Online Grounding of PDDL Domains by Acting and Sensing in Unknown Environments

Leonardo Lamanna, 1,2 Luciano Serafini, 1 Alessandro Saetti, 2 Alfonso Gerevini 2 and Paolo Traverso 1

1 Fondazione Bruno Kessler (FBK), Trento, Italy
2 Department of Information Engineering, University of Brescia, Italy

Abstract
To effectively use an abstract (PDDL) planning domain to achieve goals in an unknown environment, an agent must instantiate such a domain with the objects of the environment and their properties. If the agent has an egocentric and partial view of the environment, it needs to act, sense, and abstract the perceived data in the planning domain. Furthermore, the agent needs to compile the plans computed by a symbolic planner into low level actions executable by its actuators. This paper proposes a framework that aims to accomplish the aforementioned perspective and allows an agent to perform different tasks. For this purpose, we integrate machine learning models to abstract the sensory data, symbolic planning for goal achievement and path planning for navigation. We evaluate the proposed method in accurate simulated environments, where the sensors are RGB-D on-board camera, GPS and compass.

Introduction
An agent, in order to generate plans and achieve its goals, must instantiate a (PDDL) planning domain with the specific 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 actual objects in the environment and their specific properties. Consider, for instance, an agent 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 PDDL planning domain is a compelling challenge for three main reasons. First, in realistic environments, it is unfeasible for the agent 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, if the agent has a first-person perspective and partial view of the environment, the only way to acquire symbolic knowledge suitable for (PDDL) planning is by executing actions, observing their effects through its sensors, and mapping the low level sensed data (e.g., raw images) in a symbolic state. Third, high level actions of the planning domain are not directly executable by the agent, and therefore they need to be compiled to low level actions executable by the agent actuators. For instance, the PDDL action go closest to (table) is compiled into a sequence of agent movements and rotations which follow the path provided by a path-planner.

This paper proposes a framework that allows an agent to incrementally learn the instantiation of a planning domain by acting, sensing and planning in an unknown environment. The belief of the agent about the environment is represented by a structure with the following components: (i) the set of objects currently known by the agent and their properties expressed in PDDL, e.g., apple1, table2, and on(apple1, table2); (ii) a set of low-level features for each known object as perceived by the agent, e.g., visual features of the position of apple1 and table2; (iii) a set of global features associated to the environment state, e.g., the current knowledge about the map of a room. This structure is initially empty since we assume that the agent has no prior knowledge about the environment where it has to operate.

In this paper, we describe and experimentally evaluate the proposed framework considering the task of an agent that must move and manipulate a set of objects in a number of apartments. Examples of goals are “being close to a fridge”, “having a fridge open”, “having a pen on a desk” and “having a drawer closed”. The agent perceives the current state of the environment through a first-person RGB-D camera, and its position and orientation via a GPS and a compass. In order to abstract the sensory data into symbolic states usable by a PDDL planner, we use neural networks for the object detection/classification and relation classification. Regarding the action compilation, we use path planning for navigation, and a set of low-level actions for object manipulation, e.g. pickup(1) is compiled into pickup(, ) where is the belief position of object 1.

We propose an algorithm, called OGAMUS (Online Grounding of Action Models in Unknown Situations), which integrates model instantiation in unknown environments, state recognition, knowledge revision by action execution, symbolic planning with incomplete, incorrect and dynamic states, and online compilation of a set of symbolic actions into low-level operations executable by the agent’s actuators.

Some 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 “two apples are on a table” corresponds to formula 2xyz.on(x, z) on(y, z) apple(x) and

Abstract

GAMUS is the belief position of object 1.
apple(y) ∧ table(z) ∧ x ≠ y). 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 with four different classes of goals in the iTHOR (Kolve et al. 2017) and ROBOTHR (Deitke et al. 2020) simulated environments for embodied AI. We evaluate OGAMUS on tasks such as “go close to an object”, also called “object goal navigation”, moving an object on top of another, and opening/closing an object. We show that our approach outperforms the state of the art methods based on Reinforcement Learning (RL) on the object goal navigation task.

The paper is structured as follows: we firstly 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.

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 on PDDL 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; OGAMUS, instead, learns how to ground the domain model with new objects discovered at run time. Moreover, SDRL assumes a perfect oracle that maps low level perceptions into symbolic states, while OGAMUS deals with faulty mappings. 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. Similarity to SDRL, NSRL assumes a given and fixed abstract domain instantiation and a perfect mapping from sensory data to symbolic state. 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 that are trained off-line. Moreover some of the high level actions are pre-compiled in low level operations (e.g., pick-up an object at a given position), while policies for moving actions are computed on-line via path-planning. As the other methods mentioned above, DPDL assumes a given and fixed grounded PDDL domain. Moreover, it focuses on performing manipulation tasks in a single scene. OGAMUS, instead, can work on different scenes (we evaluate it on 35 different scenes). Finally, OGAMUS uses egocentric and dynamic views, while DPDL works with an external fixed camera.

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 constants every time the agent discovers new objects. 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 simulated environments with complex objects and egocentric images. 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.

Some works in the literature deal with action schema learning from input traces, a different but related problem w.r.t. action schema instantiation. Some approaches to action schema learning do not consider the low-level high dimensional sensory data (see e.g., (Aineto, Jimenez Celorio, and Omantia 2019; Bonet and Gefken 2020; Lamanna et al. 2021b)), while others (e.g., (Kurutach et al. 2018; Asai 2019; Lamanna et al. 2021a)) learn action models from traces composed of sequences of low-level data generated by action executions. The main difference with OGAMUS is that they learn action models, while OGAMUS focuses on solving goals in a wide class of environments by instantiating a lifted action model given in input. Moreover, in OGAMUS, the dimensionality of the ground domain is not fixed, but it is discovered by observing the environment. A further difference is that the above approaches assume that the low level actions coincide with the abstract actions and do not take into account the problem of producing low-level policies that realize high level actions.

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; 2020; Ye et al. 2021). We experimentally show that, for the ROBOTHR 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.

Framework
We start by introducing the main definitions of a symbolic planning framework. 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 \( P(x_1, \ldots, x_m) \), 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 \( M \) associates to each \( op \in \mathcal{O} \) an action schema, which is a tuple

\[ \text{https://ai2thor.allenai.org/} \]
Definition 2 (Ground action) The ground action \( op(c) \) of an operator \( op \in O \) with \( c = \{c_1, \ldots, c_n\} \) constants in \( C \) is the tuple \((pre(op(c)), eff^+(op(c)), eff^-(op(c))\) obtained by instantiating the atoms of \( pre(op) \), \( eff^+(op) \), and \( eff^-(op) \) with \( c \).

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

Definition 4 (Plan) A plan for a planning problem \((\mathcal{M}, C, s_0, \mathcal{G})\) is a sequence \((op_1(c_1), \ldots, op_n(c_n))\) such that there is a sequence \((s_1, \ldots, s_n)\) of subsets of \( \mathcal{P}(C) \) (aka states), such that for every \( 0 \leq i < n \), \( pre(op_i(c_i)) \subseteq s_i \), \( s_i = s_{i-1} \cup eff^+(op_i(c_i)) \setminus eff^-(op_i(c_i)) \), and \( s_n = \mathcal{G} \).

In order to use an abstract model, an agent needs to anchor symbols to real-world perceptions and agent physical actions (Coradeschi and Saffiotti 2003). Indeed, the agent can 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 sensors provide only a partial view of the environment. For instance, the RGB-D camera provides only an egocentric view of a portion of the room where the agent is located. Moreover, an 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. For every object that the agent is aware of at a given instant, there is a constant \( c \in C \) 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 \( c \in C \) is associated with a tuple of numeric features denoted by \( z_c \). For instance, \( z_c \) might include the estimated position of \( c \) and a set of visual features of the different views of \( c \). In addition, for each state \( s \) that has been recognized by the agent, we have a vector of state features \( z_s \), 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.

Predicate predictors. In order to map the perceptions about the 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 \( P(c) \) to be true given the features associated to \( c \) and the current state ones, i.e., \( Pr(Y_{P(c)} = True \mid z_c, z_s) \), where \( Y_{P(c)} \) is a boolean random variable associated to the atom \( P(c) \). 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 \((\mathcal{C}, z_c, s, z_s, Pr)\) where:

- \( \mathcal{C} \) is a set of constants;
- \( z_c = \{z_c\}_{c \in C} \) is a set of object feature vectors \( z_c \);
- \( s \subseteq \mathcal{P}(C) \) is the set of states that are believed to be true;
- \( z_s \) is a vector of state features;
- \( Pr = \{Pr(Y_{P(c)} | z_c, z_s)\}_{P \in P} \) is the set of probabilistic models used to predict the truth value of \( P(c) \) given the features \( z_c \) and \( z_s \) associated with the constants in \( c \).

So far, we have not considered how the set \( C \) of constants identifying objects is incrementally constructed by the agent. Indeed, we are interested in modelling the capability of an agent to discover new objects, update the anchor to an object, merge two constants anchored to the same object, and delete a constant that was erroneously identifying a non-existing object.

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

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

- The algorithm takes as input an action model \( \mathcal{M} \), a set \( Pr \) 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. The main objective of the algorithm is to reach a state that satisfies the goal.
- The agent starts by initializing all the components of its state to the emptysset (line 1). We assume indeed that the agent is not aware of any object in the environment, therefore \( C = \emptyset \), and \( z_c \) is also empty. Since \( C \) is empty, \( P(C) \) and \( s \) are also empty. The information in \( z_s \) representing the position and orientation of the agent is initialized with a vector of zeros; the information in \( z_s \) about the success of the last operation is set to nil; finally, the occupancy map of the environment in \( z_s \) is also set to empty, i.e., all the positions are freely explorables.
- 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 \( \mathcal{G} \).
Algorithm 1: O\textsc{gamus} algorithm

\textbf{Input:} \textit{M}, \textit{G}, \textit{Pr} and \textsc{MaxIter} \in \mathbb{N}.

\textbf{Output:} SUCCESS/FAIL.

1: \langle \mathcal{C}, \mathcal{Z}^c, s, z_s \rangle \leftarrow \langle \emptyset, \emptyset, \emptyset, (0, \text{nil}, \emptyset) \rangle
2: \textbf{for} \textit{i} = 0 \textbf{ to } \textsc{MaxIter} \textbf{ do}
3: \quad \textbf{if} \ s \models \mathcal{G} \textbf{ then}
4: \quad \quad \textbf{return} \text{ SUCCESS}
5: \quad \textbf{end if}
6: \quad \pi \leftarrow \text{PLAN}(\mathcal{M}, \mathcal{C}, s, \mathcal{G})
7: \quad \textbf{if} \ \pi = \text{None} \textbf{ then}
8: \quad \quad e \leftarrow \text{EXPLORE}(z_s)
9: \quad \textbf{else}
10: \quad \quad \text{op}(e) \leftarrow \text{POP}(\pi)
11: \quad \quad e \leftarrow \text{COMPILE}(\text{op}(e), z_c, z_s)
12: \quad \textbf{end if}
13: \quad e_1 \leftarrow \text{POP}(e)
14: \quad x \leftarrow \text{EXEC}(e_1)
15: \quad z_s \leftarrow \text{GETSTATEFEATURES}(x)
16: \quad \mathcal{C}_x, z_{c_x} \leftarrow \text{GETOBS}(x)
17: \quad \mathcal{C}, z_c \leftarrow \text{UPDATEOBS}(\mathcal{C}, z_c, \mathcal{C}_x, z_{c_x})
18: \quad \text{Pr}(\mathcal{Y}_{P(c)}) \leftarrow \text{PREDSTATE}(z_c, s, z_s)
19: \quad s \leftarrow \{ p(e) \in \mathcal{P}(\mathcal{C}) \mid \text{Pr}(\mathcal{Y}_{P(c)}) = \text{True} \mid z_{c} > 1\epsilon \}
20: \quad \textbf{if} \ \pi \neq \text{None} \text{ and } \text{SUCCEED}((\text{op}(e)), z_c, z_s) \textbf{ then}
21: \quad \quad s \leftarrow s \cup \text{eff}^+(\text{op}(e)) \setminus \text{eff}^-(\text{op}(e))
22: \quad \textbf{end if}
23: \quad \textbf{end for}
24: \quad \textbf{return} \text{ FAIL}

\begin{itemize}
  \item 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 \textit{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 \textit{z}_s) that it believes to be reachable. \text{EXPLORE}(z_s) calls a path planner that returns a sequence \textit{e} of low-level navigation and rotation operations, which, according to the current knowledge of the agent, moves the agent from the current position to the selected target. Clearly, such a path might fail due to the partial knowledge of the agent.
  \item If instead the planner succeeds and returns a valid plan \textit{\pi}, then the first action of \textit{\pi} is compiled into a sequence of low-level operations \textit{e}. 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 \text{pickup}(c) is compiled into the low-level operation \text{pickup}(x, y, z) where \textit{(x, y, z)} is the current (believed) position of object \textit{c}, memorized in \textit{z}_c; to compile the action \text{goclose}(c) instead, the agent calls a path-planner that provides a path from the current position of the agent (memorized in \textit{z}_s) to a position close to \textit{c}.
  \item Successively the first operation of sequence \textit{e} is executed (line 13), and a new observation \textit{x} is obtained. Then, the new state features \textit{z}_s are extracted from the sensory data \textit{x} (line 15). The information about the occupancy map is updated using the information of success/failure of the action and the depth image.
  \item At the end of the action, the planner can find a plan only if in the current state \textit{i} there is a new object \textit{c} that is needed to satisfy the goal. For instance, if the goal is to put an apple into a box, 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 agent explores the environment in order to discover new objects that are needed to satisfy the goal.
\end{itemize}
Simulator. We used the simulator, an open source interactive environment for Embodied AI. iThor provides 120 different scenes, including kitchens, living-rooms, bathrooms, and bedrooms. The scenes contain objects of 118 different types. The agent perceives the current state of the environment through an RGB-D camera that provides a photo-realistic rendering of the egocentric view of the agent. The agent also perceives its relative position and orientation via a GPS and a compass.

1. To an apple and look at it. The corresponding goal is \( \exists x (\text{Apple}(x) \land \text{CloseToAgent}(x) \land \text{Visible}(x)) \). 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 \( \exists x (\text{Drawer}(x) \land \text{Open}(x)) \). 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 the top of the other of type \( t_2 \). For instance the agent has to put an apple on a table. The corresponding goal is: \( \exists xy (\text{Apple}(x) \land \text{Table}(y) \land \text{On}(x, y)) \).

**Simulator.** We used the iThor simulator, an open source interactive environment for Embodied AI. iThor provides 120 different scenes, including kitchens, living-rooms, bathrooms, and bedrooms. The scenes contain objects of 118 different types. The agent perceives the current state of the environment through an RGB-D camera that provides a photo-realistic rendering of the egocentric view of the agent. The agent also perceives its relative position and orientation via a GPS and a compass (relative to the starting pose, which is unknown). The agent also perceives its egocentric view of the agent. The agent also perceives its egocentric view of the agent. 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\(^\circ\)). 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).

**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. 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 close, 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 \( z_s \) 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 \( z_c \) and \( z_d \) 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.

**Evaluation metrics.** The evaluation is provided by calculating a number of 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 a given task. For all the tasks we adopt the following 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 detection, for each task we consider two versions of OGame. In a version the set of objects \( C \) are those returned by our object detector; in the second version the set of objects \( C_{GT} \) are those returned by the iThor simulator, which corresponds to a ground-truth object detector. Moreover, for all tasks, we evaluate the precision \( P_C \) and recall \( R_C \) of the detected objects, and the precision \( P_P \) and recall \( R_P \) of their predicate relations. It is worth noting that \( P_P \) and \( R_P \) take into account only the objects that match with ground-truth ones. At each iteration, the matching is performed by computing the Intersection over Union (IoU) among the detected 2D bounding box 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 OGame 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 datasets generated for training the predicate classifiers and object detector.

\[ \exists t (\text{Apple}(t) \land \text{CloseToAgent}(t) \land \text{Visible}(t)) \] 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 close, 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 \( z_s \) 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 \( z_c \) and \( z_d \) 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. **Evaluation metrics.** The evaluation is provided by calculating a number of 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 a given task. For all the tasks we adopt the following 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 detection, for each task we consider two versions of OGame. In a version the set of objects \( C \) are those returned by our object detector; in the second version the set of objects \( C_{GT} \) are those returned by the iThor simulator, which corresponds to a ground-truth object detector. Moreover, for all tasks, we evaluate the precision \( P_C \) and recall \( R_C \) of the detected objects, and the precision \( P_P \) and recall \( R_P \) of their predicate relations. It is worth noting that \( P_P \) and \( R_P \) take into account only the objects that match with ground-truth ones. At each iteration, the matching is performed by computing the Intersection over Union (IoU) among the detected 2D bounding box 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 OGame 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 datasets generated for training the predicate classifiers and object detector.

- **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 detection, for each task we consider two versions of OGame. In a version the set of objects \( C \) are those returned by our object detector; in the second version the set of objects \( C_{GT} \) are those returned by the iThor simulator, which corresponds to a ground-truth object detector. Moreover, for all tasks, we evaluate the precision \( P_C \) and recall \( R_C \) of the detected objects, and the precision \( P_P \) and recall \( R_P \) of their predicate relations. It is worth noting that \( P_P \) and \( R_P \) take into account only the objects that match with ground-truth ones. At each iteration, the matching is performed by computing the Intersection over Union (IoU) among the detected 2D bounding box 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 OGame 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 datasets generated for training the predicate classifiers and object detector.

1These settings are those indicated by the simulator developers for their proposed challenges.
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. 

OGAMUS without ground-truth object detection 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 $P_C$ measures the amount of false positives in detecting objects. Although the values of $P_C$ is quite low for almost all the considered tasks, the success rate is relatively high because (i) many false positive objects are not involved in the definition of goals, and (ii) the agent acts by using the objects with the highest confidence, which usually correspond to ground-truth objects. $P_C$ is higher for the object goal navigation task, because for 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 $R_C$ measures the amount of true positive detected objects. The values for $R_C$ 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 $P_P$ and $R_P$ 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 $P_P$ 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).

### Comparison on object goal navigation

We did not find other approaches using simulator iTHOR that solve the tasks considered in our experiments. So, 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$_{action\ boost}$. 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:

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

where $N$ is the number of episodes, $p^*_i$ 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.

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 since 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 images by means of visual simultaneous localization and mapping methods (Taketomi, Uchiyama, and Ikeda 2020).
Most importantly, the usage of the validation dataset of ROBOTHOR 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 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 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 action boost provides results slightly higher than the RL baseline. Remarkably, OGAMUS outperforms DD-PPO action 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 based approaches.

Ablation study and error analysis

In Figure 1 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, 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.

Figure 2 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.

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.
References

Aineto, D.; Jiménez Celorrio, S.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artif. Intell.*, 275: 104 – 137.

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*.

Asai, M. 2019. Unsupervised Grounding of Plannable First-Order Logic Representation from Images. In *ICAPS*.

Bonet, B.; and Geffner, H. 2020. Learning First-Order Symbolic Representations for Planning from the Structure of the State Space. In *ECAI*.

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 Merriënoord, 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.; VaderBilt, 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*.

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.; VaderBilt, 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*.

Kurutach, H.; Tamar, A.; Yang, G.; Russell, S.; and Abbeel, P. 2018. Learning Plannable Representations with Causal InfoGAN. In *NIPS*.

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.; Košeká, 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*. 