# Online Grounding of Symbolic Planning Domains in Unknown Environments

**Leonardo Lamanna**[1,2] , **Luciano Serafini**[1] , **Alessandro Saetti**[2] , **Alfonso Gerevini**[2] , **Paolo Traverso**[1]

[1]Fondazione Bruno Kessler, Trento, Italy
[2]University of Brescia, Italy
{llamanna,serafini,traverso}@fbk.eu {l.lamanna,
alessandro.saetti,alfonso.gerevini}@unibs.it

## Abstract

If a robotic agent wants to exploit symbolic planning techniques to achieve some goal, it must be able to properly ground an abstract planning domain in the environment in which it operates. However, if the environment is initially unknown by the agent, the agent needs to explore it and discover the salient aspects of the environment necessary to reach its goals. Namely, the agent has to discover: (i) the objects present in the environment, (ii) the properties of these objects and their relations, and finally (iii) how abstract actions can be successfully executed. The paper proposes a framework that aims to accomplish the aforementioned perspective for an agent that perceives the environment partially and subjectively, through real value sensors (e.g., GPS, and on-board camera) and can operate in the environment through low level actuators (e.g., move forward of 20 cm). We evaluate the proposed architecture in photo-realistic simulated environments, where the sensors are RGB-D on-board camera, GPS and compass, and low level actions include movements, grasping/releasing objects, and manipulating objects. The agent is placed in an unknown environment and asked to find objects of a certain type, place an object on top of another, close or open an object of a certain type. We compare our approach with a state of the art method on object goal navigation based on reinforcement learning, showing better performances.

## 1 Introduction

Symbolic planners are powerful and flexible tools (Ghallab, Nau, and Traverso 2016) that, given a general symbolic description of an available set of actions (i.e., a planning domain) and a detailed description of an environment, are capable of generating plans for achieving ideally any goal about (known) objects in the environment. In several applications, the information about the environment required to instantiate a planning domain is not available from the beginning. In particular, when an agent is placed in a new environment, it does not know the objects that populate the environment, and therefore it does not know their specific properties and relations. Consider, for instance, a robot that has to move around and manipulate objects in a kitchen (tables, chairs, apples, etc.) without knowing which and how many objects are really in the room. In this setting, the exploitation of a planning domain is a compelling challenge for three main reasons. First, in realistic environments, it is unfeasible for the robot to acquire a complete/correct and sufficiently detailed description of the environment before starting to plan and execute actions towards the achievement of its goals. Second, a robotic agent usually has a first-person perspective and partial view of the environment (e.g., by an on-board camera), so that the only way to acquire symbolic knowledge suitable for planning is by executing actions, observing their effects through its sensors, and mapping the sensory data (e.g., raw images) in a symbolic state. Third, high level actions of the planning domain are not directly executable by the robot, and therefore they need to be compiled to low level actions executable in the environment by the agent actuators. For instance, given an object instance identified by a constant $c_0$, the action $\mathsf{goCloseTo}(c_0)$ is compiled into a sequence of robot movements and rotations, which follow the path provided by a path-planner, and moves the robot to the (nearest) location close to object $c_0$.

This paper proposes a framework for agents that incrementally instantiate a planning domain, specified in PDDL, by planning, acting, and sensing, in an unknown environment. At each time-point, the belief of the agent about the current state of the environment is represented by three components, namely: $(i)$ the set of objects currently known by the agent and their properties expressed with the predicates of the PDDL domain, e.g., $\mathsf{table}(c_0)$, $\mathsf{apple}(c_1)$, and $\mathsf{on}(c_1, c_0)$; $(ii)$ for each known object, a set of low-level features as perceived by the agent, e.g., visual features and positions of $c_0$ and $c_1$; $(iii)$ a set of global features associated to the current environment state, e.g., an occupancy map of the environment, and the current pose of the robot.

For this framework, we propose an online iterative algorithm, called OGAMUS (Online Grounding of Action Models in Unknown Situations), that allows an agent, equipped with a lifted PDDL planning domain, and placed in an unknown environment, to achieve a set of goals expressed in the language of its PDDL action model. The agent is initialized without prior knowledge about the environment where it has to operate, i.e., with the empty set of objects, the empty set of their properties, and all the points of the occupancy map set as traversable. OGAMUS attempts to achieve the goal by combining four main activities, namely: (i) *exploring the environment* to acquire the knowledge needed to achieve the goals; (ii) *abstracting the sensor information* obtained at every step into a symbolic state; (iii) performing *symbolic planning* in the abstract model grounded with the current beliefs and current abstract state; (iv) *executing* the planned abstract actions by compiling them into low-level operations suitable for the current state of the environment.

The main features of OGAMUS are the following. *Generality:* OGAMUS is able to deal with any goal that can be expressed by a (first-order) formula using the predicates of the PDDL domain. For instance the goal of putting "two apples on a table" can be specified by the formula $\exists x\, y\, z.\mathsf{on}(x, z) \wedge \mathsf{on}(y, z) \wedge \mathsf{apple}(x) \wedge \mathsf{apple}(y) \wedge \mathsf{table}(z) \wedge x \neq y$. Notice that goals are expressed with existentially quantified variables; this is because, initially, the agent is not aware of any objects in the domain. An important step, necessary to achieve a goal containing existential variables, concerns discovering the object instances of the proper types (apple and table in the previous example) for instantiating all the existential variables. *Explainability*: The behaviour of the agent, its plans, and the effects of actions are represented at a symbolic level in which the states of the PDDL domain are derived at every step by abstracting the sensory data. *Robustness:* The action model, the obtained symbolic state representation, and the action compilation are not required to be fault free. As experimentally shown in this paper, OGAMUS achieves high success rate even with low precision object detectors and classifiers.

We have implemented and experimentally evaluated OGAMUS in the ITHOR (Kolve et al. 2017) and ROBOTHOR (Deitke et al. 2020) simulated photo-realistic environments for embodied AI. We evaluate OGAMUS on different tasks including "go close to an object of a certain type" (e.g., "go close to a fridge"). In the area of Embodied AI, this class of tasks is called "object goal navigation", and it is still considered a challenging and open problem. Our framework and evaluation go beyond such a challenge, considering more complex tasks, such as "open/close an object of a certain type" (e.g., "open a laptop"), and "put an object of a type on top of an object of another type" (e.g., "put an apple on a table"). We compared our results with the state-of-the-art approaches on the object goal navigation task. In particular, we show that, in the benchmarks of the last ROBOTHOR object goal navigation challenge, our approach outperforms the state of the art methods based on Reinforcement Learning (RL).[1] The evaluation on the other tasks leads to very positive performance, but we cannot compare them with any state-of-the-art approach since, to the best of our knowledge, we are the first solving such types of tasks in the ITHOR and ROBOTHOR environment.

The paper is structured as follows: firstly we analyze the related literature, then we describe the framework and the algorithm adopted by the agent to reach a specific goal in an unknown environment; finally, we present the experimental evaluation and a comparison with RL-based approaches.

## 2 Related Work

The problem of integrating symbolic action models with low level sensory data and actions has been addressed by different approaches. Most of them are based on RL techniques. Lyu et al. (2019) propose a framework, called SDRL, which combines symbolic planning and Deep RL to learn policies that compile high level actions into low level operations. SDRL assumes that a grounded domain model is provided in input and never updated. A fundamental difference w.r.t. our approach is that OGAMUS, instead, learns how to ground the domain model with new objects discovered online. Moreover, SDRL assumes a perfect oracle that maps low level perceptions into symbolic states, while OGAMUS deals with faulty mappings without assuming to have an oracle.

NSRL (Ma et al. 2021) represents abstract domains in first-order logic and uses RL to learn high level policies. NSRL generates a compact representation of the learned policies as a set of rules via Inductive Logic Programming. Similarly to SDRL, NSRL assumes a given and fixed abstract domain instantiation and a perfect mapping from sensory data to symbolic states, while OGAMUS overcomes these assumptions.

DPDL (Kase et al. 2020) represents abstract domains in PDDL. It learns online both mappings from sensory data to symbolic states and low level policies for high level actions. In OGAMUS, instead, the mapping from perceptions to symbolic states is obtained by combining a set of neural networks trained off-line. Moreover some of the high level actions are pre-compiled in low level operations (e.g., pickup an object at a given position), while policies for moving actions are computed online via path planning. As the other methods mentioned above, DPDL assumes a given and fixed grounded PDDL domain, while this is not the case for OGAMUS. Moreover, it focuses on performing manipulation tasks in a single scene type (i.e., a kitchen). OGAMUS, instead, can work on different scenes (we evaluate it on 35 different scenes grouped by kitchens, living-rooms, bedrooms, and bathrooms). Finally, DPDL works with an external fixed camera, while OGAMUS uses egocentric and dynamic views. This makes the tasks more difficult for OGAMUS, since the agent needs to navigate, explore the environment, and find new objects outside of its current view.

Differently from the above mentioned works, in the approach proposed by Garnelo, Arulkumaran, and Shanahan (2016), like in OGAMUS, the agent instantiate the abstract domain online by augmenting the set of objects every time it discovers new ones. The states of the instantiated abstract model is represented with a set of propositional atoms on the current set of constants. However, this approach is evaluated only with an extremely simple environment, while OGAMUS is tested in accurate and photo-realistic simulated environments with complex objects and egocentric views. Furthermore, the approach in (Garnelo, Arulkumaran, and Shanahan 2016) does not take advantage of the power of symbolic planning techniques on PDDL domain descriptions, and it does not generalize over different tasks. On the contrary, OGAMUS exploits symbolic planning which allows to accomplish variegated tasks that can be specified using PDDL.

(Lamanna et al. 2021a) proposes a method for learning an extensional representation of a discrete planning domain from continuous observations. However, it focuses on discovering states as atoms, while in OGAMUS states are structured entities composed of object instances, with properties and relationships between them. In (Lamanna et al. 2021b),

[1] https://ai2thor.allenai.org/

the same authors propose an approach that deals with action schema learning, a different but related problem w.r.t. action schema instantiation. In OGAMUS, the action schema is known in advance, and we focus on mapping continuous perceptions into symbolic states, while (Lamanna et al. 2021b) does not address the problem of inferring high-level states from perceptions, i.e., it assumes to directly observe the truth value of the current state atoms.

We experimentally evaluate OGAMUS on the Object Goal Navigation task that has recently received much attention in the embodied AI community (Mirowski et al. 2017; Savva et al. 2017; Fang et al. 2019; Mousavian et al. 2019; Campari et al. 2020; Wortsman et al. 2019; Chaplot et al. 2019; Chaplot et al. 2020; Ye et al. 2021). We experimentally show that, in the benchmark of the last ROBOTHOR object goal navigation challenge, OGAMUS performs better than a method based on DD-PPO (Wijmans et al. 2019), which won the challenge using pure RL based on low-level features, without exploiting a symbolic domain.

# 3 The Framework

In this section, we start by introducing the basic definitions of our reference framework. We then introduce the OGAMUS algorithm. The section closes with the description of what are the basic components that need to be specified in order to cope with new abstract predicates or actions of the PDDL domain.

## 3.1 Preliminary Definitions

Let $\mathcal{P}$ be a set of first order predicates, $\mathcal{V}$ a set of variables (also called parameters), and $\mathcal{C}$ a set of constants. We use $\mathcal{P}(\mathcal{V})$ to denote the set of atoms $P(x_1, \dots, x_m)$, where $x_i \in \mathcal{V}$ and $P \in \mathcal{P}$, and $\mathcal{P}(\mathcal{C})$ to denote the set of atoms obtained by grounding $\mathcal{P}(\mathcal{V})$ with the constants in $\mathcal{C}$.

**Definition 1** (Action model). *Given a set of operators $\mathcal{O}$, an* action model $\mathcal{M}$ *associates to each $op \in \mathcal{O}$ an* action schema*, which is a tuple $\langle \mathsf{par}(op), \mathsf{pre}(op), \mathsf{eff}^+(op), \mathsf{eff}^-(op)\rangle$, where $\mathsf{par}(op) \subseteq \mathcal{V}$, $\mathsf{pre}(op)$, $\mathsf{eff}^+(op)$, and $\mathsf{eff}^-(op)$ are subsets of $\mathcal{P}(\mathsf{par}(op))$.*

**Definition 2** (Ground action). *The* ground action *$op(\boldsymbol{c})$ of an operator $op \in \mathcal{O}$ with $\boldsymbol{c} = \langle c_1, \dots, c_n\rangle$ constants in $\mathcal{C}$ is the tuple $\langle \mathsf{pre}(op(\boldsymbol{c})), \mathsf{eff}^+(op(\boldsymbol{c})), \mathsf{eff}^-(op(\boldsymbol{c}))\rangle$, obtained by instantiating the atoms of $\mathsf{pre}(op)$, $\mathsf{eff}^+(op)$, and $\mathsf{eff}^-(op)$ with $\boldsymbol{c}$.*

**Definition 3** (Planning problem). *A* planning problem *is a tuple $\langle \mathcal{M}, \mathcal{C}, s_0, \mathcal{G}\rangle$ where $\mathcal{M}$ is an action model, $\mathcal{C}$ is a (possibly empty) set of constants, $s_0 \subseteq \mathcal{P}(\mathcal{C})$ is the initial state, and $\mathcal{G}$ is a first order formula over $\mathcal{P}$, $\mathcal{V}$ and $\mathcal{C}$.*

**Definition 4** (Plan). *A* plan *for a planning problem $\langle \mathcal{M}, \mathcal{C}, s_0, \mathcal{G}\rangle$ is a sequence $\langle op_1(\boldsymbol{c}_1), \dots, op_n(\boldsymbol{c}_n)\rangle$ such that there is a sequence $\langle s_1, \dots, s_n\rangle$ of subsets of $\mathcal{P}(\mathcal{C})$ (aka states), such that for every $0 \leq i < n$, $\mathsf{pre}(op_i(\boldsymbol{c}_i)) \subseteq s_i$, $s_i = s_{i-1} \cup \mathsf{eff}^+(op_i(\boldsymbol{c}_i)) \setminus \mathsf{eff}^-(op_i(\boldsymbol{c}_i))$, and $s_n \models \mathcal{G}$.*

Notice that our definition of planning problem allows to express first-order goal formula $\mathcal{G}$. We say that a state $s \models \mathcal{G}$ iff $\bigwedge_{P(\boldsymbol{c}) \in s} P(\boldsymbol{c}) \wedge \bigwedge_{P(\boldsymbol{c}) \in \mathcal{P}(\mathcal{C}) \setminus s} \neg P(\boldsymbol{c}) \models \mathcal{G}$, under the assumption that all the elements of the problem are in $\mathcal{C}$.

In order to use an abstract model, an agent needs to *anchor* the symbols occurring in the states of the planning domain with the real-world perceptions, and to map abstract actions into actions executable in the real world (Coradeschi and Saffiotti 2003). We suppose that the agent can *partially* observe the current state of the environment through a set of sensors, for instance images provided by an RGB-D camera, which do not directly correspond to the states of the abstract model. Furthermore, the set of sensors provide only a partial and subjective view of the environment. For instance, the RGB-D camera provides only an egocentric view of a portion of the room visible by the agent. We also suppose that the agent interacts with the environment by executing low-level operations (e.g., move 25 cm forward, rotate 30° left, pick up or put down an object at the GPS-coordinates $(x, y, z)$), which are different from the actions in the abstract action model. We need therefore to link the abstract state to real perceptions, and the abstract actions to operations executable by the actuators of the agent. Let us first consider the relationship between abstract states and perceptions.

**Object and state anchoring.** Every object that the agent is aware of at a given instant is represented by a constant $c \in \mathcal{C}$ that is the internal identifier for such an object. Following the approaches to symbol anchoring proposed in the literature (Coradeschi and Saffiotti 2003; Persson et al. 2019), every constant $c \in \mathcal{C}$ is associated with a tuple of numeric features denoted by $\boldsymbol{z}_c$. For instance, $\boldsymbol{z}_c$ might include the estimated position of $c$ and a set of visual features of the different views of $c$. In addition, for each state $s$ determined by the agent, we have a vector of state features $\boldsymbol{z}_s$, consisting of the 3D position of the agent in the environment, the orientation of the agent relative to its initial pose, the information about the success of the last low-level operation made by the agent, and an occupancy map of the environment. The occupancy map is a 2D map of the environment storing the areas that are believed to be traversable by the robot. The occupancy map is initialized so that every point is traversable.

**Predicate predictors.** In order to map the perceptions about objects into atoms of the symbolic state, the agent associates to every predicate a probabilistic model, e.g., a neural network, that computes the probability of a certain atom $P(\boldsymbol{c})$ to be true given the features associated to $\boldsymbol{c}$ and the current state ones, i.e., $Pr(Y_{P(\boldsymbol{c})} = \mathrm{True} \mid \boldsymbol{z}_{\boldsymbol{c}}, \boldsymbol{z}_s)$, where $Y_{P(\boldsymbol{c})}$ is a boolean random variable associated to the atom $P(\boldsymbol{c})$. These probabilistic models can be updated during execution on the basis of new observations. In this paper, however, we suppose that these probabilistic models are given (e.g., a pre-trained neural network), and they are not modified during execution.

We call *belief state* the agent's knowledge about object/state anchoring and predicate predictors.

**Definition 5.** *An agent belief state is a 5-tuple $\langle \mathcal{C}, \boldsymbol{z}_{\mathcal{C}}, s, \boldsymbol{z}_s, \mathbf{Pr}\rangle$ where:*

- *$\mathcal{C}$ is a set of constants;*

**Algorithm 1** OGAMUS algorithm

```
Input: $\mathcal{M}$, $\mathcal{G}$, $\mathbf{Pr}$ and MAXITER $\in \mathbb{N}$.
Output: SUCCESS/FAIL
 1: $\langle \mathcal{C}, \boldsymbol{z}_{\mathcal{C}}, s, \boldsymbol{z}_s \rangle \leftarrow \langle \emptyset, \emptyset, \emptyset, (\mathbf{0}, nil, \emptyset) \rangle$
 2: for $1 = 0$ to MAXITER do
 3:     if $s \models \mathcal{G}$ then
 4:         return SUCCESS
 5:     end if
 6:     $\pi \leftarrow$ PLAN($\mathcal{M}, \mathcal{C}, s, \mathcal{G}$)
 7:     if $\pi =$ NONE then
 8:         $\boldsymbol{e} \leftarrow$ EXPLORE($\boldsymbol{z}_s$)
 9:     else
10:         $op(\boldsymbol{c}) \leftarrow$ POP($\pi$)
11:         $\boldsymbol{e} \leftarrow$ COMPILE($op(\boldsymbol{c}), \boldsymbol{z}_{\boldsymbol{c}}, \boldsymbol{z}_s$)
12:     end if
13:     $e_1 \leftarrow Pop(\boldsymbol{e})$
14:     $\boldsymbol{x} \leftarrow$ EXEC($e_1$)
15:     $\boldsymbol{z}_s \leftarrow$ GETSTATEFEATURES($\boldsymbol{x}$)
16:     $\mathcal{C}_{\boldsymbol{x}}, \boldsymbol{z}_{\mathcal{C}_{\boldsymbol{x}}} \leftarrow$ GETOBJS($\boldsymbol{x}$)
17:     $\mathcal{C}, \boldsymbol{z}_{\mathcal{C}} \leftarrow$ UPDATEOBJS($\mathcal{C}, \boldsymbol{z}_{\mathcal{C}}, \mathcal{C}_{\boldsymbol{x}}, \boldsymbol{z}_{\mathcal{C}_{\boldsymbol{x}}}$)
18:     $Pr(\mathbf{Y}_{\mathcal{P}(\mathcal{C})}) \leftarrow$ PREDICTSTATE($\boldsymbol{z}_{\mathcal{C}}, s, \boldsymbol{z}_s$)
19:     $s \leftarrow \{p(\boldsymbol{c}) \in \mathcal{P}(\mathcal{C}) \mid Pr(Y_{p(\boldsymbol{c})} = True \mid \boldsymbol{z}_{\boldsymbol{c}}) > 1 - \epsilon\}$
20:     if $\pi \neq$ NONE and SUCCEED($op(\boldsymbol{c})$) then
21:         $s \leftarrow s \cup \mathsf{eff}^{+}(op(\boldsymbol{c})) \setminus \mathsf{eff}^{-}(op(\boldsymbol{c})))$
22:     end if
23: end for
24: return FAIL
```

- $\boldsymbol{z}_{\mathcal{C}} = \{\boldsymbol{z}_c\}_{c \in \mathcal{C}}$ *is a set of object feature vectors* $\boldsymbol{z}_c$*;*
- $s \subseteq \mathcal{P}(\mathcal{C})$ *is the set of atoms that are believed to be true;*
- $\boldsymbol{z}_s$ *is a vector of state features;*
- $\mathbf{Pr} = \{Pr(Y_{P(\boldsymbol{c})} \mid \boldsymbol{z}_{\boldsymbol{c}}, \boldsymbol{z}_s)\}_{P \in \mathcal{P}}$ *is the set of probabilistic models used to predict the truth value of* $P(\boldsymbol{c})$ *given the features* $\boldsymbol{z}_s$ *and* $\boldsymbol{z}_{\boldsymbol{c}}$ *associated with the constants in* $\boldsymbol{c}$*.*

### 3.2 The OGAMUS Algorithm

So far, we have not considered how the set $\mathcal{C}$ of constants identifying objects is obtained by the agent. We do not assume that they are given a priori to the agent; instead, we are interested in providing the agent with the capability to discover objects by adding new constants to the representation of the environment, updating the anchor to an object, merging two constants anchored to the same object, and deleting a constant from the representation that was erroneously identifying a non existing object in the environment.

Let $\boldsymbol{x}$ be the vector that contains the data returned by the sensors (i.e., the observations) at a given time; the agent extracts from $\boldsymbol{x}$ a set of objects $\mathcal{C}_{\boldsymbol{x}}$, and for each object $c \in \mathcal{C}_{\boldsymbol{x}}$ a feature vector $\boldsymbol{z}_c$. Since the agent can also recognize objects that it has already seen, it is possible that $\mathcal{C}_{\boldsymbol{x}} \cap \mathcal{C} \neq \emptyset$.

In the following, we shortly describe the OGAMUS algorithm (Algorithm 1).

- The algorithm takes as input an action model $\mathcal{M}$, a set $\mathbf{Pr}$ of probabilistic models for predicting the predicates in $\mathcal{P}$, a goal formula $\mathcal{G}$, and a maximum number of iterations. Notice that the goal $\mathcal{G}$ cannot contain constants, since we suppose that at the beginning the agent is not aware of any object. For instance, the goal requiring that an apple is inside a box can be encoded by the PDDL expression representing formula $\exists x, y\ \mathsf{apple}(x) \wedge \mathsf{box}(y) \wedge \mathsf{in}(x, y)$.

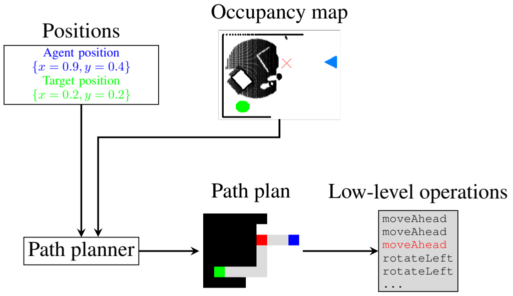

Figure 1: Example of EXPLORE($\boldsymbol{z}_s$). The occupancy map, agent position (in blue) and target position (in green) are given as input to a path planner which discretizes the occupancy map, computes a path plan, and compile the path plan into a sequence of low-level operations.

- The agent starts by initializing all the components of its state to the emptyset (line 1). We assume indeed that the agent is not aware of any object in the environment, therefore $\mathcal{C} = \emptyset$. Since $\mathcal{C}$ is empty, $\boldsymbol{z}_C$, $\mathcal{P}(\mathcal{C})$ and $s$ are also empty. The information in $\boldsymbol{z}_s$ representing the position and orientation of the agent is initialized with a vector of 0's; the information in $\boldsymbol{z}_s$ about the success of the last operation is set to *nil*; finally, the occupancy map of the environment in $\boldsymbol{z}_s$ is set to an empty map so that all the points are traversable.
- Then the agent iterates for a maximum number of steps, checking if the current state $s$ satisfies the goal (line 4); when this is the case, it returns SUCCESS.
- Otherwise, the agent invokes a planner (line 6) to solve the planning problem defined on the input action model, the current set of objects, the current state, and the input goal formula $\mathcal{G}$.
- If the planner does not find a plan that satisfies the goal, then the agent explores the environment in order to discover new objects that are needed to satisfy the goal. For instance, if the goal is to put an apple into a box, then the planner can find a plan only if in the current state $s$ there is at least one object of type apple and one of type box. For the exploration phase (line 8), the agent randomly selects a target position on the occupancy map (stored in $\boldsymbol{z}_s$) that it believes to be free from other obstacles. As shown in Figure 1, EXPLORE($\boldsymbol{z}_s$) calls a path planner that checks if such a position is reachable (if it is not reachable a new position is selected) and returns a sequence $\boldsymbol{e}$ of low-level navigation and rotation operations, which, according to the current knowledge of the agent, moves the agent from its current position to the selected target. For efficiency reasons, this path is computed in an approximated occupancy map obtained by discretizing the occupancy map through a grid. The execution of such a sequence of operations might fail due to the partial or incorrect knowledge

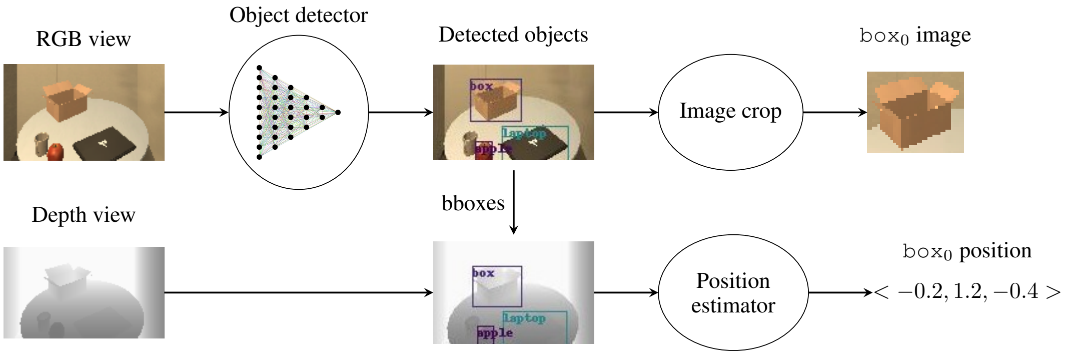

Figure 2: Example of object features extraction. The object detector takes as input the perception composed by the agent view RGB image, and returns a set of bounding boxes together with the detected object types. The object bounding boxes are used with the agent view depth image to estimate their positions (in meters w.r.t. the initial agent position). The lighter the pixels in the box, the farther the associated object.

of the agent, i.e., when the agent wrongly believes that a certain area on the path returned by the path-planner is traversable while there is an obstacle. In Figure 1, the agent fails to reach the red cell. Indeed, the execution of the third $\mathsf{moveAhead}$ action fails because the robot bumps into a table, whose size was only partially determined at the beginning of the exploration phase. The exploration terminates whenever the goal is reachable according to the learned problem. For example, consider the goal requiring that an apple is in a box and assume that an apple has already been discovered in the environment. The execution of the computed sequence of operations terminates in advance, when the agent detects the presence of a box on the table in Figure 1, as it approaches the table by executing the first operations in the sequence.

- If instead the planner succeeds and returns a valid plan $\pi$, then the first action of $\pi$ is compiled into a sequence of low-level operations $\boldsymbol{e}$ (line 11). The compilation of the action is based on the object and state features available in the agent's state. For instance, the high level action $\mathsf{pickup}(c)$ is compiled into the low-level operation $\mathsf{pickup}(x, y, z)$ where $(x, y, z)$ is the current (believed) position of object $c$, memorized in $\boldsymbol{z}_c$. The simulator executes such an operation by picking up what is present at these coordinates. To compile the action $\mathsf{goCloseTo}(c)$ instead, the agent calls a path-planner that provides a path from the current position of the agent (memorized in $\boldsymbol{z}_s$) to a position close to $c$.
- Successively, the first operation of sequence $\boldsymbol{e}$ is executed (line 14), and a new observation $\boldsymbol{x}$ is obtained. The execution of the first operation may fail or not. In both cases, the agent can acquire new knowledge (e.g., discover new objects or an obstacle), which can be used to produce a better compilation of an high-level action, and/or produce a better plan. Then, the new state features $\boldsymbol{z}_s$ are extracted from the sensory data $\boldsymbol{x}$ (line 15). The information about the occupancy map is updated using the information of success/failure of the action and the depth image.
- Then the agent runs an object detector (line 16) on the RGB image contained in observation $\boldsymbol{x}$ which returns a set of objects $\mathcal{C}_{\boldsymbol{x}}$, each associated with a vector of numeric features $\boldsymbol{z}_c$. These features include the bounding box, an estimation of the object position, and a vector of visual features extracted from the cropping of the image with the bounding box. Figure 2 gives an example of extraction of a number of objects, including a box, from the egocentric view of an agent robot, together with the bounding-box image of such a box and the estimate of its position.
- Next, at line 17, the agent merges the objects $\mathcal{C}_{\boldsymbol{x}}$ recognized in the current perception with the ones already known, i.e., $\mathcal{C}$. For every object $c' \in \mathcal{C}_{\boldsymbol{x}}$ there are two possible situations: $(i)$ $c'$ does not match with any object $c \in \mathcal{C}$, and therefore it is added to $\mathcal{C}$ with the corresponding features $\boldsymbol{z}_{c'}$; $(ii)$ $c'$ matches with a $c \in \mathcal{C}$; in this case the features $\boldsymbol{z}_c$ of $c$ are extended/updated with the features $\boldsymbol{z}_{c'}$. In the implementation, we use a very simple matching criteria which considers only the estimated position of the objects. Two objects are matched when their distance is less then a given threshold (set to 20cm). More sophisticated criteria can be adopted by defining a suitable distance measure between the entire set of object features. However, this simple criteria turned out to be sufficiently effective in our experiments.
- In line 18, the agent predicts the truth values of each atom in $\mathcal{P}(\mathcal{C})$ for the updated set of constants $\mathcal{C}$ by applying the predictors $\mathbf{Pr}$ on the features $\boldsymbol{z}_{\mathcal{C}}$. For predicate $\mathsf{closeToAgent}$, the prediction takes also as input the agent position in $\boldsymbol{z}_s$. All the atoms involving new or merged objects must be evaluated; the remaining atoms are evaluated only if the corresponding predictor takes as input some feature that has been updated after the execution of the last action. For instance, if the agent executes a move action, then all the atoms $\mathsf{closeToAgent}(c)$ for all $c \in \mathcal{C}$ must be evaluated. Each atom $\mathsf{closeToAgent}(c)$ is predicted true if the euclidean distance between the position of the object represented by $c$ in $z_c$ and the agent position in $z_s$ is lower than a given threshold (set to $140$ cm). When the action $\mathsf{open}(\mathsf{box}_0)$ is executed, the visual features of $\mathsf{box}_0$ probably change, and the truth value of predicate $\mathsf{isOpen}(\mathsf{box}_0)$ is predicted as depicted in Figure 3. Notice that, after executing an open operation it is

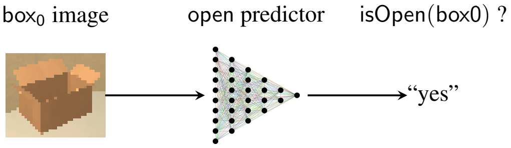

Figure 3: Example of predictor for the open predicate. The predictor takes as input the RGB images associated to box$_0$, and returns the predicted truth value of isOpen(box$_0$).

not guaranteed that the object will be open, as the action might fail.

- At line 19, the new state $s$ is created with all predicates $P(\boldsymbol{c})$ such that $Pr(Y_{P(\boldsymbol{c})} = True \mid \boldsymbol{z_c}, \boldsymbol{z_s})$ is higher than a given threshold $1 - \epsilon$ with $\epsilon \in [0, 1]$. Our approach does not assume to have access to the correct abstract state. Indeed, the agent can produce inconsistent states (e.g., a box is both on the table and on another box), or states that do not comply with action effects. Inconsistent states do not prevent OGAMUS to further plan and, whether a failure occurs, revise the agent's knowledge making them consistent. In the second case, the agent monitors the execution of high-level actions by comparing the state predicted by the PDDL model with the perceived state, and it solves inconsistencies in favour of the action effects in the model.
- At line 21, when SUCCEED$(op(\boldsymbol{c}))$ is true, i.e., the entire sequence $\boldsymbol{e}$ of operations compiling the first action $op(\boldsymbol{c})$ of $\pi$ is successfully executed, the state $s$ is updated according to the effects of $op(\boldsymbol{c})$.
- If the agent does not reach a state $s$ that satisfies the goal $\mathcal{G}$ after MAXITER steps, then the algorithm returns FAIL (line 24).

### 3.3 Knowledge Revision for New Tasks

To show the generality and the modularity of the proposed framework, we describe how it can be easily extended to cope with new tasks that can possibly involve a set of new (PDDL) actions, predicates, and object types. To allow the agent to accomplish a new task $t_{new}$, we firstly need to encode $t_{new}$ in a PDDL goal formula. If the encoding of $t_{new}$ does not require the introduction of new predicates, actions, or object types, then to solve the task it is sufficient to invoke OGAMUS with the goal formula encoding $t_{new}$. If, instead, the encoding of $t_{new}$ requires some new predicates, actions, or object types, then we have to provide the agent with the capability of (1) recognizing objects of the new types, (2) predicting the truth value of the new predicates, and (3) compiling the new actions in low-level operations executable by the agent's actuators. Consider, for instance, the case where $t_{new}$ is the task turning a lamp on: $t_{new}$ can be specified by the goal formula $\exists x.\mathsf{lamp}(x) \wedge \mathsf{turned\_on}(x)$, where lamp is a new object type and turned_on a new unary predicate.

To detect objects with the newly introduced type e.g., lamp, the object detector need to be extended and retrained with the new object type. This implies that the upper part of the detector (which is responsible of classifying the objects in their types) need to be extended with the new type and re-trained on a dataset containing also examples for this new type. The introduction of a new predicate, in our example turned_on, requires the deployment of a new classifier that predicts if the predicate holds for the objects detected from the sensory data. In case the predictor is based on a supervised learning model, then a training dataset with object labelled with positive and negative examples of the predicate need to be provided. Finally, if the new task requires adding new actions to the PDDL model, (e.g., to make the predicate $\mathsf{turned\_on}(x)$ true/false we need to introduce two new action turn_on and turn_off) we need to specify how the new action can be compiled into a sequence of low-level operations executable by the agent. For example, action $\mathsf{turn\_on}(c)$ will be compiled into a low-level operation $\mathsf{turn\_on}(x, y, z)$ where $(x, y, z)$ is the believed position of object $c$.

## 4 Experiments

We perform two sets of experiments. First, we experimentally evaluate OGAMUS with a simulated environment on four tasks that involves going close and move objects present in a number of rooms. Then, we compare OGAMUS with a state-of-the-art approach on the specific task of object goal navigation in different apartments.

### 4.1 Evaluating OGAMUS

The tasks and the corresponding goals on which we evaluate OGAMUS are the following:

1. Object goal navigation (OBJNAV $t_1$): given an object type $t_1$, the agent has to find, go close to, and look at an object of type $t_1$. For instance, the agent has to go close to an apple and look at it. The corresponding goal is $\exists x(\mathsf{Apple}(x) \wedge \mathsf{CloseToAgent}(x) \wedge \mathsf{Visible}(x))$. The agent is close to an object when the distance from the object is less than 1.5 meter.
2. Open/close an object (OPEN/CLOSE $t_1$): the agent is required to go close to an object of type $t_1$, look at it, and open/close it. For instance the agent has to open a drawer; the corresponding goal is $\exists x(\mathsf{Drawer}(x) \wedge \mathsf{Open}(x))$. In order to manipulate an object the agent need to be at a distance less than 1.5 meter.
3. Stack an object of type $t_1$ on an object of type $t_2$ (ON $t_1$ $t_2$): the agent has to find two objects of types $t_1$ and $t_2$ and put the one of type $t_1$ on top of the other of type $t_2$. For instance the agent has to put an apple on a table. The corresponding goal is: $\exists xy(\mathsf{Apple}(x) \wedge \mathsf{Table}(y) \wedge \mathsf{On}(x, y))$.

Since these tasks do not involve numerical resources or temporal constraints, we adopted propositional PDDL planning.

**Simulator.** We used the ITHOR (Kolve et al. 2017) simulator, an open source interactive environment for Embodied AI. ITHOR provides 120 different scenes, such as kitchens, living-rooms, bathrooms, and bedrooms, and allows a realistic simulation of the environment, including the physics of the objects. The scenes contain objects of 118 different types. The agent perceives the current state of the environment through an RGB-D on-board camera that provides a photo-realistic rendering of its egocentric view. The agent also perceives its position and orientation via a GPS and a compass (relative to the initial pose, which is unknown). The agent can navigate the environment by moving ahead of a given distance (set to 25cm), turn left or right, and look up or down of a given angle (set to 30°).[2] The agent can pick up objects, move them around, and change their state (e.g., a fridge can be opened or a laptop switched on).

For the object goal navigation task, we also considered a second simulator, ROBOTHOR (Deitke et al. 2020). ROBOTHOR is another simulation environment designed to develop embodied AI agents. Recently, ROBOTHOR hosted a competition that tackles an object goal navigation challenge; in our experiments, we also compared OGAMUS with the approaches that took part in the competition.

**Object detector.** As an object detector we used the Faster-RCNN model available in PyTorch 1.9 (Paszke et al. 2019), pre-trained on the COCO dataset (Lin et al. 2014) and fine-tuned on a self-generated dataset. In addition to the bounding box of the detected object, the object detector returns also the classification in one of the 118 classes. The object detector has been trained on a dataset composed by 69,095 training and validation images. The labeling of the dataset has been done by using the ground-truth provided by ITHOR. We tested it on 12,892 images obtaining a precision and recall of 50.99% and 65.18%, respectively.

**Predicate predictors.** For predicting predicate ON, we trained a feed-forward neural network (Svozil, Kvasnicka, and Pospichal 1997) with 244 input features composed by the bounding boxes coordinates of the two objects involved in the predicate relation and the 1-hot encoding of the two predicted classes returned by the object detector. For such a predicate, the training (and validation) sets is composed of 36,344 labelled pairs of objects. We evaluate the prediction of predicate ON on a test set composed of 8678 object pairs, obtaining 98.32% of both precision and recall. For predicting the unary predicate OPEN, we used a ResNet50 neural network (He et al. 2016) to extract features from the cropped object image, followed by a linear layer with input size 2048.[3] We trained it on 48,476 labelled examples, and test it on 9685 examples, obtaining 92.84% precision and 92.54% recall. The unary predicate CLOSETOAGENT, meaning that the agent is near to the object mentioned by the predicate, is computed directly from the features of the object. Specifically, we check if the distance between the agent position memorized in $\boldsymbol{z}_s$ and the object position memorized in the object feature vector is less than the manipulation distance, which is set to 1.5 meter in ITHOR and 1 meter in ROBOTHOR. Finally, we have to predict the equality predicate, i.e., when two objects $c$ and $d$ with features $\boldsymbol{z}_c$ and $\boldsymbol{z}_d$ represent the same object. To this purpose, we compute the distance between the two estimated object positions, and assign the object features to the same object instance whether such a distance is lower than a given threshold (set to 20 cm in our experiments). All the training, validation, and testing data have been extracted from a set of images collected by navigating in the ITHOR simulator.

| | *Success* ↑ | | *DTS* ↓ | | $P_{\mathcal{C}}$ ↑ | | $R_{\mathcal{C}}$ ↑ | | $P_{\mathcal{P}}$ ↑ | | $R_{\mathcal{P}}$ ↑ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\mathcal{C}$ | $\mathcal{C}_{GT}$ | $\mathcal{C}$ | $\mathcal{C}_{GT}$ | $\mathcal{C}$ | $\mathcal{C}_{GT}$ | $\mathcal{C}$ | $\mathcal{C}_{GT}$ | $\mathcal{C}$ | $\mathcal{C}_{GT}$ | $\mathcal{C}$ | $\mathcal{C}_{GT}$ |
| ON | 0.5 | 0.8 | 1 | 0.37 | 0.28 | 1 | 0.86 | 1 | 0.83 | 0.82 | 0.8 | 0.87 |
| OPEN | 0.75 | 0.87 | 0.45 | 0.25 | 0.35 | 1 | 0.78 | 1 | 0.82 | 0.81 | 0.72 | 0.82 |
| CLOSE | 0.78 | 0.89 | 0.39 | 0.16 | 0.32 | 1 | 0.8 | 1 | 0.8 | 0.79 | 0.73 | 0.82 |
| OBJNAV | 0.78 | 0.83 | 0.27 | 0.19 | 0.42 | 1 | 0.8 | 1 | 0.82 | 0.8 | 0.75 | 0.84 |

Table 1: Performance of OGAMUS with/out the ground-truth object detection, evaluated on the considered tasks in the ITHOR simulator. ↑/↓ means the higher/lower the better.

**Evaluation metrics.** The evaluation is provided by calculating a number of standard metrics over a set of episodes. For each task, an episode is obtained by randomly placing the agent in a random unseen scene and providing it a randomly generated goal for the given task. The generated goals are feasible since the object types used in their definitions are randomly chosen from a proper set of types; e.g., the goal to open a box is defined by randomly choosing the type "box" from a the set of object types that can be opened. For all the tasks we adopt the following standard evaluation metrics:

**Success rate (Success):** is equal to the fraction of successful episodes on the total number of episodes.

**Distance To Success (DTS):** For tasks (OBJNAV $t_1$), (OPEN $t_1$), and (CLOSE $t_1$), it is the average distance between the agent and the closest object of type $t_1$; for the task (ON $t_1$ $t_2$), it is the average distance between the closest pair of objects of types $t_1$ and $t_2$. If the episode succeeds such a distance is set to 0.

In order to measure the impact of errors in object detecting, for each task we consider two versions of OGAMUS. In a version the set of objects $\mathcal{C}$ are those returned by our object detector; in the second version the set of objects $\mathcal{C}_{GT}$ are those returned by the ITHOR simulator, which corresponds to a ground-truth object detector. Moreover, for all tasks, we evaluate the precision $P_{\mathcal{C}}$ and recall $R_{\mathcal{C}}$ of the detected objects, and the precision $P_{\mathcal{P}}$ and recall $R_{\mathcal{P}}$ of their predicate relations. $P_{\mathcal{P}}$ and $R_{\mathcal{P}}$ take into account only the objects that match with ground-truth ones. The matching is performed by computing the Intersection over Union (IoU) among the 2D bounding box detected during the episode and the ground-truth ones: if the IoU is higher than 50% for a ground-truth object of the same class, then the detected object matches with it.

**Experimental results.** In our experiments, a run of OGAMUS consists of 200 steps, where at each step a low-level operation is performed; we call each of these runs an episode. For all tasks, the episode dataset uses the *test* scenes of ITHOR, i.e., all environments that does not appear in the

[2] These settings are those indicated by the simulator developers for their proposed challenges.

[3] Further technical details about the hyper-parameters and datasets are available in the supplementary material.

datasets generated for training the predicate classifiers and object detector.

In Table 1, we report the average results of all tasks with and without ground-truth object detection over the considered episodes. For task on, we randomly generated 400 different goals, defining 400 episodes; for tasks open and close, we randomly generated 100 goals, defining 100 episodes for each task; for the object goal navigation task, we used the test set of goals proposed in (Wortsman et al. 2019), defining 2133 episodes. It is worth noting that, for the object goal navigation task, two different episodes often have the same goal but a different initial pose of the agent.

The impact of errors in object detecting for tasks objNav, open and close is pretty low and, as expected, it is the half of the impact for task on, since this latter task requires to detect two objects, while all other tasks requires to detect a single object. Without ground-truth object detection, Ogamus achieves the best success rate on the object goal navigation task; same or similar results are also provided in tasks open and close, since they can be seen as an extension of the object goal navigation task where, after finding and going near to an object, the agent has only to open or close the object. In the on task, the success rate decreases significantly, because it requires moving towards two objects, instead of only one, and has two additional complexities given by the facts that one object must be placed on the other one in a clear place, i.e., a place not obstructed by other objects, and that the total encumbrance of the agent increases when it carries an object, which causes more collisions during the navigation.

Metric $P_{\mathcal{C}}$ measures the amount of false positive object detections. Although the value of $P_{\mathcal{C}}$ is quite low for almost all the tasks, the success rate is relatively high because: (i) many false positive objects are not involved in the goals definition; (ii) the agent acts by using the objects with the highest confidence, which usually correspond to ground truth objects. $P_{\mathcal{C}}$ is higher for the object goal navigation task, because in this task the agent achieves the goal in fewer steps than for other tasks, and this reduces the number of predictions and the chance of detecting false positive objects.

Metric $R_{\mathcal{C}}$ measures the amount of true positive detected objects. The values for $R_{\mathcal{C}}$ are quite high, and hence the real existing objects are often detected, although in our experiments the agent sometimes fails to recognize objects when they are far from the agent. Moreover, the values of $P_{\mathcal{P}}$ and $R_{\mathcal{P}}$ are relatively high, and hence the agent can construct a symbolic state that is quite correct and complete, enabling an effective planning.

As expected, when Ogamus is provided with ground-truth object detection, all metrics are better than or similar to using our object detection. Only $P_{\mathcal{P}}$ is slightly lower when ground-truth object detection is used; we think this is due to the fact that sometimes the ground-truth object detection identifies objects which are only partially seen by the agent camera and predicting their properties more likely fails (e.g., the agent fails in predicting whether a fridge is open when it sees only a corner of the fridge).

| | *Success* ↑ | *SPL* ↑ |
|---|---|---|
| Random | 1.72% | 1.33% |
| DD-PPO | 35.11% | 17.37% |
| DD-PPO$_{boost}$ | 36.61% | 17.49% |
| Ogamus | **56.78%** | **24.87%** |

Table 2: Performance of Ogamus w.r.t. the random baseline, DD-PPO, and DD-PPO$_{boost}$, evaluated on the object goal navigation task in the RoboThor simulator.

## 4.2 Comparison on Object Goal Navigation

We did not find other approaches using simulator iThor that solve the tasks considered in our experiments. Therefore, in our experimental analysis we considered a second simulator, RoboThor (Deitke et al. 2020), for which the last challenge concerning the object goal navigation was launched in 2021.

For the object goal navigation task, we compared Ogamus with a random baseline, an RL baseline provided in the challenge, called DD-PPO, and the winner of the challenge, called DD-PPO$_{boost}$. Both the RL baseline and the winner exploit the DD-PPO algorithm (Wijmans et al. 2019) where the hidden state is computed by providing, as input to a GRU (Cho et al. 2014), the visual features of the RGB-D images computed by a ResNet-18 (He et al. 2016). The baseline and the winner approach have been trained on 108,000 episodes for 300 and about 10 million steps, respectively.

For this experiment, we adopt an additional metric, called Success weighted by Path Length (SPL) and introduced by Anderson et al. (2018). This metric measures the efficiency of the agent in reaching the goals and is defined as:

$$SPL = \frac{1}{N} \cdot \sum_{i=1}^{N} \left( s_i \cdot \frac{p_i^{\star}}{max(p_i, p_i^{\star})} \right)$$

where $N$ is the number of episodes, $p_i^{\star}$ is the shortest-path distance from the initial position of the agent to the closest goal in the $i$-th episode, $p_i$ is the length of the agent path in the $i$-th episode, and $s_i$ is a boolean variable equal to 1 when the $i$-th episode succeeds, and equal to 0 otherwise. If the path of the agent is the shortest one, the term in parenthesis is 1. The longer the path, the lower the term in parenthesis and the worse the metric.

For the experiment, we considered the validation episode dataset provided in the challenge, which is composed by 1800 episodes set in the 15 validation scenes of RoboThor. We did not consider the test episode dataset of the challenge, because for such a dataset the evaluation can be done only by the organizers of the challenge who require that the evaluated approach plays by the challenge rule. This is not the case for Ogamus because it allows the agent to perceive its pose, which is not available in the challenge. While the usage of this additional information can in principle favors Ogamus w.r.t. the approaches that took part in the challenge, it is worth noting that the agent position can be approximately derived from the RGB-D egocentric views by means of visual simultaneous localization and mapping methods (Taketomi, Uchiyama, and Ikeda 2017). Most importantly, the usage of the validation dataset of RoboThor

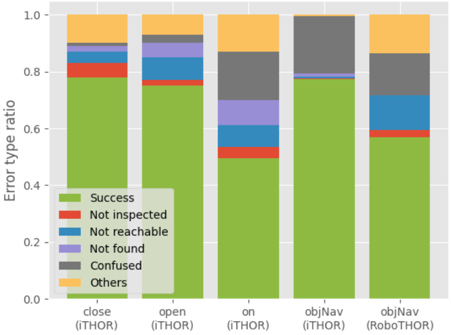

Figure 4: Ratio of the occurrences of different error types made by OGAMUS.

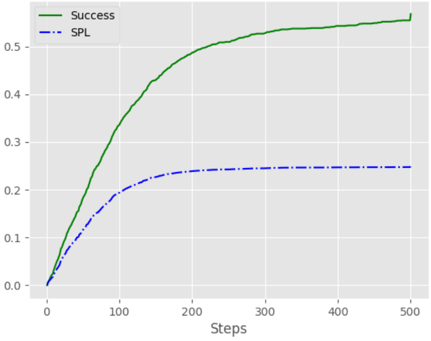

Figure 5: Average performance of OGAMUS for the goal object navigation task in the ROBOTHORsimulator, using a number of steps ranging from 0 to 500.

disfavors OGAMUS w.r.t. the other compared approaches because the object detector and predicate classifiers of OGAMUS are trained using the training and validation scenes of a different simulation environment, ITHOR, while the other compared approaches are trained and validated on the training and validation scenes of ROBOTHOR.

Each episode of the dataset consists of 500 steps, and regards finding and moving toward objects of 12 types. We trained an object detector similarly to the one for ITHOR simulator, but focused on the 12 goal object types of ROBOTHOR, which provides a performance slightly higher than the object detector trained using all the 118 object types of ITHOR, obtaining 59.02% precision and 69.06% recall.

Table 2 shows the results of the comparison. The random baseline provides poor performances. This indicates that, for the ROBOTHOR simulator, the object goal navigation task is quite challenging. The complexity of the task is confirmed by the performance of the RL baseline which is higher than the random baseline but still quite low. DD-PPO$_{boost}$ provides results slightly higher than the RL baseline. Remarkably, OGAMUS outperforms DD-PPO$_{boost}$ in terms of success rate and *SPL*. This confirms that the integration of symbolic planning with state recognition from sensory data can provide competitive results w.r.t. RL approaches.

## 4.3 Error Analysis

In Figure 4, we analyze the errors made by OGAMUS on all tasks. For few episodes, denoted as "Not inspected", the agent detects a far object of the same type as the type used for the goal definition, and subsequently approaches the object but is no more able to recognize it. This is due to the fact that either the object does not really exists, or the agent does not recognize an existing object, despite being close to and looking at it. For some episodes, namely "Not reachable", the agent finds a goal object but cannot reach a position close enough to the object. This can be due to the fact that either the agent collides or the goal object estimated position is farther than the real one. Collisions more often happen for the task ON, when the agent holds an object as the agent encumbrance increases. An error in the estimation of the object position is more likely for large objects, such as tables or televisions, since the agent considers the center of the object as its position. There are few episodes, labelled as "Not found", where the agent does not find the object, due to either an ineffective exploration of the environment or false negatives of the object detector. We observed that the latter case is more likely than the former, because the agent almost always explores the entire environment within the given number of steps. The errors labelled by "Confused" denote episodes for which the agent believes it succeeded while the task has not been completed. This is due to false positives of the object detector. Finally, "Others" denote all other task-dependent failures. E.g., for the ON, OPEN and CLOSE tasks, the agent sometimes fails to identify the object position when it has to manipulate an object. This more likely happens for small objects, such as spoons or saltshakers. Moreover, for the ON predicate an agent can fail to put an object on a table due to the fact that the target position is already occupied, or there is not enough space on the table.

Figure 5 shows the success rate and *SPL* for a number of steps ranging from 0 to 500. For almost all episodes the agent achieves the goal in 300 steps. For few episodes, the agent achieves the goal only after 500 steps. This happens because the agent is actually close to and looks at a goal object, but it fails to recognize the object.

# 5 Conclusions and Future Work

We have proposed a framework, called OGAMUS, for the online grounding of planning domains in unknown environments. Our approach enables an agent to map the sensory data into a symbolic state, allowing to perform and exploit efficient planning in a wide variety of different environments. We have tested the proposed method on different tasks obtaining better results than recent RL-based approaches. Future work will focus on learning a policy to compile the high-level actions into low-level executable operations, and on learning, online, the mapping of the sensory representations to symbolic ones.

## Acknowledgments

This work is partially supported by the EU ICT-48 2020 project TAILOR (No. 952215) and the EU H2020 project AIPlan4EU (No. 101016442).

## References

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

Campari, T.; Eccher, P.; Serafini, L.; and Ballan, L. 2020. Exploiting scene-specific features for object goal navigation. In *European Conference on Computer Vision*, 406–421. Springer.

Chaplot, D. S.; Gandhi, D.; Gupta, S.; Gupta, A.; and Salakhutdinov, R. 2019. Learning to explore using active neural slam. In *International Conference on Learning Representations*.

Chaplot, D. S.; Gandhi, D. P.; Gupta, A.; and Salakhutdinov, R. R. 2020. Object goal navigation using goal-oriented semantic exploration. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 4247–4258. Curran Associates, Inc.

Cho, K.; van Merriënboer, B.; Bahdanau, D.; and Bengio, Y. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 103–111.

Coradeschi, S., and Saffiotti, A. 2003. An introduction to the anchoring problem. *Robotics and autonomous systems* 43(2-3):85–96.

Deitke, M.; Han, W.; Herrasti, A.; Kembhavi, A.; Kolve, E.; Mottaghi, R.; Salvador, J.; Schwenk, D.; VanderBilt, E.; Wallingford, M.; Weihs, L.; Yatskar, M.; and Farhadi, A. 2020. RoboTHOR: An Open Simulation-to-Real Embodied AI Platform. In *CVPR*.

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 Conference on Computer Vision and Pattern Recognition*, 538–547.

Garnelo, M.; Arulkumaran, K.; and Shanahan, M. 2016. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

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*, 770–778.

Kase, K.; Paxton, C.; Mazhar, H.; Ogata, T.; and Fox, D. 2020. Transferable task execution from pixels through deep planning domain learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 10459–10465. IEEE.

Kolve, E.; Mottaghi, R.; Han, W.; VanderBilt, E.; Weihs, L.; Herrasti, A.; Gordon, D.; Zhu, Y.; Gupta, A.; and Farhadi, A. 2017. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*.

Lamanna, L.; Gerevini, A. E.; Saetti, A.; Serafini, L.; and Traverso, P. 2021a. On-line learning of planning domains from sensor data in pal: Scaling up to large state spaces. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11862–11869.

Lamanna, L.; Saetti, A.; Serafini, L.; Gerevini, A.; and Traverso, P. 2021b. Online learning of action models for pddl planning. In *IJCAI-2021*.

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*, 740–755. Springer.

Lyu, D.; Yang, F.; Liu, B.; and Gustafson, S. 2019. Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2970–2977.

Ma, Z.; Zhuang, Y.; Weng, P.; Zhuo, H. H.; Li, D.; Liu, W.; and Hao, J. 2021. Learning symbolic rules for interpretable deep reinforcement learning. *arXiv preprint arXiv:2103.08228*.

Mirowski, P.; Pascanu, R.; Viola, F.; Soyer, H.; Ballard, A. J.; Banino, A.; Denil, M.; Goroshin, R.; Sifre, L.; Kavukcuoglu, K.; et al. 2017. Learning to navigate in complex environments. *ICLR*.

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)*, 8846–8852. IEEE.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32:8026–8037.

Persson, A.; Dos Martires, P. Z.; De Raedt, L.; and Loutfi, A. 2019. Semantic relational object tracking. *IEEE Transactions on Cognitive and Developmental Systems* 12(1):84–97.

Savva, M.; Chang, A. X.; Dosovitskiy, A.; Funkhouser, T.; and Koltun, V. 2017. Minos: Multimodal indoor simulator for navigation in complex environments. *arXiv preprint arXiv:1712.03931*.

Svozil, D.; Kvasnicka, V.; and Pospichal, J. 1997. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems* 39(1):43–62.

Taketomi, T.; Uchiyama, H.; and Ikeda, S. 2017. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications* 9(1):1–11.

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. In *International Conference on Learning Representations*.

Wortsman, M.; Ehsani, K.; Rastegari, M.; Farhadi, A.; and Mottaghi, R. 2019. Learning to learn how to learn: Self-adaptive visual navigation using meta-learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 6750–6759.

Ye, J.; Batra, D.; Das, A.; and Wijmans, E. 2021. Auxiliary tasks and exploration enable objectnav. *arXiv preprint arXiv:2104.04112*.