POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Automated Generation of Robot Planning Tasks Observing Human Actions in Virtual Reality

Laurea Magistrale in Automation and Control Engineering - Ingegneria dell'Automazione

Author: Gianluca Clerici

Advisor: Prof. Paolo Rocco

Co-advisors: Prof. Andrea Maria Zanchettin, Ing. Niccolò Lucci

Academic year: 2021-2022

1. Introduction

Over the last few years, significant progress has been made in robot technology with the aim of making robots autonomous and able to collaborate with humans. This has contributed to the widespread of collaborative robots (cobots) into Small Medium Enterprises (SMEs). These robots are lightweight and have low speeds, low payload and low setup time. On the other hand, they need a robotic expert capable of reprogramming the cobots every time a product variation is required. For this reason, researchers are focusing on new intuitive robot programming methods that allow non-skilled operators to program the robots. Moreover, the new technologies available in the market, such as Virtual Reality (VR), can be exploited to improve the effectiveness of such robot programming methods. In fact, Virtual Reality is one of the core elements of Industry 4.0. It is used in various industrial fields such as product design, machine control and training professionals. Virtual Reality can be used to train employees. In fact, it is possible to design different virtual environments depending on the context requirements the headset is used in. In this way, the employee can be trained without stopping the production line. This minimizes the downtime and avoids all the risks that are related to the use of the physical system. Combining this technology with already known methods for reducing the robot programming effort can lead to new powerful approaches to instruct robots. Thus, we focused on developing a new approach for intuitively programming a robot with the help of Virtual Reality. The operator has to demonstrate actions in a virtual environment, and the system has to classify the actions and characterize them with relevant preconditions and postconditions. Then, the learnt actions are written in a planning domain file using the PDDL language [1]. The advantage of using PDDL is that many planners are available. A planner has the objective of scheduling the required skills to pass from the initial state to the final one. In order to define the initial state and the final one, a planning problem has to be generated. The user has to place the objects needed during the demonstration phase in the desired positions to define a goal, and the system automatically generates the planning problem. Moreover, using a planner allows performing unseen tasks and gives the

robot the flexibility it misses.

2. Methodology

2.1. Skills Classification

During the demonstration phase, the user performs actions in the virtual environment. To correctly characterize the performed skills, it is fundamental to identify what the operator is doing and also to understand when an operation is starting and ending. With this goal, we implemented a State Machine [?] able to recognize the starting and the ending of four different skills: Pick, Release, Stack, and Unstack. Every time a skill performed by the user is recognized, we store it and we characterize it. In particular, each skill has: a name, a set of preconditions and a set of postconditions. The name differentiates the recognized skill from the others. The preconditions describe when the action can be performed and the postconditions describe what are the consequences of the action. The State Machine implemented by us is displayed in Figure 1. Each circle represents a discrete state of the world. The arrows represent transitions managing the passage from one state to another when a condition is met. Looking at Figure 1, we can distinguish three different conditions: HandEmpty, ObjectAbove and ObjectInTouch. These are functions that receive as input some variables obtained by querying the world model and output a logical state. In particular, HandEmpty queries the hand state; if the hand contains an object, it returns true. ObjectAbove checks if the input object is above another without wondering about which cube is above. Finally, ObjectInTouch detects if the input object is in contact with another object, if so, it returns true.

2.2. Conditions Learning

After correctly classifying the action the user performed, it is fundamental to extract only the relevant predicates that constitute the preconditions and postconditions. In fact, a robot can execute a skill if and only if the skill preconditions are met. The postconditions verify its correct execution, and they must be checked and satisfied. Therefore we implemented a set of binary state variables (also called predicates) to describe the world state. Based on the state before and after an action execution, we set the action's preconditions and postconditions. Table 1 explains how each predicate is evaluated. Algorithm 1 shows the strategy used to set preconditions and postconditions. In particular, the effects (postconditions) of the actions performed during the demonstration by the human are learned by comparing the initial state of the objects involved with their final state, detecting what has changed. Firstly, as long as the skill is not over, it is checked if an object passes close to the operator's hands for the first time. If true, the object's initial state is evaluated and stored in the InteractingObjects list (Alg.1, Line from 4 to 9). Once the skill is terminated, the final state of each interacting object is computed and compared with its initial state. The predicates that change their values from the initial state to the final one are stored in the postconditions of the recognized skill (Alg.1, Line 10). In order to learn when the actions can be performed (preconditions) it is not sufficient to select only the predicates that change their values as for the postconditions. We also need to account for predicates that do not change their value during the skill demonstration but are fundamental for correctly representing the action. In particular, the relevant parameters for the skills are those objects on which the skill is performed. For example, during a pick demonstration, we

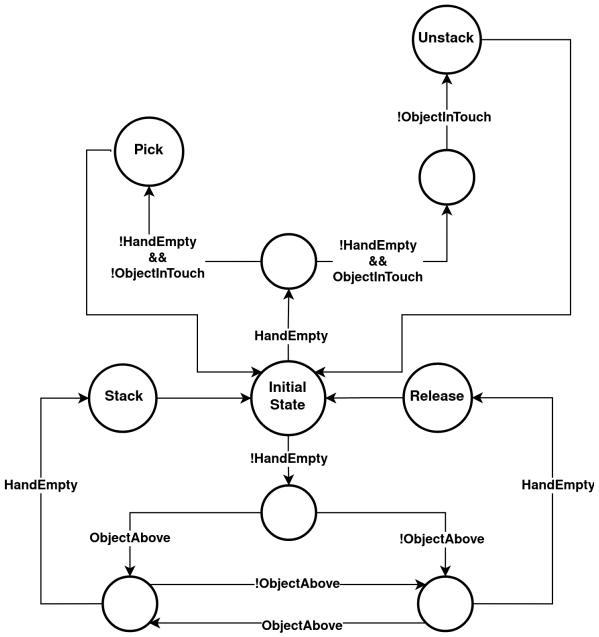

Figure 1: The figure shows the State Machine implemented.

Table 1: The table shows the defined predicates, and their respective grounding during demonstration.

predicate	Grounding
IsObjectInteractable(Obj_1*)*	An Obj_1 is interactable when no objects are on it.
IsGrasped(Obj)	An object is grasped when it is inside the user's hand.
IsReachable(Obj)	The *Obj* is reachable when is within the robot workspace.
IsAbove(Obj_1*,* Obj_2*)*	The predicate evaluates to true when Obj_1 is above Obj_2.
IsInTouch(Obj_1*,* Obj_2*)*	When Obj_1 is touching Obj_2 the predicate evaluates true.
IsGripperEmpty(Gripper)	The predicate returns true when the user's hands are free.

must learn that the picked object has to be in the reachable workspace. Otherwise, the robot will try to pick the object even if it is out of the reachable workspace. In this example, the relevant parameter is the object picked by the human during the demonstration and the value of the predicate IsReachable will not change. Such objects on which the actions are performed are called Active Objects. The Active Objects are predefined and reported in Table 2. Therefore, once the skill has been recognized, the Active Objects are set according to it (Alg.1, Line 11). Then, for each object in the Active Object list, we extract only the predicates whose parameters are all part of such Active Objects. These extracted predicates are considered relevant and are stored as preconditions of the performed skill (Alg.2, Line from 12 to 22). In addition, if the predicate is in the postconditions list previously set, the value of the predicate is negated, and it is added to the preconditions.

Skill	Active Obj1	Active Obj2
Pick	PickedObj	Null
Release	ReleasedObj	Null
Stack	ObjInHand	ObjUnderHand
Unstack	ObjInHand	ObjUnderHand

Table 2: The table shows the predefined Active Objects for each skill.

2.3. Task Planning Generation

Once the demonstrated actions are classified and characterized by the preconditions and postconditions, we focus on generalizing them through a particular language called Planning Domain Definition Language (PDDL). This language allows using a planner that schedules the learnt actions to solve the planning task. The planning task is composed of the problem and the domain. The problem describes the initial robot world state and the expected goal we try to reach. The planning domain is a collection of all possible generalized actions that can be applied to achieve the desired goal. Once the user demonstration is finished, we will have a list of all the demonstrated actions together with their preconditions and postconditions, according to what we stated in Chapter 2.2. Then, we have to generate the domain file by writing the learnt actions in the PDDL language. A fundamental rule is that actions cannot be identical in

Algorithm 1 Preconditions and Postconditions learning algorithm

```
1:  InteractingObjects ← []
2:  Postconditions ← []
3:  Preconditions ← []
4:  while Skill not over do
5:    if Object not in InteractingObjects then
6:      Obj InitState = EvalauteState(Obj)
7:      InteractingObjects ← Obj
8:    end if
9:  end while
10: SetPostconditions()
11: SetActiveObjects()
12: for all Obj ∈ ActiveObjects do
13:   for j ← 1, Obj.InitState.Count do
14:     if predicate.param ∈ ActiveObjects
        then
15:       Preconditions ← predicate
16:     end if
17:     if predicate ∈ Postconditions then
18:       Negate(predicate.value)
19:       Preconditions ← predicate
20:     end if
21:   end for
22: end for
23: SetSkillPreconditions()
```

a domain file. Thus, we need to check if the actions belonging to the demonstrated actions are already present in the domain file. If not, the actions are written in PDDL and added to the domain file. This procedure is necessary because the user can perform identical actions during the demonstration. The last step necessary to complete the task planning is to generate the problem. The problem is composed of the world initial state and the goal to be achieved. The virtual scene is composed of a set of coloured cubes near the human position, a set of coloured cubes near the robot base position and a robot. The human uses the cubes near him to perform the demonstration, while the robot uses the cubes near it to execute the actions scheduled by the planner. Thus, the initial world state is defined by evaluating the state variables reported in Table 1 for each robot cube. Regarding the goal definition, once the user has terminated the demonstration, he has to place his own set of cubes in the desired positions. Then, the state of each human side cube, and the goal will be written in the problem file in PDDL form. Figure 2 shows the virtual scene that defines the initial world state and the goal, the task planning generated and what actions the planner returns for it.

2.4. Error Handling

As discussed in the previous chapter, the planner returns a sequence of high-level actions that the robot has to execute to reach the goal state. While executing an action, there is no guarantee that the robot will accomplish it correctly. Therefore, we implemented an approach that spots and deals with possible errors during the execution. Figure 3 presents the general workflow of our system. Before calling the planner, we check if the user-defined goal is already achieved. If yes, the problem is already solved. Otherwise, the planner is called and we check if a plan is found. If the plan is not found, the problem cannot be solved meaning that the actions learnt during the demonstration phase are not sufficient to reach the desired goal starting from that initial world state. On the other hand, if a plan is found, the planner has returned a sequence of actions and the execution phase starts. We take the first action of the sequence and check if the preconditions are met.

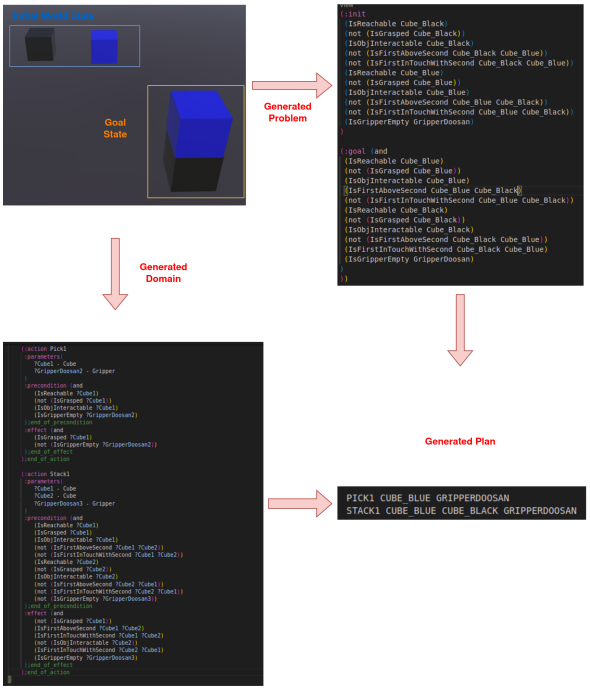

Figure 2: Example of the plan generation. The figure shows the virtual scene before the plan execution, the domain generated by the user demonstration, the problem generated and the plan generated by the Fast-Forward planner.

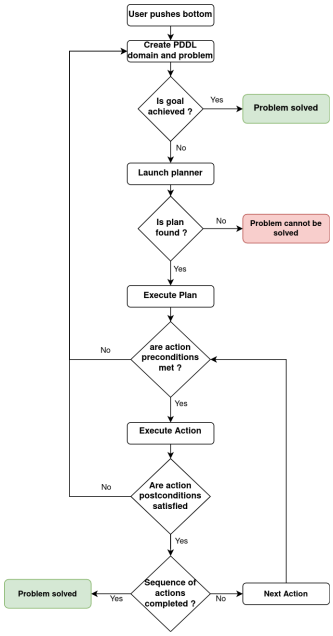

Figure 3: The figure shows the workflow of our system.

If they are not, we recompute the problem and the steps seen before are repeated. By recomputing the problem we update the initial world state of the robot side and check if something is

changed in the meanwhile. Otherwise, the robot executes the action and the postconditions are then checked. If they are not satisfied, as for the preconditions, we recompute the problem and we continue with the same procedure seen before. If they are satisfied, we check if the sequence of actions is completed. If it is not, we take the next action and repeat the steps. Vice versa, if the sequence of actions is completed, the problem is solved and we can stop. Thanks to this procedure, if an action cannot start or the robot executes it incorrectly, the plan is recomputed starting from the new initial world state until the problem is solved or the planner cannot find a new plan.

3. Experiments

3.1. Experimental setup and Use case explanation

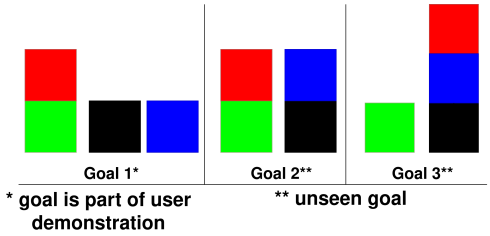

Figure 4: The figure shows the goals the robot had to reach during the experiments.

The experiments take place both in virtual and real environments. For this reason, the virtual environment has been designed to represent as close as possible the real environment. In particular, in the virtual scene a table, the Doosan A0509s robot, a set of cubes near the operator's position and another set near the robot's position are present. The human uses the set of cubes near the operator's position to perform the task demonstration and to show the actions to the robot, while the robot uses the set of cubes near it to perform the required goal. On the other hand, the real environment is composed of a table, the Doosan A0509s robot, and a set of cubes. The robot's gripper is equipped with a Realsense 435D camera that is used to monitor the scene. Furthermore, the cubes are tracked thanks to special markers attached to them, called Aruco. The difference between the real and the virtual environment is the presence of just the robot's set of cubes in the real one, as can be seen in Figure 5. This is due to the fact that to perform the experiments in the real environment, the demonstration phase is carried out in the virtual environment and only the execution phase is performed in the real environment. To test the functionality of our work, we defined three different goals the robot had to reach (Figure 4). In the training phase, we demonstrated the first goal generating a domain containing the actions: Pick, Stack, Unstack and Release.

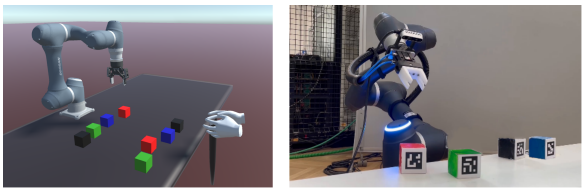

Figure 5: The left picture shows the virtual environment setup. The right picture shows the real environment setup.

3.2. Results

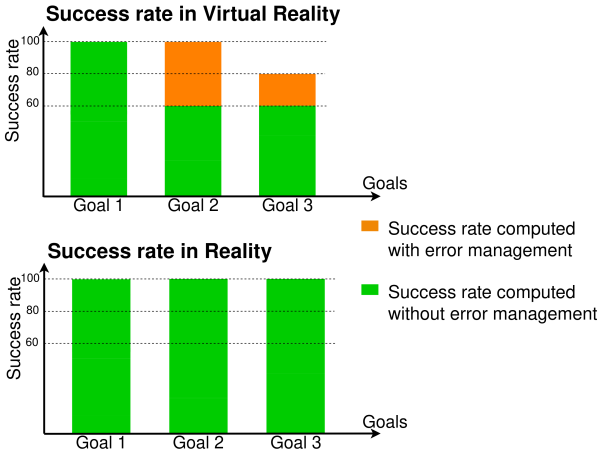

Figure 6: The figure shows the success rates achieved in the virtual and real environments for the three defined goals.

We performed each goal five times, recording the results. Figure 6 shows the success rates for the three different goals in the virtual and real environments. In green the percentage of the success rate computed without considering the error handling is represented, while in orange the percentage of the success rate achieved considering the errors management is represented. In particular, we observed that the real robot never failed to execute actions and achieved the goals without the use of the error handling procedure. In

contrast, the robot in the simulated environment required the use of the error handling procedure in the second and third goals. This is because to move the robot in the simulated environment we did use ROS, which sometimes did not find a suitable trajectory. Moreover, the packages used for the implementation of the robot in the virtual environment are in an experimental phase and sometimes cause unpredictable robot movements that do not happen with the real robot. Furthermore, during the training phase, we measured the elapsed time from the first interaction with the objects in the virtual environment to the last action performed and the time necessary to program the robot using the teach pendant which is a classic programming method in the industrial scenario. Table 3 shows the measured times necessary to program each goals. Since we could achieve the three goals with the same demonstration, the measured time necessary to program the robot is the same for each goal. As can be noticed, the required time for programming the robot with our method is lower than using the teach pendant.

Methods	Goal 1	Goal 2	Goal 3
Classical	62s	88s	117s
Our work	32s	32s	32s

Table 3: The table shows the training times required by the classical teaching and our work for programming the goal 1, goal 2 and goal 3.

Finally, we measured the time required by the planner to find a suitable plan with different PDDL domains. In particular, we generated a problem file in which the initial state describes the state of four cubes on the table, while the goal is the same as the third goal displayed in Figure 6.4. Then, we tested the planner on ten domains with an increasing number of actions. We launched the planner five times for each domain, storing the time spent to generate a plan. Figure 7 displays the results obtained, where the black line is the average time, and the red area represents the variance. It can be seen that the time required to generate a plan increases considerably as the number of skills increases. Therefore, we need to generalize as much as possible the taught actions to keep low the overall number of actions in the domain.

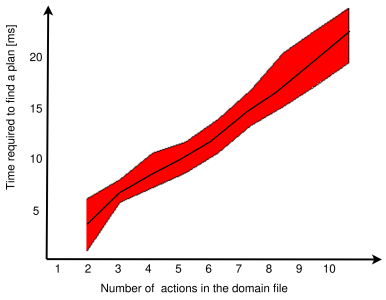

Figure 7: The figure shows the time spent by the planner to generate a suitable plan with an increasing number of actions in the domain.

4. Conclusions

In this work, a new approach is proposed that aims at programming the robots intuitively without any prior robotic knowledge. In particular, the human operator performs a series of actions in the virtual environment, and the robot understands when such actions can be performed (preconditions) and what are their consequences (postconditions). Characterizing actions with preconditions allows the robot to check if the action can be executed, while the postconditions describe what has to change after the action execution. In this way, the robot can spot and deal with possible errors and replan accordingly. The learnt actions are then generalized through a particular language called PDDL, which allows a planner to schedule the learned actions to reach a human specified goal. Finally, we performed a series of experiments to evaluate the effectiveness of our work both in simulation and with the real robot. The results highlighted how efficiently the robot learns the demonstrated actions and how fast and intuitively the operator can program the robot with respect to traditional programming methods.

References

[1] Constructions Aeronautiques, Adele Howe, Craig Knoblock, and McDermott. Pddl| the planning domain definition language. 1998.