Generic Planning Using Qualitative Causal Models for Self-Learning Autonomous Robots

Marco Van Cleemput[1][0009−0009−2948−3476], Jan Lemeire[1][0000−0002−2106−448X], and Ruben Spolmink[1][0009−0004−7096−8559]

Vrije Universiteit Brussel (VUB), Pleinlaan 2, B-1050 Brussels
marco.paolo.van.cleemput@vub.be

Abstract. This paper puts forward qualitative causal models as the basis for a cognitive architecture for autonomous agents that can learn world models and use them for control and planning. The causal relations are described qualitatively by the direction of changes of the causal influences. Context is taken into account when causal processes only take effect for particular states. The model can be learned bottom-up based on a few samples gathered during an exploration phase. During exploitation, the model is used for planning and control to perform certain sequential planning tasks, such as pick and place operations. The approach is demonstrated on a robot with a gripper that can move objects and push buttons to open boxes. This approach is more sample efficient than reinforcement learning, provides an explanation, is more general and better reflects how humans learn and reason.

Keywords: Qualitative models · open-ended learning · autonomous robots · developmental learning.

1 Introduction

Artificial intelligence (AI) is becoming increasingly important in both the workplace and academic research. Tools such as ChatGPT and other large language models (LLMs) are currently receiving significant attention. However, LLMs represent only one aspect of AI. While they demonstrate strong performance in language-related tasks, they fall short in terms of understanding and interacting with the physical world.

Embodied AI tries to address these shortcomings by using embodied agents such as autonomous robots to learn how the world works. The most used AI technologies like reinforcement learning (RL) and neural networks (NN) in the field of autonomous robots, need a lot of data and training time to achieve results. Moreover, these technologies are black-boxes that fail to provide explainability.

We advocate an alternative approach for autonomous self-learning agents: one based on qualitative models. The main paradigm, reinforcement learning is a model-free approach in the sense that the agent does not learn or use a *world*

model but it learns a policy to achieve specific goals [8]. Although successful, RL needs a lot of data and is criticized for not being scalable nor explainable. Additionally, it has become evident that the

> "RL system had learned to perform its task using heuristics specific to its training data rather than the kind of abstract, causal understanding the human trainers were trying to teach it" [4].

The need for world models for intelligent agents has been recognized by many, see for instance [6, 9]. To overcome the difficulties of modeling and learning quantitative models, such as state-space models, *qualitative models* might be the missing piece in the puzzle of creating intelligent agents that can learn, adapt, plan, reason, and explain. A qualitative model is based on the direction of causal influences, which can be positive (PLUS), negative (MINUS), or no influence (ZERO). It offers an abstract description that is correct in the area of the state space where the causal influence is a monotone function. Changes in the monotonicity in certain areas define the *context* of the causal relation. The direction of changes is easier to measure and learn, and more robust than quantitative relations. To further explore the potential of such a qualitative approach, this paper proposes the following contributions:

- qualitative causal models representing the causal relations
- models that explicitly encode contextual relations
- a task-executing algorithm based on causal inference which includes planning and control

2 Related work

Most often regarded as a Markov decision processes (MDP) [12] is formally defined as a tuple $\langle S, A, T, R \rangle$, where S is a set of states, A a set of actions, T a transition model defining the transition probabilities between states ($T(s, a, s') = p(s'|a, s)$ is the probability of reaching state s' from state s after executing action a), and $R : S \times A \to R$ is a reward signal. The goal is to learn the optimal policy π^* to optimize the long-term reward. A policy is a map from states to actions $\pi : S \to A$. The optimal policy thus gives for each state the optimal action. The agent does not know the transition probabilities T and the reward function R. Model-based methods aim at learning T and R and base their planning on the learned models, while model-free methods only estimate R and focus on learning the optimal policy. Q-learning is an example of the latter [15]. In our model-based approach, a qualitative model of T is learned without R, just relying on the agent's intrinsic motivation to understand the world around it.

The setup of Romero et al. [11] is very similar to ours. It involves a robot arm that operates in front of a table with cylinders, boxes and buttons. The robot arm must learn to perform reach, pick and place tasks. A task implies achieving a specific goal such as putting an object in a box. However, for a goal to be

attainable, specific preconditions should be achieved. Their solution is based on a model-free version of RL. This implies that for each goal, a specific plan has to be devised, which includes finding the right chain of subgoals such that the necessary preconditions are met. In our approach, a causal model is learned that includes these preconditions, which are called context. From the causal model, the subgoals can directly be inferred from the model. The model can be used for the planning of any feasible task. On the other hand, the authors build an architecture that is close to our approach. There is a specific module for seeking new capacities and retaining plans to reach novel states that are potential subgoals. These plans can then be used for building a plan for executing tasks.

The work of [5] initiated the major approach for qualitative models. It starts from a description of the system under study with ordinary differential equations (ODEs), from which qualitative constraints are extracted which are defined as follows [13]:

$$\begin{array}{ll} y = Q^+(x) \text{ means} & \frac{\partial y}{\partial x} > 0, \\ y = Q^-(x) \text{ means} & \frac{\partial y}{\partial x} < 0. \end{array} \tag{1}$$

A positive derivative means that the function $y = f(x)$ is monotonically increasing, while a negative derivative corresponds to a monotonically decreasing function. This can be extended to *multivariate monotonic qualitative constraints* [16, p. 27]. Q-constraints can be extracted from experimental data with Padé [13]. This has been used successfully for robotic control [1, 14, 17], but it is limited to a few variables for which the relations among the variables are specified a priori (manually).

From the field of *system modeling* it is known that systems described by higher-order ODEs can be converted into a *state-space representation* by expressing it as a set of coupled first-order ODEs [3][Sec. 2.2]. Additional variables might be added for this transformation, such as velocity to link acceleration with position. The resulting mathematical model is a set of input, output, and state variables related by first-order differential equations: $\dot{s}_{t+1} = f(s_t, a)$ with $\dot{s}_{t+1}$ the time derivative of the state. Quantitative descriptions are, however, most often limited to linear equations and contextual relations cannot be taken into account. Both limitations can be overcome by modeling this function in a qualitative way such as done by [2] and [10].

3 Contextual Causal Qualitative Models

This paper uses contextual causal qualitative models as a method of implementing the transition model T of MDPs and, more precisely, the function f of state space models since it involves the time derivatives. We assume that the state is known (by observation through sensors and state estimation) and that qualitative determinism holds. This model is both learned and used so that a robot can

interact with the world. The learning of the model is described in chapter 4, and the usage (planning) of this model is described in chapter 5.

3.1 Qualitative relations

The variables of the model are the state variables and their time derivatives. The action variables influence the time derivatives of the state variables through a causal relation. The causal relation between a variable v and its direct causes, denoted as the parents of the variable $Pa(v)$, is described by a deterministic qualitative function $\mathcal{Q}(v) = QF(\mathcal{Q}(Pa(v)))$. $\mathcal{Q}(v)$ gives the sign of a continuous variable, which has three possible outcomes: PLUS, MINUS and ZERO, also denoted with +, - and 0. Applied on a vector, as with $\mathcal{Q}(Pa(v))$, it returns a list with the signs of the vector's elements. A qualitative function gives the sign of a variable v based on the signs of the variable's direct causes. It is implemented as ternary truth tables.

3.2 Contextual Relations

The qualitative description of a function holds as long as a function is monotone. The qualitative relation between variables can, however, change depending on the state. This means that some relations between variables change depending on specific variables. We call this *context*. Some qualitative relationships depend on context. Such a context variable divides the state space into different *operating regions* for that relationship [16]. Operating regions are demarcated by *landmarks*, which are the border values of the context variables dividing the state space into subspaces of monotone behavior.

An example of this is the distance between a robot and an object. If the robot drives forward, this distance will decrease. This is not true if the robot is facing the other way. Then, the distance will increase. This relation has two different monotonic functions depending on the orientation of the robot. Orientation is the necessary context here.

3.3 Causal Graph

The relations among the action, state, and derivative variables can be visualized with a graph as is common practice in causal models. The edges are based on the direct causal relations, the relations between derivatives and state variables, and the contextual relations. A part of the world model is shown in Figure 1. It describes the causal and contextual relations between the variables, considering a robot with two wheels and an object in egocentric pole coordinates. The black solid arrows describe qualitative causal functions between two variables. The dashed arrows describe the contextual relations between context variables and variables. $Drive$ is an action variable, while θ and $Dist$ are state variables. They represent in polar coordinates the distance and orientation of the robot to an

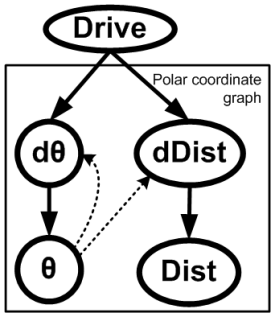

Fig. 1. Egocentric graph of the relation between the robot's actions and an object, expressed in polar coordinates.

object from an egocentric point of view. Variables starting with d are derivative variants of their state variables. This work assumes that the robot has the skill to drive as explained in section 5.2. This graph is used to represent the relation between the robot and any object in the environment.

4 The learning algorithm

The algorithm learns the causal from observational data gathered during the exploration of the environment [7]. Actions lead to changes in the system's state (comprising the agent's state and the state of the environment). For each state change that is detected, the cause-effect relation is identified when a qualitative-deterministic function QF is found between action variables and the derivatives of the state variable. For contextual relations, the state space must be partitioned to be able to identify a particular qualitative-deterministic function in each part. As soon as a different behavior is detected at different locations of the state space, the border of the contextual partition is sought and identified.

Currently, we are manually guiding the robot to the interesting locations in the environment so that it observes the necessary state changes to build the qualitative model.

5 The planning algorithm

This work proposes a planning algorithm for self-learning autonomous robots, which also provide explainability. A robot first learns a qualitative causal model of the world (section 3), after which the planner can use this model to execute tasks by providing high level planning and low level control.

5.1 Reverse Graph algorithms

The planning algorithm uses the learnt model with a goal G as input to find a sequence of actions A. A goal state Gs combines a state variable s and a goal condition Gc as $Gs = (s \rightarrow Gc)$. A goal combines one or more goal states together in a list $G = [Gs_1, Gs_2, ..., Gs_n]$. A Gc can either be a range consisting of a minimum and a maximum Gs value or a qualitative sign. For example, a goal can be executed as follows, ApproxGoal(Dist $\in [0.28, 0.02]$).

The robot executes the goals using the planning algorithm (formalised in algorithm 1). Depending on the context, the robot will have to change operating regions. For example, if the robot wants to go to an object but is not facing it, it will need to change its orientation first before it can drive forward to get to the object. In this case, turning is the action the algorithm generates in order to achieve the subgoal of transitioning from one operating region to another.

The world model is implemented as a Graph object.
The planning algorithm consists of two parts (algorithm 1 and algorithm 2 (inverseGraph)). Algorithm 1 starts with a goal, it iterates through the goal states, and executes **inverseGraph** (algorithm 2), which is a method of the Graph that traverses the Graph in reverse. Algorithm 1 finishes by executing the method **getActions** for all the action variables. This method gathers all the possible actions to execute the goal and chooses one using the control algorithm explained in section 5.2.

The method inverseGraph (algorithm 2) goes through the Graph backwards. It will start with a given variable and its condition. Then, paths in the graph that lead to the action variables will be found. During this process, the algorithm will use the method **isInContext**, which is a method of a variable, to check whether the robot is in the right context based on the sign needed (condition). If this sign is not available in the current operating region (line 4), the algorithm will execute the variable method **getcontext** (line 5) to create the correct subgoal. Example (using the same situation as the example above), The robot wants to execute DesiredsignGoal(dDist $\rightarrow -$) to decrease the distance to an object. The qualitative function that has this sign available has been found using the variable method **getcontext**. This function is inactive (not in the correct operating region) because the context variable is not in the right partition. The correct operating region states that θ needs to be between -90 and 90. A new subgoal is created to change this: Rangedgoal($\theta \rightarrow [-90, 90]$). If inverseGraph (algorithm 2) defines a novel subgoal, it will jump to the context variable to continue reversing through the Graph to get to the action variables. The variable method **Qinverese** is used to get the sign of the parent(s) of the variable variable out of the current qualitative function (line 7).

Algorithm 1: Planning Algorithm

```
1: for goalState in goal do
2:   if goalState is not attained then
3:     originalGoal = goalState
4:     Variable = Variable of goalState
5:     condition = goal condition of goalState
6:     Get state of simulation
7:     Graph.inverseGraph(originalGoalState, Variable, condition, state)
8:   end if
9: end for
10: for all ActionVariables do
11:   ActionVariable.getAction
12:   Choose action (control algorithm)
13: end for
```

Algorithm 2: InverseGraph(*originalGoalState*, *Variable*, *condition*, *state*)

```
1: create empty list Var_goalcondition_pairs
2: if Variable is not an action variable then
3:   if condition is type SIGN then
4:     if Variable.isInContext(condition) then
5:       Var_goalcondition_pairs = Variable.getcontext(condition)
6:     else
7:       Var_goalcondition_pairs = Variable.Qinverse(condition)
8:     end if
9:   else
10:    desiredSign = Sign(state(Variable) - condition)
11:    parent = Variable.getParent
12:    add {parent : desiredsign} pair to Var_goalcondition_pairs (goes 1 up)
13:  end if
14: else
15:   add {originalGoalState : condition} to Variable
16: end if
17: for Variable, condition in Var_goalcondition_pairs do
18:   Graph.inverseGraph(originalGoalState, Variable, condition, state)
19: end for
```

The function **Sign** is used to convert a number into a qualitative sign. It is used in line 10 of algorithm 2 to get the sign to reach the goal condition.

5.2 Control Algorithm

There are three parts implemented to control the robot.

The first part is the skill to drive. The action variable *Drive* can be set to drive forward, turn right or turn left using the respective qualitative signs:

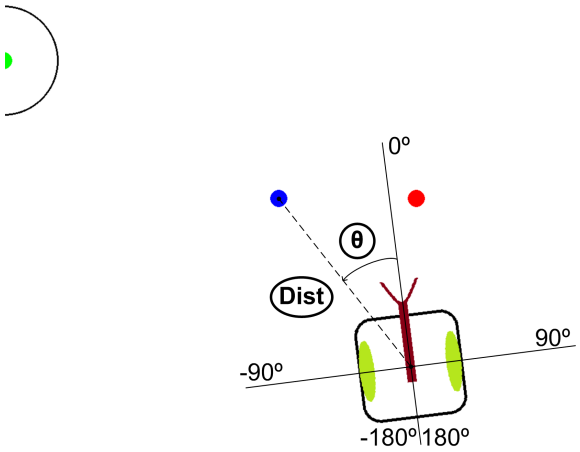

Fig. 2. The setup of the experiment with the egocentric representation in pole coordinates.

Drive = 0 means driving straight forward.
Drive = - means turning to the left around the robot's middle point.
Drive = + means turning to the right around the robot's middle point.
This value is translated to motor commands.

The second part is an algorithm implemented to keep the robot from switching actions each time step. If the robot chooses between multiple actions to execute, it decides to execute the action it chose in the previous time step.

The third part is an anti-orbit algorithm. This algorithm helps the robot reach a goal faster when the robot is stuck in an orbit around an object. The robot does not have the concept of time, which results in the orbit problem explained in 6.2. The algorithm solves this problem by adding a subgoal. When the robot wants to get closer to an object without an orientation subgoal, this algorithm adds an orientation subgoal that lets the robot face this object first.

6 Setup and Experiment

An experiment has been conducted to empirically validate the planning algorithm. This is executed in simulation.

6.1 The setup

The setup used for the experiment contains the following objects (figure 2):

1. A robot with two wheels and a gripper.
2. A movable blue object

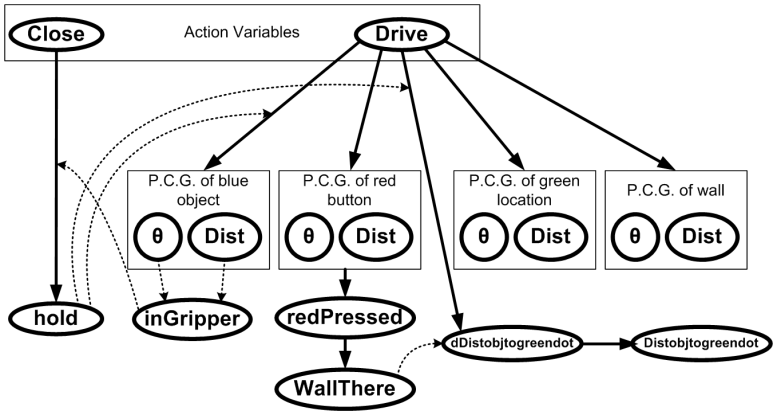

Fig. 3. The DAG of the experiment

3. A green non-movable location
4. A red non-movable button
5. A wall around the green location

A simulation is created in Python to simulate the system. It uses the motor values and calculates the next state. The simulation not only calculates the state variables that our model contains. It also calculates other state variables, which are used to draw the setup.

Although we work with egocentric pool coordinates, we visualise the setup with the absolute coordinate system as seen in figure 2. This is for a better understanding of what is happening. The egocentric representation of the setup is also visualised as in figure 2.

The simulation simulates a robot with two motors. This enables the robot to turn around its center point. As such, the robot is able to turn without changing its absolute x and y coordinates. The gripper can grab an object using a sensor that senses if something is inside the gripper. This sensor is used in order to enable a grip reflex just like humans. The object cannot go through the wall in the simulation; the robot always drops the object when the object encounters the wall. The red button makes the wall disappear when pressed.

The full model of this setup can be seen on figure 3. The P.C.G. rectangles on the dag are the Pole Coordinate Graphs of the objects. The original P.C.G. can be seen on figure 1 for one object. The P.C.G. is the same for all the objects/locations in egocentric pole coordinates.

6.2 Experiment

Now that the setup is complete and the world model describing this setup is learned, tasks can be given to the robot. We implement a pick-and-place task by giving the robot a goal to reach. The robot's goal is to move the blue object to the green location. This is achieved by giving the robot the goal ApproxGoal(distobjtogreendot $\rightarrow 0, 0.02$).

The robot can complete the goal using the planning algorithm explained in Section 5. It did this in 1020 time steps. The simulation runs at 60Hz, meaning the robot executed the goal in 17 seconds.

The robot passed through multiple operating regions to complete the goal. The plan the robot made for this experiment consists of these subgoals and their corresponding actions:

1. ApproxGoal(disttoreddot $\rightarrow 0.24, 0.02$) A: Drive = 0
2. ApproxGoal(orreddot $\rightarrow 0, 2$) A: Drive = -
3. ApproxGoal(disttoreddot $\rightarrow 0.24, 0.02$) A: Drive = 0
4. ApproxGoal(orobj $\rightarrow 0, 0$) A: Drive = -
5. ApproxGoal(disttoobj $\rightarrow 0.24, 0.02$) A: Drive = 0
6. ApproxGoal(orgreendot $\rightarrow 0, 2$) A: Drive = +
7. ApproxGoal(distobjtogreendot $\rightarrow 0, 0.02$) A: Drive = 0

The robot has to reason on the world model to generate this plan. Take the first subgoal. The robot wants to close the distance between the blue object and the green dot. This cannot be done because the robot must first change the operating region. The contexts decide that the robot has to change the sign of the variable *wallThere* to + which means changing the variable *redPressed* to + which means driving towards the red button. This is the meaning of the first two goals of the plan.

Using the planning algorithm, the robot can now travel from one operating region to the next. This provides explainability. Intuitively, this reads as follows:

1. Go to red button to press button and remove wall
2. Go to object
3. Go to green location

The result of the experiment can be seen in figure 4 where the center of the gripper is being traced to visualise the robot's movement.

We did encounter an orbit problem as explained in section 5.2. The robot enters a situation where it constantly switches between turning and driving straight. It makes the robot move into orbit around the green location. Here, the subgoal states that the robot is only able to decrease the distance between the object and the green location if the egocentric orientation of the green location in reference to the robot is between -90° and +90°. The robot should know that the distance between the object and the green location can only be 0 if the orientation is also 0°. This would mean that ApproxGoal(orgreendot $\rightarrow 0, 2$) should be a subgoal.

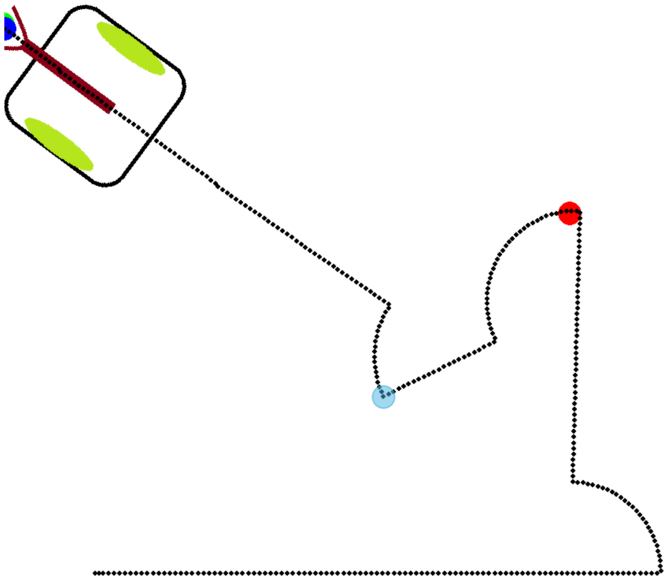

Fig. 4. The result of the experiment.

7 Conclusions

This work shows how a model-based approach can be used for the planning of typical robot tasks. Qualitative descriptions of causal relations taking context into account provide sufficient information to extract the chain of subgoals and actions needed to achieve a certain goal. The setup used is similar to [11], also generating sub goals to complete the task, but our non-blackbox approach provides explainability using less training data and time in comparison with their state of the art RL approach. This type of approach knows what, but not why, a certain action should be performed. Our approach does know why because of the context. These contexts provide sub goals that, in their turn, generate actions. These sub goals provide explainability. Even if the robot is not able to find a way to achieve a goal, it will know why.

References

1. Bratko, I.: An assessment of machine learning methods for robotic discovery. Journal of Computing and Information Technology **16**, 247–254 (01 2008)
2. Bredeweg, B., Linnebank, F., Bouwer, A., Liem, J.: Garp3 — workbench for qualitative modelling and simulation. Ecological Informatics **4**(5), 263–281 (2009). https://doi.org/https://doi.org/10.1016/j.ecoinf.2009.09.009, https://www.sciencedirect.com/science/article/pii/S1574954109000818, special Issue: Qualitative models of ecological systems
3. Franklin, G., Powell, J., Emami-Naeini, A.: Feedback Control Of Dynamic Systems, 4th edition. Prentice Hall (2002)
4. Kansky, K., Silver, T., Mély, D.A., Eldawy, M., Lázaro-Gredilla, M., Lou, X., Dorfman, N., Sidor, S., Phoenix, S., George, D.: Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics. arXiv e-prints arXiv:1706.04317 (Jun 2017). https://doi.org/10.48550/arXiv.1706.04317
5. Kuipers, B.: Qualitative simulation. Artificial Intelligence **29**(3), 289–338 (1986)
6. LeCun, Y.: Position paper: A path towards autonomous machine intelligence (2022), version 0.9.2, 2022-06-27
7. Lemeire, J., Wouters, N., Van Cleemput, M., Heirman, A.: Contextual qualitative deterministic models for self-learning embodied agents. In: Buckley, C.L., Cialfi, D., Lanillos, P., Ramstead, M., Sajid, N., Shimazaki, H., Verbelen, T., Wisse, M. (eds.) Active Inference. pp. 3–13. Springer Nature Switzerland, Cham (2024)
8. Li, K., Hopkins, A.K., Bau, D., Viégas, F., Pfister, H., Wattenberg, M.: Emergent world representations: Exploring a sequence model trained on a synthetic task. In: The Eleventh International Conference on Learning Representations (2023)
9. Mitchell, M.: Llms and world models: Part 1 (2025), accessed: 2025-03-28
10. Mugan, J., Kuipers, B.: Autonomous Learning of High-Level States and Actions in Continuous Environments. IEEE Transactions on Autonomous Mental Development **4**(1), 70–86 (Mar 2012)
11. Romero, A., Baldassarre, G., Duro, R.J., Santucci, V.G.: Autonomous discovery and learning of interdependent goals in non-stationary scenarios. In: 2024 IEEE International Conference on Development and Learning (ICDL). pp. 1–8 (2024). https://doi.org/10.1109/ICDL61372.2024.10644825
12. Sutton, R.S., Barto, A.G., et al.: Reinforcement learning: An introduction, vol. 1. MIT press Cambridge (1998)
13. Žabkar, J., Bratko, I., Demšar, J.: Learning qualitative models through partial derivatives by Padé. In: Proceedings of the 21st Annual Workshop on Qualitative Reasoning. pp. 193–202 (2007)
14. Zabkar, J., Bratko, I., Mohan, A.C.: Learning qualitative models by an autonomous robot. In: 22nd International Workshop on Qualitative Reasoning. pp. 150–157 (2008)
15. Zimmer, M., Doncieux, S.: Bootstrapping Q -Learning for Robotics From Neuro-Evolution Results. IEEE Transactions on Cognitive and Developmental Systems **10**(1), 102–119 (Mar 2018). https://doi.org/10.1109/TCDS.2016.2628817, conference Name: IEEE Transactions on Cognitive and Developmental Systems
16. Šoberl, D.: Automated planning with induced qualitative models in dynamic robotic domains. Ph.D. thesis, University of Ljubljana (2021)
17. Šoberl, D., Bratko, I.: Reactive motion planning with qualitative constraints. In: Advances in Artificial Intelligence: From Theory to Practice. pp. 41–50. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-60042-0_5