POLITECNICO DI MILANO

School of Industrial and Information Engineering

Master of Science in Automation and Control Engineering

POLITECNICO
MILANO 1863

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

Supervisor:	Prof. Paolo Rocco
Co-supervisors:	Prof. Andrea Maria Zanchettin
	Ing. Niccolò Lucci

Master Thesis dissertation of:
Gianluca Clerici

Id: 953103

Academic year 2021-2022

"Dedication makes dreams come true" - Kobe Bryant

Ringraziamenti

Mi è doveroso dedicare questo spazio del mio elaborato alle persone che hanno contribuito, con il loro instancabile supporto, alla realizzazione dello stesso.

In primis, un ringraziamento speciale al mio co-advisor Niccolò Lucci, per la sua immensa pazienza, i suoi indispensabili consigli e per l'avermi trasmesso la sua passione e dedizione a questo lavoro. Un ulteriore ringraziamento ai professori Paolo Rocco e Andrea Zanchettin che hanno guidato la stesura del mio elaborato durante tutto il tempo senza mai risparmiare utili consigli. Più in generale vorrei ringraziare tutta la comunità politecnica che ogni giorno tramite la sua passione e il suo lavoro rende il Politecnico di Milano un luogo migliore.

Ringrazio infinitamente i miei genitori e fratelli che mi hanno sempre sostenuto, appoggiando ogni mia decisione, fin dalla scelta del mio percorso di studi.

Un grazie di cuore ai miei colleghi ed amici Matteo Ubiali, Alberto Frezza, Matteo Casu, Luca Tornaquindici, Gabriele Gatti, Riccardo Bisogni con cui ho condiviso l'intero percorso universitario. È grazie a loro che ho superato i momenti più difficili. Senza i loro consigli, non ce l'avrei mai fatta.

Contents

List of Figures

List of Tables

Abstract

One among the most significant challenges for industrial robotics is to reduce the expertise and efforts required to program a robot. Usually, to program a robot, the operator needs some technical knowledge for example on Euler angles or Cartesian coordinates. These skills are not common in people who work in Small Medium Enterprises (SMEs) and this might be a barrier for a wider use of robotics. Therefore, we developed an approach that aims at programming the robots intuitively without any prior robotic knowledge. In particular, the human operator performs a series of actions in a virtual environment, and the robot understands when such actions can be performed and what are their consequences. The learnt actions are then generalized through a particular language called PDDL which allows a planner to schedule the learned actions to reach human specified goals. In fact, after the teaching phase, the user defines a goal and the symbolic planner returns a series of actions to achieve it. The generated plan is performed both in the virtual and real environment by the collaborative robot Doosan A0509s.

Sommario

Una tra le sfide più importanti per la robotica industriale è la riduzione della competenza e degli sforzi richiesti per programmare un robot. Solitamente, per programmare un robot l'operatore deve possedere una conoscenza tecnica riguardante per esempio gli angoli di Eulero o le Coordinate cartesiane. Tali conoscenze non si trovano facilmente in persone che lavorano nelle Piccole e Medie Imprese, il che può costituire una barriera a un uso pervasivo più dei robot. Motivati da questa considerazione, in questa tesi si sviluppa un approccio che consente di programmare i robot intuitivamente e senza alcuna conoscenza tecnica. In particolare, l'operatore umano esegue una serie di azioni nella realtà virtuale e il robot impara quando tali azioni possono essere eseguite e quali sono le loro conseguenze. Le azioni imparate sono poi generalizzate utilizzando un linguaggio chiamato PDDL, il quale permette ad un pianificatore di stabilire quali azioni eseguire e in che ordine per raggiungere l'obiettivo specificato dall'umano. Dopo aver eseguito la fase di insegnamento, l'utente definisce un obiettivo e il pianificatore simbolico restituisce una serie di azioni per raggiungerlo. Il piano generato è eseguito sia nell'ambiente virtuale che in quello reale grazie al robot collaborativo Doosan A0509s.

Chapter 1

Introduction

In recent years, significant progress has been made in robot technology with the aim of allowing robots to work alongside humans. These robots are called collaborative robots since they are used as a tool for factory workers. In particular, they are characterized by a lightweight design, low speeds and low payload. Their advent has revolutionized the old idea of heavy and dangerous robots as the industrial one. Figure 1.1 shows a comparison between a cobot and an industrial robot. It can be seen that the collaborative robot on the left side of the figure has a different shape from the big industrial one on the right side: it has rounded edges and it is constituted of soft and light material necessary to absorb possible impacts with the human.

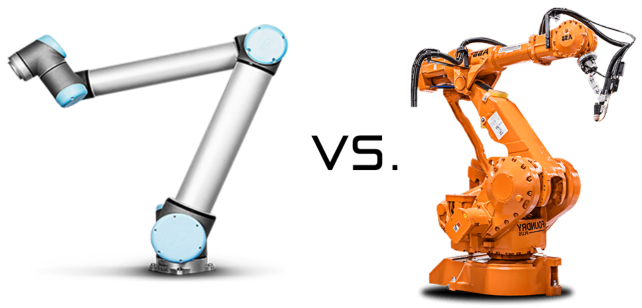


Figure 1.1: The figure shows a comparison between a cobot and an industrial robot.

The introduction of these robots in the industrial scenario has brought the possibility of introducing them into Small and Medium Enterprises alongside humans. In fact, they can be integrated in small environments since they are small, they do not need to be surrounded by cages and they are easy to set up and integrate in a production line. On the other hand, the current market demand requires high product customization. Therefore, the production line has to be highly flexible. This can be achieved by having skilled personnel capable of reprogramming the robots in order to adapt them to the new production line. Unfortunately, finding robotic experts is not that easy, thus, integrating the collaborative robot into a new production line can be difficult, time-consuming and expensive. The solution to the problem can be found using the new technologies available in the market, such as augmented reality and virtual reality, to develop new intuitive robot programming methods.

1.1 Virtual Reality in industrial systems

Virtual Reality (VR) is one among the main pillars of industry 4.0. Nowadays, multiple VR devices are available in the market at affordable prices. They do not need computers or other external devices to work and are lightweight. After wearing the headset, the user is introduced in a simulated environment where he/she can interact with objects using his/her hands thanks to the provided controllers. The first use case for virtual reality was gaming. Currently, Virtual Reality can be exploited in various industrial fields such as product design, machine control and training professionals. For example, in Figure 1.2, a virtual environment is used to see the progress in the design of a car. Virtual Reality, in this case allows seeing a preview of the car without building a physical prototype. Even if Virtual Reality is still limited, it can be used by enterprises to remain competitive in Industry 4.0, as stated in [1]. For example, it can be useful for maintenance during the industrial product lifecycle and for improving the design processes. One among the major applications of Virtual Reality in Industry 4.0 is training of the operators, in particular related to robot programming. Thanks to the possibility of customizing the virtual environment it is possible to represent different environments depending on the context requirements the headset is used in. Therefore, the operator can be trained without distracting the robot from production and then stopping the production line. This minimizes the production line downtime and avoids injuries that can happen while manipulating the tools. In this thesis work, Virtual Reality is a fundamental concept since the human operator has to demonstrate actions that the robot will use in the execution phase.

Figure 1.2: The figure shows an example of how the Virtual Reality can be used in the product design.

1.2 Automated Planning

Automated planning is a field of Artificial Intelligence which aims to solve planning problems. The robot Shakey (Figure 1.3) was the first robot to make use of this branch of Artificial Intelligence to solve tasks that necessitated planning. In particular, it was capable of autonomously rearranging objects and finding routes. This robot gave proof of the capabilities of automated planning and it inspired many researchers. Nowadays, the planning community is well-established and organizes competitions to overcome the gap between planning research and application. In order to define the planning problems, Planning Domain Definition Language (PDDL) can be used. The planning problem, also known as planning task, is composed of the problem and the domain. The problem describes the initial world state the agent is in 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. There are many different PDDL versions, and each one provides a different level of expressivity. For example, in PDDL 1.2, the problem is defined as a set of predicates describing the state that the robot starts from and the state the agent needs to reach. The planning domain is made of a set of actions, for each of them it is specified when they are applicable (preconditions) and how they modify the environment (postconditions). PDDL 2.1 introduced the possibility of defining a durative action which has similar properties of a classic action definition, but it introduces a duration parameter which models the length of time the action takes. In order to solve the planning task, a planner is required. A planner has the objective of scheduling the required skills described in the domain file to pass from the initial state to the final one descibed in the problem. In our thesis work, we chose the Fast Forward planner to devise the planning task.

Figure 1.3: The figure shows the robot Shakey.

1.3 Thesis purpose

This thesis aims at developing a new approach for intuitively programming a robot. The operator has to demonstrate actions in a virtual environment, and the system has to classify the actions and characterize them with just relevant preconditions and postconditions. Then, the learnt actions are written in a planning domain file using the PDDL language. 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 used 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 to perform unseen tasks and gives the robot the ability to reason.

1.4 Thesis achievements

In this thesis, the following objectives have been achieved:

- The system automatically generates the planning domain file from the user demonstrations. It also automatically generates the problem file requiring just the goal definition.
- The system automatically recognizes the demonstrated actions and characterizes them with just the relevant preconditions and postconditions.
- The system detects and deals with possible errors.

1.5 Thesis structure

In Chapter 2 it is discussed how the problem of intuitive robot programming is dealt with in the State-of-the-Art. Chapter 3 explains how the preconditions and postconditions of skills are set after the user demonstration. Then, how the planning task is automatically generated after the user demonstration is described in Chapter 4. Chapter 5 discusses how the robot executes the actions and how it spots and deals with eventual errors. In Chapter 6, it is explained how we conducted the validation part and experiments results are commented. Finally, Chapter 7 focuses on the conclusions and on the possible improvements.

Chapter 2

State of the art

Nowadays, one among the biggest challenges in small and medium-sized enterprises (SMEs) is reducing the robot programming effort for the operator. Some proposed solutions to overcome this problem are analyzed in [2]. Here the author focused on CAD-Based programming and Programming by demonstration with the final objective of reducing the complexity of programming a robot. Other examples of the first approach are [3], [4] and [5]. These papers show a novel robot programming paradigm based on Web Ontology Language (OWL), where the user specifies the robot tasks in terms of the involved objects and the relevant parameters, as illustrated in Figure 2.1. Then, the extrapolated parameters are used to characterize the skills necessary to complete the task. These approaches don't require a robot expert but just that the worker is aware of how to perform the task. Furthermore, this method doesn't require the real robot, consequently the robot can be programmed without removing it from its current task. This minimizes the robot production downtime. On the other hand, they don't allow process monitoring and error handling during task execution. A more in-depth understanding of tasks, and the skills that compose it, is necessary to implement a method that guarantees process monitoring and error handling. A task is a sequence of skills needed to reach the desired goal. The skills are actions that modify the state of the world, as explained in [6]. Studies have been developed in [7] and [8] to highlight the general structure of a robot skill. We can identify three fundamental parts which are reported in Figure 2.2: the skill preconditions, the

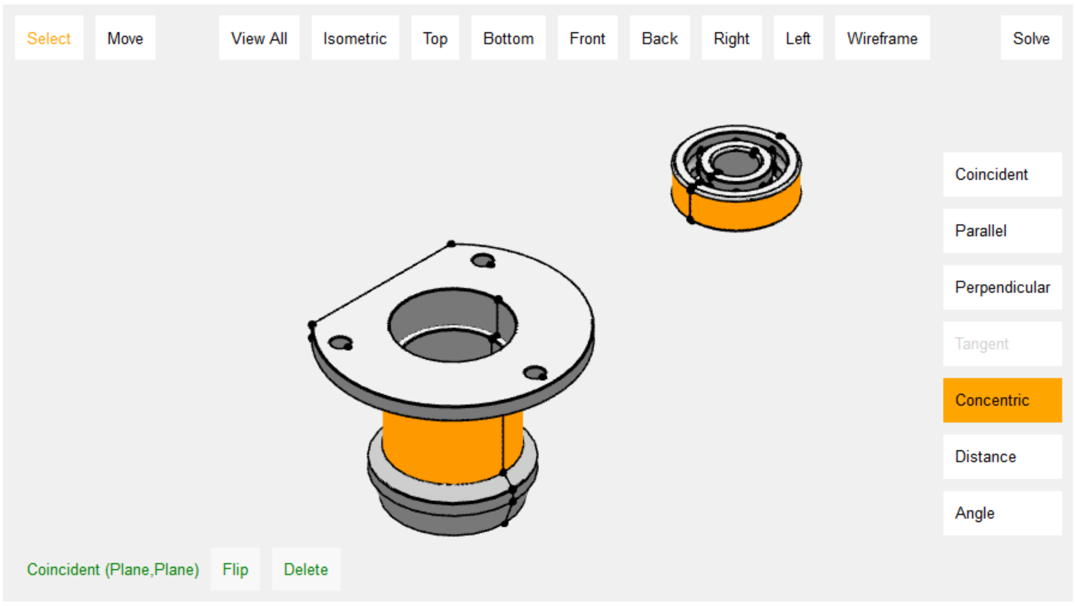


Figure 2.1: Graphic User Interface used in [3], [4] and [5].

skill execution and the skill postconditions. If the preconditions are met, the skill is executed, and then the postconditions are checked. The described procedure guarantees the process monitoring and the eventual error handling of the overall process.

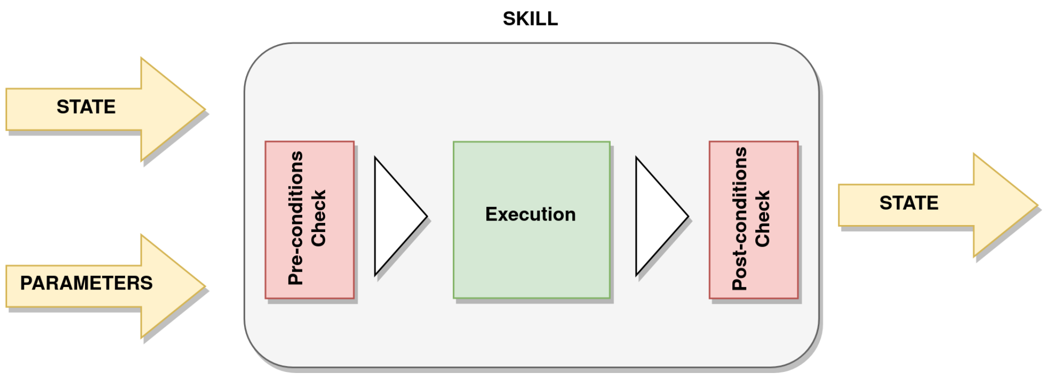


Figure 2.2: Model of a skill

An in-depth literature review has been performed based on the robot skill model presented before. In particular, Programming by Demonstration (PbD) characterizes the robot skill model through user demonstrations. In Chapter 2.1, we will present different works that use this approach. Finally, in Chapter 2.2 we will present methods that involve Virtual Reality, as a tool to simplify robot programming.

2.1 Programming by Demonstration

As stated in [9], Programming by demonstration refers to transferring new skills to a machine by relying on demonstrations from a user. The main demonstration modalities are kinesthetic teaching and observation learning. In the first one, the operator guides the robot manually, allowing the storage of data such as the trajectory and the force applied to the end-effector. Data collected during the teaching phase characterize the taught skill. A user-friendly approach for robot programming is to combine a Graphical user interface (GUI) with kinesthetic teaching, as shown in [10] and [11]. In [10], two phases are necessary to program the robot. The first one is the specification phase, where the operator uses the GUI to define the skill sequence required to complete the task. The second one is the teaching phase, where using kinesthetic teaching, the previously defined sequence is taught. In [11], kinesthetic teaching is used to infer preconditions and effects of skills which are then transformed into the Planning Domain Definition Language (PDDL)[12]. PDDL is a language that allows to describe each possible robotic action with its preconditions and effects. Consequently, after such specifications, a planner is able to sequence each of these characterized skills depending on the required task. In [11], the GUI is used to create new actions, modify inferred types or predicates, create and solve new problems with the task planner. Therefore, only one phase is required to program the robot. However, actions are taught in a real environment which involve sensor inaccuracy, and they do not account for the case where an action is performed incorrectly (error handling). Also in [13], kinesthetic teaching is used to learn the preconditions and effects of skills based on a small number of user demonstrations. The learned actions are transformed into the Planning Domain Definition Language, and then a planner generates a suitable plan to reach the specified goal starting from the initial state of the world. This procedure allowed the authors to handle skill execution errors and, if found, replan the task starting from the last state reached. Two central problems arise. The first one is that using the real environment means dealing with continuous variables, which contrasts with PDDL. The second one, instead, is the correct selection of preconditions since false positives could be included. In our work we decided to choose observation learning so to avoid the waste of time the operator produces when moving the real robot during the kinesthetic teaching phase, reduce the costs related to physical sensors and implement the demonstrations in a virtual environment, as discussed in the next chapter. In [14], the human operator provides task demonstrations in a real environment. This demonstration is processed to learn distinct sub-actions. Afterwards, the learned sub-actions are matched with predefined actions called Object-Action Complexes (OACs). The robot performs the task following the sequence of OACs obtained. The performed actions during robot guidance are difficult to segment in a real environment due to sensor inaccuracy, as explicitly stated in [15]. Furthermore, predefined actions limit the robot's performance since preconditions and effects are not learned during the demonstration. In [16] and [17], observation learning is used in a virtual environment. They infer the planning domain from the virtual demonstration. Once the planning domain is created in [16], a plan that satisfies a user-defined goal is generated using a symbolic planner. This method allows performing unseen tasks given a demonstration. On the other hand, error handling is not considered. The approach developed in [17] learns the skills preconditions and postconditions from the user demonstration and uses them to reproduce the seen task. Here, performing unseen tasks is not possible. However, the user manages errors directly when they are detected.

2.2 Virtual Reality

Building physical training environments is very costly, time-consuming and difficult. Nevertheless, tight constraints must be respected if human-robot interaction is required. Thanks to technological advancements, we can represent realistic environments using Virtual Reality (VR). A first example that combines PbD and VR is [16], where a system for automated domain generation is built.

Here a virtual robot is added to the environment, as in [17]. Therefore, all the activities are performed in VR, such as observation learning and skill execution. This is due to the fact that the real environment presents unpredictable situations such as sensor inaccuracy and object imperfections. In [18], a decision tree is used to recognize and extracts assembly activities performed by humans in VR. We implemented a similar method to classify human actions.

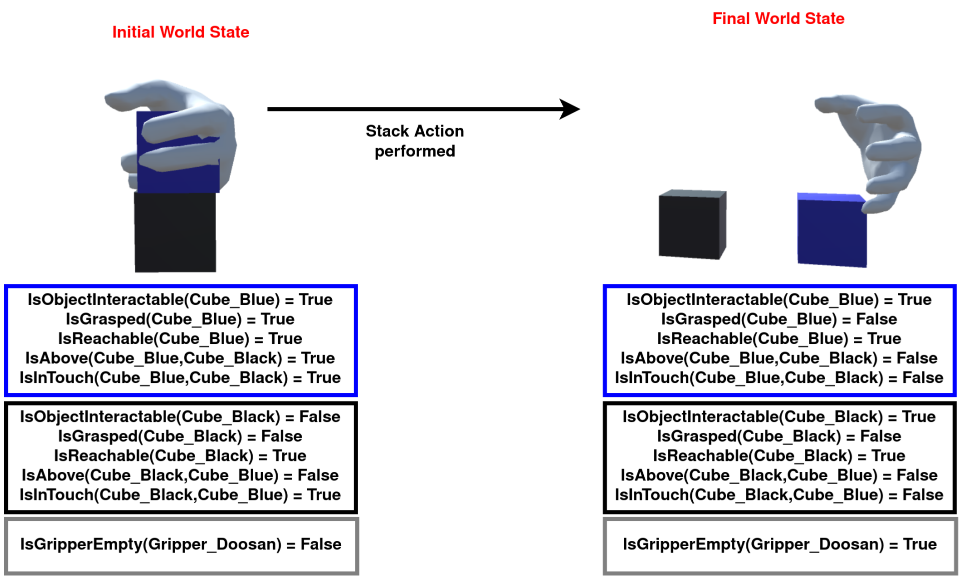
After this literature review, we decided to implement an approach based on PDDL, which extracts domain and problem automatically from human demonstrations performed inside Virtual Reality. Once each skill has been characterized, we use a symbolic planner to generate a plan necessary to reach the defined goal. In particular, the strengths of this work lies in the ability to learn relevant preconditions and postconditions from a few demonstrations. With one single demonstration for each skill, the robot is able to execute the demonstrated task and perform unseen tasks. Furthermore, during the execution phase error handling and process monitoring are considered thanks to a closed loop control performed for each skill checking preconditions and postconditions.

Chapter 3

Conditions Learning

In this Chapter, we will discuss how skill preconditions and postconditions are set. A robot can execute a skill if and only if the skill preconditions are met. Then, to verify its correct execution, postconditions must be checked and satisfied. The skill execution fails if at least one postcondition is not satisfied. For example, if the robot has to pick a cube, the robot's gripper must be free (*IsGripperEmpty*), and the cube must be in the robot's reachable workspace (*IsReachable*). If even one of these conditions is not satisfied, the skill cannot be applied. Conversely, if the robot has picked the cube, the robot's gripper must be full. If this condition is not met, the skill execution is considered as failed. When the operator performs an action the world state changes, passing from the state before the action execution (inital state) to the state after the action execution (final state), as illustred in Figure 3.1. Since we aim to use the PDDL, we need to learn preconditions and postconditions based on the initial and final world state.

We decided to implement a set of binary state variables (also called predicates) to describe the world state. The major challenge is to extract only the relevant predicates that characterize the performed actions. Returning to our previous example, if the user demonstrates a pick skill, there is no reason to consider the condition where the cube is touching another cube ($IsInTouch$) as a precondition for the pick skill. In fact, if we set $IsInTouch$ as a precondition for the pick skill, the pick skill will be more specific, and we will need two different demonstrations for it, one when the cube is not touching another cube and one when the



IsObjectInteractable(Cube_Blue) = True
IsGrasped(Cube_Blue) = True
IsReachable(Cube_Blue) = True
IsAbove(Cube_Blue,Cube_Black) = True
IsInTouch(Cube_Blue,Cube_Black) = True

IsObjectInteractable(Cube_Black) = False
IsGrasped(Cube_Black) = False
IsReachable(Cube_Black) = True
IsAbove(Cube_Black,Cube_Blue) = False
IsInTouch(Cube_Black,Cube_Blue) = True

IsGripperEmpty(Gripper_Doosan) = False

IsObjectInteractable(Cube_Blue) = True
IsGrasped(Cube_Blue) = False
IsReachable(Cube_Blue) = True
IsAbove(Cube_Blue,Cube_Black) = False
IsInTouch(Cube_Blue,Cube_Black) = False

IsObjectInteractable(Cube_Black) = True
IsGrasped(Cube_Black) = False
IsReachable(Cube_Black) = True
IsAbove(Cube_Black,Cube_Blue) = False
IsInTouch(Cube_Black,Cube_Blue) = False

IsGripperEmpty(Gripper_Doosan) = True

Figure 3.1: The world state changes due to the stack action performed by the operator.

cube is touching another cube.
On the other hand, if we don't set $IsInTouch$ as a precondition for the pick skill, this can be executed both if the cube is touching another cube or if it is not touching another cube by executing the skill once. It is clear then that it is fundamental to correctly extract the relevant predicates to have actions that are as general as possible.
In Chapter 3.1, we will present the list of predicates together with their evaluations that characterize the environment. In Chapters 3.2 and 3.3, the concept of active objects for skill postconditions and preconditions is introduced, and then the learning algorithm is presented.

3.1 List Of Predicates

This section shows the list of predicates, their evaluation and some illustrative examples that clarify their functioning.

3.1.1 IsGrasped(*Object*)

IsGrasped takes as input an object, and returns true when the object is grasped by the operator's left hand, otherwise it returns false. A pratical example is shown in Figure 3.2, where the red cube is grasped by the operator's left hand and the blue cube is not grasped.

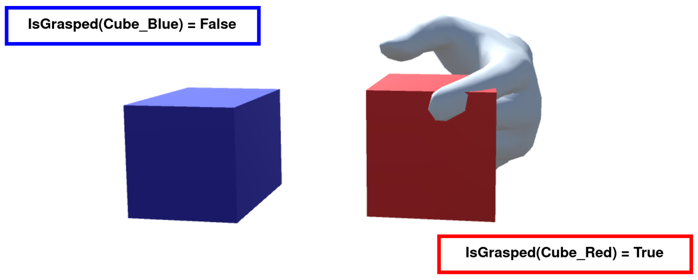


Figure 3.2: This picture shows the evaluation of IsGrasped during a operator demonstration. The operator's left hand is grasping the red cube, thus IsGrasped evaluated on the red cube returns true. While the blue cube is not grasped, so IsGrasped evaluation on the blue cube returns false.

3.1.2 IsGripperEmpty(*Gripper*)

IsGripperEmpty takes as input a gripper object, and returns true when an interaction between the gripper and an object is detected, otherwise it returns false. In this work, we use only one robot; therefore, the same gripper will always be evaluated during execution. However, we implemented it generically for future improvements regarding the use of more robots. During the demonstration, the operator performs actions on objects therefore we set the boolean value of this predicate, looking at the operator's left hand as if it is the robot gripper. It returns true when a object is inside the hand, vice versa it returns false. Figure 3.3 shows IsGripperEmpty evaluation during a demonstration, the operator's left hand is grasping the red cube therefore the value of the predicate will be true.

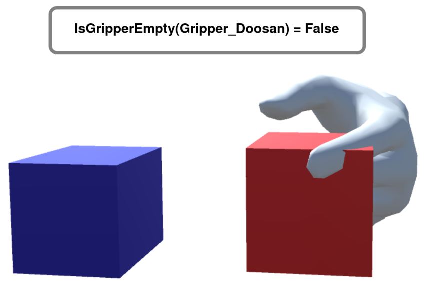


Figure 3.3: The picture shows the evaluation of IsGripperEmpty during a operator demonstration. The operator's left hand is grasping the red cube, thus IsGripperEmpty returns false.

3.1.3 IsAbove($Object_1, Object_2$)

IsAbove takes as input two objects, and returns true when the $Object_1$ is above $Object_2$, otherwise it returns false. To evaluate the value of this predicate, a ray is projected downwards starting from Object1, and the true value is returned if the ray intersects with Object2, otherwise false. Figure 3.4 shows a situation in which the black cube is above the blue cube and the red cube on the ground.

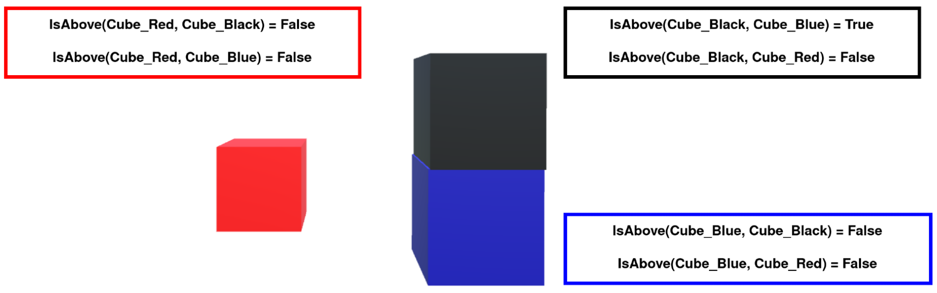

IsAbove(Cube_Red, Cube_Black) = False
IsAbove(Cube_Red, Cube_Blue) = False
IsAbove(Cube_Black, Cube_Blue) = True
IsAbove(Cube_Black, Cube_Red) = False
IsAbove(Cube_Blue, Cube_Black) = False
IsAbove(Cube_Blue, Cube_Red) = False

Figure 3.4: The picture shows the evaluation of IsAbove. The black cube is above the blue one, thus IsAbove($Cube-Black, Cube-Blue$) returns true. The other evaluations of IsAbove return the false value since the blue and red cube are not above other cubes.

3.1.4 IsInTouch($Object_1, Object_2$)

IsInTouch takes as input two objects, and returns true when $Object_1$ is in touch with $Object_2$, otherwise it returns false. Each object in the scene has a list of objects that are in contact with it. If $Object2$ is in the list of objects in contact with $Object1$, IsInTouch returns true, otherwise false. When contact between objects is detected, the respective lists of objects in contact are updated. On the contrary, when contact is lost the objects are removed from the respective lists. Figure 3.5 illustrates an example where the black cube is touching the blue cube and the red cube is not touching other cubes.

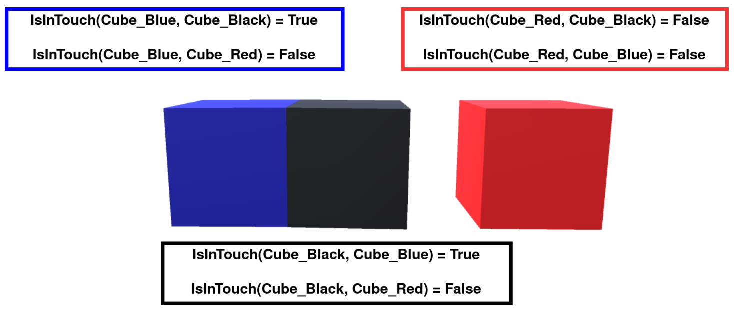

IsInTouch(Cube_Blue, Cube_Black) = True
IsInTouch(Cube_Blue, Cube_Red) = False
IsInTouch(Cube_Red, Cube_Black) = False
IsInTouch(Cube_Red, Cube_Blue) = False
IsInTouch(Cube_Black, Cube_Blue) = True
IsInTouch(Cube_Black, Cube_Red) = False

Figure 3.5: The picture shows the evaluation of IsInTouch. The black cube is touching the blue one and vice versa. Therefore, the IsInTouch predicate, having as input the blue and black cube, returns true, while the evaluation of IsInTouch for the red cube with the black and the blue one returns false.

3.1.5 IsObjectInteractable($Object$)

IsObjectInteractable takes as input an object, and returns true when no objects are above it, otherwise it returns false. The predicate implementation makes use of IsAbove seen in sub-chapter 3.1.3 to detect if other cubes are above it. In Figure 3.6 the red and black cubes are interactable since there are no cubes above them while the blue cube is not interactable because the black cube is above it.

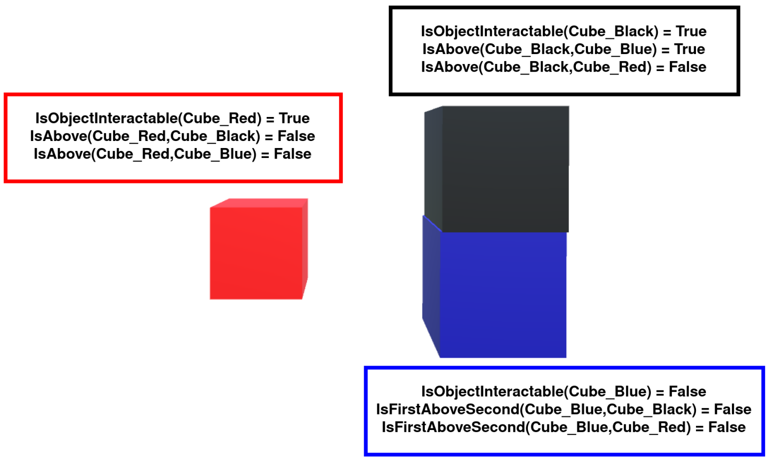

IsObjectInteractable(Cube_Black) = True
IsAbove(Cube_Black,Cube_Blue) = True
IsAbove(Cube_Black,Cube_Red) = False
IsObjectInteractable(Cube_Red) = True
IsAbove(Cube_Red,Cube_Black) = False
IsAbove(Cube_Red,Cube_Blue) = False
IsObjectInteractable(Cube_Blue) = False
IsFirstAboveSecond(Cube_Blue,Cube_Black) = False
IsFirstAboveSecond(Cube_Blue,Cube_Red) = False

Figure 3.6: The picture shows the evaluation of IsObjectInteractable. The black cube is above the blue one, thus the IsAbove($Cube-Black, Cube-Blue$) returns true while IsObjectInteractable evaluated for the blue cube returns false. The black and red cube have no cubes above them, so IsObjectInteractable evaluation for them returns true.

3.1.6 IsReachable($Object$)

IsReachable takes as input an object, and it returns true if the object is in the robot workspace (rw), vice versa it returns false. If the inequality (3.13) holds the object is in the robot workspace, and the output of the predicate will be true, otherwise false. The following formula regulates the predicate output

$$||p_{obj} - p_{robot}||_2 < rw \tag{3.1}$$

where p_{robot} is the robot base position and p_{obj} is the object position. In Figure 3.7 the green line delimitates the robot workspace, the black cube is inside it while the blue cube is outside.

A final example that shows how the world's state is described by the predicates is illustrated in Figure 3.8

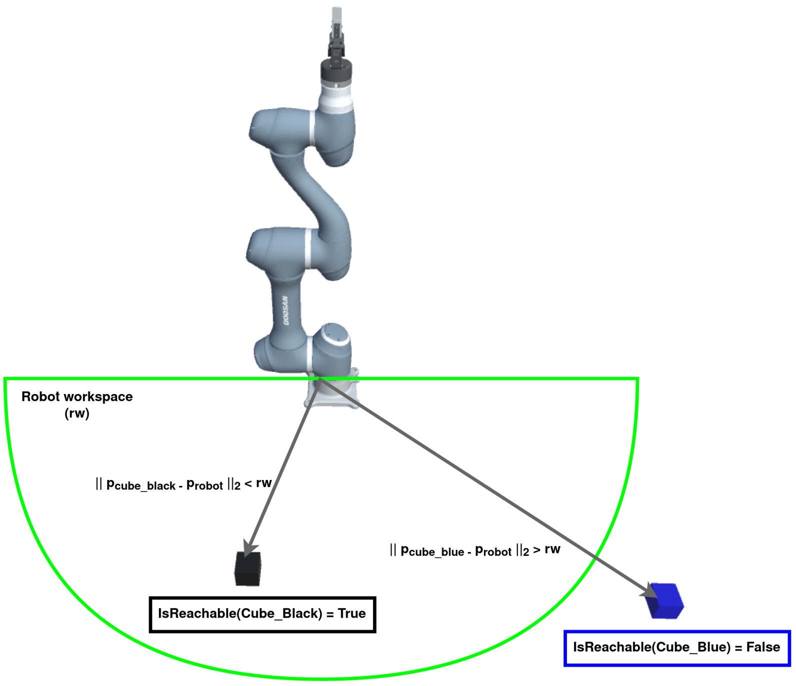

Robot workspace
(rw)
|| Pcube_black - Probot ||2 < rw
|| Pcube_blue - Probot ||2 > rw
IsReachable(Cube_Black) = True
IsReachable(Cube_Blue) = False

Figure 3.7: The picture shows the evaluation of IsReachable. The black cube is inside the robot workspace while the blue one is outside it. Therefore, IsReachable returns true only for the black cube.

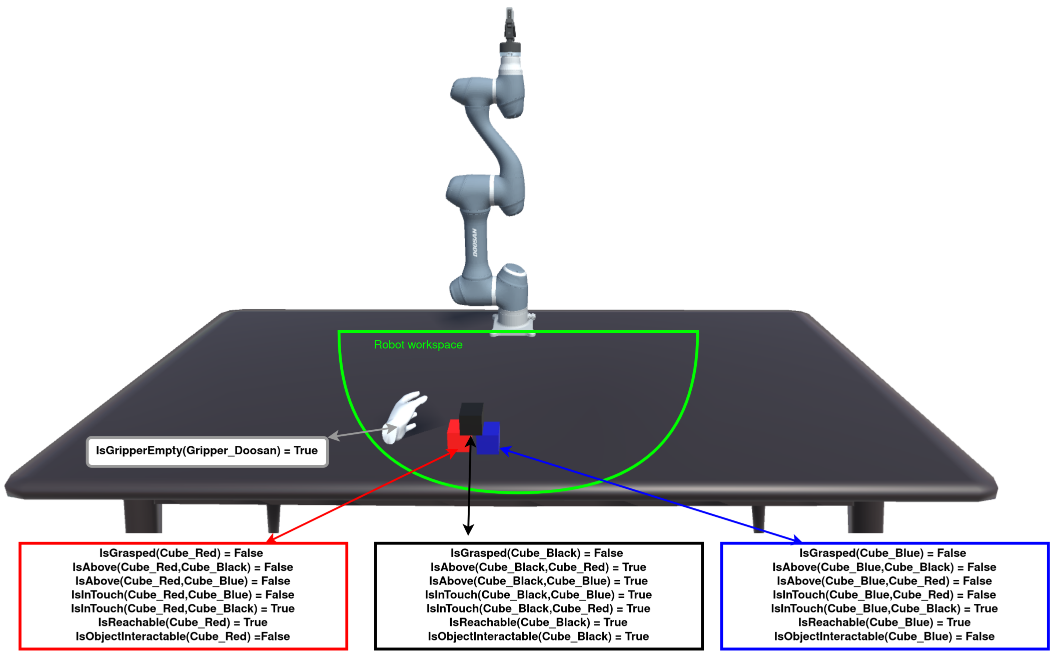

Robot workspace
IsGripperEmpty(Gripper_Doosan) = True
IsGrasped(Cube_Red) = False
IsAbove(Cube_Red,Cube_Black) = False
IsAbove(Cube_Red,Cube_Blue) = False
IsInTouch(Cube_Red,Cube_Blue) = False
IsInTouch(Cube_Red,Cube_Black) = True
IsReachable(Cube_Red) = True
IsObjectInteractable(Cube_Red) =False
IsGrasped(Cube_Black) = False
IsAbove(Cube_Black,Cube_Red) = True
IsAbove(Cube_Black,Cube_Blue) = True
IsInTouch(Cube_Black,Cube_Blue) = True
IsInTouch(Cube_Black,Cube_Red) = True
IsReachable(Cube_Black) = True
IsObjectInteractable(Cube_Black) = True
IsGrasped(Cube_Blue) = False
IsAbove(Cube_Blue,Cube_Black) = False
IsAbove(Cube_Blue,Cube_Red) = False
IsInTouch(Cube_Blue,Cube_Red) = False
IsInTouch(Cube_Blue,Cube_Black) = True
IsReachable(Cube_Blue) = True
IsObjectInteractable(Cube_Blue) = False

Figure 3.8: The picture shows the evaluation of the state of the world.

3.2 Postconditions learning Algorithm and Active Objects

The postconditions learning algorithm uses the same strategy as in [17]. In particular, the effects 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. Since the operator demonstrates skills with his hands, if an object passes close to them for the first time, the object's state is evaluated and stored in a list of $InteractingObjects$ as long as the skill is not over. Such list contains all the objects that the hand has interacted with. To define when an object is near the operator's hands, we decided to use a fictitious sphere placed in the centre of the hands, as Figure 3.9 shows. Once the skill is terminated, for each interacting object, its initial state is compared with its final state which is computed once an action has finished. The predicates that change their values are stored in the postconditions of the recognized skill. Finally, the object is removed from the interacting object list. The procedure discussed above is reported in Algorithm 1.

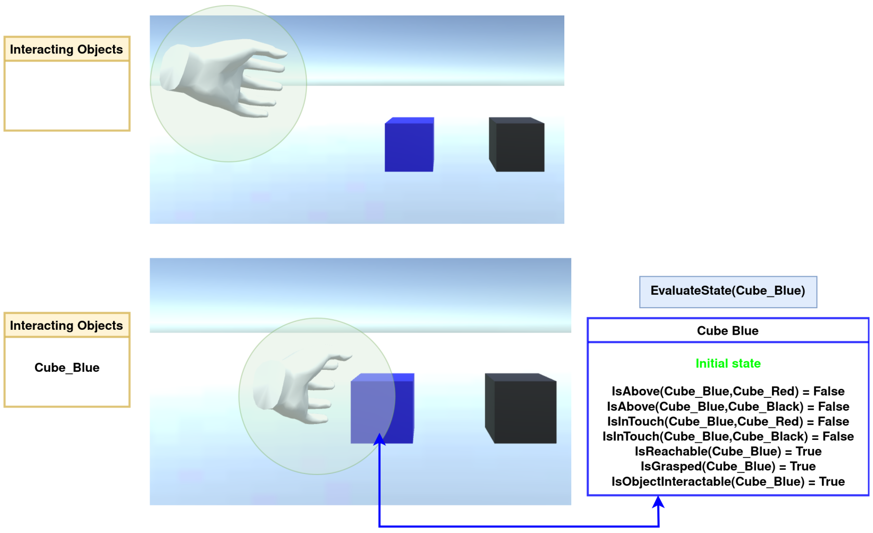

Interacting Objects
Interacting Objects
Cube_Blue
EvaluateState(Cube_Blue)
Cube Blue
Initial state
IsAbove(Cube_Blue,Cube_Red) = False
IsAbove(Cube_Blue,Cube_Black) = False
IsInTouch(Cube_Blue,Cube_Red) = False
IsInTouch(Cube_Blue,Cube_Black) = False
IsReachable(Cube_Blue) = True
IsGrasped(Cube_Blue) = True
IsObjectInteractable(Cube_Blue) = True

Figure 3.9: In the upper image the operator's left hand is far away the blue cube. Then the operator moves it near the blue cube, thus the state for the blue cube is evaluated and finally it is added to the interacting objects list.

Algorithm 1 Postconditions learning algorithm

```
 1: $InteractingObjects \leftarrow []$
 2: $Postconditions \leftarrow []$
 3: **while** Skill not over **do**
 4:     **if** InteractedObjects not in InteractingObjects **then**
 5:         $Obj\ InitState = EvalauteState(Obj)$
 6:         $InteractingObjects \leftarrow Obj$
 7:     **end if**
 8: **end while**
 9: **for each** $Obj \in InteractingObjects$ **do**
10:     $Obj\ FinState = EvalauteState(Obj)$
11:     **if** $Obj\ InitState\ ! = Obj\ FinState$ **then**
12:         $MaintainChangedstates(FinState)$
13:         $Postconditions.Add(FinState)$
14:     **end if**
15:     $Obj = InteractingObjects.RemoveFirst()$
16: **end for**
17: $SetSkillPostconditions()$
```

A series of figures that illustrates the postconditions learning algorithm is presented. The figures show the postconditions setting during a Stack Skill.
In Figure 3.10, there are two cubes in the scene and the operator's left hand, one of the two cubes is in the sphere collider of the operator's left hand. Therefore, the initial state of the objects and the hand is computed and represented by the coloured boxes in the figure.
Figure 3.11 shows that the operator's left hand has moved the blue cube near the black cube. Consequently, the black cube enters into the sphere and its initial state is evaluated.
In the end, figure 3.12 shows the final step of a stack skill, namely when the operator releases the blue cube above the black one. Once the stack operation has finished, the predicates that have changed their values are extracted and set as postconditions for the stack skill.

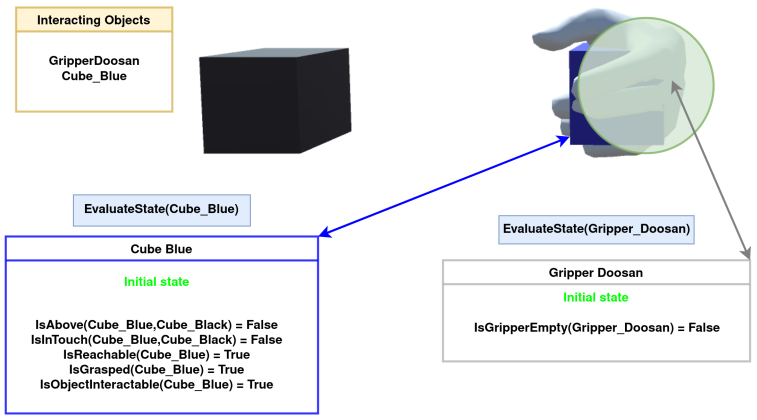

Interacting Objects
GripperDoosan
Cube_Blue
EvaluateState(Cube_Blue)
Cube Blue
Initial state
IsAbove(Cube_Blue,Cube_Black) = False
IsInTouch(Cube_Blue,Cube_Black) = False
IsReachable(Cube_Blue) = True
IsGrasped(Cube_Blue) = True
IsObjectInteractable(Cube_Blue) = True
EvaluateState(Gripper_Doosan)
Gripper Doosan
Initial state
IsGripperEmpty(Gripper_Doosan) = False

Figure 3.10: Initial state of a Stack skill. The operator's left hand is grasping the blue cube. The initial state of the blue cube, black cube and operator's left hand are computed and showed in the coloured boxes.

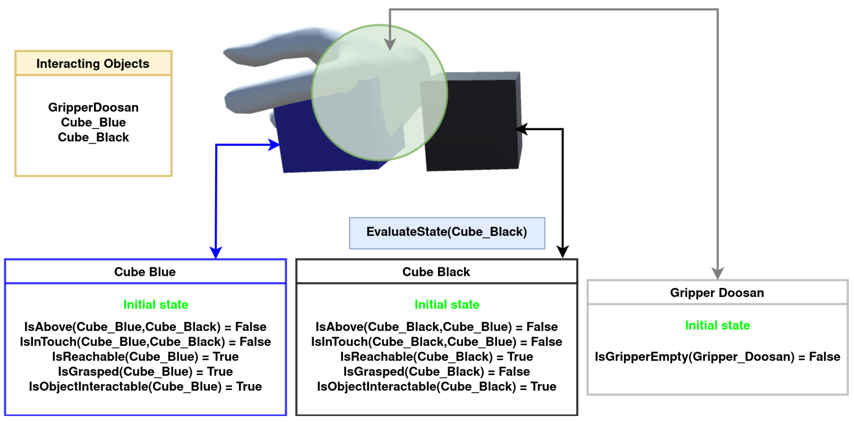

Interacting Objects
GripperDoosan
Cube_Blue
Cube_Black
EvaluateState(Cube_Black)
Cube Blue
Initial state
IsAbove(Cube_Blue,Cube_Black) = False
IsInTouch(Cube_Blue,Cube_Black) = False
IsReachable(Cube_Blue) = True
IsGrasped(Cube_Blue) = True
IsObjectInteractable(Cube_Blue) = True
Cube Black
Initial state
IsAbove(Cube_Black,Cube_Blue) = False
IsInTouch(Cube_Black,Cube_Blue) = False
IsReachable(Cube_Black) = True
IsGrasped(Cube_Black) = False
IsObjectInteractable(Cube_Black) = True
Gripper Doosan
Initial state
IsGripperEmpty(Gripper_Doosan) = False

Figure 3.11: The intermediate step of a stack skill. The operator is moving the blue cube towards the black cube. The black cube initial state is evaluated and shown by the black box once entered in the sphere.

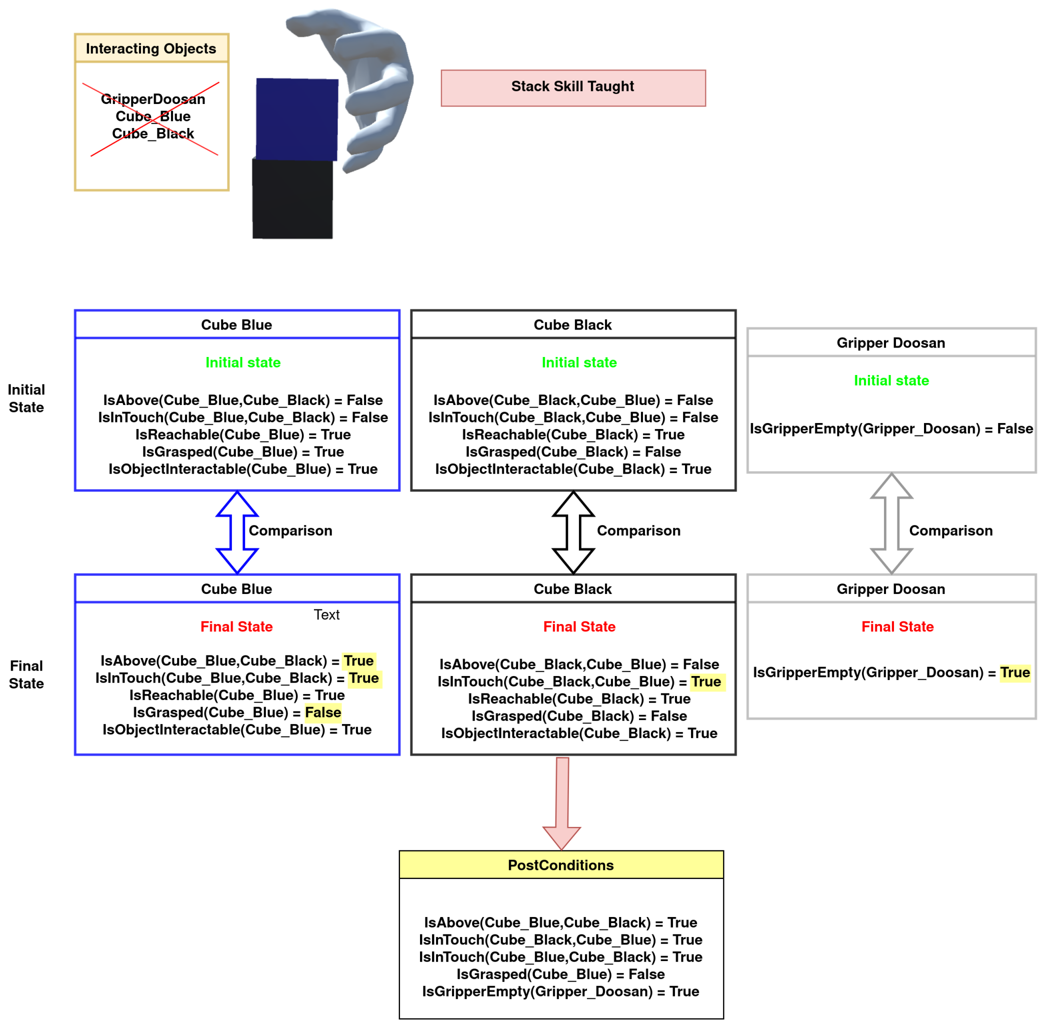

Interacting Objects
GripperDoosan
Cube_Blue
Cube_Black
Stack Skill Taught
Initial State
Cube Blue
Initial state
IsAbove(Cube_Blue,Cube_Black) = False
IsInTouch(Cube_Blue,Cube_Black) = False
IsReachable(Cube_Blue) = True
IsGrasped(Cube_Blue) = True
IsObjectInteractable(Cube_Blue) = True
Cube Black
Initial state
IsAbove(Cube_Black,Cube_Blue) = False
IsInTouch(Cube_Black,Cube_Blue) = False
IsReachable(Cube_Black) = True
IsGrasped(Cube_Black) = False
IsObjectInteractable(Cube_Black) = True
Gripper Doosan
Initial state
IsGripperEmpty(Gripper_Doosan) = False
Comparison
Comparison
Comparison
Final State
Cube Blue
Text
Final State
IsAbove(Cube_Blue,Cube_Black) = True
IsInTouch(Cube_Blue,Cube_Black) = True
IsReachable(Cube_Blue) = True
IsGrasped(Cube_Blue) = False
IsObjectInteractable(Cube_Blue) = True
Cube Black
Final State
IsAbove(Cube_Black,Cube_Blue) = False
IsInTouch(Cube_Black,Cube_Blue) = True
IsReachable(Cube_Black) = True
IsGrasped(Cube_Black) = False
IsObjectInteractable(Cube_Black) = True
Gripper Doosan
Final State
IsGripperEmpty(Gripper_Doosan) = True
PostConditions
IsAbove(Cube_Blue,Cube_Black) = True
IsInTouch(Cube_Black,Cube_Blue) = True
IsInTouch(Cube_Blue,Cube_Black) = True
IsGrasped(Cube_Blue) = False
IsGripperEmpty(Gripper_Doosan) = True

Figure 3.12: The final step of a stack skill. The operator releases the blue cube above the black one. The postconditions box shows the relevant predicates selected for the taught skill.

3.3 Preconditions learning Algorithm and Active Objects

In the previous work [17], preconditions were hard-coded for each skill. Therefore, no matter the world state, each skill will always have the same preconditions. Since we aim to use the PDDL, this is not the best approach. When the operator demonstrates the skill, we need to learn preconditions based on the current world state. In order to learn the preconditions of the performed skill, 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 a correct representation of the action. In particular, the relevant parameters for a skill are those objects on which the skill is performed. For example, during a pick demonstration, we 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 action is performed are called Active Objects. The Active Objects are predefined and reported in Table 3.1. We used the same concept of interacting objects discussed in the previous chapter to evaluate the object state. That is, when the operator's hand passes close to an object for the first time, the object state is evaluated and stored in $InteractingObjects$ as long as the skill is not over (Alg.2, Line from 4 to 9). Then, the postconditions are set as previously explained, and the Active Objects are evaluated according to the recognized skill (Alg.2, Line 10,11). For example, if a stack skill is performed, the operator will have a cube in hand and he will release it above another one. Therefore, the operator's hand will be close to the grasped object (ObjectInHand) and the object above which the grasped object is stacked on (ObjectUnderHand). They are the Active Objects defined in Table 3.1 for the stack skill, and since the hand has passed close to them, their state is evaluated and stored in the interacting objects list. 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 precondition of the performed skill (Alg.2, Line from 12 to 22). In addition, if the predicate is in the postconditions list previously set (Alg.2, Line 10), the value of the predicate is negated, and the predicate is added to the preconditions. This because a predicate that changes value during the demonstration is considered relevant. If none of the above conditions is satisfied, the object's predicate is ignored. In the end, the preconditions learning algorithm will return a list of predicates which will constitute the skill preconditions.

Skill	**ActiveObject1**	**ActiveObject2**
Pick	PickedObject	null
Release	ReleasedObject	null
Stack	ObjectInHand	ObjectUnderHand
Unstack	ObjectInHand	ObjectUnderHand

Table 3.1: Table that shows the active objects for each skill. The Pick and Release skills have only one Active Object therefore their third element is null.

A series of figures that illustrate the preconditions learning algorithm is presented. The figures show the preconditions setting during a Stack Skill. In Figure 3.13, there are three cubes in the scene and the operator's left hand, two of the three cubes are in the fictitious sphere of the operator's left hand. Therefore, the initial state of the objects and the hand is computed and represented by the coloured boxes in the figure.

Figure 3.14 shows that the operator's left hand has moved the blue cube near the black cube. Consequently, the black cube is entered into the trigger sphere collider and its initial state is evaluated.

Algorithm 2 Preconditions learning algorithm

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

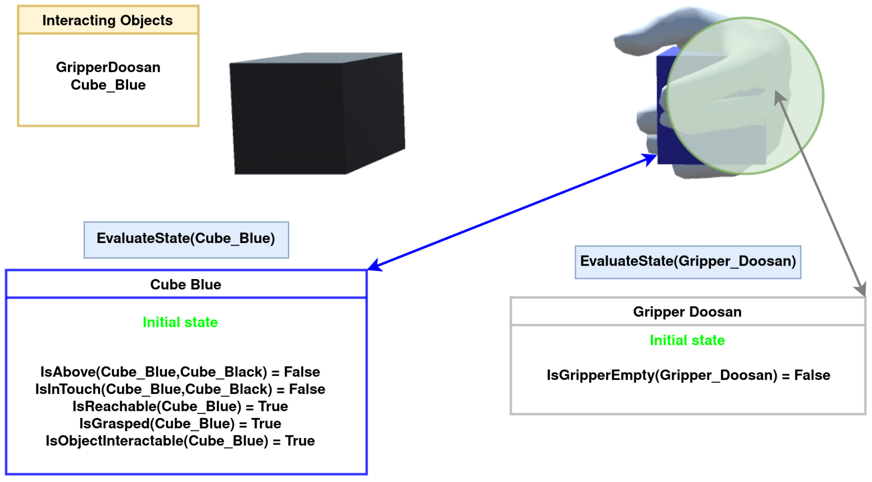

Interacting Objects
GripperDoosan
Cube_Blue
EvaluateState(Cube_Blue)
Cube Blue
Initial state
IsAbove(Cube_Blue,Cube_Black) = False
IsInTouch(Cube_Blue,Cube_Black) = False
IsReachable(Cube_Blue) = True
IsGrasped(Cube_Blue) = True
IsObjectInteractable(Cube_Blue) = True
EvaluateState(Gripper_Doosan)
Gripper Doosan
Initial state
IsGripperEmpty(Gripper_Doosan) = False

Figure 3.13: Initial state of a Stack skill. The operator's left hand is grasping the blue cube. The initial state of the blue cube, black cube and operator's left hand are computed and showed in the coloured boxes.

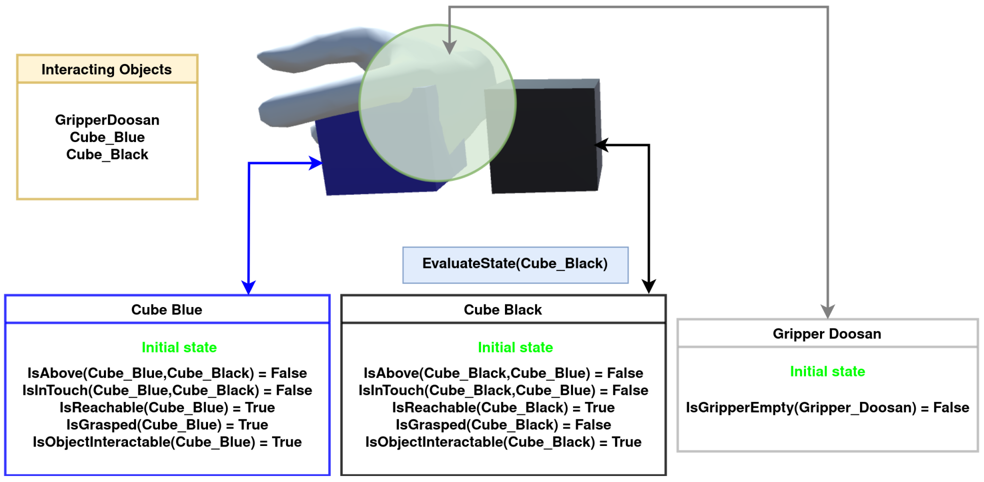
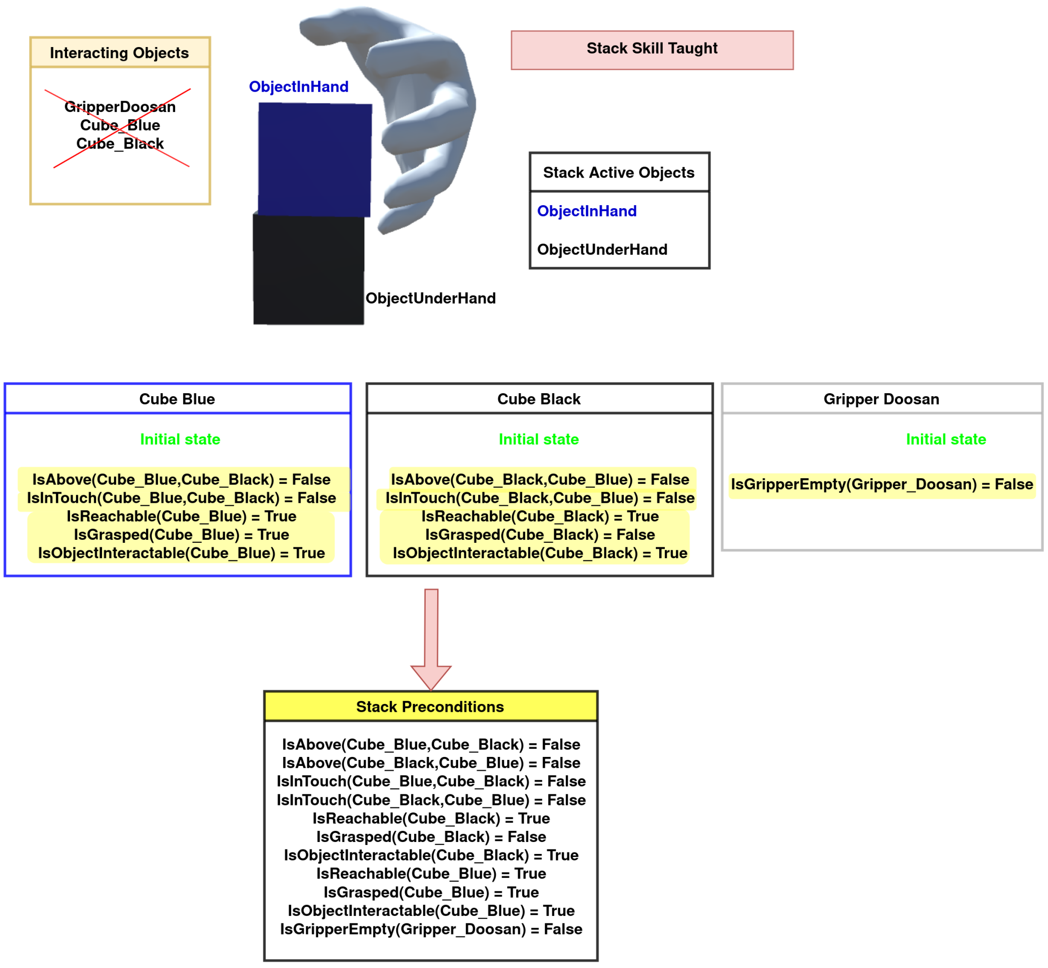

Interacting Objects
GripperDoosan
Cube_Blue
Cube_Black
EvaluateState(Cube_Black)
Cube Blue
Initial state
IsAbove(Cube_Blue,Cube_Black) = False
IsInTouch(Cube_Blue,Cube_Black) = False
IsReachable(Cube_Blue) = True
IsGrasped(Cube_Blue) = True
IsObjectInteractable(Cube_Blue) = True
Cube Black
Initial state
IsAbove(Cube_Black,Cube_Blue) = False
IsInTouch(Cube_Black,Cube_Blue) = False
IsReachable(Cube_Black) = True
IsGrasped(Cube_Black) = False
IsObjectInteractable(Cube_Black) = True
Gripper Doosan
Initial state
IsGripperEmpty(Gripper_Doosan) = False

Figure 3.14: The intermediate step of a stack skill. The operator is moving the blue cube towards the black cube. The black cube initial state is evaluated and shown by the black box once entered in the sphere.

In the end, figure 3.15 shows the final step of a stack skill, namely when the operator releases the blue cube above the black one. Once the stack operation has finished, the relevant predicates are extracted and set as preconditions for the stack skill.



Interacting Objects
~~GripperDoosan~~
~~Cube_Blue~~
~~Cube_Black~~
ObjectInHand
ObjectUnderHand
Stack Skill Taught
Stack Active Objects
ObjectInHand
ObjectUnderHand
Cube Blue
Initial state
IsAbove(Cube_Blue,Cube_Black) = False
IsInTouch(Cube_Blue,Cube_Black) = False
IsReachable(Cube_Blue) = True
IsGrasped(Cube_Blue) = True
IsObjectInteractable(Cube_Blue) = True
Cube Black
Initial state
IsAbove(Cube_Black,Cube_Blue) = False
IsInTouch(Cube_Black,Cube_Blue) = False
IsReachable(Cube_Black) = True
IsGrasped(Cube_Black) = False
IsObjectInteractable(Cube_Black) = True
Gripper Doosan
Initial state
IsGripperEmpty(Gripper_Doosan) = False
Stack Preconditions
IsAbove(Cube_Blue,Cube_Black) = False
IsAbove(Cube_Black,Cube_Blue) = False
IsInTouch(Cube_Blue,Cube_Black) = False
IsInTouch(Cube_Black,Cube_Blue) = False
IsReachable(Cube_Black) = True
IsGrasped(Cube_Black) = False
IsObjectInteractable(Cube_Black) = True
IsReachable(Cube_Blue) = True
IsGrasped(Cube_Blue) = True
IsObjectInteractable(Cube_Blue) = True
IsGripperEmpty(Gripper_Doosan) = False

Figure 3.15: The final step of a stack skill. The operator releases the blue cube above the black one. The preconditions box shows the relevant predicates selected for the taught skill.

Chapter 4

Automated Generation of the Planning Task

In this chapter, we will discuss how the planning task is automatically generated after the user demonstrations. The planning task is composed of the planning problem and the planning domain. The planning problem describes the initial 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. A planner has the objective of scheduling the required skills to pass from the initial state to the final one.
Chapter 4.1 describes how the virtual demonstration is carried out. In Chapter 4.2, we will discuss the method used to perform skill recognition and classification. Then, in Chapter 4.3 we explain how the domain and problem files are generated after the user demonstration, and finally, in Chapter 4.4 the planner choice is discussed.

4.1 Virtual Demonstration

As already stated in Chapter 2, the operator demonstrates actions in a virtual environment using the Oculus quest 2 headset and its controllers. When the operator wears the headset, he sees a scene and thanks to the provided controllers, he can move and close the virtual hands, grasp, drag and drop objects. In Figure 4.1, we can see the virtual scene 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 set of cubes near the human position are called human cubes since the human uses them to perform the task demonstration. On the other hand, the set of cubes near the robot base position are used by the robot to perform the learned actions. Once the operator finishes the demonstration, the interface shows to him the name of actions performed with the involved objects. Thanks to this visual feedback, the operator knows what actions have been recognized by the system. Figure 4.2 illustrates how the UI shows the action learned.

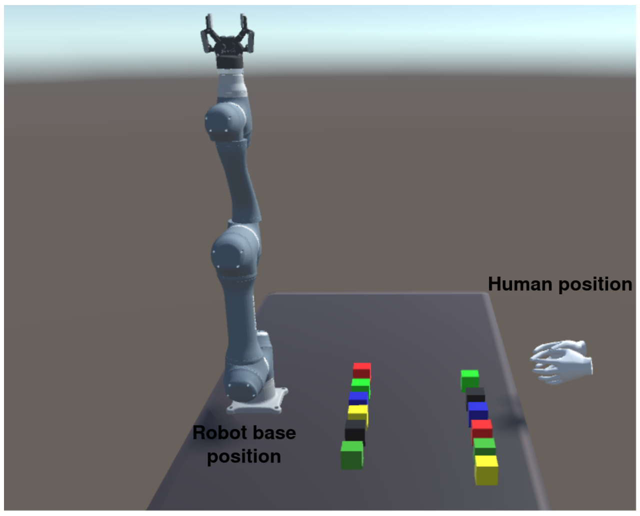


Figure 4.1: Side view of the virtual scene.

4.2 Skills classification

The previous chapter described the virtual scene in which the operator demonstrates the actions. Now we discuss the method used to recognize and classify the

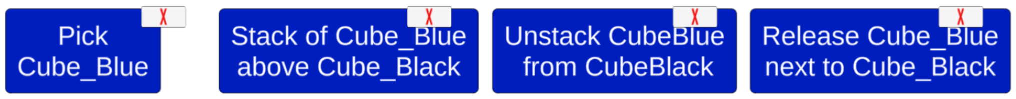

Pick Cube_Blue
Stack of Cube_Blue above Cube_Black
Unstack CubeBlue from CubeBlack
Release Cube_Blue next to Cube_Black

Figure 4.2: UI visual feedback showed after that the operator grabs the blue cube, releases it above the black one, pick the blue cube and releases it near the black one.

demonstrated actions. In order to assign correctly skill preconditions and postconditions, it is fundamental not only to know which skill has been performed but also when it started and ended, as discussed in Chapter 3. For this purpose, we decided to implement a State Machine as in the previous work [17] with slight changes to adapt it to our work. This approach is able to recognize the starting and the end of four different skills: Pick, Release, Stack, and Unstack. Every time a skill performed by the user is recognized, we store it in a list containing all the demonstrated skills together with their preconditions and postconditions. The deterministic State Machine implemented by us is illustrated in Figure 4.3. Each circle represents a discrete state of the world. Arrows represent transitions managing the passage from one node to another when a condition is met. A fundamental concept of a deterministic State Machine is that it can only have a single active state at any given time. Therefore, if two transitions start from the same state, the conditions associated with them are mutually exclusive. Looking at Figure 4.3, we can distinguish three different conditions: $HandEmpty$, $ObjectAbove$ and $ObjectInTouch$. These are first order logic 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 the true value. $ObjectAbove$ is an Open Predicate; therefore, it 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.

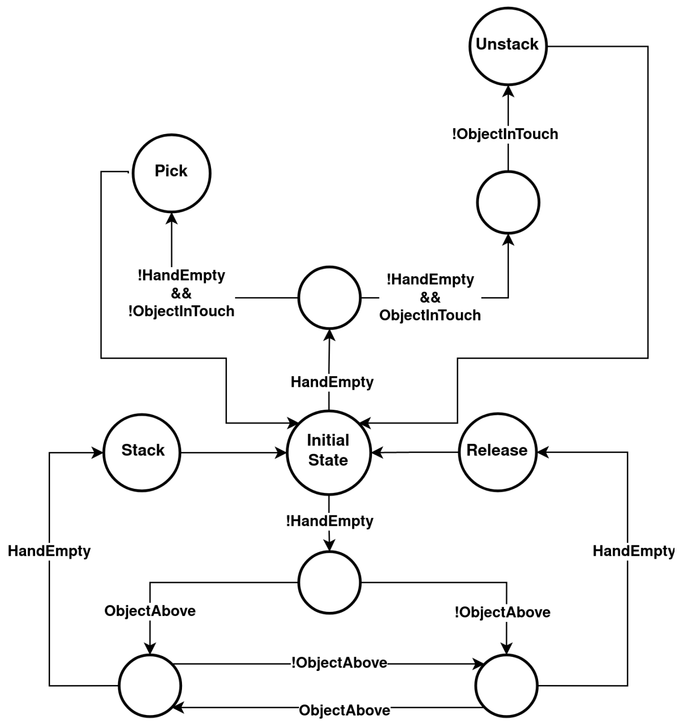


Figure 4.3: State Machine

An example of how the state machine works during a demonstration is reported in Figure 4.4 and Figure 4.5. At the beginning of the demonstration, the operator's left hand is grasping the blue cube. As a consequence, the predicate HandEmpty returns the true value activating the transition represented by the arrow pointing downwards. Then, the operator moves the blue cube above the black cube. The ObjectAbove predicate is queried, it returns the true value activating the transition represented by the left arrow pointing downwards. Finally, the operator releases the blue cube, and the stack skill performed is recognized.

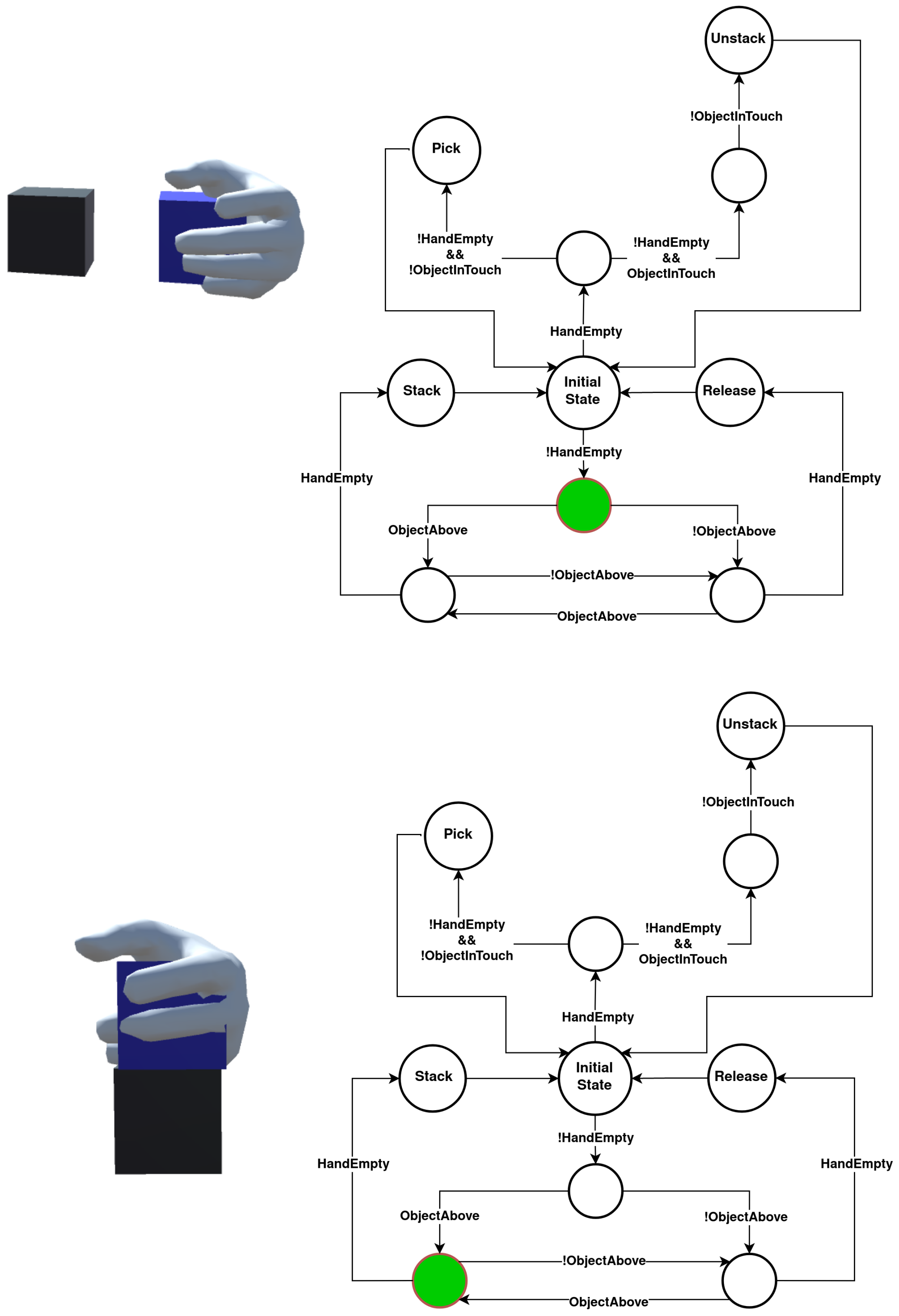


Figure 4.4: On the left are represented the action performed by the operator during the demonstration. On the right the state machine is represented with the active state that is coloured of green.

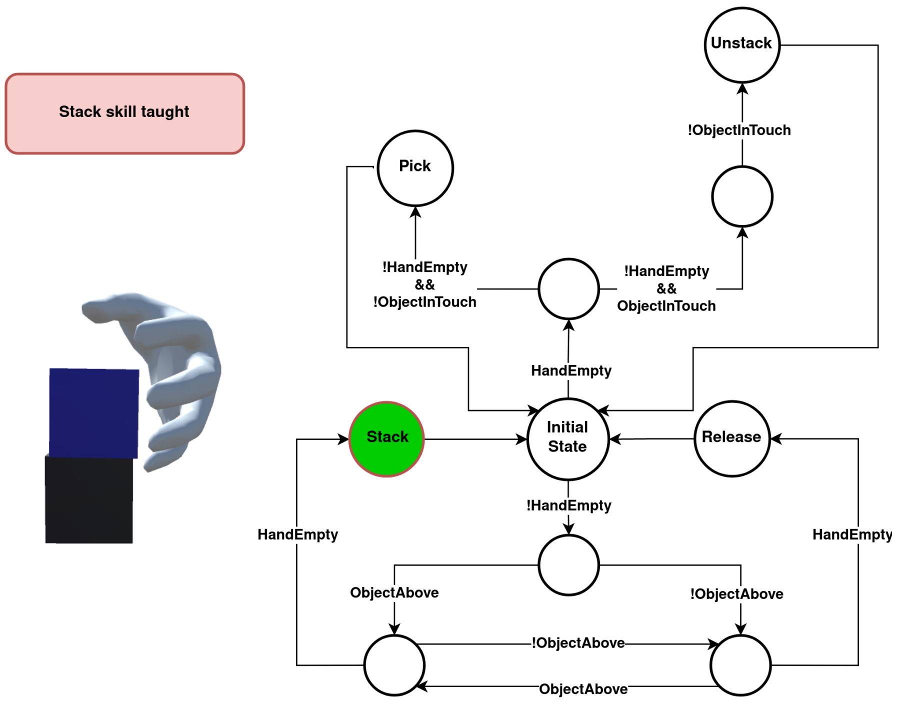
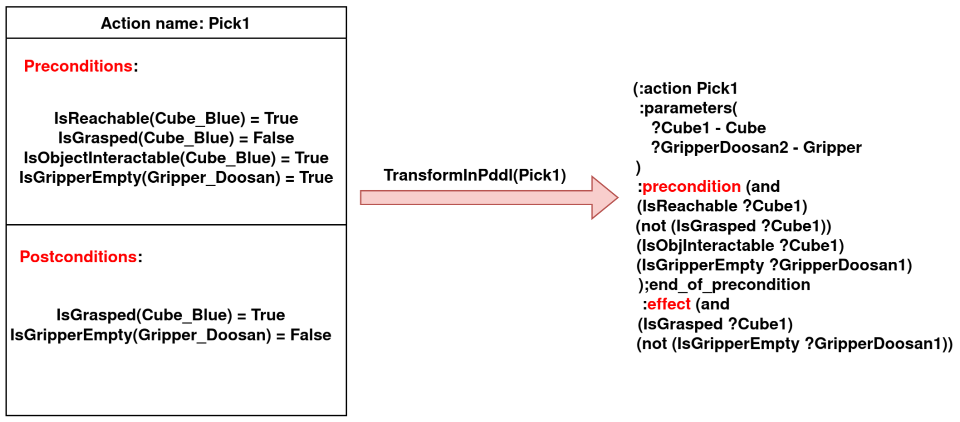


Figure 4.5: The operator releases the blue cube above the black one. Therefore, the active state is the stack one meaning that the skill has been taught.

4.3 Domain and Problem Generation

As we stated at the beginning of this chapter, the planning task comprises a planning domain and a problem. The planning domain is a collection of all possible actions that can be applied to achieve the desired goal. Actions are represented in the PDDL language with a unique name, a set of parameters, a list of preconditions and a list of effects. Once the user demonstration is finished, we will have a list of all the demonstrated actions together with their preconditions and postconditions. We have to write them in the PDDL language inside the domain file. Actions cannot be identical in a domain file. Thus, we need to check if the learnt 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. For example, if a user wants to build a pile of three cubes during the demonstration, he has to pick a cube, stack the cube above another, pick the third cube, and stack it above the other two. If the two picked cubes have the same initial state, the two pick actions performed will be identical, having the same preconditions and postconditions. In order to write an action in the PDDL language, we need generalize it and abstract from the particular objects used during the demonstration. Figure 4.6 shows an example of the content of a generalised action in PDDL form.

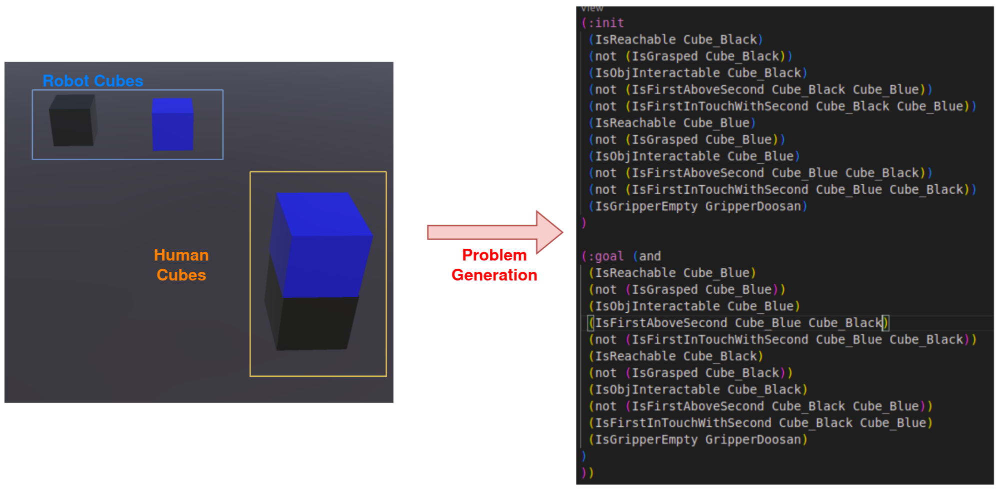


```
Action name: Pick1

Preconditions:

IsReachable(Cube_Blue) = True
IsGrasped(Cube_Blue) = False
IsObjectInteractable(Cube_Blue) = True
IsGripperEmpty(Gripper_Doosan) = True

Postconditions:

IsGrasped(Cube_Blue) = True
IsGripperEmpty(Gripper_Doosan) = False
```

```
(:action Pick1
 :parameters(
   ?Cube1 - Cube
   ?GripperDoosan2 - Gripper
)
:precondition (and
(IsReachable ?Cube1)
(not (IsGrasped ?Cube1))
(IsObjInteractable ?Cube1)
(IsGripperEmpty ?GripperDoosan1)
);end_of_precondition
 :effect (and
(IsGrasped ?Cube1)
(not (IsGripperEmpty ?GripperDoosan1))
```

Figure 4.6: On the left an example of how a demonstrated pick is stored in DemonstratedActions. Then, on the right the string containing the pick skill written in the PDDL language is showed.

In order to complete the task planning, the problem has to be generated. The problem is composed of the world initial state and the goal to be achieved. In the virtual scene implemented by us, the robot has its own set of cubes. Therefore, the initial world state is defined by evaluating the predicates implemented in Chapter 3 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. An example of how the problem is generated is reported in Figure 4.7

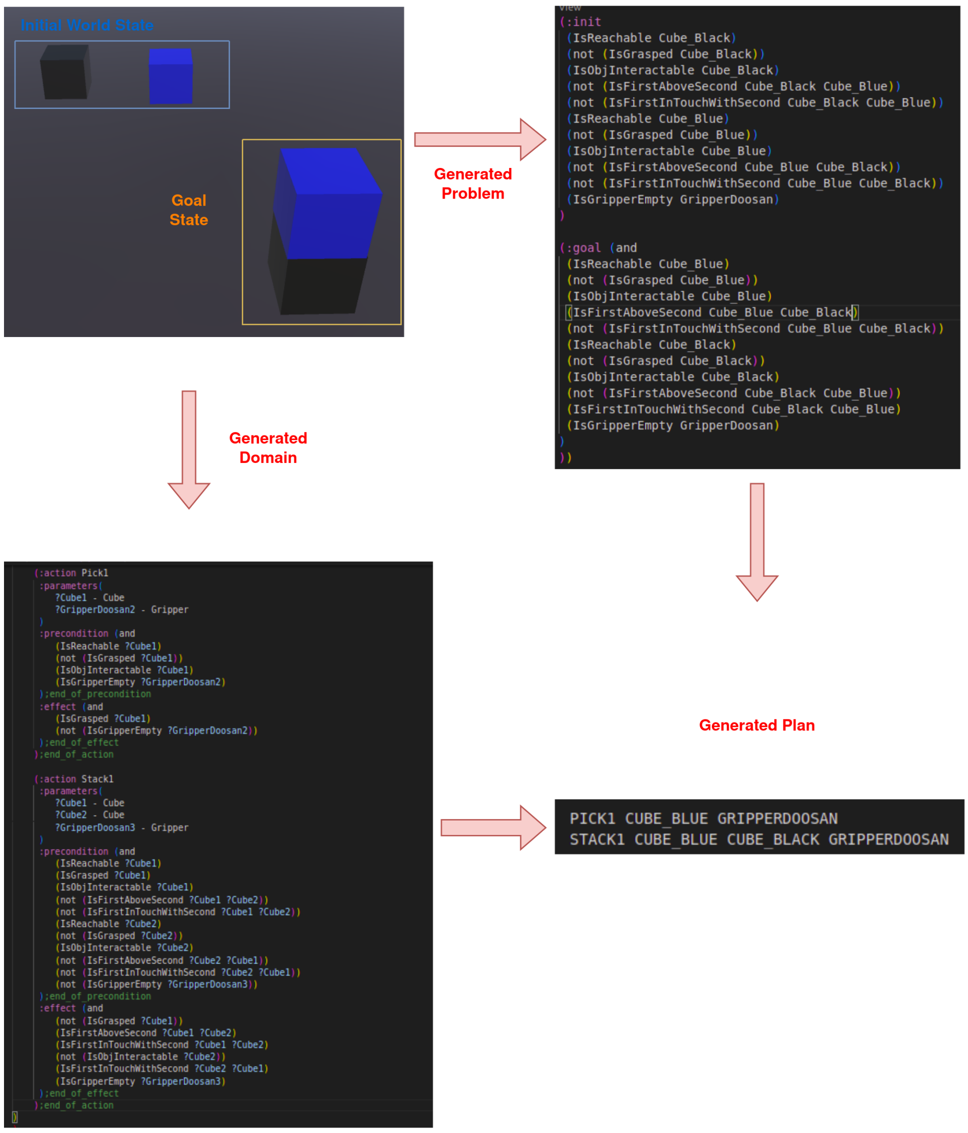


```
(:init
 (IsReachable Cube_Black)
 (not (IsGrasped Cube_Black))
 (IsObjInteractable Cube_Black)
 (not (IsFirstAboveSecond Cube_Black Cube_Blue))
 (not (IsFirstInTouchWithSecond Cube_Black Cube_Blue))
 (IsReachable Cube_Blue)
 (not (IsGrasped Cube_Blue))
 (IsObjInteractable Cube_Blue)
 (not (IsFirstAboveSecond Cube_Blue Cube_Black))
 (not (IsFirstInTouchWithSecond Cube_Blue Cube_Black))
 (IsGripperEmpty GripperDoosan)
)

(:goal (and
 (IsReachable Cube_Blue)
 (not (IsGrasped Cube_Blue))
 (IsObjInteractable Cube_Blue)
 (IsFirstAboveSecond Cube_Blue Cube_Black)
 (not (IsFirstInTouchWithSecond Cube_Blue Cube_Black))
 (IsReachable Cube_Black)
 (not (IsGrasped Cube_Black))
 (IsObjInteractable Cube_Black)
 (not (IsFirstAboveSecond Cube_Black Cube_Blue))
 (IsFirstInTouchWithSecond Cube_Black Cube_Blue)
 (IsGripperEmpty GripperDoosan)
)
))
```

Figure 4.7: Example of the problem generation. On the left a screenshot of the virtual scene is showed, the human cubes define the goal, while the robot cubes define the initial world state. On the right the initial world state and the goal represented in the PDDL language are showed.

4.4 Plan Generation

Once the planning task is generated, we need to find a sequence of high-level actions that the robot will execute in order to reach the goal state. For this purpose, a planner is used in the execution step. Since we used the PDDL language to define the planning problem, a large variety of planners are available. We used the STRIPS [19] formalism to define the planning task. This is the classical formalism for the PDDL definition, thus, we had no tight constraints on the planner choice, and we selected the Fast-Forward planner [20]. A final example is reported in Figure 4.8, where the virtual scene is shown that defines the initial world state and the goal, the task planning generated and the plan returned by the Fast-Forward planner.



```
(:init
(IsReachable Cube_Black)
(not (IsGrasped Cube_Black))
(IsObjInteractable Cube_Black)
(not (IsFirstAboveSecond Cube_Black Cube_Blue))
(not (IsFirstInTouchWithSecond Cube_Black Cube_Blue))
(IsReachable Cube_Blue)
(not (IsGrasped Cube_Blue))
(IsObjInteractable Cube_Blue)
(not (IsFirstAboveSecond Cube_Blue Cube_Black))
(not (IsFirstInTouchWithSecond Cube_Blue Cube_Black))
(IsGripperEmpty GripperDoosan)
)

(:goal (and
(IsReachable Cube_Blue)
(not (IsGrasped Cube_Blue))
(IsObjInteractable Cube_Blue)
(IsFirstAboveSecond Cube_Blue Cube_Black)
(not (IsFirstInTouchWithSecond Cube_Blue Cube_Black))
(IsReachable Cube_Black)
(not (IsGrasped Cube_Black))
(IsObjInteractable Cube_Black)
(not (IsFirstAboveSecond Cube_Black Cube_Blue))
(IsFirstInTouchWithSecond Cube_Black Cube_Blue)
(IsGripperEmpty GripperDoosan)
)
))
```

```
(:action Pick1
 :parameters(
    ?Cube1 - Cube
    ?GripperDoosan2 - Gripper
 )
 :precondition (and
    (IsReachable ?Cube1)
    (not (IsGrasped ?Cube1))
    (IsObjInteractable ?Cube1)
    (IsGripperEmpty ?GripperDoosan2)
 );end_of_precondition
 :effect (and
    (IsGrasped ?Cube1)
    (not (IsGripperEmpty ?GripperDoosan2))
 );end_of_effect
);end_of_action

(:action Stack1
 :parameters(
    ?Cube1 - Cube
    ?Cube2 - Cube
    ?GripperDoosan3 - Gripper
 )
 :precondition (and
    (IsReachable ?Cube1)
    (IsGrasped ?Cube1)
    (IsObjInteractable ?Cube1)
    (not (IsFirstAboveSecond ?Cube1 ?Cube2))
    (not (IsFirstInTouchWithSecond ?Cube1 ?Cube2))
    (IsReachable ?Cube2)
    (not (IsGrasped ?Cube2))
    (IsObjInteractable ?Cube2)
    (not (IsFirstAboveSecond ?Cube2 ?Cube1))
    (not (IsFirstInTouchWithSecond ?Cube2 ?Cube1))
    (not (IsGripperEmpty ?GripperDoosan3))
 );end_of_precondition
 :effect (and
    (not (IsGrasped ?Cube1))
    (IsFirstAboveSecond ?Cube1 ?Cube2)
    (IsFirstInTouchWithSecond ?Cube1 ?Cube2)
    (not (IsObjInteractable ?Cube2))
    (IsFirstInTouchWithSecond ?Cube2 ?Cube1)
    (IsGripperEmpty ?GripperDoosan3)
 );end_of_effect
);end_of_action
)
```

```
PICK1 CUBE_BLUE GRIPPERDOOSAN
STACK1 CUBE_BLUE CUBE_BLACK GRIPPERDOOSAN
```

Figure 4.8: 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.

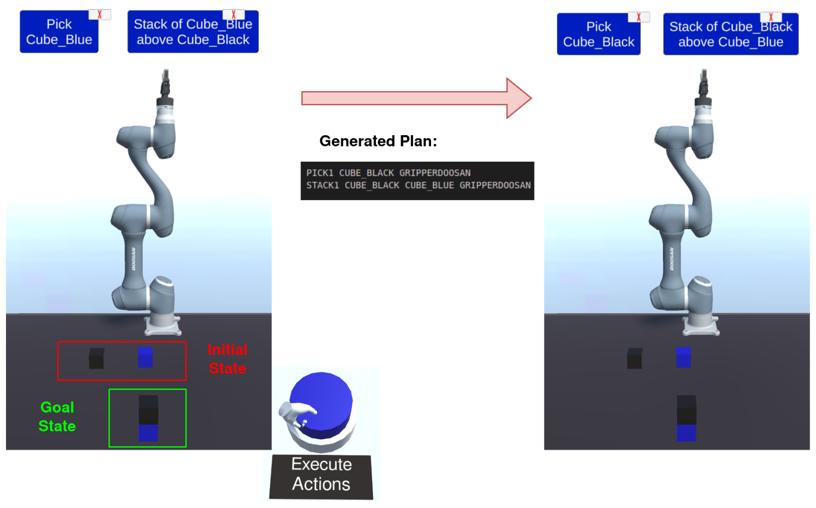
Chapter 5

Plan Execution and Recovery

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. This is even more likely once we switch to the real environment, since measurement are typically difficult, unavailable or noisy. Let's make an example of a robot that has to stack a cube above another one. In such scenario, it could happen that the vision system returns an inaccurate position of the base cube and the action fails. Therefore, we implemented an approach that spots and deals with the possible errors during the execution.
In Chapter 5.1, we discuss how the robot carries out the plan execution. Then, Chapter 5.2 explains the error management approach and illustrates some examples.

5.1 Plan Execution in VR

After the demonstration phase, the user defines the goal by sorting his cubes in the desired position and launches the planner. The planner parses the generated domain and problem, returns a sequence of actions that modify the environment from the initial state to the final one. The visual feedback showing the demonstrated actions is replaced by a visual feedback that shows the robot actions to be performed and their list of involved objects, as illustrated in Figure 5.1. Finally, the robot has to execute the actions. In order to move the robot in the virtual environment, we used ROS [21], which provides services such as message-passing, package management, low-level device control, and hardware abstraction. Figure 5.2 illustrates how the communication between Unity and ROS works. On the ROS side, a TCP endpoint running as ROS node handles all message passing. On the Unity side, a ROS component provides the necessary functions to publish, subscribe, or call a service using the TCP endpoint ROS node. During the execution, we just need to send messages from Unity to ROS regarding the position and the orientation that the robot has to reach in order to move it. Each action execution is predefined and discussed in detail in the following subchapters.



```
PICK1 CUBE_BLACK GRIPPERDOOSAN
STACK1 CUBE_BLACK CUBE_BLUE GRIPPERDOOSAN
```

Figure 5.1: The left the figure shows the user defined goal, the initial world state and the demonstrated actions showed by the User Interface. The user pushes the button to start the execution phase and the plan actions returned by the planner are displayed. Then, on the right, the demonstrated actions are replaced by the planned actions.

5.1.1 Pick Execution

The pick execution is divided into three steps: first, the robot moves in the approach position, which is the position above the cube to be picked. Then, it moves linearly toward the cube and grasps it. The pick action finishes when the robot

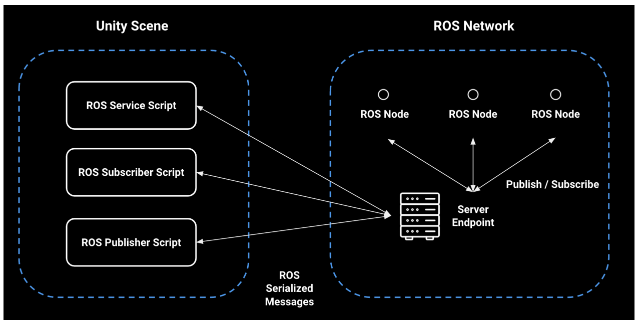


Figure 5.2: The figure shows how the communication between Unity and ROS works.

returns to the pick approach position grasping the cube. Figure 5.3 shows the three steps of a pick execution.

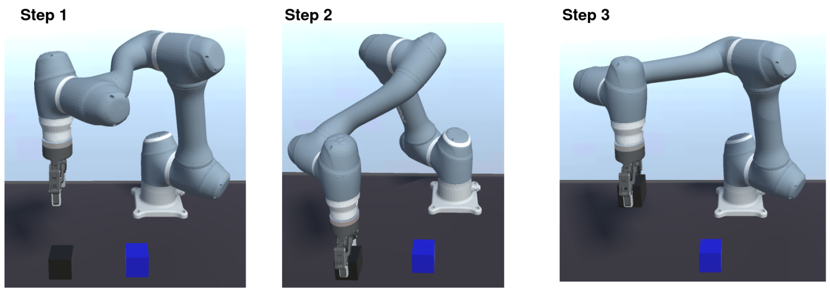


Figure 5.3: In the figure, the robot has to pick the black cube. In the first step, the robot has moved to the pick approach position, which is the one above the black cube. In the second step, the robot has moved linearly toward the black cube and grasps it. Finally, the robot has returned to the approach position with the black cube in the gripper.

5.1.2 Stack Execution

The stack execution is divided into three steps: first, the robot with a cube in the gripper, moves in the approach position, which is the position above the base cube. Then, it moves linearly toward it and stacks the desired cube above the base one. Finally, the robot completes the stack execution by returning to the stack approach position. Figure 5.4 shows the three steps of a stack execution.

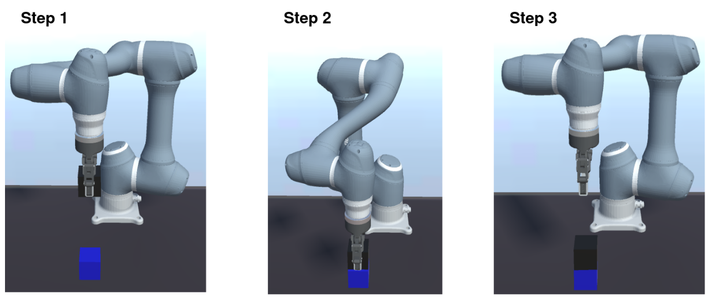


Figure 5.4: In the figure, the robot has to stack the black cube. In the first step, the robot has moved to the approach position, which is the one above the blue cube. In the second step, the robot has moved linearly toward the blue cube and has stacked the black cube on it. Finally, the robot has returned to the stack approach position.

5.1.3 Unstack Execution

The unstack execution is divided into three steps: first, the robot moves in the approach position, which is the position above the cube to be unstacked. Then, it moves linearly toward, closes the gripper, and unstacks the desired cube from the base one. Finally, the robot completes the unstack execution by returning to the approach position grasping the desired cube. Figure 5.5 shows the three steps of a unstack execution.

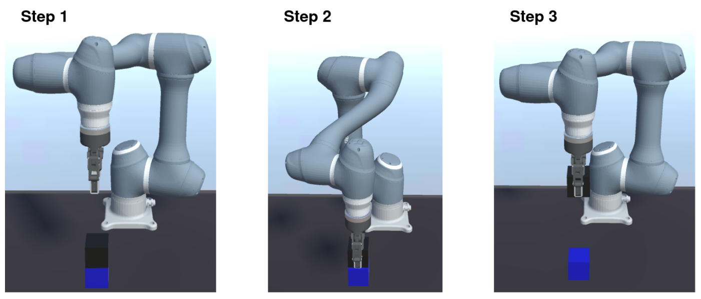


Figure 5.5: In the figure, the robot has to unstack the black cube. In the first step, the robot has moved to the approach position, which is the one above the black cube. In the second step, the robot has moved linearly toward the black cube and has unstacked the black cube from the blue one. Finally, the robot has returned to the approach position grasping the black cube.

5.1.4 Release Execution

The release execution is divided into three steps: first, the robot moves in the approach position, which is the position above the buffer while it has a cube in the gripper. The buffer is a special area in the table near the robot position where the robot releases the cubes. In the second step, the robot moves linearly toward the buffer position and releases the cube. Finally, the robot completes the execution by returning to the release approach position. Figure 5.6 shows the three steps of a release execution.

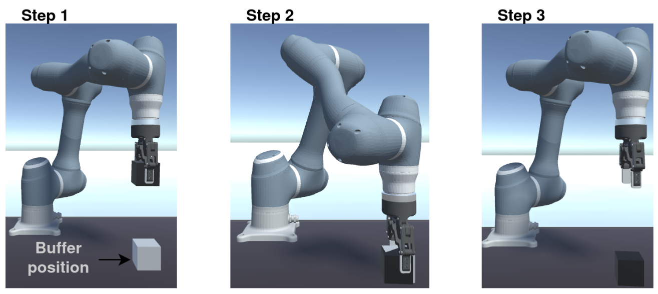


Figure 5.6: In the figure, the robot has to release the black cube. In the first step, the robot has moved to the approach position, which is the one above the buffer. In the second step, the robot has moved linearly toward the buffer and has released the black cube in it. Finally, the robot has returned to the approach position.

A final example of how the execution phase is carried out in the virtual environment is illustrated in Figur 5.7. The figure shows the steps of the robot's execution phase, where in the inital state the blue and black cubes are on the ground and the goal is to stack the black cube above the blue one. In particular, the robot executes the pick and stack actions in three steps: at first, it moves in the pick approach position (step 1), which is the position above the black cube. Then, it moves linearly toward the black cube and grasps it (step 2). The pick action finishes when the robot returns to the pick approach position grasping the black cube (step 3). Furthermore, the figure shows how the User interface changes the colour of the box representing the pick action during its execution. During the action execution, the box colour is orange. When the robot has completed the action, the box is coloured green. This can be useful to the human operator to understand and constantly track what is happening. As soon as the robot completes the pick action, the stack action starts and its corresponding box is coloured orange. The robot moves in the stack approach position (step 4), which is the position above the blue cube. Then, it moves linearly toward it and stacks the black cube above the blue one (step 5). Finally, the robot completes the task by returning to the stack approach position (step 6), and the box corresponding to the stack action is coloured in green.

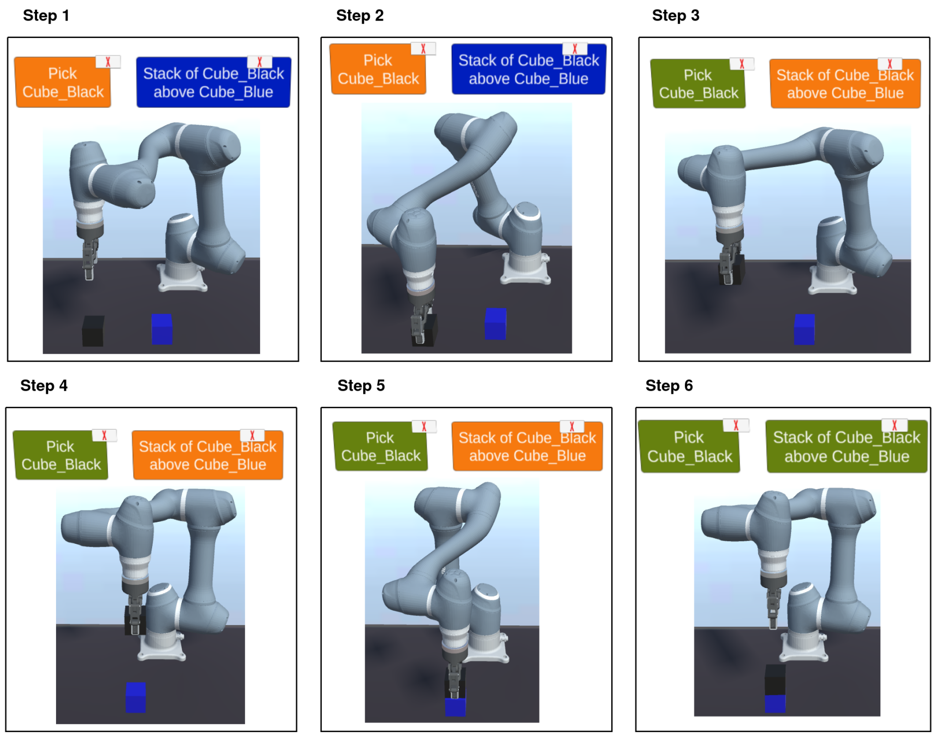

Step 1
Pick Cube_Black
Stack of Cube_Black above Cube_Blue
Step 2
Pick Cube_Black
Stack of Cube_Black above Cube_Blue
Step 3
Pick Cube_Black
Stack of Cube_Black above Cube_Blue
Step 4
Pick Cube_Black
Stack of Cube_Black above Cube_Blue
Step 5
Pick Cube_Black
Stack of Cube_Black above Cube_Blue
Step 6
Pick Cube_Black
Stack of Cube_Black above Cube_Blue

Figure 5.7: The figure shows how the robot executes the pick and stack actions.

5.2 Error handling

In the previous chapter, we discussed how the robot executes the plan if no errors occur during the execution. This chapter focuses on the approach developed to spot errors. Figure 5.9 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. An example of this is illustrated in Figure 5.8, where the user demonstrated the pick action only and defined a goal where also the stack skill is required. Since the demonstrated action is not sufficient to reach the desired goal, when the planner is called, it returns an empty plan which means that a solution to the proposed problem does not exist.

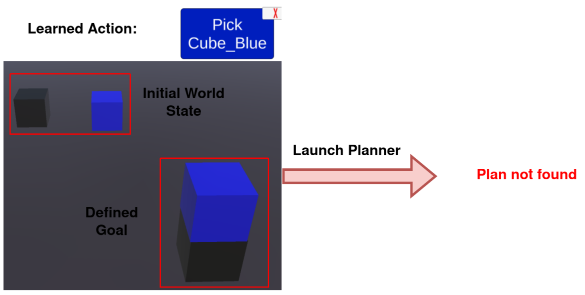


Figure 5.8: The figure shows how the system behaves in case an insufficient domain is provided to the robot.

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. If they are not, we recompute the problem and the steps seen before are repeated. Recomputing the problem allows us to 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 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.

Figure 5.10 and 5.11 show an example of the error handling execution. Figure

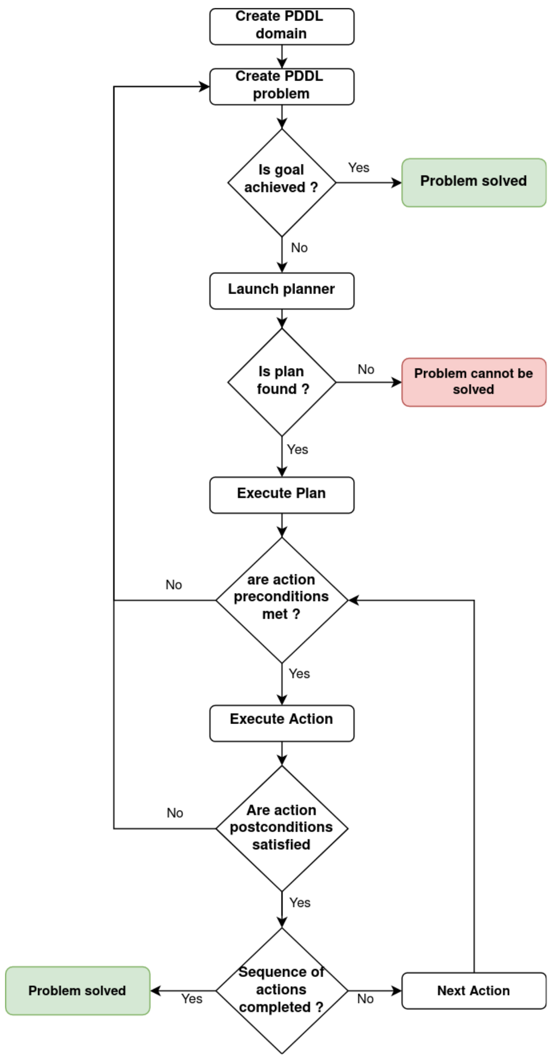
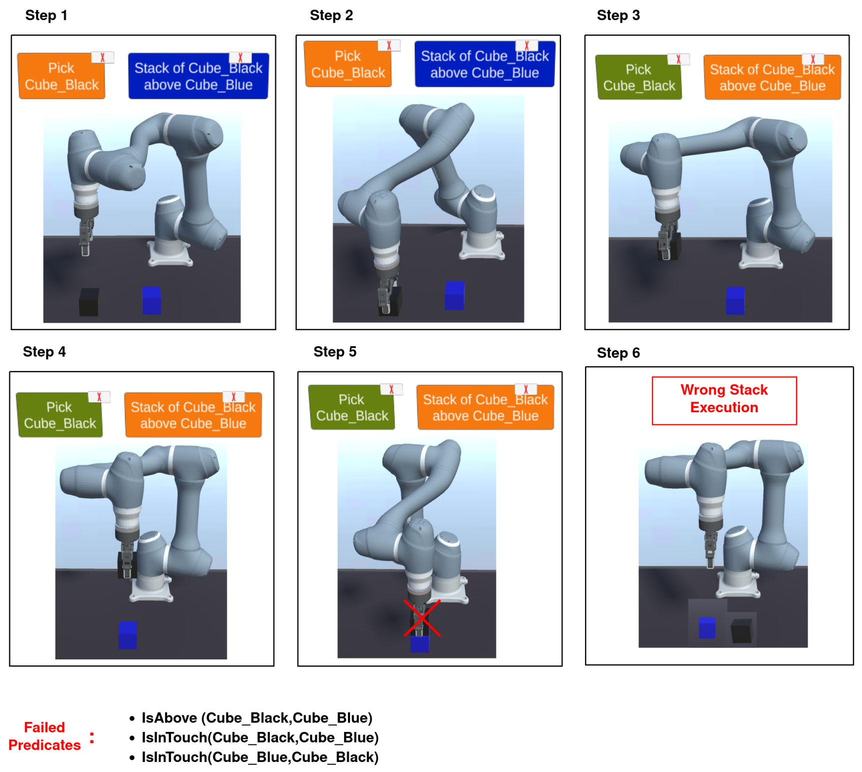


Figure 5.9: The figure shows the workflow of our system.

5.10 shows the steps that the robot executes in the execution phase. The robot executes the pick action correctly, while the stack action failed causing the fall of the black cube. Since the postconditions $IsAbove(Cube-Black, Cube-Blue)$, $IsInTouch(Cube-Black, Cube-Blue)$ and $IsInTouch(Cube-Blue, Cube-Black)$ are not satisfied, the problem is recomputed and the planner re-launched. In Figure 5.11, the robot executes correctly the actions of the recovered plan achieving the user-defined goal.



Step 1
Pick Cube_Black
Stack of Cube_Black above Cube_Blue
Step 2
Step 3
Step 4
Step 5
Step 6
Wrong Stack Execution
Failed Predicates :
• IsAbove (Cube_Black,Cube_Blue)
• IsInTouch(Cube_Black,Cube_Blue)
• IsInTouch(Cube_Blue,Cube_Black)

Figure 5.10: In the figure the robot has to pick the black cube and stack it above the blue one, but it fails the stack action.

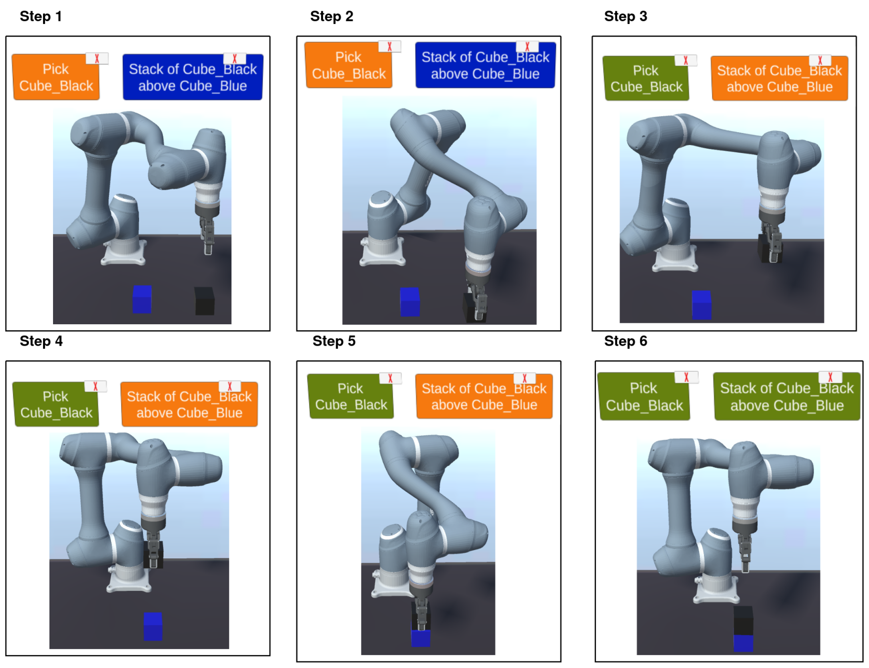


Figure 5.11: the figure shows how the plan is recovered after that the stack execution failed.

Chapter 6

Experimental Use Case

In this Chapter we will discuss how the experiments have been set up, the objectives of each experiment, how they are structured, and finally, the results.

6.1 Experimental Setup

The same experiments have been carried out both in virtual and real environments. For this reason, the virtual environment has been designed to represent as close as possible the real one. Figure 6.1 shows the virtual and real environments respectively. The left side figure shows how the virtual environment has been set up. In particular, on the right hand side of the table, the hands represent the operator's position. On the opposite side of the table, the Doosan A-series 0509s can be seen. Furthermore, two sets of cubes are present in the virtual scene, one near the operator's position and one near the robot's position. 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. On the other hand, the set of cubes near the robot base position is used by the robot to perform the required task.The picture on the right side of Figure 6.1 shows how the setup of the real environment has been designed. The robot used is the same as the virtual one, that is the Doosan A-series 0509s. 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 main difference between the real and the virtual environment is the presence of just the robot's set of cubes in the real one. 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.

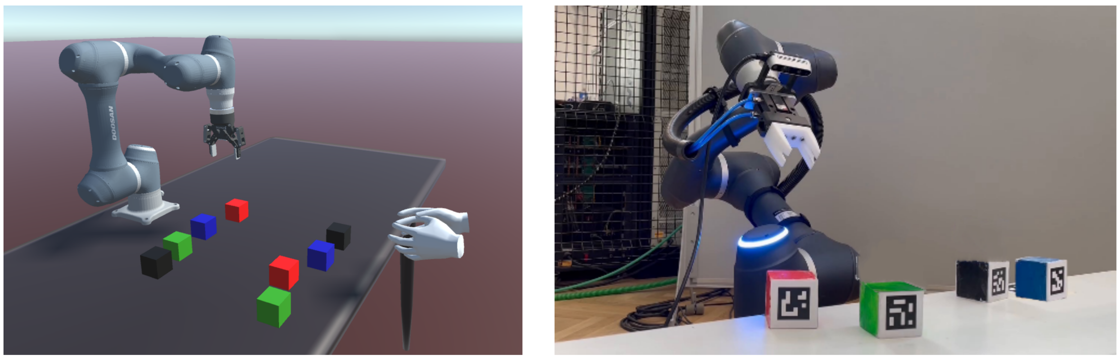


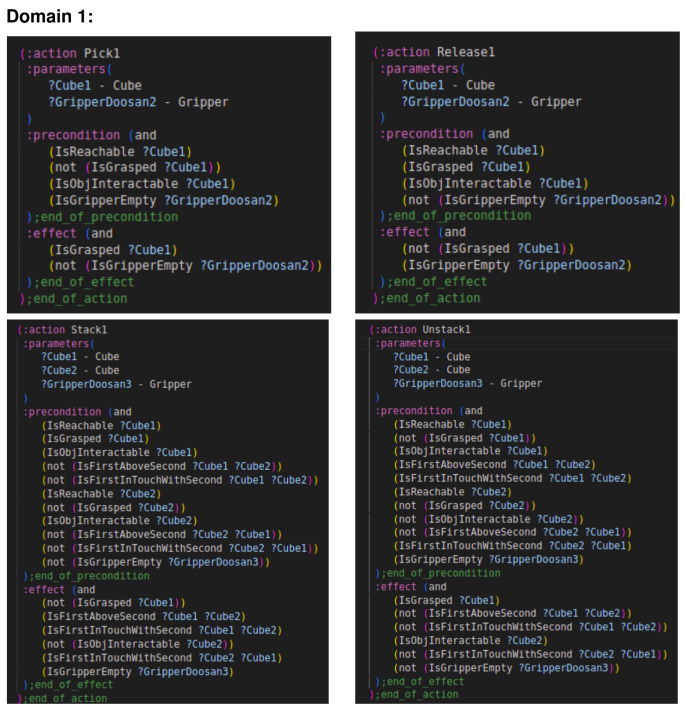
Figure 6.1: The left picture shows the virtual environment setup. The right picture shows the real environment setup.

The virtual reality headset used to perform the experiments is the Oculus Quest 2 (Figure 6.2). It is used as a standalone device, hence it needs no computer or phone to work.

Figure 6.2: The figure shows the Oculus Quest 2 device.

6.2 Use Case explanation

To properly validate our work, we defined three different goals the robot had to reach, using the demonstrated actions during the training phase in VR saved in the domain file shown in Figure 6.3. The domain file is generated from a demonstration in which we manipulated the cubes in order to have one demonstration for each action. Figure 6.4 shows the goals. In particular, the first goal consists in stacking the red cube on the green one starting from the initial state, that is, the one where all the cubes are on the table and not in contact with each other . Then, starting from the position of the cubes in goal 1, the robot has to pass to Task 2 which consists in stacking the blue cube on the black one. Finally, to complete the third goal, the robot has to unstack the red cube from the green one and stack it on the blue. Each goal has been performed five times in the virtual and real environments. Furthermore, goal 1 was demonstrated during the demonstration phase, while the remaining goals were not demonstrated.



```
(:action Pick1
 :parameters(
    ?Cube1 - Cube
    ?GripperDoosan2 - Gripper
 )
 :precondition (and
    (IsReachable ?Cube1)
    (not (IsGrasped ?Cube1))
    (IsObjInteractable ?Cube1)
    (IsGripperEmpty ?GripperDoosan2)
 );end_of_precondition
 :effect (and
    (IsGrasped ?Cube1)
    (not (IsGripperEmpty ?GripperDoosan2))
 );end_of_effect
);end_of_action
```

```
(:action Release1
 :parameters(
    ?Cube1 - Cube
    ?GripperDoosan2 - Gripper
 )
 :precondition (and
    (IsReachable ?Cube1)
    (IsGrasped ?Cube1)
    (IsObjInteractable ?Cube1)
    (not (IsGripperEmpty ?GripperDoosan2))
 );end_of_precondition
 :effect (and
    (not (IsGrasped ?Cube1))
    (IsGripperEmpty ?GripperDoosan2)
 );end_of_effect
);end_of_action
```

```
(:action Stack1
 :parameters(
    ?Cube1 - Cube
    ?Cube2 - Cube
    ?GripperDoosan3 - Gripper
 )
 :precondition (and
    (IsReachable ?Cube1)
    (IsGrasped ?Cube1)
    (IsObjInteractable ?Cube1)
    (not (IsFirstAboveSecond ?Cube1 ?Cube2))
    (not (IsFirstInTouchWithSecond ?Cube1 ?Cube2))
    (IsReachable ?Cube2)
    (not (IsGrasped ?Cube2))
    (IsObjInteractable ?Cube2)
    (not (IsFirstAboveSecond ?Cube2 ?Cube1))
    (not (IsFirstInTouchWithSecond ?Cube2 ?Cube1))
    (not (IsGripperEmpty ?GripperDoosan3))
 );end_of_precondition
 :effect (and
    (not (IsGrasped ?Cube1))
    (IsFirstAboveSecond ?Cube1 ?Cube2)
    (IsFirstInTouchWithSecond ?Cube1 ?Cube2)
    (not (IsObjInteractable ?Cube2))
    (IsFirstInTouchWithSecond ?Cube2 ?Cube1)
    (IsGripperEmpty ?GripperDoosan3)
 );end_of_effect
);end_of_action
```

```
(:action Unstack1
 :parameters(
    ?Cube1 - Cube
    ?Cube2 - Cube
    ?GripperDoosan3 - Gripper
 )
 :precondition (and
    (IsReachable ?Cube1)
    (not (IsGrasped ?Cube1))
    (IsObjInteractable ?Cube1)
    (IsFirstAboveSecond ?Cube1 ?Cube2)
    (IsFirstInTouchWithSecond ?Cube1 ?Cube2)
    (IsReachable ?Cube2)
    (not (IsGrasped ?Cube2))
    (not (IsObjInteractable ?Cube2))
    (not (IsFirstAboveSecond ?Cube2 ?Cube1))
    (IsFirstInTouchWithSecond ?Cube2 ?Cube1)
    (IsGripperEmpty ?GripperDoosan3)
 );end_of_precondition
 :effect (and
    (IsGrasped ?Cube1)
    (not (IsFirstAboveSecond ?Cube1 ?Cube2))
    (not (IsFirstInTouchWithSecond ?Cube1 ?Cube2))
    (IsObjInteractable ?Cube2)
    (not (IsFirstInTouchWithSecond ?Cube2 ?Cube1))
    (not (IsGripperEmpty ?GripperDoosan3))
 );end_of_effect
);end_of_action
```

Figure 6.3: The figure shows the learnt actions composing the domain file with their preconditions and postconditions.

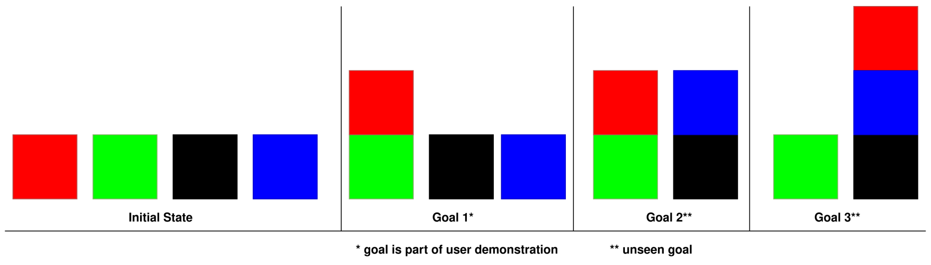


Figure 6.4: The figure shows the goals the robot had to reach and the initial state.

6.3 Results

During the training phase necessary to extract the domain of actions displayed in Figure 6.3, we measured the elapsed time from the first interaction with the objects in the virtual environment to the last action performed. 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. To make a comparison with the traditional style of robot programming, we measured the time necessary to program the robot using the teach pendant. To program the robot, we moved it from point to point, using the buttons on the pendant to move it around and save each position individually. Thus, Table 6.1 has been obtained and it illustrates how the required time for programming the robot with our method is lower than using the Tech Pendant.

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

Table 6.1: The table shows the training times required by the classical teaching and our work for programming the goal 1, goal 2 and goal 3.

Then, we observed the success rate in reaching the required goals during the execution phase in the virtual environment. Figure 6.5 illustrates the success rates for the three goals in VR. In green the percentage of the success rate computed without considering the error handling procedure described in Chapter 5.2 is represented, while in orange the percentage of the success rate achieved considering the errors management is represented. In particular, the robot achieved the first goal without any problem. In two cases, achieving the second goal required the error handling procedure because the robot failed to stack the cubes due to a wrong trajectory returned by ROS. In that cases, the cubes fell, but it was possible to recover the plan and achieve the goal using the learnt actions. Finally, the robot achieved the third goal in four out of five cases. In the failed attempt the robot could not achieve the goal since a cube fell from the tower touching the base cube. Since we did not demonstrate an action in which we grasped a cube in touch with another one the robot was stuck and did not know how to solve the situation. The obtained results show how the error handling procedure can improve the effectiveness of the robot's performance, especially when performing difficult tasks and even when unpredictable situations occur. Moreover, the robot performed the unseen tasks without the help of a human. This highlights how a demonstration can be sufficient to extract generalized actions that allow a successful plan generation for unseen tasks.

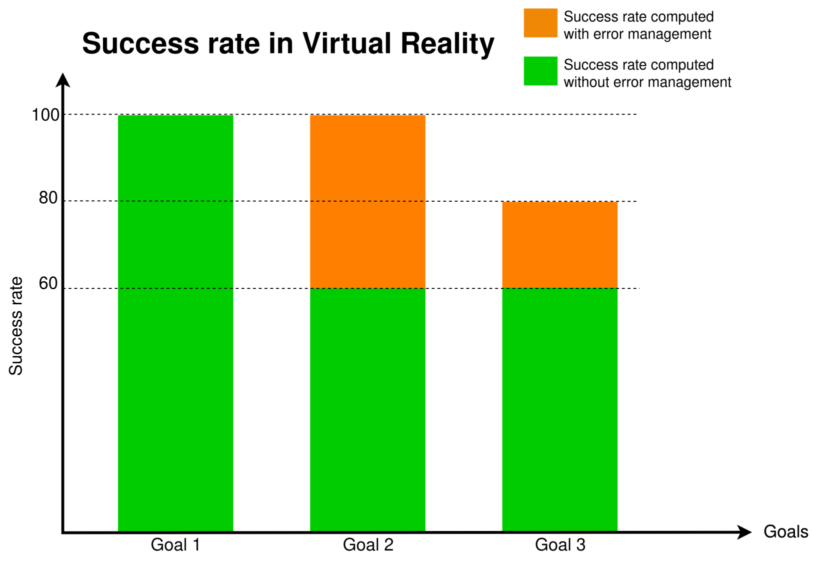


Figure 6.5: The figure shows the success rates achieved in the virtual environment for the three defined goals.

Figure 6.6 illustrates how many times the robot achieved the required goals during the execution phase with the real robot. In green the percentage of the success rate computed without considering the error handling procedure is represented, while in orange the percentage of the success rate achieved considering the errors management is represented. We observe that the real robot never failed to execute actions and achieved the goals without the use of the error handling procedure. This is because to move the robot in the real environment we did not use ROS, thus the computed trajectories were not faulty. 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.

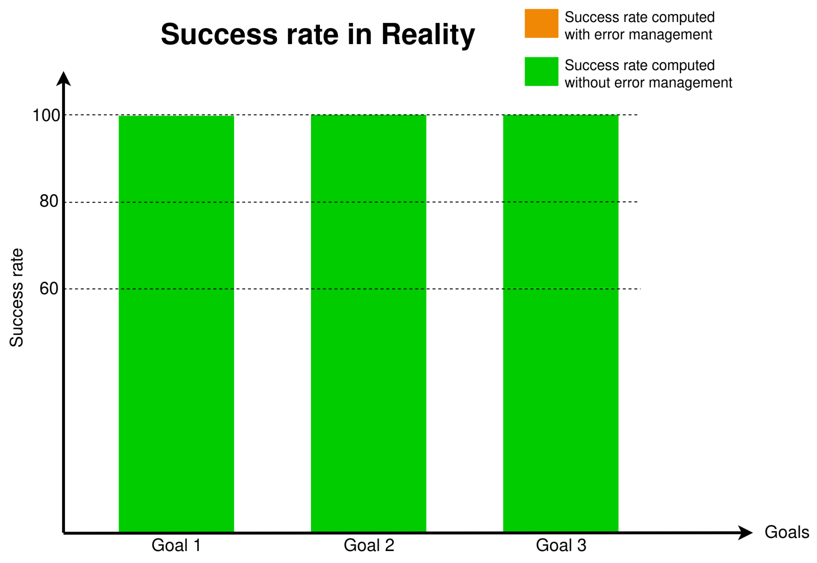


Figure 6.6: The figure shows the success rates achieved in the real environment for the three defined goals.

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 6.7 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 the overall number of actions in the domain low.

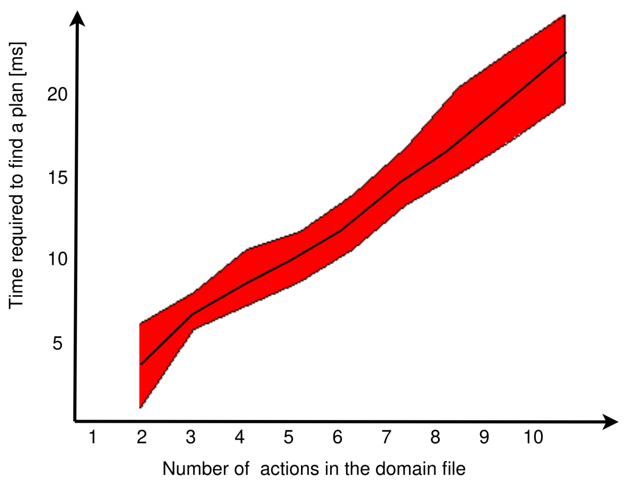


Figure 6.7: The figure shows the time spent by the planner to generate a suitable plan with an increasing number of actions in the domain.

Chapter 7

Conclusions

Traditional assembly lines are too rigid for today's market demand. In fact, in the last years, the market demands moved toward the production of customized products requiring assembly lines to allow for rapid changeovers and efficient teaching. In this context, the time required to teach new processes represents a fundamental aspect to adapt robot to new product variants. Moreover, small and medium enterprises are increasingly adopting robot-based industrial automation that is not easy reprogrammable by non-experts . Programming robots in a more abstract way that does not require robot expert operators is essential today.
This work focuses on developing a new programming method that allows non-skilled shop floor workers to program the robot intuitively. In particular, the operator exploits Virtual Reality during the teaching, demonstrating the actions the robot needs to learn and what is the new task. Then, the system recognizes and classifies the actions, extracts the relevant preconditions and effects and automatically generates a planning domain. Characterizing actions with preconditions allows the robot to check if the action can be executed, while the effects describe what has to change after the action execution. This allows to check if the action has been performed correctly. In this way, the robot can spot and deal with possible errors and replan accordingly. We performed a series of experiments to evaluate the effectiveness of our work both in simulation and with the real robot. In particular, the robot had to reach different predefined goals using the learnt action during the demonstration phase. The results highlighted how

fast and intuitively the operator can program the robot with respect to traditional programming modes.
As future works, several aspects could be improved.
Firstly, it is possible to replace the cubes with more complicated objects or even consider more industrial applications. Another possible improvement deals with the release action. In our work, the robot is able to release the cubes only in the predefined positions, thus implementing a new method to place the objects in non-predefined positions could improve the robot performances. Moreover, we used PDDL to define the actions in the domain file. Using more complicated languages such as Probabilistic PDDL or PDDL 2.1 could allow the implementation of more complex skills.

Bibliography

[1] Ziyue Guo et al. “Applications of virtual reality in maintenance during the industrial product lifecycle: A systematic review”. In: *Journal of Manufacturing Systems* 56 (2020), pp. 525–538. ISSN: 0278-6125.

[2] Gregory F Rossano et al. “Easy robot programming concepts: An industrial perspective”. In: *2013 IEEE international conference on automation science and engineering (CASE)*. IEEE. 2013, pp. 1119–1126.

[3] Alexander Perzylo et al. “Intuitive instruction of industrial robots: Semantic process descriptions for small lot production”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 2293–2300.

[4] Alexander Perzylo et al. “Toward efficient robot teach-in and semantic process descriptions for small lot sizes”. In: *Proceedings of the Workshop on Combining AI Reasoning and Cognitive Science with Robotics, Robotics: Science and Systems (RSS)*. 2015.

[5] Nikhil Somani et al. “Constraint-based task programming with CAD semantics: From intuitive specification to real-time control”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 2854–2859.

[6] Simon Bøgh et al. “Does your Robot have Skills?” English. In: *Proceedings of the 43rd International Symposium on Robotics*. null ; Conference date: 29-08-2012 Through 31-08-2012. VDE Verlag GMBH, Aug. 2012.

[7] Mikkel Rath Pedersen et al. "On the integration of hardware-abstracted robot skills for use in industrial scenarios". In: *Proceedings of the IEEE/RSJ International Conference on Robots and Systems, Workshop on Cognitive Robotics Systems: Replicating Human Actions and Activities*. 2013.

[8] Franz Steinmetz and Roman Weitschat. "Skill parametrization approaches and skill architecture for human-robot interaction". In: *2016 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2016, pp. 280–285.

[9] Sylvain Calinon. "Learning from demonstration (programming by demonstration)". In: *Encyclopedia of robotics* (2018), pp. 1–8.

[10] Casper Schou et al. "Human-robot interface for instructing industrial tasks using kinesthetic teaching". In: *IEEE ISR 2013*. IEEE. 2013, pp. 1–6.

[11] Ying Siu Liang et al. "End-user programming of low-and high-level actions for robotic task planning". In: *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE. 2019, pp. 1–8.

[12] Constructions Aeronautiques et al. "PDDL| The Planning Domain Definition Language". In: *Technical Report, Tech. Rep.* (1998).

[13] Nichola Abdo et al. "Learning manipulation actions from a few demonstrations". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 1268–1275.

[14] Mirko Wächter et al. "Action sequence reproduction based on automatic segmentation and object-action complexes". In: *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE. 2013, pp. 189–195.

[15] Paul Bakker, Yasuo Kuniyoshi, et al. "Robot see, robot do: An overview of robot imitation". In: *AISB96 Workshop on Learning in Robots and Animals*. Vol. 5. Citeseer. 1996.

[16] Maximilian Diehl, Chris Paxton, and Karinne Ramirez-Amaro. "Automated Generation of Robotic Planning Domains from Observations". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 6732–6738.

[17] Giuseppe Fabio Preziosa. "Learning human actions semantic in virtual reality for a better human-robot collaboration". In: *MSc thesis in Mechanical Engineering, Politecnico di Milano* (2022).

[18] Ning Zhang, Tao Qi, and Yongjia Zhao. "Real-time learning and recognition of assembly activities based on virtual reality demonstration". In: *Sensors* 21.18 (2021), p. 6201.

[19] *STRIPS*. https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/fox03a-html/node2.html.

[20] *Fast-Forward planner*. https://fai.cs.uni-saarland.de/hoffmann/ff.html.

[21] *ROS*. https://www.ros.org.