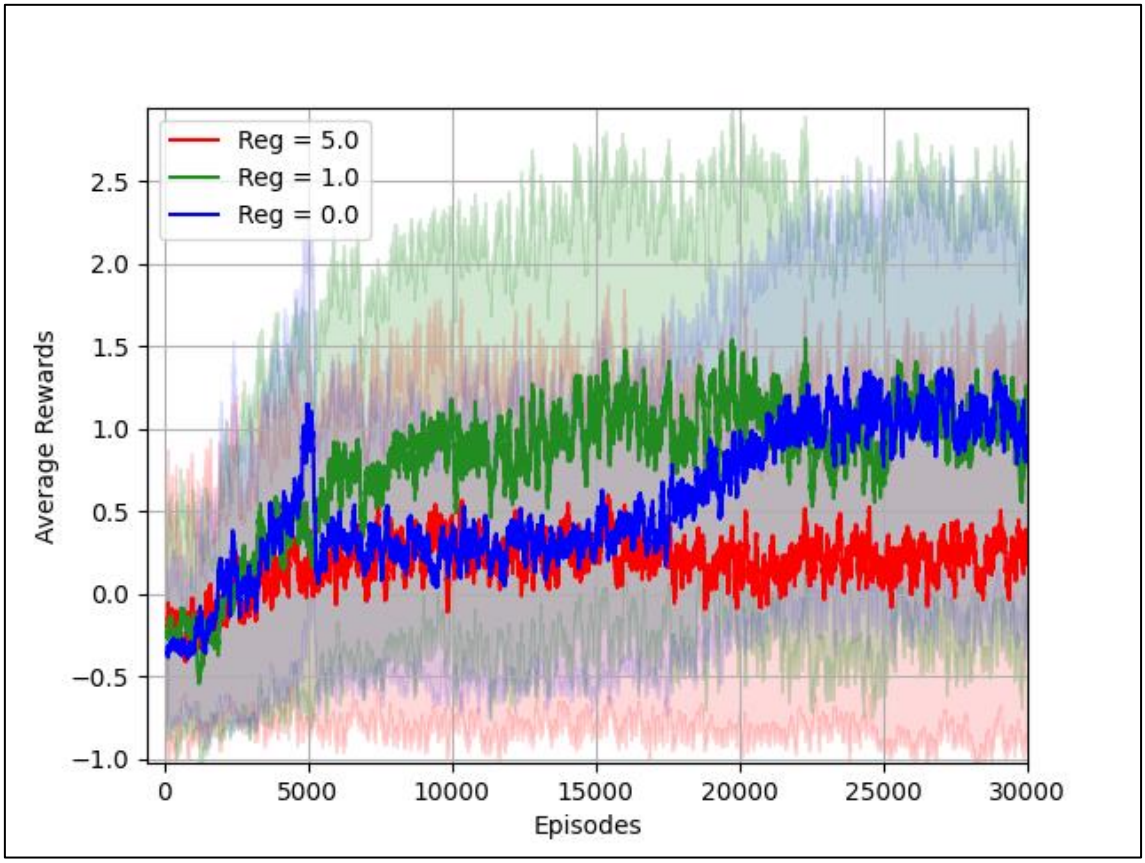

Figure 10 – Moving average of the rewards obtained during each training. Each vertical light-colored interval represents the moving standard deviation. A window of 100 episodes was used for both moving average and moving standard deviation.

regularization term is too big (red line), it can hinder the robot's performance or even make the robot not learn at all. When using high regularization values, the robot would ignore the reward and optimize only the regularization term, which resulted in the robot going straight into walls.

5.3 Hybrid Navigation

To verify the robustness of the hybrid navigation method proposed in this work, we performed a long-range navigation mission in a simulated environment. We compared our work with the state-of-the-art motion planner *Move Base*. Whereas our hybrid autonomous navigation algorithm needs only 23 sparse laser range readings, Move Base utilizes all the 690 ranges. Hence, we also compared our work with a *Constrained Move Base*, which only received the same 23 range readings as our algorithm. Finally, we also tested how the pure RL policy would deal with a long-range mission. The task environment is shown in Figure 11. The robot should start from one of the edges of the building and navigate without collisions until the opposing end. The total path has approximately 140 meters. We performed the same task 10 times for each planner. The summary of this experiment is shown in Table 3.

We can see that the pure RL policy was not able to complete a long-range task without colliding. Due to the RL algorithm receiving only the Euclidian distance to the goal, the relative angle would make the robot want to turn back, go against a wall and then try to avoid hitting it, repeating this process in a loop. This behavior ultimately led to the robot going too close to a wall and colliding with it. Using only the RL policy resulted in the robot failing to leave the first corridor on all tries.

Table 3 – Navigation Task Results

Method	Success		Failure	Success Rate
	Average Distance (m)	Average Time (s)	Average Distance (m)	
Move Base	141.04	123.31	—	100%
Constrained Move Base	140.21	125.19	26.87	60%
Pure RL	—	—	14.7	0%
Hybrid PRM–RL	149.71	160.45	102.4	90%

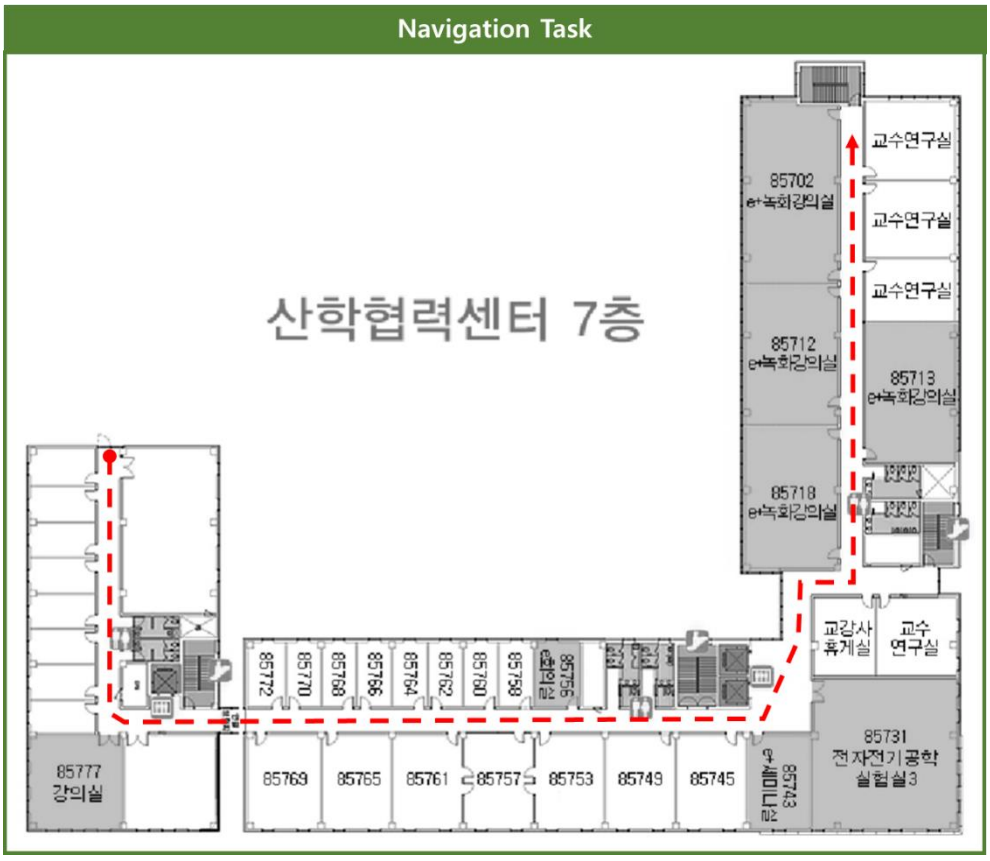

Figure 11 – Long–range navigation task.

The baseline algorithm (i.e., Move Base) was able to perform the task seamlessly in the shortest time and 100% success rate. Nonetheless, after reducing the laser scanner resolution, the algorithm performed poorly. Due to the lower resolution, the robot failed to perceive the obstacle when turning the first corner, hitting it 40% of the time (usually when entering the curve too close to the wall). Our proposed hybrid PRM-RL approach was able to complete the task 90% of the time, using only 23 sparse laser scans to avoid obstacles, and taking 7ms to generate a control command. The single failure happened due to the PRM algorithm sampling a point too close to obstacles, which led to a collision. The algorithm, however, could not perform as well as the original Move Base algorithm, taking more time and some times a longer path. The main cause was the localization algorithm miscalculating the robot position, then suddenly updating to the correct position further ahead. Occasionally, this would cause the robot to miss one of the PRM goals, turning back, reaching that goal, then turning back again to continue the original path. Such behavior increased the average time and distance taken to finish the task.

This validation experiment mimics the application proposed on this work, where the mental simulation can be used to verify the feasibility of a plan. If the sampled path contains a node too close to an obstacle, which would cause a collision in the simulation, the roadmap can be re-sampled and a new path generated. This would avoid the robot the need of the robot performing such action in the real world, enhancing its overall safety.

6.Conclusion and Future Work

In this paper, a novel automatic mental simulation framework was proposed. We have shown the usage of TOSM for environmental modeling and extended its capabilities to represent robotic agents as well. This TOSM-based data was used to generate a simulated environment without the need for human aid, depending only on the robot known information. Such simulation is robust to new objects, generating placeholders whenever an object has no 3D model available. This allows our approach to be used in unknown environments, regardless of the availability of 3D models for the encountered objects. We showed that the mental simulation system could be used to train RL policies. We also proposed an autonomous navigation policy that uses a sparse laser range scan and a relative position in order to navigate in an unknown environment. This policy used a custom loss with a regularization term that improved the learning speed of the algorithm when compared with the baseline. We also proposed a hybrid navigation algorithm that integrates a sampling-based planner with an RL policy, enabling real-time long-range navigation. Even though the path taken by our proposed algorithm was less optimal than the state-of-the-art baseline, it showed a higher success rate when both algorithms were using only a sparse input. We showed that mental simulation could be used as a validation step before acting on the real world, enhancing the robot's safety.

As an ongoing work, we will continue to further improve the RL policy by adding dynamic obstacles to the environment to further enhance the robot's reactiveness. We will also port the mental simulation to a mobile robot to fully

integrate our proposed method with physical agents. The mental simulation can be a central component of the new generations of robots. By using low latency 5G connections, robots can communicate with the cloud in real time. Thus, the mental simulation can be run on a centralized cloud solution, validating robot plans on real-time, and improving the overall safety of mobile robot operations.

References

[1] D. Kahneman and A. Tversky, "The Simulation Heuristic," *STANFORD UNIV CA DEPT Psychol.*, vol. TR-5, 1981.

[2] T. Suddendorf and J. Busby, "Mental time travel in animals?," *Trends Cogn. Sci.*, vol. 7, no. 9, pp. 391–396, 2003.

[3] P. Boyer, "Evolutionary economics of mental time travel?," *Trends Cogn. Sci.*, vol. 12, no. 6, pp. 219–224, 2008.

[4] N. Burgess, "Spatial cognition and the brain," *Ann. N. Y. Acad. Sci.*, vol. 1124, pp. 77–97, 2008.

[5] G. Hesslow, "The current status of the simulation theory of cognition," *Brain Res.*, vol. 1428, pp. 71–79, 2012.

[6] R. M. Gordon, "Folk psychology as simulation," *Mind Lang.*, vol. 1, no. 2, pp. 158–171, 1986.

[7] A. M. TURING, "COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, no. 236, pp. 433–460, Oct. 1950.

[8] M. Polceanu and C. Buche, "Computational mental simulation: A review," *Comput. Animat. Virtual Worlds*, vol. 28, no. 5, pp. 1–15, 2017.

[9] I. Kostavelis, K. Charalampous, A. Gasteratos, and J. K. Tsotsos, "Robot navigation via spatial and temporal coherent semantic maps," *Eng. Appl. Artif. Intell.*, vol. 48, pp. 173–187, 2016.

[10] A. Cosgun and H. I. Christensen, "Context-aware robot navigation using interactively built semantic maps," *Paladyn*, vol. 9, no. 1, pp. 254–276, 2018.

[11] R. Waibel, M. and Beetz, M. and Civera, J. and D'Andrea, R. and Elfring, J. and Galvez-Lopez, D. and Haussermann, K. and Janssen, R. and Montiel, J.M.M. and Perzylo, A. and Schiessle, B. and Tenorth, M. and Zweigle, O. and van de Molengraft, "RoboEarth-A World Wide Web for Robots," *Robot. Autom. Mag. IEEE*, vol. 18, no. June, pp. 69–82, 2011.

[12] B. L. Douglas, "CYC: A Large-Scale Investment in Knowledge Infrastructure," *Commun. ACM*, vol. 38, no. 11, pp. 33–38, 1995.

[13] I. Niles and A. Pease, "Towards a standard upper ontology," *Form. Ontol. Inf. Syst. Collect. Pap. from Second Int. Conf.*, pp. 2–9, 2001.

[14] R. Gupta and M. J. Kochenderfer, "Common sense data acquisition for indoor mobile robots," *Proc. Natl. Conf. Artif. Intell.*, vol. 94041, pp. 605–610, 2004.

[15] M. Firdaus-Nawi, O. Noraini, M. Y. Sabri, A. Siti-Zahrah, M. Zamri-Saad,

and H. Latifah, “Encoder–Decoder with Atrous Separable Convolution for Semantic Image Segmentation,” *Pertanika J. Trop. Agric. Sci.*, vol. 34, no. 1, pp. 137–143, 2011.
[16] J. Han, L. Yang, D. Zhang, X. Chang, and X. Liang, “Reinforcement Cutting-Agent Learning for Video Object Segmentation,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 9080–9089, 2018.
[17] S. Brahimi, N. Ben Aoun, A. Benoit, P. Lambert, and C. Ben Amar, “Semantic segmentation using reinforced fully convolutional densenet with multiscale kernel,” *Multimed. Tools Appl.*, vol. 78, no. 15, pp. 22077–22098, 2019.
[18] M. Tenorth and M. Beetz, “KNOWROB – Knowledge processing for autonomous personal robots,” *2009 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS 2009*, no. November 2009, pp. 4261–4266, 2009.
[19] M. Beetz, M. Tenorth, and J. Winkler, “Open-EASE,” *Proc. – IEEE Int. Conf. Robot. Autom.*, vol. 2015-June, no. June, pp. 1983–1990, 2015.
[20] M. Beetz, D. Bessler, A. Haidu, M. Pomarlan, A. K. Bozcuoglu, and G. Bartels, “Know Rob 2.0 – A 2nd Generation Knowledge Processing Framework for Cognition-Enabled Robotic Agents,” *Proc. – IEEE Int. Conf. Robot. Autom.*, pp. 512–519, 2018.
[21] T. Lombrozo, “‘Learning by Thinking’ in Science and in Everyday Life,” *Sci. Imagin.*, p. 230, 2019.
[22] L. Kunze and M. Beetz, “Envisioning the qualitative effects of robot manipulation actions using simulation-based projections,” *Artif. Intell.*, vol. 247, pp. 352–380, 2017.
[23] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” *Proc. – IEEE Int. Conf. Robot. Autom.*, pp. 3389–3396, 2017.
[24] L. Tai, G. Paolo, and M. Liu, “Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation,” *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 2017-Septe, pp. 31–36, 2017.
[25] P. Shah, M. Fiser, A. Faust, J. C. Kew, and D. Hakkani-Tur, “FollowNet: Robot Navigation by Following Natural Language Directions with Deep Reinforcement Learning,” 2018.
[26] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, “Self-Supervised Deep Reinforcement Learning with Generalized Computation Graphs for Robot Navigation,” *Proc. – IEEE Int. Conf. Robot. Autom.*, pp. 5129–5136, 2018.

[27] G. A. Miller, “WordNet: A Lexical Database for English,” *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.
[28] H. Liu and P. Singh, “ConceptNet — a practical commonsense reasoning tool-kit,” *BT Technol. J.*, vol. 22, no. 4, pp. 211–226, Oct. 2004.
[29] K. Sterelny, *The representational theory of mind: an introduction*. B. Blackwell, 1991.
[30] J. Decety, “Brain Structures Participating in Mental,” *Acta Psychol. (Amst).*, vol. 73, pp. 13–34, 1990.
[31] A. Tversky and D. Kahneman, “Availability: A heuristic for judging frequency and probability,” *Cogn. Psychol.*, vol. 5, no. 2, pp. 207–232, 1973.
[32] H. B. Kappes and C. K. Morewedge, “Mental Simulation as Substitute for Experience,” *Soc. Personal. Psychol. Compass*, vol. 10, no. 7, pp. 405–420, 2016.
[33] M. Hegarty, “Mechanical reasoning by mental simulation,” *Trends Cogn. Sci.*, vol. 8, no. 6, pp. 280–285, 2004.
[34] C. J. Bates, I. Yildirim, J. B. Tenenbaum, and P. W. Battaglia, “Humans predict liquid dynamics using probabilistic simulation,” *CogSci 2015*, no. July, pp. 172–177, 2015.
[35] B. Bergen and K. Wheeler, “Grammatical aspect and mental simulation,” *Brain Lang.*, vol. 112, no. 3, pp. 150–158, 2010.
[36] L. J. Speed and A. Majid, “An Exception to Mental Simulation: No Evidence for Embodied Odor Language,” *Cogn. Sci.*, vol. 42, no. 4, pp. 1146–1178, 2018.
[37] J. E. Laird, “It knows what you’re going to do,” 2001, pp. 385–392.
[38] D. Buchsbaum, B. Blumberg, C. Breazeal, and A. N. Meltzoff, “A simulation-theory inspired social learning system for interactive characters,” *Proc. - IEEE Int. Work. Robot Hum. Interact. Commun.*, vol. 2005, pp. 85–90, 2005.
[39] N. L. Cassimatis, J. G. Trafton, M. D. Bugajska, and A. C. Schultz, “Integrating cognition, perception and action through mental simulation in robots,” *Rob. Auton. Syst.*, vol. 49, no. 1-2 SPEC. ISS., pp. 13–23, 2004.
[40] M. Polceanu, M. Parenthöen, and C. Buche, “ORPHEUS: Mental simulation as support for decision-making in a virtual agent,” *Proc. 28th Int. Florida Artif. Intell. Res. Soc. Conf. FLAIRS 2015*, pp. 73–78, 2015.
[41] C. Buche, N. Le Bigot, and M. Polceanu, “Simulation within simulation for agent decision-making: Theoretical foundations from cognitive science to

operational computer model," *Cogn. Syst. Res.*, vol. 40, pp. 46–58, 2016.
[42] P. Vicente, L. Jamone, and A. Bernardino, "Online body schema adaptation based on internal mental simulation and multisensory feedback," *Front. Robot. AI*, vol. 3, no. MAR, 2016.
[43] D. Vanderelst and A. Winfield, "An architecture for ethical robots inspired by the simulation theory of cognition," *Cogn. Syst. Res.*, vol. 48, pp. 56–66, 2018.
[44] B. Gundersen, "Forming artificial memories during sleep," *Nat. Neurosci.*, vol. 18, no. 4, p. 483, 2015.
[45] H. F. Ólafsdóttir, F. Carpenter, and C. Barry, "Coordinated grid and place cell replay during rest," *Nat. Neurosci.*, vol. 19, no. 6, pp. 792–794, 2016.
[46] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, no. 1–2, pp. 99–134, 1998.
[47] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.
[48] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, 2016.
[49] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," *35th Int. Conf. Mach. Learn. ICML 2018*, vol. 4, pp. 2587–2601, Feb. 2018.
[50] P. Long, T. Fanl, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 6252–6259, 2018.
[51] S. D. Pendleton *et al.*, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, 2017.
[52] M. Šeda, "Roadmap Methods vs . Cell Decomposition in Robot Motion Planning," pp. 127–132, 2007.
[53] J. Lee, Os. Kwon, L. Zhang, and S. E. Yoon, "A selective retraction-based RRT planner for various environments," *IEEE Trans. Robot.*, vol. 30, no. 4, pp. 1002–1011, 2014.
[54] L. Kavraki and J. C. Latombe, "Randomized preprocessing of configuration space for fast path planning," in *Proceedings - IEEE International Conference on Robotics and Automation*, 1994, no. pt 3, pp. 2138–2145.
[55] S. M. Lavalle and S. M. Lavalle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1998.
[56] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal

motion planning," *Int. J. Rob. Res.*, vol. 30, no. 7, pp. 846–894, 2011.
[57] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014.
[58] A. Faust *et al.*, "PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 5113–5120, 2018.
[59] M. Fyhn, S. Molden, M. P. Witter, E. I. Moser, and M. B. Moser, "Spatial representation in the entorhinal cortex," *Science (80-.).*, vol. 305, no. 5688, pp. 1258–1264, 2004.
[60] S.-H. Joo, S. Manzoor, Y. G. Rocha, H.-U. Lee, and T.-Y. Kuc, "A Realtime Autonomous Robot Navigation Framework for Human like High-level Interaction and Task Planning in Global Dynamic Environment," 2019.
[61] M. A. Musen, "The protégé project," *AI Matters*, vol. 1, no. 4, pp. 4–12, Jun. 2015.
[62] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science (80-.).*, vol. 362, no. 6419, pp. 1140–1144, 2018.
[63] OpenAI *et al.*, "Dota 2 with Large Scale Deep Reinforcement Learning," 2019.
[64] R. Bellman, "The theory of dynamic programming," 1954.
[65] E. W. Dijkstra and others, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, 1959.

논문요약

온톨로지 기반 모델링 프레임워크를 이용한 자율 학습 및 계획을 위한 정신 시뮬레이션

Yuri Goncalves Rocha
전자전기콤퓨터공학과
성균관대학교

인지과학의 발견은 인간이 사고로 시뮬레이션 환경을 구성하는 뛰어난 능력을 가지고 있다는 것을 보여주었다. 이러한 시뮬레이션 환경은 미래를 전망하고, 과거의 기억을 재실행하며, 알려진 상황에 대해 다른 결과를 시뮬레이션하고, 궁극적으로 실제 물리 환경에서 동작하지 않고도 계획된 행동을 학습하고 시험하는데 사용될 수 있다. 이러한 인지 능력은 현재 로봇 시스템을 이와 유사한 방식으로 발전시켜, 로봇 시스템을 실행하기 전에 계획된 행동의 결과를 예측할 수 있게 할 수 있다. 또한 강화학습 알고리즘을 수행하기 위해 사람이 관여하지 않고도 자동으로 실행하는 플랫폼을 제공함으로써 점진적으로 기술을 향상시킨다. 그러기 위해서 정보가 풍부한 서술적 프레임워크를 사용하여 환경과 로봇 자체를 모델링해야 한다.

본 연구에서는 "트리플렛 온톨로지 시맨틱 모델"을 활용하여 로봇과 그 주변 환경을 표현하였다. 이 모델링된 데이터는 사람의 간섭없이 자동적으로 시뮬레이션 환경을 생성하는데 사용되었으며 생성된 시뮬레이션 환경은 자율 주행을 위한 강화 학습에 사용되었다. 또한 고전적인 플래너와 강화학습 알고리즘을 결합한 하이브리드 내비게이션 방식도 제안하여 long-term 내비게이션 기능을 개선했다.

실험 결과는 사고 시뮬레이션이 강화학습 알고리즘을 트레이닝하고 계획

타당성을 검증하는 데 사용될 수 있다는 것을 보여준다. 제안된 방법은 sparse 데이터만을 사용하여 long-term 내비게이션 작업을 수행할 수 있었고, 제어 명령을 실행하는 데 7ms가 소요되었다.

주제어: 사고 시뮬레이션, 자율 주행, 강화학습, 경로 계획, Probabilistic Roadmap

Master's Thesis	Mental Simulation for Autonomous Learning and Planning	2020	Yuri Goncalves Rocha