KB4RL: Towards a Knowledge Base for automatic creation of State and Action Spaces for Reinforcement Learning

Lobna Joualy[1,*], Eric Demeester[2] and Nikolaos Tsiogkas[1]

[1]*Department of Computer Science, KU Leuven, 3590 Diepenbeek, Belgium*
[2]*Department of Mechanical Engineering, KU Leuven, 3590 Diepenbeek, Belgium*

Abstract
Reinforcement Learning (RL) is a promising approach for creating adaptive solutions for robotic tasks that are difficult to design directly. Unlike traditional approaches that rely on designing explicit behaviors, RL allows agents to learn skills by interacting with their environment. Inspired by human learning processes, agents acquire knowledge through trial and error guided by the rewards gained from their actions through various experiences. A key step when solving planning problems with RL algorithms is the definition of state and action spaces. Typically, state and action spaces are manually designed in a domain-specific manner to ensure effective problem solving. In this paper, we present an innovative approach to define state and action spaces through a *Knowledge Base* (KB) encoding information about the environment and physics. We propose a structured and extensible form for knowledge description, and we show how to extend the action space through action learning. This allows the robot to reach its goal in those states where unexpected scenarios are encountered and where, therefore, the action space obtained through the KB is not sufficient.

Keywords
Knowledge graphs, Reinforcement learning, Automatic state space, Automatic action space

1. Introduction

Reinforcement learning (RL) has emerged as a promising approach to create adaptive solutions to challenging robotic tasks that are difficult to tackle with conventional approaches, such as manipulation [1], navigation [2] and locomotion [3, 4]. Unlike traditional methodologies that rely on designing explicit behaviours [5, 6], RL allows agents to learn skills autonomously through interaction with the environment [7, 8, 9, 10]. This learning process, inspired by human cognition, involves the acquisition of skills through iterative trial and error, guided by the rewards gained from their actions. Leveraging various experiences, the agents can effectively explore complex environments and achieve desired goals.

A key step when solving planning problems with RL algorithms is the definition of state and action spaces, which are used by the agent to learn the desired policy. Typically, state and action spaces are manually designed in a domain-specific manner to ensure effective problem solving.

In our work, we present a novel approach in which the definition of state and action spaces occurs through a *Knowledge Base* (KB) that encodes information about the environment and physics via *Resource Description Framework* (RDF) triples. We propose a structured and extensible form for the knowledge description, that allows the creation of new state and action spaces, based on structured information that is added to the KB, using specific prepared queries. Next, we introduce a solution for extending the action space to include unforeseen situations that the agent may encounter during the policy learning. The agent can learn new actions, which allow it to reach a successful state, by using low-level reinforcement learning based on motor commands. We define a successful state as a state in which the initial action space is sufficient to reach the goal, and for which a policy is therefore already known. Lastly, a final training is performed, this time considering both the original state and action space, as well as the newly learnt actions.

Workshop at ESWC '24: Actionable Knowledge Representation and Reasoning for Robots, May 27, 2024, Hersonissos, Crete, Greece
*Corresponding author.
lobna.joualy@kuleuven.be (L. Joualy); eric.demeester@kuleuven.be (E. Demeester); nikolaos.tsiogkas@kuleuven.be (N. Tsiogkas)
0000-0003-2408-4387 (L. Joualy); 0000-0001-6866-3802 (E. Demeester); 0000-0003-2842-7316 (N. Tsiogkas)

2. Preliminaries and related work

2.1. Knowledge Graphs

Knowledge Graphs (KG) are networks of interconnected data points, where each node represents a piece of information and each edge represents a connection between those pieces of information. They are a representation that is easy for humans as well as for machines to understand, and they can provide new inferences and remodel themselves with the addition of new data over time.

Ontologies define the vocabulary within a given domain. Its main components are *classes* (entities are classified within a hierarchy of classes, e.g. in the context of robotics, classes could include robot types, action types, etc.), *relationships* (they define how entities or classes are related to each other) and *attributes* (properties that describe an individual class).

RDF[1] and Web Ontology Language (OWL)[2] are key standards of the Semantic Web. Represented in RDF, knowledge graphs excel at integrating, unifying, linking and reusing data.

2.2. Reinforcement Learning

Reinforcement Learning (RL) is a type of Machine Learning that makes an agent learn from interactions with the environment without explicit examples or external instructors [11, 7, 8]. The environment is typically modelled as a *Markov Decision Processes* (MDP) or a partially observable MDP (POMDP). Formally, an MDP is denoted as a tuple of five elements (S, A, P, R, γ) where S represents the space of states (i.e. the set of possible states), A represents the space of actions (i.e. the set of possible actions), P represents the probability of transition from one state to another given a particular action, R represents the reward function and $\gamma \in [0, 1]$ is the discount factor that determines the importance of future rewards. The agent interacts with the environment in discrete time steps, $t = 0, 1, 2, ...$. At each time step t, the agent obtains a representation of the environmental state $S_t \in S$, takes an action $A_t \in A$, moves to the next state S_{t+1} and receives a scalar reward $R_{t+1} \in R$. The agent's behaviour is described as a policy $\pi : S \times A$, where $\pi(s|a) = P(A_t = a|S_t = s)$ is the probability of taking an action $a \in A$ given state $s \in S$. The agent's goal is to maximise the expected cumulative discounted reward, which is denoted as G_t:

$$G_t = \sum_{n=0}^{\infty} \gamma^n R_{t+n+1}$$

The optimal behaviour of taking the best action in each state to maximise the reward over time is called the optimal policy, $\pi*$.

RL can be divided in two distinct computational strategies, namely model-free RL and model-based RL.

Model-free RL assumes that learning occurs without access to any internal representation of the causal structure of the environment [12]. The agent merely stores estimates of the expected values of actions available in each state or context, modelled by a history of direct interactions with the environment, without building a model of the environment.

Model-based RL, on the other hand, assumes that the agent has an internal model that predicts the outcomes of actions and estimates the immediate reward associated with specific situations [8, 13]. Decisions are not made on the basis of stored action values, but through planning: after learning, the agent is able to make predictions about what the next state and reward will be before taking each action and is thus able to compare candidate courses of action.

In our work, we utilise both types of learning. We employ model-based RL to learn policies and model-free RL to learn new actions.

[1]https://www.w3.org/TR/rdf-mt/

[2]https://www.w3.org/TR/owl-semantics/

2.3. Related work

Learning a representation of the state is an active research topic, under the term *State Representation Learning* (SRL) [14, 15, 16, 17]. In [14], prior knowledge about interaction with the physical world is used to learn state representations that are consistent with physics. The method extracts task-relevant state representations from high-dimensional observations. In [16], the authors focus on partially observable environments and propose to learn a minimal set of state representations that, given observational sequences, capture sufficient information for decision making, termed *Action-Sufficient state Representations* (ASRs).

Previous work has also demonstrated the feasibility of combining skill learning with external knowledge. In reference to the work of Wardenga et al. [18], knowledge is introduced in the form of RDF knowledge graphs during the RL process. This is achieved through the introduction of a *KG wrapper*, which utilises observations from the environment (text, images, heterogeneous data...) to extract the most relevant features and define the observation state. This is then inputted to the RL method of choice. In [19], the authors propose an integrated learning process for the robot, in which knowledge is incorporated into the learning process. They distinguish between two types of knowledge: *explicit knowledge*, which is extracted from prior information; and *tacit knowledge*, which is acquired through the robot's direct interactions with the environment. Similarly, [20, 21] exploit knowledge graphs in reinforcement learning. In [20], the authors combine the agent's belief with commonsense knowledge from the ConceptNet knowledge graph in order to act in the world.

3. Proposed approach

We present a framework that allows to create a state space S and action space A as input of a reinforcement learning problem. The core of the framework is a knowledge base that encodes information about the environment and physics via RDF triples. Given the robot type and the task to be learned, via queries to the KB and a low-level RL action training, the agent learns the state and action space, which are then used to obtain the policy π. To the best of our knowledge we are the first ones who propose an approach that tries to create state and action spaces starting from a KB. Our approach has the following advantages:

- *generality*: The knowledge related to the task is independent from the robot type;
- *extensible*: The KB can be extended with new knowledge; we give a structure to be followed when adding new information, making it possible to use provided predefined queries;
- *almost fully automatic process*: The only human input is in the definition of the KB and the reward function. We expect that encoding manually all of the knowledge needed to scale to real-world applications will not be feasible, so in the future we would like to introduce methods for acquiring and integrating knowledge from different sources;
- *the ability to learn a policy is not limited by the KB*: If the actions obtained from the KB are not sufficient to achieve the goal, new actions can be learned through action learning.

3.1. Knowledge Base

The KB provides the mechanisms to store and retrieve information about actions, objects, the environment, the physics, their properties and relations. The knowledge base is defined via RDf triples, where each element is represented by a Uniform Resource Identifier (URI). We use RDFlib [3] for handling RDF data using the Python programming language.

3.1.1. General predicates

To describe the relations between the concepts we define the following predicates:

[3]https://rdflib.readthedocs.io/en/stable/index.html

- **requires**: property to describe the requirements of the task. Fig. 1a shows the structure of the task definition.
- **has_action**: property to describe the actions executable by the subject.
- **has_state**: property to describe the state of the subject.
- **has_components**: property to describe the components of the subject.
- **state_type**: property to describe the type of state; the state can either be discrete or continuous.
- **can_be**: property to describe what the discrete state can be.

The task definition structure is defined as a set of triples having the form of *<task/action, requires, requirement>*. For example, a task that requires a robot to "pick_up" an object, is defined by the requirements of knowing what is used to pick up the object, what is the object to be picked up, and where to pick up the object. A graphical representation of the generic task definition and the "pick_up" example can be seen in fig. 1a and 1b respectively. This task definition is general and does not depend on the robot type.

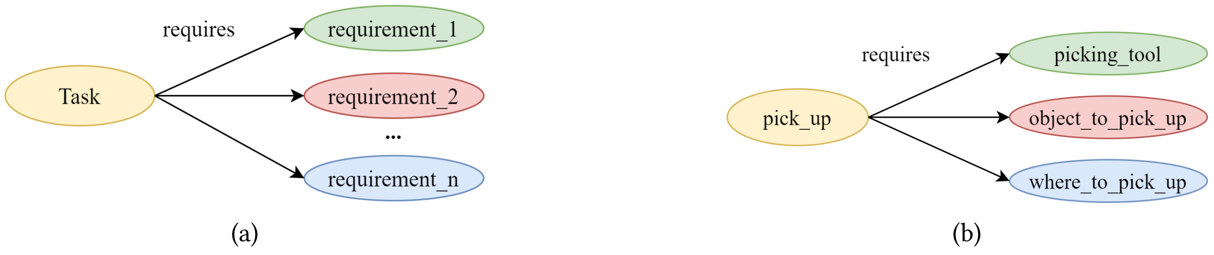

Figure 1: Task definition structure: (a) General structure; (b) Example of task definition for a *"pick_up"* task.

In a similar way, we define the description of a robot as a set of triples having the form *<robot, requirement, object>*. As can be seen, the "requirement", which was the object in the task definition, becomes a predicate in the robot definition, denoting that the robot has that capability. Examples of robot definitions can be found in figures 2 and 3, where fig. 2 describes a forklift and fig. 3 a robotic arm. It can be seen that there are identical task predicates in both contexts.

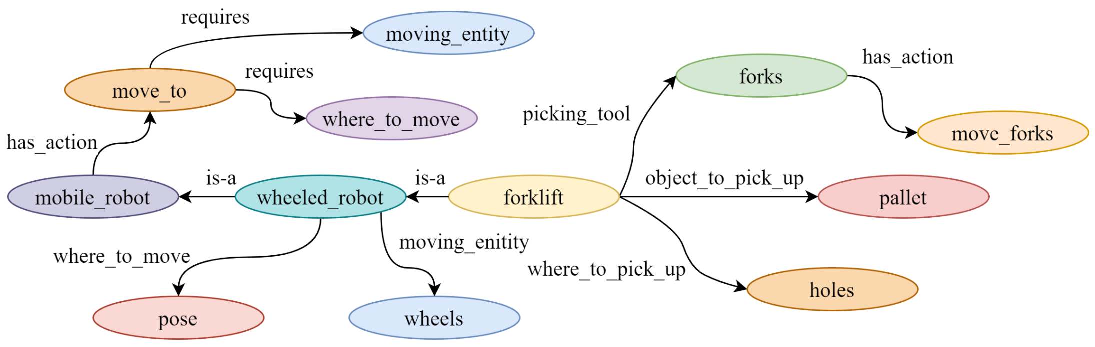

Figure 2: Example of requirements definition for a forklift. It can be seen that the forklift has the *"requirements"* needed to perform the *"pick_up"* task.

3.2. Prepared queries

We define a set of queries that allows to extract the relevant information from the KB. We used the *prepared queries* (parametrized SPARQL queries) provided by the RDFlib library, this avoids re-parsing and translating the query into SPARQL algebra each time. The following are the principal predefined queries used to obtain state and action spaces.

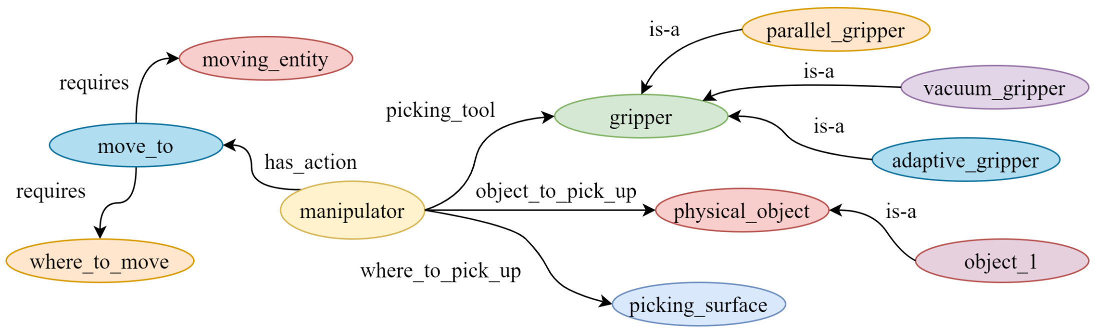

Figure 3: Example of requirements definition for a robotic manipulator. Also the robotic manipulator has the *"requirements"* needed to perform the *"pick_up"* task.

I. Get state space.
Initial bindings: robot, task

```
PREFIX krl: <http://kb4rl.org/>
SELECT ?entity ?state
WHERE {
{ {?task krl:requires ?requirement .
?robot ?requirement ?entity .
?entity krl:has_state ?state }
{?state krl:type_state krl:discrete}
UNION
{?state krl:type_state krl:continuous} }
UNION
{ {?task krl:requires ?requirement .
?robot ?requirement ?entity .
?entity krl:has_state ?m_state .
?m_state krl:has_components ?state }
{?state krl:type_state krl:discrete}
UNION
{?state krl:type_state krl:continuous} }
}
```

II. State space into array.
Initial bindings: state

```
PREFIX krl: <http://kb4rl.org/>
SELECT ?can_be WHERE {
?state krl:type_state krl:discrete .
?state krl:can_be ?can_be
}
```

III. Decode the state array.
Initial bindings: state

```
PREFIX krl: <http://kb4rl.org/>
SELECT ?can_be WHERE {
?state krl:can_be ?can_be .
?can_be krl:id ?id
}
```

IV. Get actions.
Initial bindings: robot

```
PREFIX krl: <http://kb4rl.org/>
PREFIX rdfs:
<http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?action WHERE {
{ ?robot ?p ?o .
?o ?has_action ?action }
UNION
{ ?robot rdfs:subClassOf ?robot_class .
?robot_class ?has_action ?action }
}
```

The first query retrieves the state space; it takes as inputs the task to be learned and the robot that needs to learn how to perform the task. The second query converts the state, which is currently an array of URIs, into a numerical array. The third query translates the numeric array back into a URIs array, which is useful when learning new actions. Finally, the fourth query retrieves the action space.

3.3. Methodology

The framework consists of three distinct parts, each including a training phase: the state and action space extraction from the KB, the action space expansion via low level training and the final training. Fig. 4 shows the conceptual structure of our framework.

State and action space extraction from the KB

The state and action space extraction is the core of the framework (first block of Fig. 4). After inputting the robot type and the action or task to be learned, queries are sent to the KB and a state space S_{KB} and action space A_{KB} are obtained. The agent is then trained using S_{KB} and A_{KB} and a policy π_I is obtained.

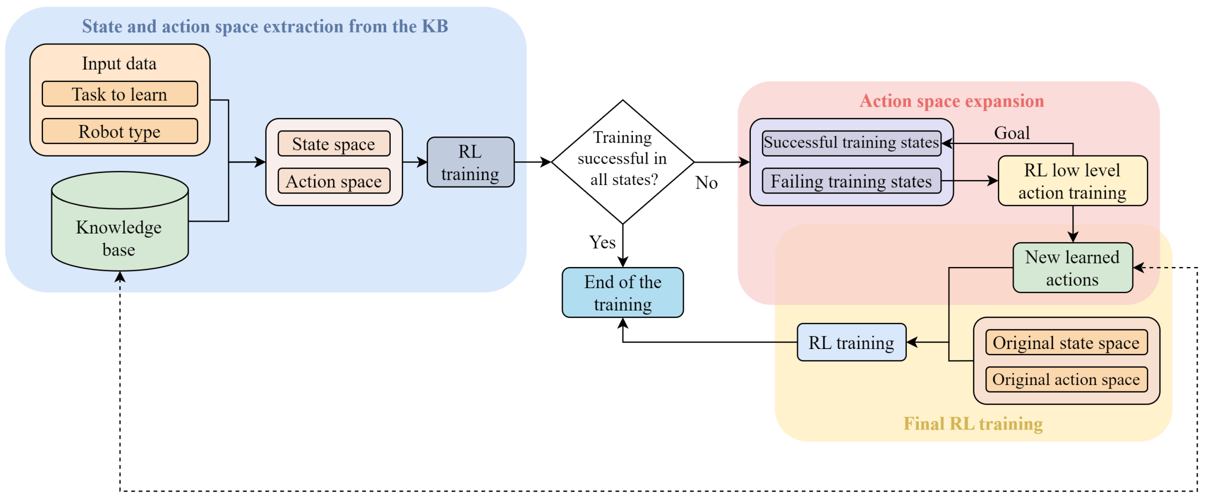

Figure 4: Conceptual structure of our framework: three blocks are highlighted. Starting from *the state and action space extraction from the KB* block, state and action space are extracted from the knowledge base and the first policy training is performed. Then, in the *action space expansion* block, the agent learns new skills. Finally, in the last block, the agent combines the new acquired skills with the actions obtained via the KB and a *final training* is performed.

Action space expansion via low-level training

Training the agent using the state and action space obtained from the KB may not succeed to find a policy reaching the goal from all states S. This is due to the fact that the action space A_{KB} obtained via the KB may not be large enough and therefore may not include unforeseen situations that the robot would encounter while performing the task. Hence a second training is introduced in the framework, to teach the agent actions that enable it to reach a state where policy π_I is applicable. Fig. 5 shows the action-learning framework.

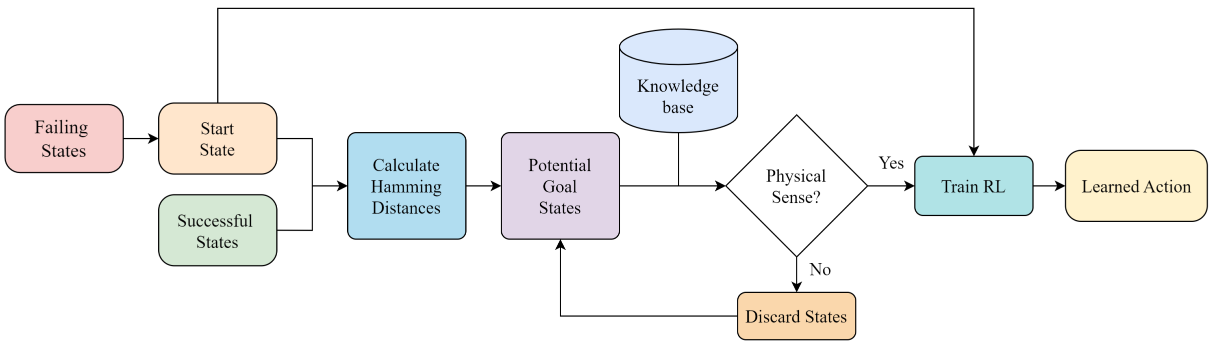

Figure 5: Action-learning framework: the failing states $S_{failing}$ are compared to the successful states $S_{successful}$ in order to learn the actions the allows to reach the “closest” successful state, for which a policy π_I is already known.

The states are divided in successful states $S_{successful} \in S$ (i.e. states in which the goal can be reached with A_{KB}) and failing states $S_{failing} \in S$ (i.e. states in which the goal cannot be reached with A_{KB}). At each training step an initial state is chosen between the failing states $s_{start} \in S_{failing}$. Next, the *Hamming distances* to the successful states are calculated, and the closest states (i.e. states with the minimum distance) are taken as potential goal states. Each potential goal state is assessed to determine if it is physically reachable from the initial state, i.e. whether the transition is physically possible. All states that are not physically reachable are discarded, and only the physically reachable states are kept as possible goal states. If none of the closest successful states is physically meaningful, the second closest successful states are considered as potential goal states and an evaluation is performed again to see if transitions are possible. This process is repeated until at least one physically significant successful state is identified. If a state cannot be found, the original goal used in training the first policy will be set as the goal. Finally, the goal state s_{goal} is selected from the closest physically significant successful states and the agent is trained to learn an action to reach the goal state. When learning actions, the KB is used to identify similarities between the new actions (dashed arrow Fig. 4).

For example, consider a task where a forklift needs to pick up a pallet. The high-level training would be able to find a policy from states in which one of the two faces of the pallet is free or states in which the holes are hidden by a deformable object (the holes are accessible by the forks). However, the training would fail in those states in which the pallet holes are hidden in both faces by a rigid obstacle, for example, a fallen box in the warehouse. When comparing the successful states $S_{successful}$ with the failing states $S_{failing}$, two types of state transition would emerge: the state of the hole has to go from hidden to free or the object has to change from rigid to deformable. However, only one of the two types of transition makes physical sense in this case, as usually a rigid object cannot change to a deformable one.

Therefore, the $goal$ to achieve with the low-level training would be to free the holes in one of the two faces of the pallet, thus learning an obstacle removal action. Once the holes are free, the agent could simply use the policy π_I obtained with the first training.

Final training

In the last RL training (third block of Fig. 4) the agent is trained by using both the old action space A_{KB} as well as the newly learnt actions $A_{extended}$ and a policy π_{II} is obtained. This time the agent opts for the most convenient action to take.

Returning to the example of the forklift that has to pick up a pallet, let us consider a state in which the holes are hidden in just one face; this is a successful state, i.e. a state in which the original policy π_I was enough to reach the goal. By following π_I the agent would have gone to the free face of the pallet as it did not know any action to move the obstacles, but now with π_{II} the agent may find it more efficient to move the obstacle rather than going to the free face of the pallet.

4. Discussion

In this work, we have presented an approach to automate the creation of state and action space for Reinforcement Learning via a Knowledge Base that provides the mechanisms to store and retrieve information about actions, objects, the environment, the physics, their properties and relations. We have also introduced a solution for extending the action space to include unforeseen situations that the agent may encounter during the policy learning.

At the moment we have no experimental evaluations to support our approach, apart from human evaluation of the generated policies. In the near future we plan to test our method on a real robot. Additionally, we would like to automate the retrieval of knowledge from different sources and also to automate the definition of the reward function via the KB.

Acknowledgments

This work was funded by the Research Foundation of Flanders (FWO), Grant Number 1SHHJ24N.

References

[1] L. Sciavicco, B. Siciliano, Modelling and control of robot manipulators, Springer Science & Business Media, 2012.

[2] F. Gul, W. Rahiman, S. S. Nazli Alhady, A comprehensive study for robot navigation techniques, Cogent Engineering 6 (2019) 1632046.

[3] S. Rutishauser, A. Sprowitz, L. Righetti, A. J. Ijspeert, Passive compliant quadruped robot using central pattern generators for locomotion control, in: 2008 2nd IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics, IEEE, 2008, pp. 710–715.

[4] S. Aoi, K. Tsuchiya, Locomotion control of a biped robot using nonlinear oscillators, Autonomous robots 19 (2005) 219–232.

[5] Y.-q. Jiang, S.-q. Zhang, P. Khandelwal, P. Stone, Task planning in robotics: an empirical comparison of pddl-and asp-based systems, Frontiers of Information Technology & Electronic Engineering 20 (2019) 363–373.

[6] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, T. Lozano-Pérez, Integrated task and motion planning, Annual review of control, robotics, and autonomous systems 4 (2021) 265–293.

[7] L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement learning: A survey, Journal of artificial intelligence research 4 (1996) 237–285.

[8] A. S. Polydoros, L. Nalpantidis, Survey of model-based reinforcement learning: Applications on robotics, Journal of Intelligent & Robotic Systems 86 (2017) 153–173.

[9] K. Zhu, T. Zhang, Deep reinforcement learning based mobile robot navigation: A review, Tsinghua Science and Technology 26 (2021) 674–691.

[10] J. Kober, J. A. Bagnell, J. Peters, Reinforcement learning in robotics: A survey, The International Journal of Robotics Research 32 (2013) 1238–1274.

[11] R. S. Sutton, A. G. Barto, et al., Introduction to reinforcement learning, volume 135, MIT press Cambridge, 1998.

[12] Y. Jiang, F. Yang, S. Zhang, P. Stone, Task-motion planning with reinforcement learning for adaptable mobile service robots, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2019, pp. 7529–7534.

[13] I. P. Pinto, L. R. Coutinho, Hierarchical reinforcement learning with monte carlo tree search in computer fighting game, IEEE transactions on games 11 (2018) 290–295.

[14] R. Jonschkowski, O. Brock, State representation learning in robotics: Using prior knowledge about physical interaction., in: Robotics: Science and systems, 2014.

[15] T. Lesort, N. Díaz-Rodríguez, J.-F. Goudou, D. Filliat, State representation learning for control: An overview, Neural Networks 108 (2018) 379–392.

[16] B. Huang, C. Lu, L. Leqi, J. M. Hernández-Lobato, C. Glymour, B. Schölkopf, K. Zhang, Action-sufficient state representation learning for control with structural constraints, in: International Conference on Machine Learning, PMLR, 2022, pp. 9260–9279.

[17] D. Ha, J. Schmidhuber, Recurrent world models facilitate policy evolution, Advances in neural information processing systems 31 (2018).

[18] R. Wardenga, L. Kovriguina, D. Pliukhin, D. Radyush, I. Smoliakov, Y. Xue, H. Müller, A. Pismerov, D. Mouromtsev, D. Kudenko, Knowledge graph injection for reinforcement learning (2023).

[19] M. Mayr, F. Ahmad, K. Chatzilygeroudis, L. Nardi, V. Krueger, Combining planning, reasoning and reinforcement learning to solve industrial robot tasks, arXiv preprint arXiv:2212.03570 (2022).

[20] K. Murugesan, M. Atzeni, P. Shukla, M. Sachan, P. Kapanipathi, K. Talamadupula, Enhancing text-based reinforcement learning agents with commonsense knowledge, arXiv preprint arXiv:2005.00811 (2020).

[21] S. Zhou, X. Dai, H. Chen, W. Zhang, K. Ren, R. Tang, X. He, Y. Yu, Interactive recommender system via knowledge graph-enhanced reinforcement learning, in: Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval, 2020, pp. 179–188.