TÉCNICO LISBOA

# Active Advice Inverse Reinforcement Learning in Large State Spaces

**Gonçalo Miguel Carrola da Silva**

Thesis to obtain the Master of Science Degree in

**Electrical and Computer Engineering**

Supervisor: Prof. Alexandre José Malheiro Bernardino

## Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Prof. Alexandre José Malheiro Bernardino
Member of the Committee: Dr. Tiago Santos Veiga

**November 2021**

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

# Acknowledgments

I would like to thank my parents for all the affection and support they have given me over these 5 long years, and especially for the education and conditions they have provided me throughout my life so that I can be what I am today. Likewise, I thank all my family, brother, parents, grandparents, uncles, cousins and friends for helping me to overcome the difficulties and for helping me to enhance my qualities and overcome my flaws.

I would also like to give a special thanks to my closest friends at IST, they know perfectly well what we had to go through to get here and this was only possible through supporting each other, as no one finishes the course alone. I clearly feel that this whole journey has made me grow a lot as a person and as an engineer and I am very proud to say in the future that I graduated as an engineer at Instituto Superior Técnico.

Last but not least, I would like to thank my thesis advisor, who was always available to help with anything I needed, even with so many students under his wing. Without Professor Alexandre Bernardino, this work would not have been done because many problems would still be unresolved. His experience in the field made me broaden my horizons and understand more clearly the importance and coverage of this area in professional scope.

To each and every one of you - Thank you.

# Abstract

Autonomous agents usually need a decision system in order to know which data should be sent to the actuators depending on the data received in the sensors. This decision system works as the agent's brain, and can be obtained using several learning approaches. In learning from demonstration, the agent learns with the information given by a demonstrator, which shows correct behavior about how to proceed. The main problem of this approach is the need for too many demonstrations to learn large problems. This thesis aims to minimize this issue by using maximum likelihood inverse reinforcement learning followed by active query selection to learn large problems by demonstration and advice, minimizing the number of "trips" to the demonstrator. The implemented algorithm is applied to a grid-world and a grasping problem, which have a large discrete state and action sets. It is proposed a process of clustering of the states to label clusters in order to speed up the process of learning and a query selection method based on uncertainty to give to the demonstrator the most uncertain cluster to label.

# Keywords

Inverse Reinforcement Learning; Reinforcement Learning; Learning from Demonstration; Active Learning; Advice Learning; Large State Spaces.

# Resumo

Agentes autónomos necessitam sempre de um sistema de decisão de modo a saber que dados enviar aos atuadores dependendo dos dados recebidos pelos sensores. Esse sistema de decisão funciona como se fosse o cérebro do agente, e pode ser obtido através de diversas abordagens de aprendizagem. Na aprendizagem por demonstração, o agente aprende através da informação fornecida por um demonstrador que demonstra comportamentos corretos sobre como proceder. O maior problema desta abordagem deve-se à necessidade de muitas demonstrações para aprender problemas grandes. Esta tese tem como objetivo minimizar este problema realizando aprendizagem por reforço inversa por máxima verossimilhança seguido de escolha ativa de dados para aprender por demonstração e por conselho para problemas grandes minimizando o número de idas ao demonstrador. O algoritmo implementado é aplicado a um labirinto e a um problema de agarre, que contém um grande conjunto discreto de estados e de ações. Esta proposta contém um processo de agrupamento de estados para rotular conjuntos de forma a agilizar o processo de aprendizagem e um método de escolha do conjunto baseado em incerteza para dar ao demonstrador o conjunto mais incerto para ser rotulado.

# Palavras Chave

Aprendizagem por reforço inversa; Aprendizagem por reforço; Aprendizagem por demonstrações; Aprendizagem Ativa; Aprendizagem por conselho; Espaços de estados grandes

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Acronyms

| | |
|---|---|
| **ACS** | Active Class Selection |
| **AFAC** | Active Feature Acquisition and Classification |
| **AcL** | Active Learning |
| **AdL** | Advice Learning |
| **ApL** | Apprenticeship Learning |
| **BIRL** | Bayesian Inverse Reinforcement Learning |
| **DH** | Devanit-Hartenberg |
| **DoF** | Degrees of Freedom |
| **EMC** | Expected Model Change |
| **EQL** | Equivalence Query Learning |
| **FK** | Forward Kinematics |
| **IK** | Inverse Kinematics |
| **IRL** | Inverse Reinforcement Learning |
| **LfD** | Learning from Demonstration |
| **LP** | Linear Programming |
| **MDP** | Markov Decision Process |
| **MLIRL** | Maximum Likelihood Inverse Reinforcement Learning |
| **MPC** | Model Parroting and Compression |
| **MQS** | Membership Query Synthesis |
| **NN** | Neural Network |
| **PbAS** | Pool-based Active Sampling |
| **QbC** | Query-by-Committee |

| | |
|---|---|
| **RL** | Reinforcement Learning |
| **ROS** | Robot Operating System |
| **SbSS** | Stream-based Selective Sampling |
| **SSL** | Semi-Supervised Learning |
| **STL** | Standard Triangle Language |
| **URDF** | Unified Robot Description Format |
| **US** | Uncertainty Sampling |

# 1

# Introduction

## Contents

Nowadays it is increasingly necessary to automate processes in order to facilitate people's lives and maximize the performance of tasks. The introduction of autonomous agents in carrying out tasks is an important step to perform that. These autonomous agents must be able to adapt to different situations that may occur outside of their routine. To do this, they need to analyze the environment where they are inserted, evaluate the possible consequences of each action they may take and decide for one that accomplishes in the best way what it is purposed. Basically, what it is wanted in this case is that an autonomous agent analyze a situation based on its knowledge and experience and then make a decision that maximizes the consequence intended.

## 1.1 Motivation

To achieve the purpose of taking the best decision to a certain situation, learning from a human that already has the optimal actions is a great and efficient solution. An agent benefits from learning with human demonstrations, instead of going through trial and error. We can see this idea as having a human being who teaches a robot in the same way that a teacher teaches his student to perform a certain task. For example, a human being wants to teach a robot to wave back to his neighbor whenever he waves to him. The human being warns the robot that whenever he detects the neighbor's waving (for example, if he raises an arm), he must also perform the movement of raising his arm. Once this is done, the robot will learn to this experience and will be able to perform this action whenever this situation is detected.

Humans are able to sense a given event, through neurological signals given and received to the brain. The brain will then decide how to proceed based on the neurological signals it receives, whether it is raising an arm or turning the head up for example. A robot can be designed to look as much as possible as a human being in reacting to external stimulation, so it is possible to make an analogy. The senses of the robot are obtained through sensors that measure data from the environment, his actions will be done by actuators and his decisions will be made by a decision system that will act like the robot's brain. This decision system stores the knowledge taken by previous experience and with that information makes the decision he thinks it is the best for a certain situation. That is why learning from demonstrations seems so natural: a robot must learn from someone more experienced, in order to achieve optimal decisions faster. In this way, robots can eventually perform the tasks taught by humans and thereby help them in solving tasks more efficiently.

## 1.2 Problem Statement

Learning from demonstration is a natural way of thinking about learning processes, as it is similar to the way humans learn. However, this approach has some particular problems that are not desirable in the area of robot learning. First of all, a lot of training is necessary to obtain a strong model that allows the robot to know what it has to do in a given situation. In other words, the demonstrator (that is often an human being) will have to give to the robot a lot of demonstrations about the behavior of the environment. An agent can perform several actions in a certain state configuration of the environment, however it will exist one action that brings the best consequence. This consequence is called a reward in machine learning literature. However, it is more desirable to achieve better rewards after a sequence of actions, even if that means that the reward is lower in the beginning, so it is introduced the concept of action values, that are nothing more than accumulated rewards. A policy is the decision rule that selects the action that the robot must perform for each state. The agent only needs a policy to decide which action to take. If the policy has not all optimal decisions, the agent can perform bad actions, so what is intended is to make the policy the most optimal possible. However, the agent can have limited memory resources, so it is crucial that this information be the most informative possible.

The problem that is presented becomes worse for large state spaces, which makes that it is necessary an even bigger amount of data to perform well. To fight this issue, the concept of Inverse Reinforcement Learning (IRL) was born [3] [4] [5] [6] [7] [8] [9] [2] [10] [11] [12] [13], a concept that obtains a reward function from the currently known human demonstrations, which will give the information about the consequence of performing the action known as the best for a given state, following the given policy. From this information it is possible to update the action values and consequently change the policy at an iterative way. In very large state spaces it is also difficult to perform enough experiments on all of the states. There are states that rarely happen, so they do not need as many samples, or even states that do not need much learning because it is evident what is the best action to take. Even with these considerations, the fact is that the policy will not be good enough, because even for the states rarely visited a bad decision is always preferable to avoid. To minimize the problems mentioned, it is crucial that the samples given to the robot are as informative as possible, so that less data is needed to obtain similar results. If the agent is not so certain of what action it should take in the respective state, then it is best to request to the user the correct information instead of unrelated demonstrations that end up becoming contradictory or redundant. This search for the most informative features is the goal of Active Learning (AcL) [1]. To choose the most informative features it is necessary to first evaluate them according to its informativeness. The most used measure of informativeness in the literature is uncertainty, as will be explained later. Another problem consists of the possible errors made by the demonstrator. The demonstrator is usually an human being, and like all humans it can make mistakes. This issue must be considered, because it can lead to contradictions on some demonstrations and the learning

iteration will not be as much effective as we expect. Advice Learning (AdL) makes it possible to provide information to the agent that is less susceptible to errors, thus being able to correct possible suboptimal demonstrations. Advice in this context is given by a set of preferences in state/action/reward spaces. Advice differs from demonstration in the sense that advice can cover more information about the state, namely whether a certain feature is beneficial or detrimental to the problem, whereas a demonstration only refers to the best action for a certain state.

The goal of this work is to minimize the impact of these issues, applying methods that make the policy converge to optimal as soon as possible, knowing its limitations. Later we will see that information in the form of advice can help not only to correct some contradictory demonstrations but also to know more about non visited states, since this form of information is very cheap to produce.

## 1.3 Objectives

The goal of this work is to speed up the process of learning from demonstration, applying methods that make the policy converge to optimal more quickly in large state spaces.

To perform that, it will be used some techniques to maximize the informativeness of the "lessons" given by the user, by focusing on the areas of the state space that the agent does not know so well. If we focus on those less certain areas to request and then learn, we will need less demonstrations to perform a good policy. The algorithm will also incorporate information in the advice format to complement learning by demonstration and we will see the impact of advice on non visited states and on the correction of contradictory demonstrations.

The proposed method will be implemented on a simple grid-world problem and a more complicated grasping problem, with a large discrete state and action spaces to test the algorithm effectiveness.

## 1.4 Organization of the Document

The thesis is organized as follows: in Chapter 1 it is given a motivation and the problem definition of the thesis. In Chapter 2 it is introduced a framework of the theme in the specific scientific area of machine learning. In Chapter 3 it is introduced some background of the concepts and known algorithms that inspire the final framework of this work. In Chapter 4 it is explained the methodology and the definition of the problem, with a view to solve the problem that is intended. In Chapter 5 it is explained in detail all the process of development of the work thesis. In Chapter 6 it is presented some expected results in addition to analyzing the results of the final experiments. Chapter 7 concludes this work by summarizing it.

# 2

# Related work

## Contents

The aim of the work is to create a decision system based on the knowledge acquired by the robot from its "master". A decision system is often implemented throughout the literature using Reinforcement Learning (RL) concepts. In RL, a policy is obtained using a reward function, which works as the consequence of performing a certain action *a* from a state *s*. However, in learning from demonstrations the goal is to obtain a reward function using a set of demonstrations given by the user, and then update the policy with the obtained reward function.

## 2.1 Reinforcement Learning

*Sutton and Barto (2018)* [14] presents all the theory and concepts behind RL. The main idea of RL is to find an optimal policy that maximizes an accumulated reward for each state to be considered. This accumulated reward is called state value ($V^{\pi}(s)$). As already mentioned, a policy $\pi$ is a representation of which action should an agent perform for each state. This policy is calculated from the recurring update of its state values. Another important concept is the Markov Decision Process (MDP). An MDP is a set of information that describes the environment involved, and in which the robot will interact to obtain the "learning material", that is, the current state, the action to perform, the next state and its reward. An MDP always contains a set of states $S$ and a set of actions $A$, and may also contain other additional information to complement the representation of the problem such as a transition model (usually in the form of a transition probability matrix from $s$ to $s'$ performing $a$), a discount factor, a policy or a reward function. RL has several algorithms associated, and the approach to use depends on the information given to the MDP. We can obtain a policy through on-policy or off-policy methods, with known or unknown environment model, depending on the problem we want to solve.

## 2.2 Learning from Demonstration

Learning from Demonstration (LfD) is a learning concept that allows to obtain a policy through a set of demonstrations of an human being. *Argall et al. (2008)* [15] presents a survey about LfD, that defines design choices in terms of the demonstrator, the state space and the performance and analyzes several examples and techniques for the derivation of the policy. They end their work by mentioning the main limitations of LfD, such as the already mentioned scarcity of demonstrations in certain states, or the possible poor quality of the existing demonstrations.

## 2.3 Inverse Reinforcement Learning

IRL is a derivation of RL, where instead of trying to compute a policy from a reward function, the opposite is intended, that is, the reward function is the goal of the algorithm. IRL is even the most used method to approach the problem of LfD. From demonstrations, the agent computes the necessary rewards to update the action values $Q^{\pi}(s, a)$ for each state.

*Russell (1998)* [4] was one of the first published works on the subject of IRL. The work proposes an architecture based on RL that can handle stochastic, partially observable environments. But the main input of this work to the area of LfD is the first use of IRL to provide a good model of animal and human learning. *Russell et al. (2000)* [5] introduces the basic concepts of IRL. They propose three algorithms for IRL. Two deal with cases when policy is entirely known, although one of them considers a finite state space and the other considers a linear functional approximation of the reward function over an infinite state space (i.e. a continuous state space). The third algorithm deals with the case when the policy is partially known only through a finite set of observed trajectories (i.e. demonstrations). In all three algorithms it is proposed a linear programming formulation for the IRL problem in order to pick a reward function that differentiates the observed policy from the other (possible) suboptimal policies. The reward function will be computed with the resolution of an optimization problem where the demonstration will act like constraints to maximize the reward function.

IRL algorithms can be applied in many other approaches that need to obtain a reward function. *Neu et al. (2009)* [10] show how IRL algorithms can be applied to structured prediction, in particular to parser training. *Abbeel et al. (2004)* [12] introduce the application of IRL to resolve an Apprenticeship Learning (ApL) problem. They create an IRL algorithm to learn the reward function of a task demonstrated by an expert. This reward function is expressible as a linear combination of known features, and the algorithm tries to recover it in a small amount of iterations. In this framework the reward function is represented as a linear combination between the known features and unknown weights that are computed to maximize the reward. However, apprenticeship IRL is not a good choice for determining deterministic policies, as the policies derived from this framework are always stochastic.

To sum up, *Gao et al. (2012)* [13] present a survey of IRL techniques with an overview of the theoretical background, applications and some refinements of IRL. This work can be seen as an introduction guide of theory and applications of IRL.

### 2.3.1 Inverse reinforcement Learning for large state spaces

As will be seen in Chapter 3, performing IRL for large state spaces is a very difficult task, since the basis of IRL consists in the realization of optimization problems in order to maximize the reward and at the same time ensure the Bellman's Optimality Equation (explained in Section 3.2). Problems with too many components and constraints will make the computation too heavy to be feasible.

*Russell et. al (2000)* [5] also presents the first solution for performing IRL in large state spaces. In their framework, the reward function is presented as a linear combination of previously known state functions, and the goal is to determine the weights for each component in order to complete the entire reward function based on the Bellman Optimality Equation. Although this solution substantially reduces the number of features to be computed, the number of constraints is still an unsolved issue, a problem that is alleviated in this work by considering only a subset of states to comply with Bellman's condition.

Other works were also developed in order to reduce the impact of the increase in the state space in computational terms, with *Burdick & Li (2017)* [3] being the one with the most success in this reduction. To avoid solving the computationally expensive reinforcement learning problems in reward learning, this work computes a function approximation method to ensure that the Bellman Optimality Equation always holds, and then obtains a function to maximize the likelihood of the observed demonstrations (known as Maximum Likelihood Inverse Reinforcement Learning (MLIRL)). The time complexity of the proposed method is linearly proportional to the cardinality of the action set, thus it can handle large state spaces efficiently. The work shows that the developed framework is more accurate than existing methods and significantly better in scalability. The algorithm developed in *Burdick & Li (2017)* [3] will be the basis of the algorithm developed in this thesis.

## 2.4 Active Learning

AcL consists of choosing a set of the most informative unlabelled features to be labelled, in order to minimize the amount of data necessary to train a model. The strategy of giving queries to an oracle (i.e. the demonstrator) so he can label them is a practical and useful method in modern learning literature. This strategy is used in many areas such as Semi-Supervised Learning (SSL), Equivalence Query Learning (EQL), Active Class Selection (ACS), Active Feature Acquisition and Classification (AFAC), Model Parroting and Compression (MPC), and of course RL.

AcL is a special case in machine learning where a learning algorithm can interactively query an user to label new data points with the desired outputs. There are situations in which unlabeled data is abundant but manual labeling is expensive. In such a scenario, learning algorithms can actively query the user for labels. With AcL it is possible to obtain the same results with less necessary labelled data. Those queries can be chosen using several different strategy frameworks based on informativeness

analysis. *Burr Settles (2009)* [1] introduces the whole basic concepts about AcL. In the work are defined some different scenarios such as Membership Query Synthesis (MQS), Stream-based Selective Sampling (SbSS) or Pool-based Active Sampling (PbAS) and strategy frameworks such as Query-by-Committee (QbC), Expected Model Change (EMC), the already mentioned Uncertainty Sampling (US), among others. US is the most used query strategy, and consists of choosing the feature where the learning algorithm is less certain about a task, so it needs more labelled information.

Several works also used the uncertainty query strategy to learn interactively. *Racca et al. (2018)* [16] propose an AcL framework for modelling user preferences about task execution. The proposed approach interactively gathers information by asking questions expressed in natural language. The results show that active strategies are applicable for learning preferences in temporal tasks from non-expert users.

## 2.5 Active Learning applied to Reinforcement Learning

Several works in the literature already propose algorithms that involve combinations between AcL and RL. *Mihalkova and Mooney (2006)* [17] proposes a new framework for aiding a reinforcement learner by allowing it to relocate to a state that it selects so as to decrease the number of necessary steps to obtain an effective policy. This framework requires a minimal amount of human involvement and assumes a cost for each relocation. This relocation is specially applied on Q-learning, that is an off-policy method. The addition of this ability can bring some advantages in some domains that do not involve the knowledge of a transition model. *Fang et al. (2017)* [18] introduces a new formulation by reframing the active learning as a RL problem and explicitly learning a data selection policy, where the policy takes the role of the active learning heuristic. They demonstrate their method in cross-lingual named entity recognition and they obtain some improvements compared to traditional AcL.

## 2.6 Inverse Reinforcement Learning for active querying

*Lopes et al. (2009)* [6] uses a variant of IRL called Bayesian Inverse Reinforcement Learning (BIRL) which consists of obtaining a reward function that presents the maximum *a posteriori* probability given a set of $D$ samples. IRL is cast as an inference problem, in which the agent is provided with a noisy sample of the desired policy from which it must estimate a reward function explaining the policy. However, the main input of that work is the introduction to the idea of using AcL to update the policy recursively, asking the demonstrator about new samples that involve more uncertain states. When choosing samples that are more targeted to certain states, a policy is learned more quickly with less necessary data. The metric used in [6] for choosing the query was uncertainty (more explicitly entropy), although there are many other query strategies in the literature. However, the approach taken in [6] is still not fast, as

a single demonstration will not make a short-term difference in the update of the policy, making the convergence towards an optimal policy still slow.

Other works have proposed IRL methods with active queries, but using alternatives to US. *Brown and Niekum (2018)* [7] proposes a sampling method based on BIRL that uses demonstrations to allow an agent to bound the difference in expected return between the agent's own policy and the optimal policy for the task, under the expert's unknown reward function. This method determines an high-confidence policy performance bounds in a IRL setting, where the true reward function is unknown and only demonstrations are given. Later, *Brown et al. (2019)* [8] extends this work by applying the active query technique used in [6] to minimize the performance risk of the policy the robot is learning. They propose a risk-aware active IRL algorithm that focuses active queries on areas of the state space with the potential for large generalization error. They achieve better results than some standard active IRL frameworks that do not consider performance, while also provides a performance-based stopping criteria. This stopping criteria is an advantage because it allows to stop giving demonstrations to safely perform a task.

## 2.7 Advice Learning: learning by advice vs learning by demonstrations

After the work of *Lopes et al. (2009)* [6] had been published, several works start to appear to improve even more the idea of maximizing the speed of convergence of a policy to its optimal version. *Kunapuli et al. (2013)* [9] proposed a new extension for IRL by introducing the concept of AdL in IRL problems. It is important to emphasize that learning by advice and learning by demonstrations are different solutions, however very easily combined in the context of IRL. Advice is usually given by an expert, whereas demonstrations are given by a demonstrator. However, the natural thing is to be the same human being to perform both roles. AdL consists of assigning preferences to certain features (which can be actions, states or even rewards). Two subsets of features are then created: a subset of preferable features **Pref** and a subset of features to avoid **Av**. These preferences can be applied in three different ways. *Action Preferences* assigns preferred actions and to be avoided by the robot for a certain state. *State Preferences* gives preference to a certain subset of states to be explored. *Reward preferences* express preferences for certain immediate rewards, rather than their long-term values (i.e. state values). The combination between IRL and AdL allows the IRL algorithm to be performed through not only demonstrations but also the set of advice. *Kunapuli et al.* [9] was inspired by the work done in [5] to propose a new version of the IRL formulation through an optimization problem. This approach was proposed to minimize two assumptions that exist in LfD problems: demonstrator with optimal decisions and a sufficiently large set of demonstrations. In fact this approach is proved to lead to better results compared to

traditional LfD algorithms, specially with noisy demonstrations [9]. However, the algorithm proposed has some practical problems, specially the computational cost of giving advice to all states that exist in the space state.

*Odom and Natarajan (2016)* [2] propose a strong solution to this issue. They applied the idea of choosing queries to obtain new specific features. Instead of giving these queries to the demonstrator, they will be given to the expert so it can sample new advice to the query chosen. This approach allows not only greater flexibility in the information given by the human annotator to the agent, but also allows a better interaction with him, since the advice method is less prone to human errors. This flexibility also allows the attribution of a greater amount of information to the agent for each interaction. For each interaction of the algorithm proposed, a set of new preferable and avoidable features is immediately given. In this way, it is necessary to interact less often with the human annotator to obtain the same quality of policy, which goes in favour to the solution of the proposed problem. In addition, the algorithm proposed in [2] (called *ADVISE*) also allows the division of the state space into clusters according to the similarity in state's characteristics. With this clustering in mind, the query that is chosen will be a cluster of states instead of a single state. These clusters allow us to capture more information in each interaction, since the expert's answer for all the cluster is very likely to be the same. This active advice-seeking approach was also used in some other works like *Odom et al. (2016)* [11], where they introduce active advice-seeking for relational domains. Relational logic allows for compact, but expressive interaction between the human expert and the learning algorithm.

## 2.8 Research Gaps

All the state of the art mentioned refer strategies to speed up the computation of an optimal policy. Some assumptions are made on primordial IRL literature, and some works try to minimize and solve the intended problems without these considerations. However, it is also possible to achieve more learning rate or less necessary data to achieve an optimal policy. The algorithms implemented in some works would not be experienced on such a large discrete state space as a simple grasping problem, and can be upgraded to even more informativeness of the queries and even better cluster assignment. Other IRL algorithms deal well with large state spaces, but do not present alternatives for the lack of information that exists about the policy (namely lack of demonstrations or other forms of information such as advice).

# 3

# Background

## Contents

In this chapter I intend to give the reader a brief feedback about the fundamental theory for this work. Several notations and learning components will be addressed and the *ADVISE* algorithm will also be discussed in more detail, as well as its limitations in relation to large state spaces. The algorithm described in *Burdick & Li (2017)* [3] will also be explained, as it is fundamental for handling IRL in large state spaces.

## 3.1 Reinforcement Learning notation

Let's start with the notation used in this work regarding the learning components. A policy $\pi$ consists of a mapping of the action space in the state space. In other words, a policy in this context is a vector where, for each state, the action that the agent must perform for that situation is given (i.e. deterministic), thus functioning as the agent's decision system. The action value matrix $Q^{\pi} : S \times A$ represents the learning values for each state-action pair. The state value vector $V^{\pi}$ defines the largest action value for each state. Finally, a reward function $R$ corresponds in this context to a vector where a certain reward is given when the environment reaches a certain state. The better for the goal to reach that state, the greater the value of the reward function will be.

Another component that will be present in the problem's MDP is the transition model $P : S \times A \times S$, characteristic of the environment responsible for attributing a certain probability of occurring a state $s'$ when performing the state-action pair $(s, a)$. The transition model is not always available in a MDP, so there are model-free learning algorithms based only on demonstrations responsible for knowing the environment's behavior and learning a policy at the same time, however it is not the case in this work where the transition model is learned by exploration of the environment.

The policy of a given state $s$ will then be the action that presents the highest action value in the matrix $Q^{\pi}$, that is, $Q^{\pi}(s, a^*) = V^{\pi}(s) \Rightarrow \pi(s) = a^*$. The reward function will then be fundamental to compute the optimal policy, as it is necessary for updating the action values and, consequently, updating the policy. However, in many environments it is difficult to obtain an accurate reward function.

## 3.2 Bellman equations

As previously said, IRL is useful in domains where defining the reward function is difficult, but providing demonstrations of the correct behavior is more practical. First, I will introduce some important and fundamental concepts for learning. When the environment transition model is known, the state values $V^{\pi}$ and the action values $Q^{\pi}$ of a certain policy can be updated using the two Bellman equations:

$$V^{\pi}(s) = \sum_{s'} P_{sa^*}(s') \times (R(s') + \gamma \times V^{\pi}(s')) \qquad , \tag{3.1}$$

$$Q^{\pi}(s,a) = \sum_{s'} P_{sa}(s') \times (R(s') + \gamma \times V^{\pi}(s')) \quad , \tag{3.2}$$

where $\gamma$ is the discount factor, which denotes a discount on the previous state values in order to prioritize the actual reward, and $a^*$ is the optimal action for a state $s$ (i.e. $\pi(s)$). The policy is then updated using the Bellman optimality:

$$\pi(s) = \underset{a \in A}{\operatorname{argmax}}\ Q^{\pi}(s,a) \quad , \tag{3.3}$$

i.e., the action mapped in the policy $\pi$ for a given state $s$ is the one with the highest action value for that state. From Bellman's equations, we can guarantee that a given policy $\pi$ is optimal iif the reward function $R$ satisfies the following inequality (proof in Appendix B.1):

$$(P_{a^*} - P_a) \times (I - \gamma \times P_{a^*})^{-1} \times R \succeq 0 \quad . \tag{3.4}$$

## 3.3 IRL for finite state spaces

One natural way of finding a reward function that makes a policy optimal is to demand that it satisfies (3.4). So, *Ng & Russell (2000)* [5] propose one so as to maximize:

$$\max \left( \sum_{s \in S} \quad Q^{\pi}(s,a^*) - \max_{a \in A \setminus a^*} Q^{\pi}(s,a) \right) \quad , \tag{3.5}$$

which is the same as:

$$\max \left( \sum_{s \in S} \quad \min_{a \in A \setminus a^*} \left( (P_{a^*} - P_a) \times (I - \gamma \times P_{a^*})^{-1} \times R \right) \right) \quad . \tag{3.6}$$

In short, according to *Ng & Russell (2000)* [5], the optimization problem consists of maximizing (3.6) with the constraint that $R$ must fulfill (3.4). The optimization problem is then given by:

$$\begin{aligned} \max_{R} \quad & -\lambda ||R||_1 + \sum_{s \in S} \quad \min_{a \in A \setminus a^*} \left( (P_{a^*}(s) - P_a(s)) \times (I - \gamma \times P_{a^*})^{-1} \times R \right) \\ \text{s.t.} \quad & (P_{a^*}(s) - P_a(s)) \times (I - \gamma \times P_{a^*})^{-1} \times R \succeq 0\,, \qquad \forall a \in A \setminus a^* \quad , \\ & |R_i| \leq R_{max}\,, \qquad i = 1, \ldots, N \quad . \end{aligned} \tag{3.7}$$

## 3.4 Advice Learning (AdL)

In order to obtain the reward function from the optimization problem (3.7), it is necessary to obtain a policy, in order to know which actions are optimal for each state, since IRL finds a reward function that best fits with a certain policy. However, the aim of this work is precisely to obtain the optimal policy for a grasping problem. Bearing in mind that the policy itself is initially unknown, it is necessary to learn this policy through demonstrations. Therefore, learning via demonstrations presents two problems, as already mentioned: the high possibility of there being a lack of demonstrations in problems with larger state spaces and the consideration that these belong to the optimal policy. To minimize these two problems, *Kunapuli et al. (2013)* [9] introduced the idea of using advice as a complement to trajectories (i.e. demonstrations) so that fewer trajectories are needed and minimize the problem of their inevitable sub-optimality. The advice will be given in the form of action preferences, which consist of two subsets of actions (preferable or avoidable) for each state. An action can not belong to both subsets, since we should guarantee that:

$$\min_{a \in Pref_s} Q^{\pi}(s,a) \geqslant \max_{a' \in Avd_s} Q^{\pi}(s,a') \quad . \tag{3.8}$$

Advice allow us to give more comprehensive information about each state, which allows to cross off certain actions and prioritize others in search of the best action for each situation. That's why it is possible to use advice as a complement to LfD. Taking into account that trajectories are now considered as sub-optimal, the constraint used in (3.7) is relaxed in order to ensure feasibility if the expert's action preferences possibly contradict the trajectories. The new IRL algorithm proposed by *Kunapuli et al. (2013)* [9] is given by the following optimization problem:

$$\begin{aligned}
\max_{R,\xi_i,\zeta_j} \quad & -||R||_1 + \lambda_d \sum_{i=1}^{N} \xi_i + \lambda_a \sum_{j \in S_e} \zeta_j \quad , \\
\text{s.t.} \quad & (P^i_{a^*} - P^i_a) \times (I - \gamma \times P_{a^*})^{-1} \times R \geqslant \xi_i \,, && \forall a \in A \setminus a^*, \quad i = 1, \ldots, N \quad , \\
& (P^j_a - P^j_{a'}) \times (I - \gamma \times P_{a^*})^{-1} \times R \geqslant \zeta_j + \delta_j \,, && \forall a \in Pref_j \,, \quad a' \in Av_j \,, \quad j \in S_e \quad , \\
& \zeta_j \geqslant 0 \,, && \forall j \in S_e \quad , \\
& |R_i| \leqslant R_{max} \,, && \forall i = 1, \ldots, N \quad .
\end{aligned} \tag{3.9}$$

$P^i_a$ is the transition probability row vector of a state $i$ taking the action $a$. $P_a$ is the transition matrix from state $s$ to state $s'$, given the action $a$. $a^*$ is the optimal action associated to state $i$. $R_{max}$ is the maximum value that a reward can reach, while $R_i$ is the reward associated to state $i$. $\xi_i$ and $\zeta_j$ are variables that are intended to maximize. The goal of these variables is to maximize the minimum value of the constraints, for the demonstrations and advice respectively. $Pref_j$ and $Av_j$ are the sets of preferable

and avoidable actions for the state $j$ respectively. $\delta_j$ is a regulable parameter that controls the hardness of each preference for a state $j$. If we set $\delta_j > 0$, this enforces a hard margin on the preferences; a larger $\delta_j$ makes the constraint harder to violate. Alternately, if we set $\delta_j \leqslant 0$, the constraint becomes softer advice, which is easier to satisfy. Thus, $\delta_j$ reflects how rigorously the expert prefers the advice to be used. $a^*$ is the action with the highest value of $Q(s, a)$ for a given state $s$. $S_e$ corresponds to the set of states with provided advice, while $N$ is the total number of states visited. $\lambda_d$ and $\lambda_a$ are adjustable parameters that define the degree of confidence that one has in relation to the demonstrations and advice given respectively. Finally, the $l_1$-regularizer in $R$ helps to learn sparse rewards. Note that $\xi_i$ does not have to be greater than 0 anymore, as they relax the constraint to ensure feasibility if the expert's action preferences possibly contradict the trajectories.

## 3.5 IRL for infinite state spaces

Although the concept of advice helps in the presence of insufficient demonstrations (both in quality and quantity), we still have a troubling problem for large state spaces, as there will certainly be problems in linear programming due to excessive constraints and excessive dimension of the variables involved in the problem. For example, performing an inversion of a large matrix is very expensive in time, which prevents the algorithm from being done for computational reasons. Furthermore, it will be very expensive to optimize such a large reward function.

One way of trying to solve these limitations would be to consider the reward function as a linear combination of base state functions, passing the reward function from discrete to continuous, as referred by *Ng. & Russell (2000)* [5]. Assuming that $\Phi = \phi_1, \ldots, \phi_d$ are fixed, known bounded basis functions mapping the state space that evaluate a certain state according to some criteria, the new aim of IRL is to find a set of weights $\Omega = \omega_1, \ldots, \omega_d$ in order to fit the reward function to a certain policy, assuming that:

$$R(s) = \omega_1 \times \phi_1(s) + \ldots + \omega_d \times \phi_d(s) \qquad . \tag{3.10}$$

By the linearity of expectations, the value function $V^\pi$ when the reward function is given as (3.10) is also a linear combination such as:

$$V^\pi(s) = \sum_{k=1}^{d} \omega_k \times V_k^\pi(s) \qquad , \tag{3.11}$$

where $V_k^\pi$ is the value function when the reward function is $R = \phi_k$. However, this formulation maintains the problem of excess constraints in the optimization problem. The simplest solution will then be to reduce the number of constraints by considering only a subset of states. This is possible due to

the fact that the reward function is now continuous, which makes it possible to compute it even for states not considered in the optimization problem. The subset of states $S_0$ must therefore be as representative as possible of all possible situations in the environment. The continuous state space IRL algorithm proposed by *Ng. & Russell (2000)* [5] is then given by:

$$\begin{aligned} \max_{\Omega} \quad & \sum_{s \in S_0} \min_{a \in A \setminus a^*} p\left( \sum_{s'} \quad (P_{sa^*}(s') - P_{sa}(s')) \times V^{\pi}(s') \right) \\ \text{s.t.} \quad & |\omega_k| \leqslant 1\,, \qquad k = 1, \ldots, d \quad , \end{aligned} \tag{3.12}$$

where $p$ is given by $p(x) = x$ if $x \geqslant 0$ and $p(x) = 2x$ otherwise, and penalizes violations of the constraints. Although this adaptation allows a more accessible computation of the IRL problem, the computational cost continues to be excessively high for the problem at hand.

## 3.6 Maximum Likelihood Inverse Reinforcement Learning (MLIRL)

MLIRL is another IRL approach based on probabilistic methods. This method consists of maximizing the probability of a given set of demonstrations taking place. Thus, the goal is to optimize a set of parameters that allow to obtain the maximum likelihood of a given set of demonstrations. MLIRL was introduced by *Vroman (2014)* [19], where we aim to optimize the weights of the state base functions, as in (3.10). However, this optimization is done by obtaining the maximum log likelihood of the set of demonstrations:

$$L(D|\Omega) = \sum_{d \in D} \sum_{(s,a) \in d} \log \pi_{\Omega}(s,a) \qquad , \tag{3.13}$$

where $\pi_{\Omega}(s,a) = \frac{e^{\beta Q_{\Omega}(s,a)}}{\sum_{a' \in A} e^{\beta Q_{\Omega}(s,a)}}$ and $\beta$ is the Boltzmann constant. A gradient ascent method can be used to learn the parameter $\Omega$:

$$\Omega = \Omega + \alpha \times \nabla_{\Omega} L(D|\Omega) \qquad , \tag{3.14}$$

where $\alpha$ is the learning rate. The parameter $\Omega$ will then be constantly updated until it converges, then we can update $R$, $V$ and $Q$ using (3.10), (3.1) and (3.2) respectively.

## 3.7 Inverse Reinforcement Learning via Function Approximation

Although the IRL methods mentioned in the previous sections present good results for small state spaces, the same does not happen for very large state spaces, mainly due to the computational cost of performing RL problems in reward learning. Algorithms based on Linear Programming (LP) always

end up having too many constraints in order to solve the problem in a reasonable time horizon, such as explained in the previous sections.

In order to handle large state spaces, *Burdick & Li (2017)* [3] propose a function approximation method to avoid this issue by ensuring that the Bellman Optimality Equation always holds and by estimating a function to maximize the likelihood of the demonstration based on MLIRL method.

### 3.7.1 Function Approximation Framework

Given the action space and the transition model, a reward function leads to a unique optimal value function. To learn the reward function from the observed demonstrations, instead of directly learning the reward function, they use a parameterized function, named as VR function, to represent the summation of the reward function and the discounted optimal value function:

$$f(s, \Theta) = R(s) + \gamma \times V^*(s) \quad , \tag{3.15}$$

where $\Theta$ denote the parameters of VR function. Substituting (3.15) into (3.1) and (3.2), we will obtain equations of $R$, $V$ and $Q$ that only depend on the VR function:

$$V^\pi(s) = \sum_{s'} \quad P_{sa^*}(s') \times f(s', \Theta) \quad , \tag{3.16}$$

$$Q^\pi(s, a) = \sum_{s'} \quad P_{sa}(s') \times f(s', \Theta) \quad , \tag{3.17}$$

$$R(s) = f(s, \Theta) - \sum_{s'} \quad P_{sa^*}(s') \times f(s', \Theta) \quad . \tag{3.18}$$

This approximation method is related to value function approximation method in reinforcement learning, but the proposed method can compute the learning parameters without solving a set of linear equations in stochastic environments. If we apply the MLIRL algorithm using the VR function approximation, the likelihood of the demonstrations is given by:

$$L(D|\Theta) = \sum_{(s,a) \in D} \left( \beta \times \sum_{s'} P_{sa}(s') \times f(s', \Theta) - \log \sum_{a' \in A} e^{\beta \sum_{s'} P_{sa'}(s') \times f(s', \Theta)} \right) \quad . \tag{3.19}$$

Using the gradient ascent to update the parameter $\Theta$ as in (3.14) until reaching convergence allow us to obtain a VR function that ensures the Bellman Optimality condition and that maximizes the likelihood of the demonstrations. To compute $f(s, \Theta)$ we will assume that the VR function is a neural network where $\Theta$ are the weights and biases of the neural network and $L(D|\Theta)$ is the Loss function.

# 4

# Methodology

## Contents

In order to implement the idea proposed in the thesis, the goal is to design a decision system applied to a simple grid-world problem and to a grasping problem. The real motivation for this thesis is the implementation of this decision system based on features given by the user, which will have the role of demonstrator and expert simultaneously. Initially the MDP is presented, where the entire state space, possible actions for each state and a transition model are defined.

The grid-world problem consists on a grid where the goal state is the center of the grid. Each cell is a state, and the possible actions correspond to right, up, left and down. This grid will have several different sizes. The grasping problem consists on a tabletop scenario with a robotic hand with only 2 fingers that can grasp several objects. The goal of the problem is to successfully grasp a certain object, and then move it to a certain goal pose according to some criteria defined by a human being. First it is necessary to successfully make a solid grasp of the object (i.e. grasp the object without fall or change its orientation). Then, several actions can be performed to reach the goal, be it rotating the gripper and then moving to the desired position or the opposite, as long as the object ends up in the desired pose (position + orientation). The robot used to perform the grasping problem was firstly the *KINOVA Gen3 Lite*, a simple 6 Degrees of Freedom (DoF) manipulator with a 2-fingered gripper in the tip. Therefore, after some problems and considerations with the mentioned manipulator, it was decided to change the manipulator to its improved model: the *KINOVA Gen3* is a simple 7-DoF manipulator without an integrated gripper, however a gripper with 2 fingers (*robotiq_2f_85*) was added to it. This last manipulator also contains an *Intel Realsense D410* camera, including a color sensor and a depth sensor enabled by both stereo vision and active IR sensing, that was not used in this work. Next, it is presented the methodology used to tackle this problem, including the tools in previous works.

## 4.1 MDP Definition

The MDP of the grid-world problem consists of a discrete finite set of states, which represent each cell of the grid, and a set of actions that consists of moving to left, right, up or down.

The MDP of the problem consists of a discrete finite set of states, which represent a description of the object to be grasped and its arrangement in the environment, and a set of actions that consists in a first instance of the possible grasping options, and then the options to rotate or move the robotic arm to a specific goal-state. The problem description was motivated by the work of *Coelho (2013)* [20], which proposes an MDP and a simple environment to be used in planning grasp and push tasks. The environment consists of a table, an object placed on top of the table and the manipulator used. This environment will be simulated using *Gazebo* simulation environment. The table is presented through a fixed 3D grid. A table representation example can be found in Figure 4.1.

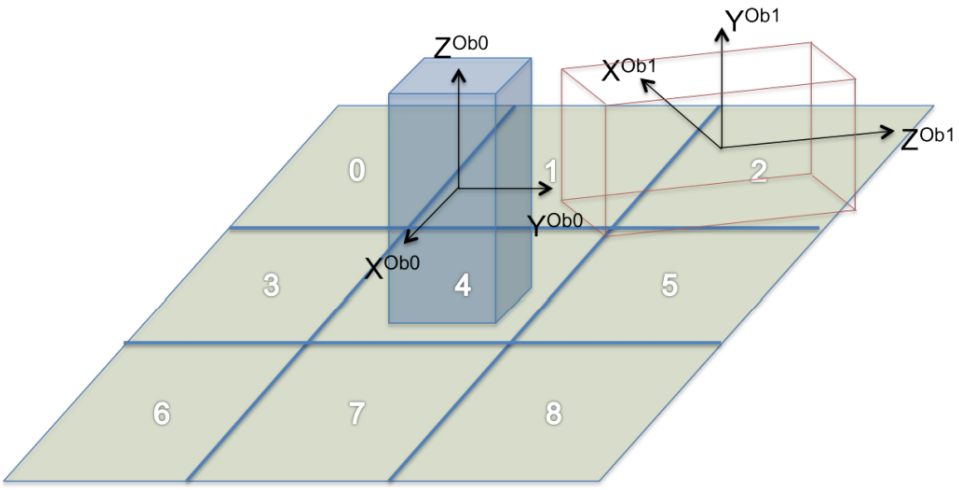



**Figure 4.1:** Grid representation on a 3D view (Coelho 2013).

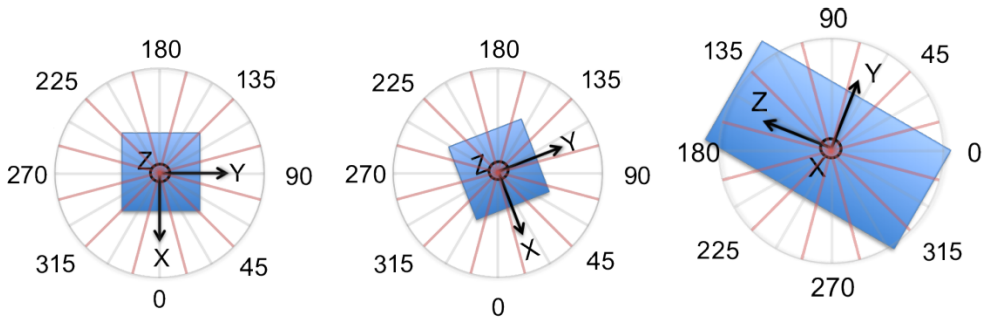
### 4.1.1 State Space

For the grid-world problem, the state is the description of a single cell and has features $s = [x, y]$, $x, y \in [0, 1]$. For the grasping problem, as already mentioned, the state is the description of the object in the environment. Each state has the following parameters: pose (orientation + position in table coordinates) and current situation of the grasp (whether or not the gripper is already grasping the object). If the object is being grasped by the gripper, the gripper orientation is added as parameter too. A $4 \times 4 \times 4$ cm cube is always considered as the target object to be grasped. The object's reference frame has the same orientation as the world reference frame.

The first component of the state corresponds to the object's pose (position + orientation). The position of the object corresponds to the cell where the object origin is situated. Since this component must be also discretized, we will impose x as $\{$0.5, 0.55, 0.6$\}$m, y as $\{$-0.1, 0.0, 0.1$\}$m and z as $\{$0.55, 0.6$\}$m, giving 18 possible positions. The orientation of the object will be discretized to obtain 48 different combinations. Table 4.1 identifies all possible orientations of an object. The different object orientations are represented by the rotation angles $\alpha$, $\beta$ and $\gamma$ around gripper's axis X, Y, Z, respectively (according to Z-Y-X Euler notation: see Subsection 5.7.2). Figure 4.2 shows some examples of possible object orientations.

**Table 4.1:** Different orientations of the object, represented by the rotation angles $\alpha$, $\beta$ and $\gamma$ (in degrees).

| Rotation index (i) | $\alpha$ | $\beta$ | $\gamma$ | Reference axis |
|---|---|---|---|---|
| $i = 0, \ldots, 7$ | 0º | 0º | 45º i | z up |
| $i = 8, \ldots, 15$ | 90º | 0º | 45º (i - 8) | y up |
| $i = 16, \ldots, 23$ | 0º | 270º | 45º (i - 16) | x up |
| $i = 24, \ldots, 31$ | 180º | 0º | 45º (i - 24) | z down |
| $i = 32, \ldots, 39$ | 270º | 0º | 45º (i - 32) | y down |
| $i = 40, \ldots 47$ | 0º | 90º | 45º (i - 40) | x down |




**(a)** Object's reference frame; **(b)** $(\alpha, \beta, \gamma) = (0, 0, 21°) \approx (0, 0, 45°)$; **(c)** $(\alpha, \beta, \gamma) = (0, -90°, 70°) \approx (0, 270°, 90°)$;

**Figure 4.2:** Examples of object's orientation.

The second component of the state is a binary condition *is_grasp* and it informs whether the robot is already grasping or not. This component is crucial to differentiate the possible set of actions that the state can take. This statement will be explained in more detail when defining the action space.

If *is_grasped* is true, another component is added to the state (as already denoted before): the gripper orientation, represented as a discretization of 18 possible orientations of the gripper (see Table 4.2). A limitation added to the problem is the condition that the gripper can never be oriented upwards. This prevents the object from "hitting" the gripper as it falls, a behavior that is to be avoided at all costs. The same limitation is performed to avoid the gripper to be oriented backwards, because any joint positions with that set of poses will result in a less comfortable position of the manipulator. A third limitation is that the fingers and the camera should not hit the table in any position. When considering these limitations, it is possible to calculate the number of possible discrete orientations for the gripper (18). In short, the state space is defined by a set of states:

- $Ori = \{0, \ldots, 47\}$: object orientation.
- $Pos = \{0, \ldots, 17\}$: object position on the table grid.
- $is\_grasp = \{False, True\}$: if the grasp is already made.
- $Grip = \{0, \ldots, 17\}$: gripper orientation.

Since the object can not be in $z = 0.6$ without being grasped, this situation would not be considered as a valid state, that is, it does not exist a state where $z = 0.6$ and $is\_grasp = False$. In conclusion, the number of possible states of the problem is **15984**.

### 4.1.2 Action Space

For the grid-world problem, an action corresponds on moving to other cell to the right, left, up or down. For the grasping problem, the set of actions for each state will depend of the condition *is_grasp*. There are two main different sets of actions that are assigned depending of the second component of the state.

If *is_grasp* is false, it means that the object is not already grasped. In that case, the set of actions consists of 18 possible grasps, as represented in Table 4.2. Note that the angles in Table 4.2 are represented in X-Y-Z Euler notation (see Subsection 5.7.1).

**Table 4.2:** Different gripper orientations.

| Rotation index (i) | roll | pitch | yaw |
|---|---|---|---|
| $i = 0$ | $0°$ | $90°$ | $0°$ |
| $i = 1, 2$ | $-90°$ | $45° \cdot (i - 1)$ | $90°$ |
| $i = 3, 4$ | $90°$ | $45° \cdot (i - 3)$ | $-90°$ |
| $i = 5$ | $0°$ | $135°$ | $0°$ |
| $i = 6$ | $-135°$ | $0°$ | $90°$ |
| $i = 7$ | $-135°$ | $45°$ | $125°$ |
| $i = 8$ | $135°$ | $0°$ | $-90°$ |
| $i = 9$ | $135°$ | $45°$ | $-125°$ |
| $i = 10, \ldots, 17$ | $0°$ | $180°$ | $45° \cdot (i - 10)$ |

Like in the case of the object, the gripper's reference frame is also the same as the world reference frame, for the sake of simplicity. When performing one of the grasping actions, it will be simulated by the robot simulator whether or not the grasp was successful. If the grasp is successful, it means that there has been a transition to a state where the *is_grasp* condition is true. If the grasp is not successful (i.e. the object falls or changes its orientation), it means that the resulting state will either be the same or another state with a different pose.

If *is_grasp* is true, then it means that the robot has already performed a grasp successfully, which means that now the goal is to move the grasped object to a goal state that will contain a particular pose that the object must end. The actions now correspond only to changes in the gripper pose and do not involve grasping parameters. Therefore, the action can be to change the rotation of the gripper, change the position of the gripper or drop the object (open the gripper). Again, it is necessary to check after each of these actions if the object continues to be grasped by the gripper, because in the case of the gripper has dropped the object, the parameter *is_grasp* is false again. Translation and rotation action's features correspond to the gripper's goal position and orientation respectively. The number of rotation actions correspond to the number of possible gripper orientations. In total there are **55** possible actions in the defined MDP: 18 of grasping an object ($Grip = \{0, ..., 17\}$, $is\_grasp = False$), 18 of gripper translation ($Pos = \{0, ..., 17\}$, $is\_grasp = True$), 18 of gripper rotation ($Grip = \{0, ..., 17\}$, $is\_grasp = True$) and 1 object drop.

### 4.1.3 State transition model

The state transition model corresponds to the probability that a given state $s$ goes to $s'$ taking action $a$. This model characterizes the environment and therefore it is also part of the MDP. Being a feature of the environment, it must be obtained before the learning process is performed through the algorithm that is proposed in section 4.2.

For the grid-world problem, the transition model gives a 100% chance that the transitioned state is expected (that is, if the action goes to the right, the next state will certainly be the cell to the right of the current one).

The proposed grasping model will only be a binomial distribution, since for each state there is only the possibility of the grasp attempted to be successful or not. However, in case the grasp is not successful, there are several states to which it can go, just that the object has changed its discrete pose as soon as it falls. The transition model is initiated by Bernoulli experiments on the *Gazebo* simulator. For each object, the model is learned through several grasping attempts for several different poses of the gripper. For each experiment, the gripper tries first to grasp the object through a fixed orientation and then raise it to a small fixed position (consider 5 cm), maintaining the orientation of the gripper. If the gripper manages to lift the object without dropping it or changing its orientation, it counts as a successful grasp. Several attempts of rotating or translating the grasped object will also be performed. Since the state space is large, it will be very difficult to perform the Bernoulli experiments to all **15984** states, so it will be created a Kernel function so that many states can be trained to each experiment, based on state's similarity (see Subsection 5.9.1). This process is repeated randomly, and the probability is then given by normalizing the number of times for each state resulting from the action attempt.

Both transition models are stored in sparse matrices due to their large size.

## 4.2 Proposed Learning Algorithm

The proposed algorithm will now be introduced. The aim is to obtain a policy $\pi$ that decides the best action for each situation in order to reach the goal state with maximum reward. A scheme of the proposed learning algorithm is shown in Figure 4.3. Since we are working with a very large state space, a suitable solution to the IRL problem will be to consider a VR function as a function approximation of the summation between the reward function and the discounted state value function such as (3.15). As the VR function is an approximate function, it can be computed from the optimization of the parameters of a neural network using a gradient ascent method, as explained in Subsection 3.7.1.

A Neural Network (NN) is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons. NN are capable of fitting its weights and biases using back

propagation methods, which consists of a parameter update that depends on the resulting output. To perform the gradient ascent method, a back propagation method must always contain a loss function, which we must maximize in order to find the maximum likelihood in this case.

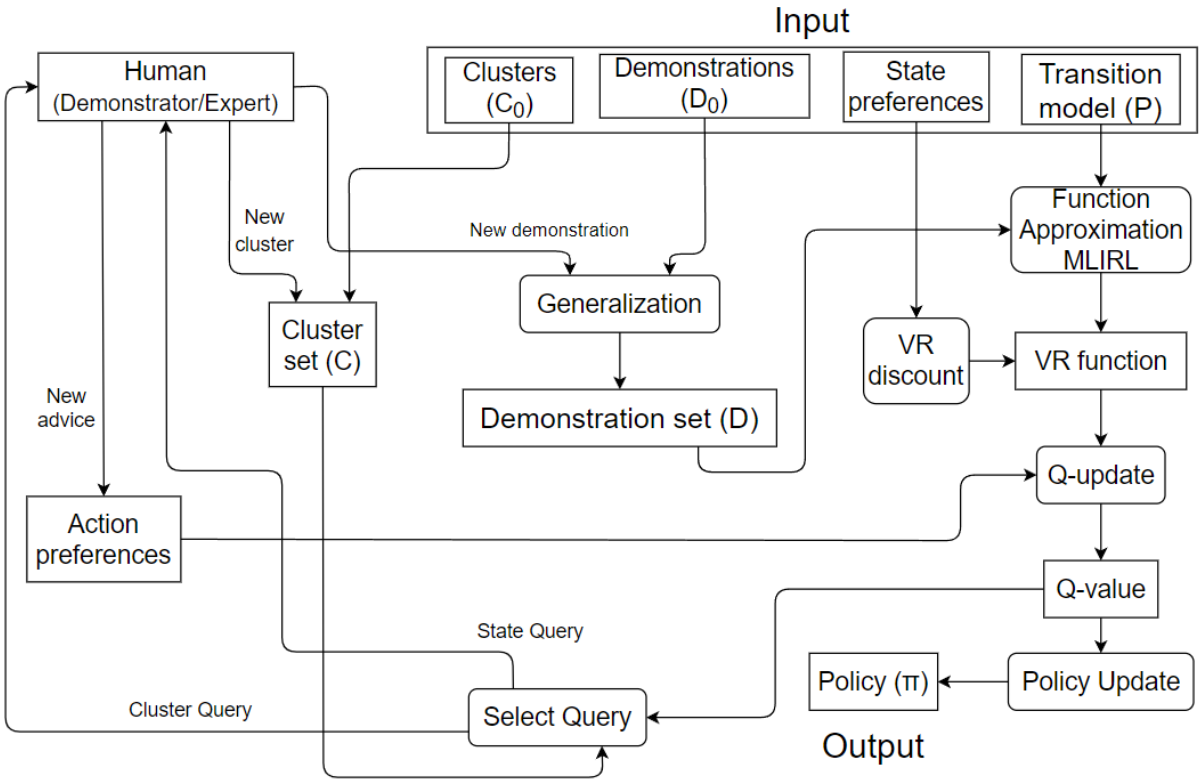



**Figure 4.3:** A framework for the algorithm proposed.

To perform the function approximation method described in Subsection 3.7.1 we will use a NN to optimize a parameter set $\Theta$ in order to get the VR function that maximizes the log likelihood of the demonstrations, as performed in *Burdick & Li (2017)* [3]. However, a set of AcL and AdL approaches will be added to this method, asking the demonstrator for new "optimal" responses and advice to less certain states.

A predefined set of state preferences is given in both problems. For the grid-world problem, it will be preferred to go to the center and the limit states will be avoided. Section 5.12 explains the state preference framework for the grasping problem. Once the final VR function is found, a 20% benefit/discount is given to the VR values of the preferable or avoidable states respectively. After this discount, only the $Q$ matrix is computed using (3.17) (the other learning parameters are not necessary both for obtaining the policy and for query selection).

An *Expert* has the role of labelling what actions should be set to the preferable set of actions and which ones should be set as the avoidable actions for the queried cluster. No action can belong to both subsets. A set of demonstrations $D$ and a set of $clusters$ must also be given to the algorithm. In this framework, a demonstration is represented as state-action pair and should be considered as optimal (explained in more detail in Section 5.10). A set of $clusters$ would be given in the beginning, as in this case it is quite easy to know some states with similar behavior in policy (explained in more detail in Section 5.11). The sets of states where the possible actions will be grasping (i.e. *is_grasp* is false) and that have the same orientation will be grouped for example. At the algorithm it will be also possible by the expert to add new clusters to the cluster set. However, for cases where this clustering process is not easy, *Jain et al. (1988)* [21] can be a good guide for implementing any kind of clustering algorithms. A good clustering algorithm can also be implemented in this framework to obtain even more specific clusters, as it is proved in [2] that a good cluster quality obtains better results. The process of selecting a query is based on the strategy of Pool-based Active Sampling (PbAS), which considers that there is a small set of labeled data and a large set of non-labeled data. Figure 4.4 shows a scheme of the PbAS representation.

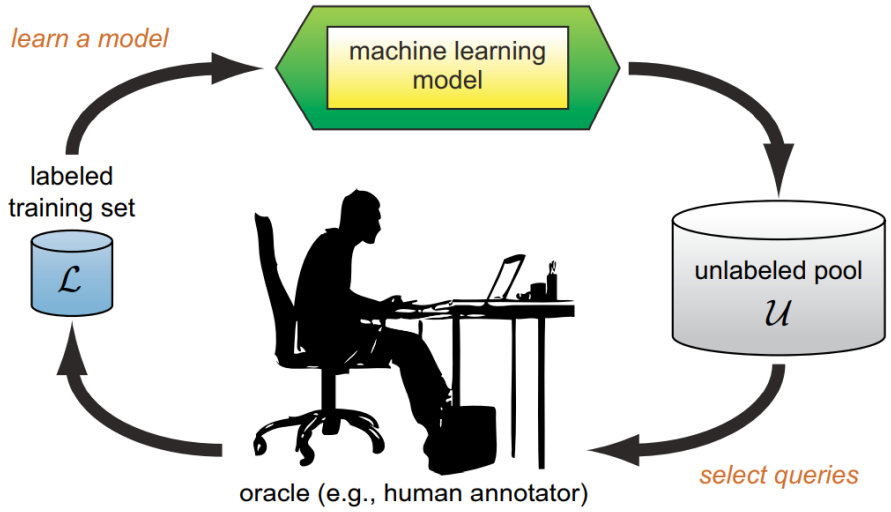



**Figure 4.4:** Pool-based Active Sampling cycle. This image is used in [1].

Among the large state space, two queries are then chosen to be labelled: a state query for demonstration labelling and a cluster query for advice labelling. The less certain state will be labelled by the demonstrator, and later added to the demonstration set after a generalization process based on a Kernel function (explained in Subsection 5.10.1). This choice is made through measures of informativeness. US is the most widely used measure in the literature on a large scale and also the one that will be used on this approach. US is a technique where the agent queries the instance that is least certain

how to label. The uncertainty technique itself involves different strategies, but one of the most used is undoubtedly entropy. The entropy used to obtain the uncertainty is given by the following expression:

$$E(s) = -\sum_{a \in A} P_s(a) \cdot \log P_s(a) \quad . \tag{4.1}$$

In this case, $E(s)$ is the uncertainty w.r.t. policy $\pi$, so the probability $P_s(a)$ will be given by the following expression:

$$P_s(a) = \frac{e^{Q(s,a)}}{\sum_{b \in A} e^{Q(s,b)}} \quad . \tag{4.2}$$

In addition to the most uncertain states, the most uncertain cluster will also be queried in order to complement the information provided with advice given by an expert (also a human being). As in *Odom & Natarajan (2016)* [11], an expert will be responsible for providing advice in form of action preferences for a given cluster. These actions will be held in preferred or avoidable action sets, and a $20\%$ benefit or $100\%$ discount (i.e. the action is excluded of the equation) is given to the action value $Q(s, a)$, when updating the matrix $Q$. Algorithm 4.1 explains in pseudo-code the proposed framework. Note that for the grid-world problem the action preference framework was not used, so only state queries are given.

This framework manages to combine the idea of obtaining a policy in a more accelerated way from a method of function approximation with the idea of providing more information in quality and quantity in an iterative way, thus accelerating the learning process. Furthermore, the state and action preferences do not affect the training of the NN, as it only depends on the demonstrations given for the Loss function. The given advice serves only as a push for some states after obtaining the VR function. In this way, both processes are carried out independently, so that there is no interference between them.

**Algorithm 4.1:** Active FA-MLIRL algorithm

**Function** FA-AL-MLIRL(Budget, Demonstrator, Expert, $clusters$, $state\_pref$, $D_0$, $P$):

$D = D_0$;
$advice = \emptyset$;
create variable $\Theta_0$ as neural network parameter;
build $f(\Theta)$ as the output of the neural network;
initialize $\Theta_0$ arbitrarily;
$Q$, $\Theta$ = FunctionApprox($D$, $P$, $\Theta_0$);

**for** $k = 1$ to Budget **do**
$state\_query$, $cluster\_query$ = SelectQuery($Q$, $clusters$);
$New\_D$ = Demonstrator($state\_query$);
$New\_D$ = Generalization($New\_D$);
$D$.append($New\_D$);
$new\_advice$ = Expert($cluster\_query$);
$advice$.append($new\_advice$);
$Q$, $\Theta$ = FunctionApprox($D$, $P$, $\Theta$);
**end**

$\pi$ = PolicyUpdate($Q$);

return $\pi$;
**end**

**Function** FunctionApprox($D$, $P$, $\Theta$, $state\_pref$, $advice$):

Compute neural network using Equation (3.19) as loss function $L(D|\Theta)$;

**while** $\Theta$ not converging **do**
$\Theta$ = $\Theta$ + $\alpha$ $\nabla_\Theta L(D|\Theta)$;
**end**

$f(\Theta)$ = DiscountState($f(\Theta)$, $state\_pref$);
$Q$ = UpdateQ($f(\Theta)$, $advice$);

return $Q$, $\Theta$;
**end**

**Function** SelectQuery($Q$, $clusters$):

  **for** $s \in S$ **do**
    E(s) = 0;
    **for** $a \in A$ **do**
      $P_s(a)$ = $\frac{e^{Q(s,a)}}{\sum_{b \in A} e^{Q(s,b)}}$;
      $E(s)$ -= $P_s(a) log P_s(a)$;
    **end**
  **end**

  **for** $c \in clusters$ **do**
    $U(c) = \frac{\sum_{s \in c} E(s)}{len(c)}$;
  **end**

  $state\_query = argmax_s \ E$;
  $cluster\_query = argmax_c \ U$;

  return $state\_query$, $cluster\_query$;
**end**

**Function** DiscountState($f(\Theta)$, $state\_pref$):

  **for** $s \in S$ **do**
    **if** $s \in state\_pref.pref$ **then**
      $f(\Theta, s) = f(\Theta, s) + 0.2 \times |f(\Theta, s)|$;
    **end**
    **if** $s \in state\_pref.avd$ **then**
      $f(\Theta, s) = f(\Theta, s) - 0.2 \times |f(\Theta, s)|$;
    **end**
  **end**

  return $f(\Theta)$;
**end**

**Function** UpdateQ($f(\Theta), advice$):

  **for** $s \in S$ **do**
    **for** $a \in A$ **do**
      $Q(s, a) = \sum_{s'} P(s, a, s') \times f(\Theta, s')$;
      **if** $a \in advice.pref(s)$ **then**
        $Q(s, a) = Q(s, a) + 0.2 \times |Q(s, a)|$;
      **end**
      **if** $a \in advice.avd(s)$ **then**
        $Q(s, a) = -2$;
      **end**
    **end**
  **end**

  return $Q$;
**end**

**Function** PolicyUpdate($Q$):

  **for** $s \in S$ **do**
    $\pi(s) = argmax_a \ Q(s)$;
  **end**

  return $\pi$;
**end**

# 5

# Development of grasping problem

## Contents

Section 5 is addressed to explain all the processes regarding the development of the grasping problem proposed in this work. I will start by the description of the processes carried out for the manipulation of the *KINOVA Gen3 Lite* and *KINOVA Gen3* robots and construction of the simulated environment in *Gazebo*. The table has $0.4 \times 0.4 \times 0.5$ meters and the robot is placed on top of a $0.5$ meters high wooden box, such as illustrated in Figure 5.1.

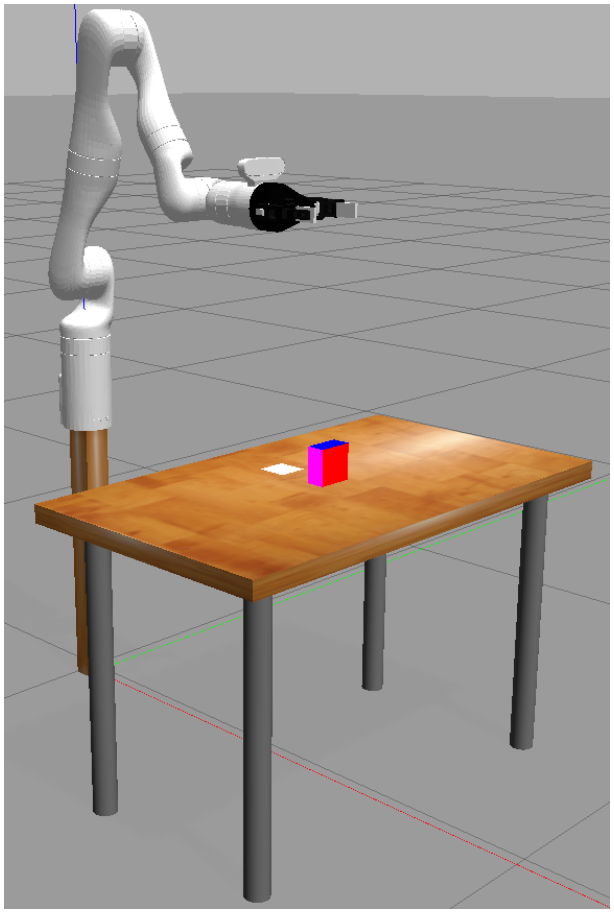


**Figure 5.1:** Gazebo Environment with *KINOVA Gen3*.

All robot simulation are performed using ROS language to communicate with *Gazebo* simulator. The ROS package of both *KINOVA Gen3 Lite* and *KINOVA Gen3* is *KORTEX ROS* official package [1] [22].

Since Robotics is going to be used to build a means to extract information necessary for the implementation of the Algorithm 4.1 (namely data for obtaining the reachability map, the state transition model and the demonstration set), it is recommended that the reader consult the book of John J. Craig [23] to deepen the basics about Robotics, namely the transform's notation and kinematics.

[1] https://github.com/Kinovarobotics/ros_kortex/

## 5.1 *KINOVA Gen3 Lite*

At the beginning of the work, the robot chosen to perform object manipulation was the *KINOVA Gen3 Lite*. This choice was mainly due to the fact that it features a universal gripper that is factory included. Later it was seen that the use of this simple manipulator has more disadvantages than advantages. The main disadvantage of this robot is related to its inverse kinematics, not only because it has only 6 DoF but also because the solver present in the package has recurring errors. As explained in Appendix A, an attempt was made to alleviate this problem but the result was not the best for what is intended in this work. Even so, all the work is possible with the *KINOVA Gen3 Lite*, however the results obtained will always be limited by the developed kinematics. Figure 5.2 shows an illustration of this manipulator. In Appendix A it is also described this robot's specifications, as well as the description of the attempt to improve its inverse kinematics.


**Figure 5.2:** *KINOVA Gen3 Lite* manipulator.

## 5.2 *KINOVA Gen3*

In the middle of the work, it was decided that the best solution would be to use another robot, one that had better inverse kinematics (i.e. more DoF). The chosen robot was the *KINOVA Gen3*, a robot with 7 DoF and that does not have any gripper by default. For the choice of this robot, the fact of the existence of such a model in the laboratories of ISR-Lisboa (Institute For Systems and Robotics in Lisbon) was also considered, which would allow the physical application of the work performed in this thesis. The gripper added to this robot was the *robotiq_2f_85* model. Another advantage of using this manipulator is its integrated camera over its wrist, that could be used to integrate a visual component to this work in order to identify the current state of the environment. Figure 5.3 shows an illustration of the manipulator used in this work. Although both the visual component and the transition to the real robot have not been implemented, the use of this robot allows the implementation of these components for future work.

**Figure 5.3:** *KINOVA Gen* manipulator with *robotiq_2f_85* gripper integrated.

### 5.2.1 Useful Specifications

In order to be able to work with a certain manipulator, it is essential to read its documentation to know as much as possible its qualities and limitations. Specifications such as the size of its components and the limit of the values of the joints lead to a finite number of possible poses of the end-effector (which in this case is its gripper). Figure 5.4 shows a representation of the main components of the robot, including the name and locations of each actuator.

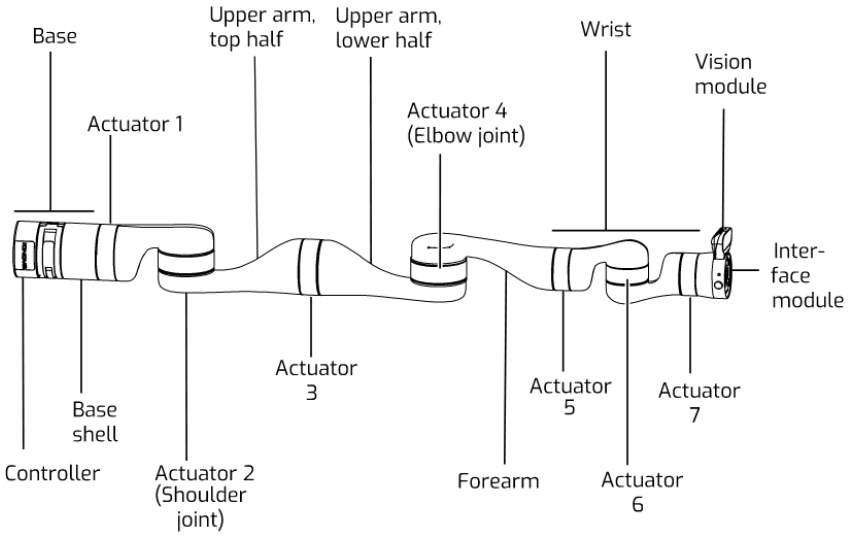



**Figure 5.4:** *KINOVA Gen3* main components.

#### 5.2.1.A Devanit-Hartenberg parameters

A commonly used convention for selecting frames of reference in robotics applications is the Devanit-Hartenberg (DH) convention which was introduced by Jacques Denavit and Richard S. Hartenberg. This convention is used by robotics experts to mathematically describe the size and range of robot's "bones". In this convention, coordinate frames are attached to the joints between two links such that one transformation is associated with the joint, and the second is associated with the link. Table 5.1 presents the DH parameters of *KINOVA Gen3*.

**Table 5.1:** *KINOVA Gen3* DH parameters.

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $a_i$ [m] | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $b_i$ [m] | -0.2848 | -0.0118 | -0.4208 | -0.0128 | -0.3143 | 0 | -0.1674 |
| $\alpha_i$ [°] | 90 | 90 | 90 | 90 | 90 | 90 | 180 |
| Lower Limit [°] | - $\infty$ | -128.9 | - $\infty$ | -147.8 | - $\infty$ | -120.3 | - $\infty$ |
| Upper Limit [°] | $\infty$ | 128.9 | $\infty$ | 147.8 | $\infty$ | 120.3 | $\infty$ |

To complete the DH information, we represent the rotation matrix of each joint as:

$$Q_i = \begin{bmatrix} cos(\theta_i) & -cos(\alpha_i) \cdot sin(\theta_i) & sin(\alpha_i) \cdot sin(\theta_i) \\ sin(\theta_i) & cos(\alpha_i) \cdot cos(\theta_i) & -sin(\alpha_i) \cdot cos(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) \end{bmatrix} \quad , \tag{5.1}$$

and the position vector as:

$$k_i = \begin{bmatrix} a_i \cdot cos(\theta_i) & a_i \cdot sin(\theta_i) & b_i \end{bmatrix}^T \quad . \tag{5.2}$$

Considering **q** as the vector of the values of each joint, $\theta$ is given by the transformation contained in Table 5.2.

**Table 5.2:** Conversion between **q** and $\theta$.

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $\theta_i$ [°] | $q_0$ | $q_1$ + 180 | $q_2$ + 180 | $q_3$ + 180 | $q_4$ + 180 | $q_5$ + 180 | $q_6$ + 180 |

Figure 5.5 outlines the dimensions and frames of each joint/link pair of the manipulator. To see more useful specifications of this manipulator, see its documentation [2] [24].

[2] https://www.kinovarobotics.com/sites/default/files/UG-014_KINOVA_Gen3_Ultra_lightweight_robot_User_guide_EN_R06_0.pdf

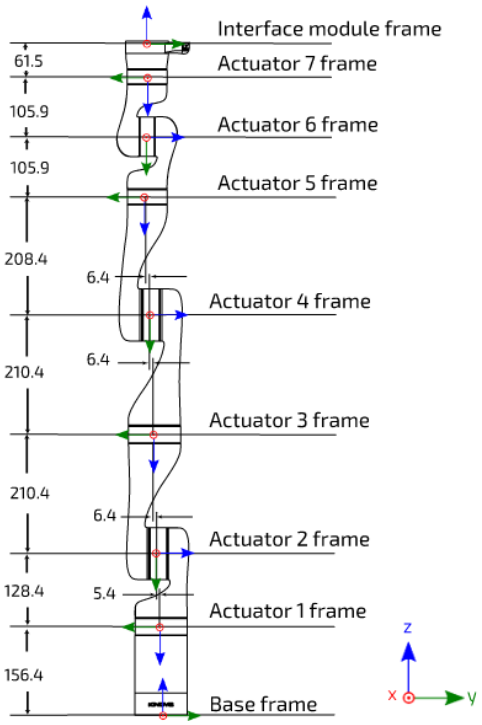



**Figure 5.5:** *KINOVA Gen3* link dimensions and joint frames.

## 5.3 *Kortex* ROS package overview

In this subsection, the components and functionalities provided by the *Kortex* ROS package will be summarized.

- *Kortex_control* contains the configuration files for the *ros_control* controllers used to control the simulated arms;
- *Kortex_description* contains the Unified Robot Description Format (URDF), Standard Triangle Language (STL) and configuration files for the Kortex-compatible robots;
- *Kortex_driver* allows communication between a *ROS* node and the robot;
- *Kortex_gazebo* contains files to simulate the *KINOVA Gen3* and *KINOVA Gen3 Lite* robots in *Gazebo*;
- *Kortex_move_it_config* contains all the auto-generated *MoveIt!* configuration ROS packages. These packages have been generated using the *MoveIt! Setup Assistant*. This component was not used in this thesis;

- *Kortex_examples* contains several examples of some functionalities of the robot, including actuator configuration, arm movement, vision module configuration and some *MoveIt!* examples.

## 5.4 Kinematics

Before moving on to building the MDP proposal, it is strictly necessary to implement the kinematics of the robot, so we can move it to whenever feasible pose of the environment.

### 5.4.1 Forward Kinematics

The forward kinematic problem, i.e. the cartesian position **p** and orientation matrix of the tool **Q**, are straight forward and can be written as:

$$\mathbf{p} = \sum_{i=0}^{n-1} \left( \left( \prod_{j=0}^{i} Q_j \right) + k_i \right) \quad , \tag{5.3}$$

$$\mathbf{Q} = \prod_{i=1}^{n} Q_i \quad , \tag{5.4}$$

where $Q_0$ is the 3x3 identity matrix and n is the number of joints. Given a vector **q**, we can easily compute **p** and **Q** using (5.1), (5.2), (5.3) and (5.4). In this work, the Forward Kinematics (FK) problem is solved recurring to the *'base_feedback'* robot message, which give the actual end-effector's pose directly.

### 5.4.2 Inverse Kinematics

The Inverse Kinematics (IK) problem is much more complex and difficult to compute comparing to the forward kinematic problem, since the problem of solving the kinematic equations is a nonlinear one. Given the desired tool pose, the goal is to obtain the set of joint angles which will achieve that pose. As with any nonlinear set of equations, we must concern ourselves with the existence of multiple solutions or even the possibility of any feasible solution taking into account the limits of the joints. Even if there is a solution, it may require complex and expensive computations to find it. The question of whether any solution exists at all raises the question of the manipulator's workspace.

In kinematics, workspace is the volume of space that the end-effector of the manipulator can reach. For this work, where we want to take a given pose (i.e. position+orientation), we need to consider an even more restricted space around the manipulator where the end-effector can reach with a desired range of orientations, since it is not enough to reach a certain position. So, the workspace in this work has to be all space where the manipulator can reach with at least one orientation contained in the set of

possible action orientations defined in the MDP. A reachability map will be built in order to identify the workspace of the problem, as it will be explained in Section 5.5.

To compute the IK, the ROS *Kortex* package provides a function to calculate it directly using a IK solver based on gradient descent. Since this solver uses a gradient descent-based method, it has to take into account the current pose of the end-effector, which makes this solver a bad approach in cases where the end-effector is still a long way from the desired pose and the manipulator has few DoF. This fact leads us to many cases where this robot gives errors because it cannot find a solution, despite its existence, such as mentioned before. In *KINOVA Gen3 Lite*, this consequence was quite visible, so a new approach was developed (see Appendix A for details). However, for *KINOVA Gen3* this effect was not visible. Since more solutions are provided for this manipulator, it is enough to use this provided robot solver to compute IK for this manipulator and for the considered workspace.

## 5.5 Reachability map

In order to compute the problem's workspace, a reachability map was built. This reachability map consists on a large table where for each existing gripper pose in the action space, the IK was previously performed. This table will tell us if the desired pose in a respective action is possible to be performed by the manipulator, thus defining the problem's workspace. In other words, the reachability map resulted from a pre-computation of all the poses defined in the action space as possible poses of the manipulator.

For *KINOVA Gen3*, the process consists of performing IK for each pose using the provided robot IK gradient-based solver. If the pose of the end-effector corresponds approximately to the desired pose, then its result is 1, otherwise is 0.

Every time an action is performed, it has to be confirmed by the reachability map if the manipulator's goal pose is feasible, if not, the action is invalid. If the goal pose does not exist on the reachability map, a single experiment of reaching it will be done and noted in the reachability map.

Since the notation of the gripper is X-Y-Z Euler (see Section 5.6), the rotation in $Z$ is the last to be performed. Since the $Z$-axis always represents the direction of the gripper, a rotation in relation to this axis simply consists of the rotation of the last joint (wrist) of the robotic arm. As this joint presents complete freedom of movement for *KINOVA Gen3* (or it did not have the limits at $]-\infty, \infty[$), the rotation in relation to the $Z$-axis of the gripper no longer has any effect on the gripper's feasibility to reach a certain pose. In this way, it is possible to consider that, regardless of the $Z$-rotation performed in the experiment, the result is the same for all possible values in this variable, which means that more than one pose is trained at once. However, this consideration can only be made for poses where the height value ($Z$ coordinate) is greater than $0.55$m, because for heights at $0.55$m (or less) the gripper can hit the table depending on the rotation around the $Z$-axis.

Naturally the reachability map is unique for each manipulator, so the reachability map obtained for *KINOVA Gen3 Lite* cannot be used for *KINOVA Gen3* and vice versa.

## 5.6 Orientation notation

A special care that was necessary in this work and in Robotics in general is the notation that each module has in terms of orientation. As we are working in a 3-dimensional space, there are several possible notations to represent rotation. For more details on the angle conversions used in this thesis, see Section 5.7. Table 5.3 shows the orientation notation used for each work module.

**Table 5.3:** Orientation notations for each module.

| Module/Notation | X-Y-Z Euler | Z-Y-X Euler | Quaternion |
|---|---|---|---|
| FK robot solver | | X | |
| IK robot solver | | X | |
| Object | | X | |
| Gripper | X | | |
| Reachability map | X | | |
| Object State in *Gazebo* | | | X |

Note that the robot solvers provided by the *ROS Kortex* package have an offset of 90º in z-axis, so it is necessary to take this into account and take the necessary precautions.

## 5.7 Angle conversion

As the MDP construction involves a lot of Robotics, it is necessary to be especially careful with the notation of the angles used, as previously mentioned. There are several different notations to represent orientations in 3 dimensions, with some of the best known being the Euler, Fixed-angle, Quaternion or Tait-Bryan notations. Each notation can have several conventions, for example, Euler's notation can have X-Y-Z or Z-Y-X convention.

In this work, the notations used were X-Y-Z Euler, Z-Y-X Euler and Quaternion. In order to interrelate all the angles of the problem, it was necessary to perform several conversions. Next, all the angle conversions used in this work will be described, including the transformations of each notation to rotation matrix form and vice versa.

### 5.7.1 X-Y-Z Euler transformations

The Euler notations consist of rotating an object around its respective axis in a certain order. In other words, an Euler rotation consists of the product of 3 rotation matrices: each in relation to the $X$, $Y$ or $Z$ axis. The Euler convention defines the order of each rotation. For the X-Y-Z Euler case, we should

rotate the object first around the $X$-axis, then a rotation around $Y$-axis and finally a rotation around $Z$-axis. However, a disadvantage of Euler notations is that Euler angles suffer from aliasing, that is, the same rotation can be represented with multiple Euler angle combinations. To work around this problem, a conversion is made for any Euler angle to rotation matrix, and then again reconverted to the pretended Euler notation. In this way, a certain orientation will contain only one representation in Euler's notation, since even if there is another representation that corresponds to the same orientation, it will be converted to the only representation. Figure 5.6 shows an illustration of this rotation convention.

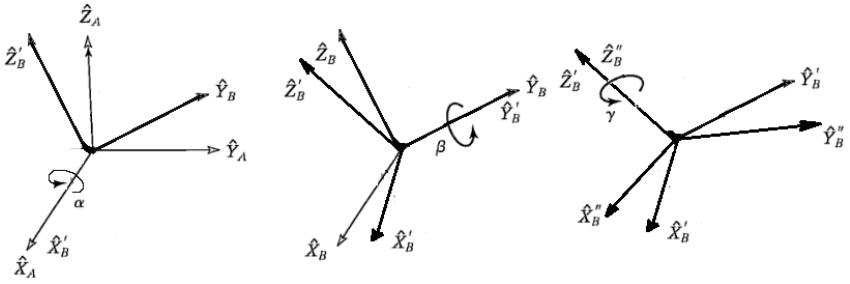



**Figure 5.6:** X-Y-Z Euler notation.

The conversion of this notation to rotation matrix is computed as follows:

$$R = R_X(\alpha) \cdot R_Y(\beta) \cdot R_Z(\gamma) = \begin{bmatrix} c\beta \cdot c\gamma & -c\beta \cdot s\gamma & s\beta \\ c\alpha \cdot s\gamma + s\alpha \cdot s\beta \cdot c\gamma & c\alpha \cdot c\gamma - s\alpha \cdot s\beta \cdot s\gamma & -s\alpha \cdot c\beta \\ s\alpha \cdot s\gamma - c\alpha \cdot s\beta \cdot c\gamma & s\alpha \cdot c\gamma + c\alpha \cdot s\beta \cdot s\gamma & c\alpha \cdot c\beta \end{bmatrix} \quad , \qquad (5.5)$$

where $c\alpha = \cos\alpha$ and $s\alpha = \sin\alpha$. The following pseudocode represents the conversion to X-Y-Z Euler notation from a rotation matrix.

```
if r13 < 1:
    if r13 > -1:
        alpha = arctan2(-r23,r33)
        beta = arcsin(r13)
        gamma = arctan2(-r12,r11)
    else:
        alpha = - arctan2(r21,r22)
        beta = - pi/2
        gamma = 0
else:
    alpha = arctan2(r21,r22)
    beta = pi/2
    gamma = 0
```

, where $arctan2(y,x)$ is given by:

$$arctan2(y,x) = \begin{cases} arctan\left(\frac{y}{x}\right), & \text{if } x > 0, \\ arctan\left(\frac{y}{x}\right) + \pi, & \text{if } x < 0 \text{ and } y \geq 0, \\ arctan\left(\frac{y}{x}\right) - \pi, & \text{if } x < 0 \text{ and } y < 0, \\ \frac{\pi}{2}, & \text{if } x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2}, & \text{if } x = 0 \text{ and } y < 0, \\ N/A, & \text{if } x = 0 \text{ and } y = 0. \end{cases} \quad . \tag{5.6}$$

### 5.7.2 Z-Y-X Euler transformations

For the Z-Y-X Euler case, we should rotate the object first around the $Z$-axis, then a rotation around $Y$-axis and finally a rotation around $X$-axis. Figure 5.7 shows an illustration of this rotation convention.

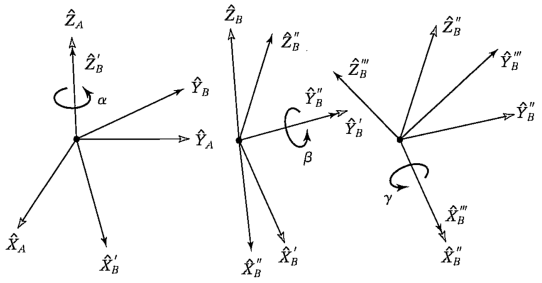



**Figure 5.7:** Z-Y-X Euler notation.

The conversion of this notation to rotation matrix is computed as follows:

$$R = R_Z(\alpha) \cdot R_Y(\beta) \cdot R_X(\gamma) = \begin{bmatrix} c\alpha \cdot c\beta & c\alpha \cdot s\beta \cdot s\gamma - s\alpha \cdot c\gamma & c\alpha \cdot s\beta \cdot c\gamma + s\alpha \cdot s\gamma \\ s\alpha \cdot c\beta & s\alpha \cdot s\beta \cdot s\gamma + c\alpha \cdot c\gamma & s\alpha \cdot s\beta \cdot c\gamma - c\alpha \cdot s\gamma \\ -s\beta & c\beta \cdot s\gamma & c\beta \cdot c\gamma \end{bmatrix} \quad . \tag{5.7}$$

The following pseudocode represents the conversion to Z-Y-X Euler notation from a rotation matrix.

```
if r31 < 1:
    if r31 > -1:
        gamma = arctan2(r32,r33)
        beta = arcsin(-r31)
        alpha = arctan2(r21,r11)
    else:
        gamma = 0
        beta = pi/2
        alpha = - arctan2(-r23,r22)
else:
```

```
11      gamma = 0
12      beta = - pi/2
13      alpha = arctan2(-r23,r22)
```

### 5.7.3 Quaternion transformations

The Quaternion notation is a compact, efficient, and numerically stable way of representing a 3D rotation. Compared to Euler angles, they are simpler to compose and avoid the problem of gimbal lock [3]. However, they are not as intuitive and easy to understand as Euler angles. Due to the periodic nature of sine and cosine, rotation angles differing precisely by the natural period will be encoded into identical quaternions and recovered angles will be limited to $[0, 2\pi]$.

A Quaternion angle consists on a rotation of $\theta$ radians around a fixed point in space $(x, y, z)$. A spatial rotation around a fixed point of $\theta$ radians about a unit axis $(X, Y, Z)$ that denotes the Euler axis is given by the quaternion $(c, X \times s, Y \times s, Z \times s)$, where $c = \cos(\theta/2)$ and $s = \sin(\theta/2)$. Figure 5.8 shows an illustration of this rotation convention.

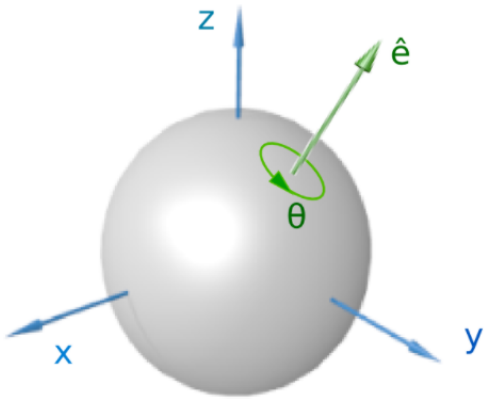



**Figure 5.8:** Quaternion notation.

The conversion of this notation to rotation matrix is computed as follows:

$$R = \begin{bmatrix} w^2 + x^2 - y^2 - z^2 & 2(x \cdot y - w \cdot z) & 2(w \cdot y + x \cdot z) \\ 2(x \cdot y + w \cdot z) & w^2 - x^2 + y^2 - z^2 & 2(y \cdot z - w \cdot x) \\ 2(x \cdot z - w \cdot y) & 2(w \cdot x + y \cdot z) & w^2 - x^2 - y^2 + z^2 \end{bmatrix} \quad , \tag{5.8}$$

where $[w, x, y, z]$ are given by $[c, X \times s, Y \times s, Z \times s]$. The following pseudocode represents the conversion to Quaternion from a rotation matrix.

[3] Gimbal lock (GL) is the loss of a degree of freedom in a three-dimensional mechanism that occurs when the axes of two of the gimbals (SC) are brought into a parallel configuration, "locking" the system so that it can only rotate in one two-dimensional space. Source: https://en.wikipedia.org/wiki/Gimbal_lock

```
tr = r11 + r22 + r33

    if tr > 0.0:
        S = (tr + 1.0)**0.5 * 2
        w = 0.25*S
        x = (r32 - r23) / S
        y = (r13 - r31) / S
        z = (r21 - r12) / S
    else if r11 > r22 and r11 > r33:
        S = (1.0 + r11 - r22 - r33)**0.5 * 2
        w = (r32 - r23) / S
        x = 0.25*S
        y = (r21 + r12) / S
        z = (r13 + r31) / S
    else if r22 > r33:
        S = (1.0 - r11 + r22 - r33)**0.5 * 2
        w = (r13 - r31) / S
        x = (r12 + r21) / S
        y = 0.25*S
        z = (r23 + r32) / S
    else:
        S = (1.0 - r11 - r22 + r33)**0.5 * 2
        w = (r21 - r12) / S
        x = (r13 + r31) / S
        y = (r23 + r32) / S
        z = 0.25*S
```

## 5.8 MDP development

To implement an algorithm based on reinforcement learning, it is necessary to define the MDP of the problem, as previously mentioned in Section 4.1. For this problem, the MDP contains a set of states, a set of actions, a reachability map, a state transition model and a demonstration set.

The state and action spaces are stored in *.csv* files and loaded at table and hash table format. The state/action table is used to directly access the discretized state/action through its index, which will be beneficial for the translation of the transition model. The state/action hash table allows to search more quickly and efficiently for a discrete state/action's index.

The same data structure is used for the reachability map storage, where the table will be used to

store new experiments that are made during the simulation in order to be written in their respective file at the end of the simulation. The hash table will be used to search for experiments already done and to obtain its result in a faster way than using linear search.

The construction of the reachability map has already been mentioned in Section 5.5 and it is important to define the possible poses that the gripper can have throughout the problem at the moment. The sets of states and actions are already defined in Section 4.1, however how can the robot understand what state the environment is in? And how does the robot check whether a given action is possible or not?

### 5.8.1 State identification

At any moment the agent must identify which state the environment is in, in order to be able to act in a certain way depending on that same state. In this framework, the agent identifies the current pose of the object from the reading of an existing topic in the *Gazebo* simulator ('$/gazebo/model\_states$'), which accurately returns the pose of all *Gazebo* models with Quaternion notation. The identification of the state is then done as follows: the pose is obtained in real time by that topic (and then converted to Z-Y-X Euler notation) and is being grasped if the position of the end-effector is the same as that of the object. The end-effector pose of the gripper is also obtained by subscribing a *ROS Kortex* topic ('$robot\_name/base\_feedback$') in Z-Y-X Euler notation (with an offset of 90º in z-axis due to incorrect specification of the simulator, as explained in Section 5.2) and then converting to X-Y-Z Euler notation.

For this work, subscribing the *Gazebo* topic is sufficient (and presents better results) than implementing a classifier based on image analysis for real robots. The transition model is a characteristic of the environment and that is why it is important that the values used for this training are the least error-prone possible. Bearing this in mind and due to lack of time for implementation, the object classifier has not been developed and will be left for future work.

### 5.8.2 Action's description and verification

Although there is already a set of actions defined in the MDP, not all actions are possible to be performed: the manipulator may try to grasp on a certain pose that is not highlighted in the reachability map, or the rotation of the object to a certain orientation may not be possible due to limitations in the movement of the gripper. For these and other reasons, it is necessary to carry out a verification of the possibility of performing an action before it is performed, in order to avoid errors in the robot's IK. If the action's pose does not exist in the reachability map, then a new IK experiment will be performed before conducting the action and its result will be added in the reachability map's data structures.

As mentioned in Subsection 4.1.2, the actions can be of 4 different types: grasp the object (if it is not

already grasped), rotate, translate and drop (if the object is already grasped). For the grasping action, a check of the gripper's end-effector pose on the reachability map is enough for the verification. For the translating and rotating actions it is also only necessary to check the gripper's end-effector goal pose, as the only change is in the position and orientation respectively. For the dropping action, no validation is necessary, since the only action to do is to open the gripper. Of course, for any action verification, a discretization of the gripper's pose is performed, in order to be able to compare with the reachability map's parameters.

If the action is valid, it is then discretized so that it corresponds to one of the possible actions defined in the problem's MDP.

## 5.9 Transition model training

As the agent is already able to identify states and actions and to perform all the defined types of action, all the tools are provided to start the process of training the problem's state transition model.

The transition model consists of a dictionary with all percentages different to 0 where the state-action-state trio is used as key. This representation was found to alleviate the problem of the real size of the transition model. The real transition model would consist of a matrix of 3 dimensions (state-action-state) of size $15984 \times 55 \times 15984$, which corresponds to $1.41 \times 10^{10}$ elements. A matrix with so many elements is too big to be represented in its dense form. However, as most elements have a value of 0 and many actions are not possible for certain states (for example, grasp an already grasped object), it can be represented as a sparse matrix, which consists of representing only the matrix's values different from 0. Two auxiliary matrices (one with the counts of each initial state-action pair and other with the set of states with probability greater than zero for each state-action pair) are created in the same way in order to assist the calculation of the probabilities in the transition model. All dictionaries are stored in *.json* files.

Every time a training process starts, all *.json* files are read and their respective dictionaries are loaded in the respective data structures. With the actual model stored, the Bernoulli experiments can start. The state is identified and discretized to obtain its index and the action is given, verified its feasibility and then discretized to obtain the respective index. After performing the given action, the identification of the new state is performed in order to obtain the new state index. Obtained the respective trio state-action-state, we only need to update the transition matrix by incrementing the number of occurrences of that state-action-state trio and then update the probability using the auxiliary matrices, such as explained in Subsection 4.1.3.

This process corresponds to a single Bernoulli experiment, and more experiments just like this will be performed but with different state-action pairs. Each Bernoulli experiment will also count other state-

action-state trios according to the similarity with the trio obtained. For example, if the robot tries to grasp an object with a certain pose and it is successful, the same behavior is expected if the object and the gripper are slightly deviated with the same pose. As already described in Subsection 4.1.3, a kernel function was developed in order to train a set of cases by amounting those trios with similar behavior. Although this process can make the transition model represent cases that are not the reality, that is necessary in order to accelerate the process of training the transition model, since it is unfeasible to train the model for all possible situations. When the training process ends, the updated model and its auxiliary matrices are saved when rewriting the respective *.json* files.

Despite all these methods of accelerating the learning of the transition model, it was not possible to train it for all states due to lack of time. In order to complete the state transition model, a function that estimates the next state $s'$ by performing a state-action pair $(s, a)$ was developed. Although this estimate may not correspond to the actual behavior of the environment, it is necessary to complete the transition model for all states. Possible less correct results should take into account the fact that the transition model is not complete, as it may impact the final experiments. Another possible solution to this issue can be the development of a model-free version of the proposed algorithm. This idea will be left for future work.

### 5.9.1 Kernel function criteria

To accelerate the training process, it is extremely necessary to build a kernel function in order to be able to train several state-action-state trios for each Bernoulli experiment. In the case of the transition model, the kernel function consists of assigning certain state-action-state trios as being similar to the goal trio. This similarity in relation to the state-action-state sequence performed is determined from a set of matching rules between features. The clustering criteria for the transition model are as follows:

- For any successful action, the same result is expected regardless of the initial object's position in x and y, maintaining the remaining state components.
- The same result is considered regardless of the orientation performed on the fixed axes x and y with orientation of 90°, 180° or 270° around the fixed Z-axis in relation to the orientation of the initial state. For example, a grasp made on an object with an orientation [0°,0°,0°] has similar behavior compared to a grasp made with object orientation [0°, 0°, 90°], [90°, 0°, 0°], [0°, 90°, 180°], etc.
- After the cluster formation, if the action performed is a translation and the goal height pose is higher than 0.55 meters (sensibly 5 cm above the table), then the same result is expected in trios where the initial state and the final state only differs in gripper orientation from any of the states amounted by the kernel function.

Of course only the trios with feasible action poses (according to the reachability map) will be considered. If the goal action's pose does not exist in the reachability map, no cluster is formed and only the goal trio is considered.

## 5.10 Demonstration set

As previously mentioned in Section 4.2, the demonstration set in this context consists of a set of state-action pairs with optimal behavior. Each state-action pair is called a demonstration, and all demonstrations are stored on a *.csv* file. For this work, the construction of the demonstrations for the first demonstration set performs the following steps:

- The object is placed somewhere on the table, with a certain pose. An attempt is then made to grasp that object. The choice of which action to take is made by the demonstrator which will choose the considered optimal grasp action for that situation;
- If the object was successfully grasped, the next action to be taken will be rotation, translation or dropping the object. This sequence of events is done until the object is no longer grasped;
- If the object was not grasped, a new grasp action will be experimented;
- Repeat the previous steps until the object is in the goal state (non grasped object with pose [0.5m, 0m, 0.55m, 0°, 0°, 0°]). After reaching the goal state, a new object will be randomly placed and all the process is repeated successively.

Considering the large size of the state space, it is unaffordable to perform an initial demonstration set large enough to cover a large part of the state space. Then, a generalization of the demonstrations created based on the similarity between states will be performed, following the idea of kernel criteria used to train the transition model. With generalization, it is possible to obtain a substantially larger number of trajectories, speeding up the process of building the demonstration set. When querying the more uncertain states to the demonstrator, it responds with the action it thinks is optimal for that state and the resulting trajectory is again submitted to a generalization.

### 5.10.1 Generalization of trajectories

The current demonstration set consists of a sequence of state-action pairs, where, for a given situation, the demonstrator (where it is assumed that it owns the optimal policy) tells the agent the best action to take. The problem in this context is the scarcity of demonstrations that will exist, which will never be enough for a good performance of the proposed algorithm.

To reduce this problem, a generalization of the demonstrations obtained from the access to the environment is performed. This generalization is performed in a similar way to the transition model training acceleration process. For each demonstration obtained, more demonstrations are created that show the same behavior for similar events. Thus, let us consider the generalization criteria:

- For states where the object is not being grasped, the same grasping action applies wherever the object is, as long as the object's orientation remains, except when the gripper points left at $y = -0.1$m or points right at $y = 0.1$m (since the gripper is less comfortable in those poses).
- If on a given trajectory the action is to translate the object to $x = 0.5$m and $y = 0.0$m (target position) or to drop it, the same action is expected for any state that maintains all state components of the trajectory except the x and y positions, as long as it does not present one of the following less comfortable poses of the gripper:
  - The gripper points left at $y = -0.1$m;
  - The gripper points right at $y = 0.1$m.
- The same happens for any rotation, as long as both the initial state and the predicted final state meet the requirements presented here.

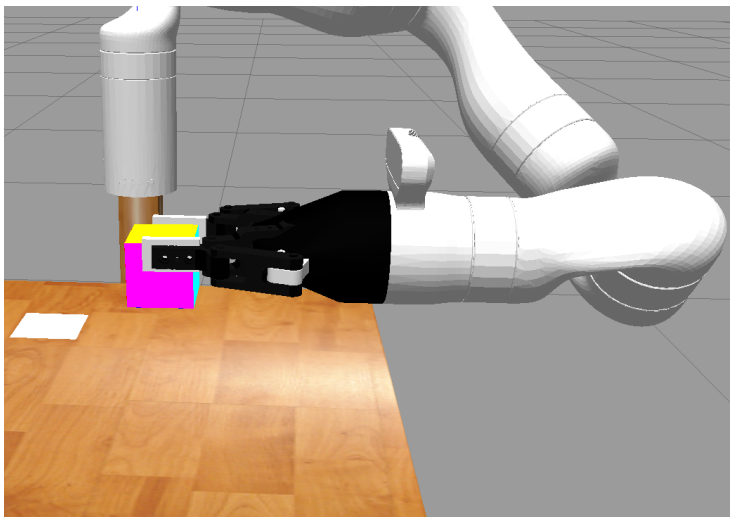


**Figure 5.9:** Example of a less comfortable gripper pose (gripper pointing right at $y = 0.1$m).

## 5.11 Action preferences: state clustering

In order for the expert to be able to advise certain actions for a set of states instead of an individual state, it is necessary to perform a pre-grouping of all the problem states in clusters according to their similarity. In this way, learning the decision model will be performed much more quickly in relation to advice, and will maintain performance if this grouping is perfect. It is very unlikely that the grouping of states will be perfect (i.e., the consequence would be exactly the same if a certain action is taken for two similar states), so it is necessary to find a balance. Generally, the more states that contain a cluster, the smaller the similarity between all of them, which will effectively lose training quality. However, it is also intended to create clusters with the largest possible number of states, as this would be synonymous with faster training. Thus, the objective will be to create a representation of clusters with the largest possible number of states, but without losing the similarity between them.

Taking this into account, several clusters were created in order to amount the various states according to a certain feature. As explained in Algorithm 4.1, the Active Learning method will give us a feature to be advised by the expert: the cluster with the greatest uncertainty. The expert is then given the decision to choose whether to give advice to the cluster or to create a new cluster according to the features that it is wanted to cover for the advice. If the choice of the expert falls on the creation of a new cluster, then the set of all available clusters will be given by the agent. The expert will thus be able to choose which clusters he wants to cover in the advice process, or even if he wants to obtain an intersection of several of these clusters. Some of the features considered to create several clusters are:

- Division by the *is_grasp* parameter;
- Division by object's position;
- Division by object's orientation;
- Division by gripper's orientation;

Table 5.4 shows the description of all 20 primordial clusters of the problem. With the cluster query chosen, the next step is to assign advice to the agent. The agent will show on the command line all the actions contained in the MDP and will first ask the expert to select which actions are preferable to assign to the query and then which actions should be avoided. These new advice will be added to the states belonging to the chosen query.

**Table 5.4:** Description of each cluster.

| Feature | Description |
|---|---|
| Division by *is_grasp* | 0 |
| | 1 |
| Division by gripper orientation | Gripper pointing down: $R[2][2] < -0.9$ |
| | Gripper pointing front: $R[0][2] > 0.9$ |
| | Gripper pointing right: $R[1][2] < -0.9$ |
| | Gripper pointing left: $R[1][2] > 0.9$ |
| Division by object orientation | $x = 0°$ or $x = 180°$, $y = 0°$ |
| | $x = 0°$, $y = 90°$ or $y = 270°$ |
| | $x = 90°$ or $x = 270°$, $y = 0°$ |
| | $z = 0°$ or $z = 180°$ |
| | $z = 45°$ or $z = 225°$ |
| | $z = 90°$ or $z = 270°$ |
| | $z = 135°$ or $z = 315°$ |
| Division by table zones | back limit: $x = 0.5$m |
| | front limit: $x = 0.6$m |
| | right limit: $y = -0.1$m |
| | left limit: $y = 0.1$m |
| | under limit: $z = 0.55$m |
| | upper limit: $z = 0.6$m |
| | center: $x = 0.55$m and $y = 0$m |

## 5.12 State preferences

*Kunapuli et al. (2013)* [9] does not only present the type of advice based on preferred actions or to be avoided for each state. In this work another type of advice is also introduced: state preferences. This type of advice consists of assigning states that should be benefited in the training process or that should be avoided. The choice of preferred and avoidable states is a form of information that can be used as a complement to demonstrations and action preferences. This form of advice will be known at the beginning of the algorithm and will not undergo any querying process.

A certain state will belong to the preferred group if it meets the following requirements:

- Object orientation is $(0°, 0°, 0°)$;
- Object position is ($0.5$m, $0.0$m, z).

Likewise, a state will belong to the group to be avoided if it meets the following requirements:

- There is no way to transition the state to another where the object's orientation is $(0°, 0°, 0°)$;
- Object at $y = 0.1$m and gripper pointing to the right;
- Object at $y = -0.1$m and gripper pointing to the left;

# 6

# Results

## Contents

Section 6 is addressed to show and explain the experiments performed in this thesis. It will be also addressed some expected results from the conclusion of other works. A problem about LfD is related to two assumptions that rarely are achieved in large state spaces: the presence of enough demonstrations to learn a good policy and the guarantee that these demonstrations given by the user really demonstrate a good behavior.

## 6.1 Expected results

What is intended with the thesis proposal is to bring the solution closer to both assumptions: maximize the features to be advised in order to be more informative and reduce the impact that a user error will have on the learning process. With the proposed algorithm, it is expected that the query selection process will allow to obtain the same quality of the policy with less necessary data coming from the user, which makes the policy to definitely converge faster to an optimal solution. In fact, this reasoning is proven and demonstrated in several works like *Lopes et al. (2009)* [6] and *Odom and Natarajan (2016)* [2].

*Lopes et al. (2009)* goes even further in this conclusion and finds in its results that the effectiveness of Active Learning (AcL) in the scenario of Bayesian Inverse Reinforcement Learning (BIRL) can depend a lot on prior knowledge about policy or reward function. To approach the second assumption, the concept of Advice Learning (AdL) was introduced in this area, since advice allows fewer errors to be taken by the user. It is natural to assume that it is easier for a human being to declare which actions will be more preferable or avoidable than to claim which is the best action. In fact, the literature about Learning from Demonstration (LfD) shows that the IRL approach without advice does not deal well with noisy trajectories, as it is generally considered the assumption of considering the demonstrator as optimal. *Kunapuli et al. (2013)* [9] tries precisely that experiment: if we consider noisy trajectories in the demonstrations, adding advice makes the results not decrease so much, while adding noise to advice (although less likely in a real case) does not affect the results so much compared to noise-free advice.

*Odom and Natarajan (2016)* [2] performed a series of experiments with different simulated discrete worlds and in all of them they obtained better results when learning using advice instead of exclusively demonstrations and even better results if the chosen queries are clusters instead of single states. In Figure 6.1 an example of a result of this work is presented, in the stochastic domain of *Sailing*. Those results prove that 6 iterations of their algorithm were enough for the goal reached to be approximately equal to 100%.

*Odom and Natarajan (2016)* [2] also state that the quality of clustering can be a great source of improvement in the results obtained. If the separation of the states becomes very well defined and divided, the preferred and avoidable actions are even more representative of the entire cluster, which will

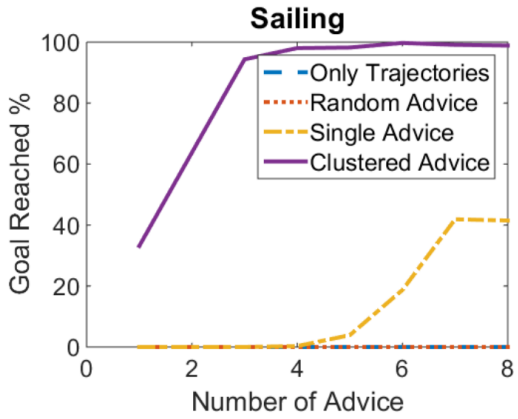



**Figure 6.1:** Results for varying the amount of advice on Sailing domain [2]. Using thoughtful advice effectively leads to faster learning. Clustered advice allow us to retain more information at once, so performance improvements will be seen even sooner. Performance is measured in domain-specific performance measures.

for sure bring more accurate information to the IRL agent. Figure 6.2 shows the impact that the quality of clustering can have in the results. The closer the clustering algorithm has to the ideal separation, the better is the ability to assign the most representative actions of each cluster.

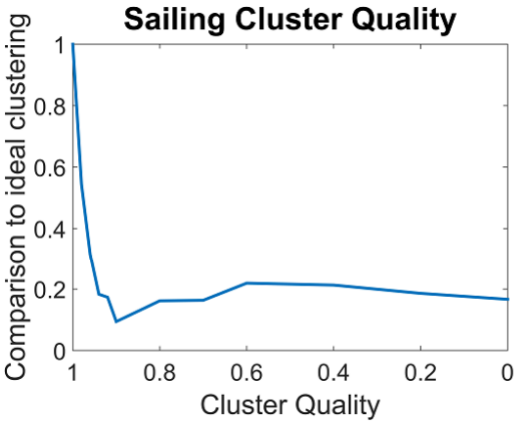



**Figure 6.2:** Results for varying the quality of the clustering on Sailing domain. In [2], the IRL algorithm was tested with different levels of clustering quality. Performance is measured relative to the ideal clustering.

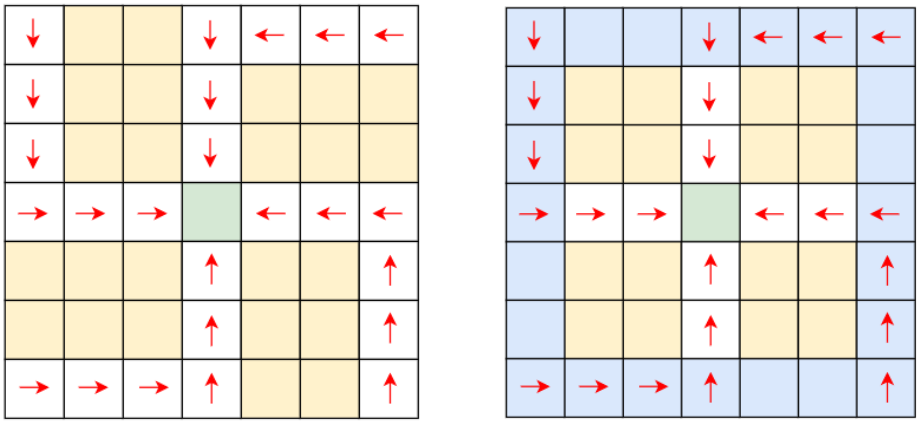
However, the algorithm proposed in *Odom & Natarajan (2016)* [11] is not efficient for large state spaces, since it is a LP problem and therefore the number of constraints will always be unbearable. In this way, the basic idea will be introduced in works involving the manipulation of large state spaces such as the case of *Burdick & Li (2017)* [3]. The MLIRL approach with approximation function proposed in the work of *Burdick & Li (2017)* [3] was compared to other iterative IRL algorithms in computational terms and there was a significant improvement in computation time with the increase in the number of states, as it is shown in Table 6.1.

**Table 6.1:** Computation time (in seconds) of one iteration of gradient method under different number of states with different methods (taken from [3]). "MaxEnt" denotes maximum entropy method, "DeepMaxEnt" denotes the deep IRL approach, "BGI" denotes Bellman Gradient Iteration method, and "FAIRL" denotes the function approximation IRL with neural network approach. The "FAIRL" approach has a much lower computation time compared to any of the other referred approaches in large state spaces.

| States (#) | MaxEnt | DeepMaxEnt | BGI | FAIRL |
|---|---|---|---|---|
| 25 | 0.017 | 0.012 | 0.0313 | 0.197 |
| 225 | 1.831 | 0.178 | 2.031 | 0.397 |
| 625 | 24.151 | 0.95 | 20.963 | 0.724 |
| 1225 | 133.189 | 3.158 | 102.46 | 0.921 |
| 2025 | 474.907 | 8.119 | 352.007 | 0.776 |
| 3025 | 1319.365 | 20.253 | 1061.147 | 0.762 |
| 4225 | 3030.723 | 59.279 | 2630.309 | 2.468 |
| 5625 | 6197.718 | 101.434 | 5228.343 | 2.831 |
| 7225 | 12234.417 | 229.752 | 10147.628 | 2.217 |
| 9025 | 20941.9 | 10466.784 | 16345.874 | 3.347 |

## 6.2 Grid-world problem

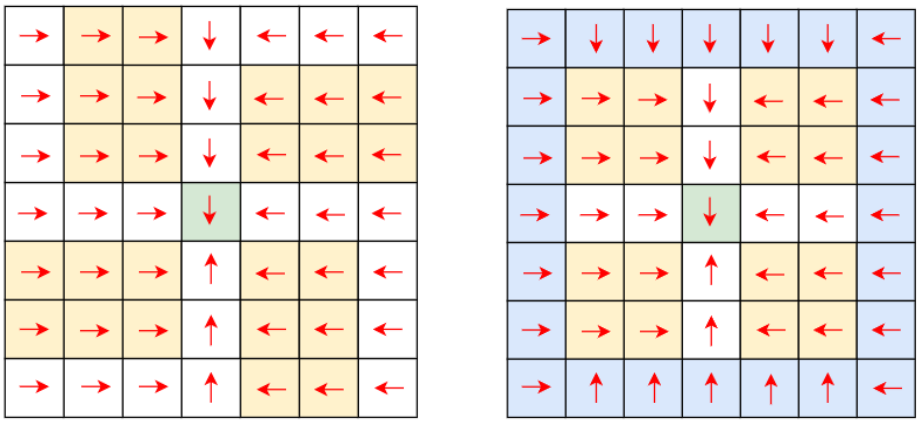
To show the impact of the method introduced in the thesis, it is performed the Algorithm 4.1 applied to a simple grid-world problem. The grid-world problem consists on a grid where the goal state is the center of the grid. Each cell is a state, and the possible actions correspond to $A = \{RIGHT, UP, LEFT, DOWN\}$. First, a transition model is built and several demonstrations are computed by the demonstrator. Figure 6.3 denotes the problem that is intended to solve. The green cell is the goal state, the red arrows represent the demonstration set. The yellow cells are the states that were never visited by the demonstration set.



**(a)** without state preferences **(b)** with state preferences

**Figure 6.3:** Demonstrations given to $7 \times 7$ grid. Red arrows are the given demonstrations. The yellow cells are the non-demonstrated states. The blue cells are the states to be avoided in state preference framework.

Figure 6.3(b) shows the states to avoid in blue and the preferred state (target state) in green. We can observe that this new information can have a lot of weight in the final policy learned, as it can even surpass the information given in the demonstrations. In Figure 6.4 it is possible to observe that with state preferences the trajectories move away from the limits of the grid, even when the demonstrations say so. This example shows that state preferences can fix suboptimal demonstrations. These results show that we do not need in this case to use the AcL framework, which can be a good sign to achieve fast convergence on larger problems.



**(a)** without state preferences **(b)** with state preferences

**Figure 6.4:** Final policy of $7 \times 7$ grid. Red arrows are the given demonstrations. The yellow cells are the non-demonstrated states, the green cell is the preferred state and the blue cells are the states to be avoided in the state preference framework.

Then, experiments were performed to evaluate the scalability of the algorithm as the number of states increased. The same experiments were done on $25 \times 25$, $35 \times 35$ and $101 \times 101$ grids. The NN used for the 3 different state spaces has the same format: 2 hidden layers with *ReLU* activation function, an output layer with *Tanh* activation function and learning rate $\alpha = 0.01$. The same format was used in order to make it easier to compare the computational costs of the 3 different grids and *Tanh* was chosen in order to output values between 1 and -1. Each NN input consists of a vector of 2 values that correspond to the 2 features of this problem: $s = [x, y]$, $x, y \in [0, 1]$.

**Table 6.2:** Algorithm scalability with increasing number of states (2000 iterations).

| States (#) | Time (s) | Time/Iteration (s/iter) |
|---|---|---|
| 49 | 21.24 | 0.0106 |
| 625 | 75.45 | 0.0377 |
| 1225 | 108.20 | 0.0541 |
| 10201 | 169.02 | 0.0845 |

In Table 6.2, it is possible to conclude that the increase in the size of the state space makes the algorithm scale in a linear factor less than 1, regarding the computational cost for each iteration, since the algorithm increases only with the cardinality of the action space.

Following the example of Figure 6.3, demonstrations were given in order to perform the same 4 trajectories from a corner to the center of the grid. As the grid size increases, the percentage of demonstrated states decreases. Figure 6.5 shows the obtained policy for the $101 \times 101$ grid.

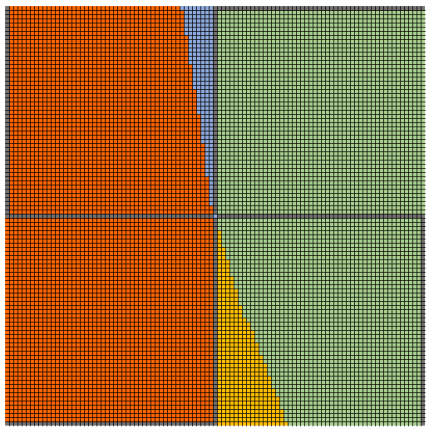


**Figure 6.5:** Final policy of $101 \times 101$ grid. Orange: Right; Blue: Down; Green: Left; Yellow: Up; Grey: Demonstrations.

Throughout the iterations, runs of the grid-world environment will be performed with all cells as initial state and following the policy obtained, and an accuracy score will be obtained for the number of runs that reached the goal state after $N$ state transitions (where $N = 7$ for the 7x7 grid for example). Fig. 6.6 represents the evolution along the iterations of the policy's accuracy score for different grid sizes.

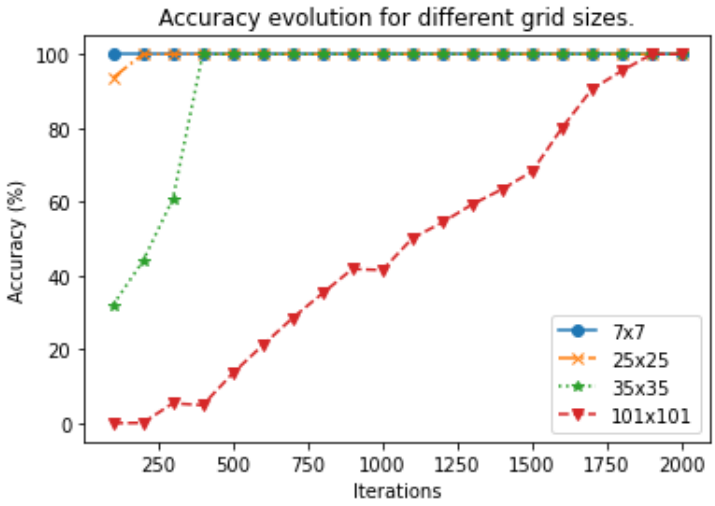



**Figure 6.6:** Accuracy evolution for different grid sizes.

In Fig. 6.6, it is possible to conclude that, even with a decrease in the percentage of states demonstrated with the increase of the grid size, the algorithm always manages to reach excellent performances with the increase in the number of iterations in the proposed grid-world problem. For the $101 \times 101$ grid, only 3.92% of the demonstrated states were needed for the algorithm to reach an optimal policy in 1900 iterations. The choice of gray states in Fig. 6.5 also helped this result, since if only states to the right of the goal state had been demonstrated, the algorithm would hardly learn the states to the left of the goal state, for example. That is why AcL is also important in this framework: the fact that demonstrations were given scattered all over the grid allowed the algorithm to need fewer demonstrations to achieve 100% accuracy.

## 6.3 Grasping problem

In this section, the proposed algorithm is applied to the grasping problem. The network was trained from a set of 981 initial demonstrations (Section 5.10) and after 10000 iterations an initial policy is obtained. Then, new demonstrations and new advice in action preference format will be introduced every 100 iterations to the most uncertain states/clusters labeled by the human demonstrator. All the advice in state preference format is given in the beginning.

The neural network consists of two hidden layers with *ReLU* activation function, and the output layer has *Tanh* activation function, as performed in the grid case. Hidden layers have 150 and 50 neurons respectively. The input is given by 15984 states with 10 features each state, so giving 159840 entries from the network. The 10 features correspond to the state components normalized to values between 0 and 1: the first 6 features correspond to the pose of the object, the seventh feature corresponds to the *is_grasp* component (0 or 1) and the last 3 features correspond to the orientation of the gripper ([0,0,0] if the object is not grasped).

It was noticed an increase in the computation time of each iteration compared to all grid-world cases, since the number of actions is greater. Three experiments were performed with different additional demonstration set sizes. In the first experiment, no demonstrations were added, so the network was trained only with the 981 initial demonstrations and with 10000 iterations. In the second experiment, additional demonstrations and advice in action preference format were given iteratively to random states and clusters. In the third experiment the additional demonstrations and advice in action preference format were given iteratively using the proposed query selection process based on uncertainty.

For this problem, a validation set consisting of 100 demonstrations is built by the demonstrator. The accuracy score in this case will be the percentage of times where the states shown in this validation set match the policy in the optimal action. Table 6.3 shows the obtained results for these three experiments.

**Table 6.3:** Accuracy score with or without state and action preferences for 3 different demonstration sets. The first line corresponds to the percentage of states demonstrated in the demonstration set (training set).

| | Init set | Set without AcL | Set with AcL |
|---|---|---|---|
| States demonstrated (%) | 6.14 | 14.25 | 8.11 |
| Without Advice (%) | 8 | 15 | 19 |
| Action Advice (%) | 9 | 15 | 19 |
| State Advice (%) | 14 | 18 | 21 |
| Action and State Advice (%) | 15 | 18 | 21 |

Through Table 6.3, it is possible to observe the effect of choosing the query through AcL: even with fewer demonstrations, the results of the third experiment network are better than those of the second experiment. This is because the demonstrations added in the third experiment are more informative as the states shown are more uncertain. In other words, the new demonstrations show behaviors that are more pronounced and less known by the neural network.

The inclusion of state and action preferences in the iterative process to update the $Q$ matrix also improves the accuracy, as it consists in the introduction of information that is not covered by the demonstration set. Even so, the results are not as expected when compared to the grid case. These results may be caused by some error in the construction of the MDP that was not identified by the author, however there are other reasons identified:

- The transition model is not the most correct as it was not possible to perform enough Bernoulli experiments to obtain a robust and complete model.

- The fact that there are actions that can not be performed for certain states (such as grasping an object that is already being grasped) may also have an influence on the following states.

- Demonstrations may contain certain contradictions, as they are susceptible to labeling errors by the demonstrator or in the generalization process. These contradictions can lead to a more difficult interpretation of the intended behaviors in the final policy. However, the given advice was able to correct some of these possible errors by the demonstrator, as can be seen in the improved accuracy.

- The choice of the given states for demonstration may also not have been the best, since it is more difficult to identify desirable behaviors with the set of state features presented in the grasp problem. Also for this reason, improvements are expected with the use of a query choice technique that better identifies the states that need to be demonstrated for a better identification of these behaviors.

## 6.4 Discussion

In conclusion, although accuracy values are lower than expected for the grasping problem, it is possible to confirm certain aspects referred in both problems:

- The choice of the states that will be demonstrated is very important in obtaining the optimal policy. Choosing the right states allows capturing the most crucial information for the neural network to be able to identify the desirable behaviors.
- The completion of learning through demonstrations with advice allows not only to correct some suboptimal demonstrations, but also to obtain optimal actions for some states not included in the demonstrations, accelerating the process of obtaining a good policy.
- The computational cost of the neural network increases with the increase in the number of actions, as seen in [3].

# 7

# Conclusion

## Contents

In the work it was proposed an approach to implement a decision system based on the knowledge acquired by the robot from human demonstrations and advice. The aim was to design the decision system applied to a simple grasping problem with a large state and action sets, where a lot of training is necessary to obtain a robust model that allows the robot to know what to do in a given situation. In large state spaces it is difficult to perform experiments on all of the states, so it is crucial that the samples given to the robot are as informative as possible.

## 7.1 Summary

The algorithm proposed in the work implement the idea on learning by demonstrations and advice simultaneously, in order to obtain more information from an expert with less iterations. The algorithm consisted on applying AcL and AdL on a MLIRL framework, obtaining an approximation of the VR function (sum of the reward function with the discounted state value) using a NN and consequently a new updated policy. Then a query selection model based on uncertainty was implemented to obtain a cluster query that was given to the expert to label it in the form of action advice and a state query to label it in demonstration form.

Through the obtained results it was possible to observe the importance of the informativeness of the demonstrations in the proposed algorithm, since the choice of the right states to demonstrate will present better results with less necessary demonstrations (as is the case of grids). It was also possible to observe an increase in accuracy when using advice as a complement to the demonstrations.

## 7.2 Main Risks

For a good functioning of the implemented system it is necessary a good clustering algorithm to cluster the states with the same policy behavior in order to accelerate the visit to all states of the environment and to give the best label action to each state. If the clustering is not well defined, some states do not have associated the best action in its policy. It is also necessary to give the less certain queries to the expert in order to reach optimality faster, so better query strategy alternatives to US can be addressed to represent better informativeness. Another risk taken is that it is not possible to obtain a good transition model that represents the behavior of the environment well, since it is considered a large discretized state space. In the same way, there is a risk on demonstration generalization, since contradictions in the desirable behavior for the set of demonstrations can more easily occur.

## 7.3 Future work

After experimenting with the proposed algorithm to accelerate the learning process in two different scenarios, many other approaches can be improved or added to this work for the grasping problem case.

The transition model can be improved by performing Bernoulli experiments for all the problem states a sufficient number of times to obtain a good representation of the behavior of the environment. In this work, it was not possible to compute the sufficient number of experiments through lack of time. Considering the difficulty in creating a good transition model, the ideal would be to successfully adapt the proposed algorithm to a model-free version by replacing the $Q$-matrix update formula in order to avoid the presence of a transition model $P$ (see [25] for motivation). The NN itself can be optimized, as there was no time to explore the effect of the number of layers and weights or different activation functions. Other settings may better optimize getting the VR function.

Other possible improvements to this framework would be to improve clustering quality or implement a query selection strategy more appropriate for this context than US.

Finally, the baseline of the grasping problem could be transferred to the real world, thus implementing a visual component to identify the state (namely where the object is and if it is being effectively grasped) using the camera integrated in the wrist of the *KINOVA Gen3* manipulator. This algorithm can also be tested for an even larger state space, as there was no time to run the algorithm with a broader problem.

# Bibliography


[1] B. Settles, "Active learning literature survey," *Computer Sciences Technical Report 1648, University of Wisconsin-Madison*, 2009.

[2] P. Odom and S. Natarajan, "Active advice seeking for inverse reinforcement learning," in *Proceedings of the 2016 international conference on autonomous agents & multiagent systems*, 2016, pp. 512–520.

[3] K. Li and J. W. Burdick, "Inverse reinforcement learning in large state spaces via function approximation," 2017.

[4] S. Russell, "Learning agents for uncertain environments," in *Proceedings of the eleventh annual conference on Computational learning theory*, 1998, pp. 101–103.

[5] A. Y. Ng, S. J. Russell *et al.*, "Algorithms for inverse reinforcement learning." in *Icml*, vol. 1, 2000, p. 2.

[6] M. Lopes, F. Melo, and L. Montesano, "Active learning for reward estimation in inverse reinforcement learning," in *European Conference on Machine Learning (ECML/PKDD)*, 2009, pp. 31–46.

[7] D. Brown and S. Niekum, "Efficient probabilistic performance bounds for inverse reinforcement learning," *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[8] D. Brown, Y. Cui, and S. Niekum, "Risk-aware active inverse reinforcement learning," in *Conference on Robot Learning (CoRL)*, 2019.

[9] G. Kunapuli, P. Odom, J. W. Shavlik, and S. Natarajan, "Guiding autonomous agents to better behaviors through human advice," in *2013 IEEE 13th international conference on data mining*. IEEE, 2013, pp. 409–418.

[10] G. Neu and C. Szepesvári, "Training parsers by inverse reinforcement learning," *Machine learning*, vol. 77, no. 2-3, p. 303, 2009.

[11] P. Odom and S. Natarajan, “Actively interacting with experts: A probabilistic logic approach,” in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2016, pp. 527–542.

[12] P. Abbeel and A. Y. Ng, “Apprenticeship learning via inverse reinforcement learning,” in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 1.

[13] Y. Gao, J. Peters, A. Tsourdos, S. Zhifei, and E. M. Joo, “A survey of inverse reinforcement learning techniques,” *International Journal of Intelligent Computing and Cybernetics*, 2012.

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[15] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, “A survey of robot learning from demonstration,” *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[16] M. Racca and V. Kyrki, “Active robot learning for temporal task models,” in *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, 2018, pp. 123–131.

[17] L. Mihalkova and R. Mooney, “Using active relocation to aid reinforcement learning,” in *Proccedings of the Florida Artificial Intelligence Research Society (FLAIRS)*. AAAI Press, 2006, pp. 580–585.

[18] M. Fang, Y. Li, and T. Cohn, “Learning how to active learn: A deep reinforcement learning approach,” *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

[19] M. C. Vroman, *Maximum likelihood inverse reinforcement learning*. Rutgers The State University of New Jersey-New Brunswick, 2014.

[20] R. Coelho, “Planning push and grasp actions: Experiments on the icub robot,” 2013.

[21] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[22] A. Vannobel, “Ros kortex,” open-source official ROS package for Kortex robots, accessed on 10/03/2021. [Online]. Available: https://github.com/Kinovarobotics/ros_kortex

[23] J. J. Craig, *Introduction to Robotics: Mechanics and Control*. Pearson Education International, 2005.

[24] K. R., “Kinova gen3 documentation,” official documentation of KINOVA Gen3, accessed on 21/04/2021. [Online]. Available: https://www.kinovarobotics.com/sites/default/files/UG-014_KINOVA_Gen3_Ultra_lightweight_robot_User_guide_EN_R06_0.pdf

[25] A. Boularias, J. Kober, and J. Peters, “Relative entropy inverse reinforcement learning,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 182–189.

[26] K. Robotics, “Kinova gen3 lite documentation,” official documentation of KINOVA Gen3 Lite, accessed on 10/03/2021. [Online]. Available: https://www.kinovarobotics.com/sites/default/files/UG-017_KINOVA_Gen3_lite_robot_USER_GUIDE_EN_R01_0.pdf

[27] H. M. Zohour, B. Belzile, and D. St-Onge, “Kinova gen3-lite manipulator inverse kinematics: optimal polynomial solution,” 2021.

# A

# *KINOVA Gen3 Lite* description

## A.1 Useful Specifications

In the current section some specifications of the *KINOVA Gen3 Lite* robot will be explained. Figure A.1 shows a representation of the main components of the robot, including the name and locations of each actuator. *KINOVA Gen3 Lite* contains 6 DoF and a 2-fingered gripper as its end-effector.

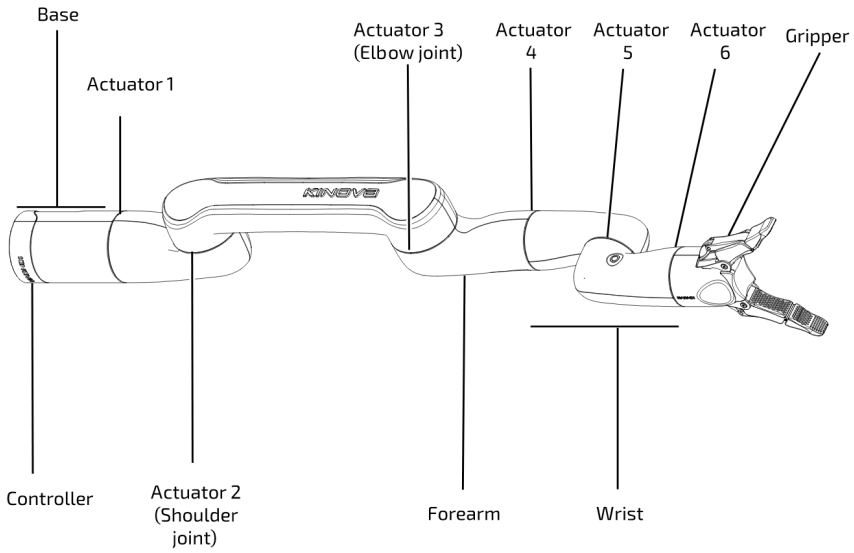



**Figure A.1:** *KINOVA Gen3 Lite* main components.

### A.1.1 Devanit-Hartenberg parameters

Table A.1 presents the DH parameters of *KINOVA Gen3 Lite*.

**Table A.1:** *KINOVA Gen3 Lite* DH parameters.

| i | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $a_i$ [m] | 0 | 0.28 | 0 | 0 | 0 | 0 |
| $b_i$ [m] | 0.2433 | 0.03 | 0.02 | 0.245 | 0.057 | 0.235 |
| $\alpha_i$ [°] | 90 | 180 | 90 | 90 | 90 | 0 |
| Lower Limit [°] | -154 | -150 | -150 | -149 | -145 | -149 |
| Upper Limit [°] | 154 | 150 | 150 | 149 | 145 | 149 |

Considering **q** as the vector of the values of each joint, $\theta$ is given by the transformation contained in Table A.2.

**Table A.2:** Conversion between **q** and $\theta$.

| i | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $\theta_i$ [°] | $q_0$ | $q_1$ + 90 | $q_2$ + 90 | $q_3$ + 90 | $q_4$ + 180 | $q_5$ + 90 |

Figure A.2 outlines the dimensions and frames of each joint/link pair of the manipulator. To see more useful specifications of this manipulator, see its documentation [26].

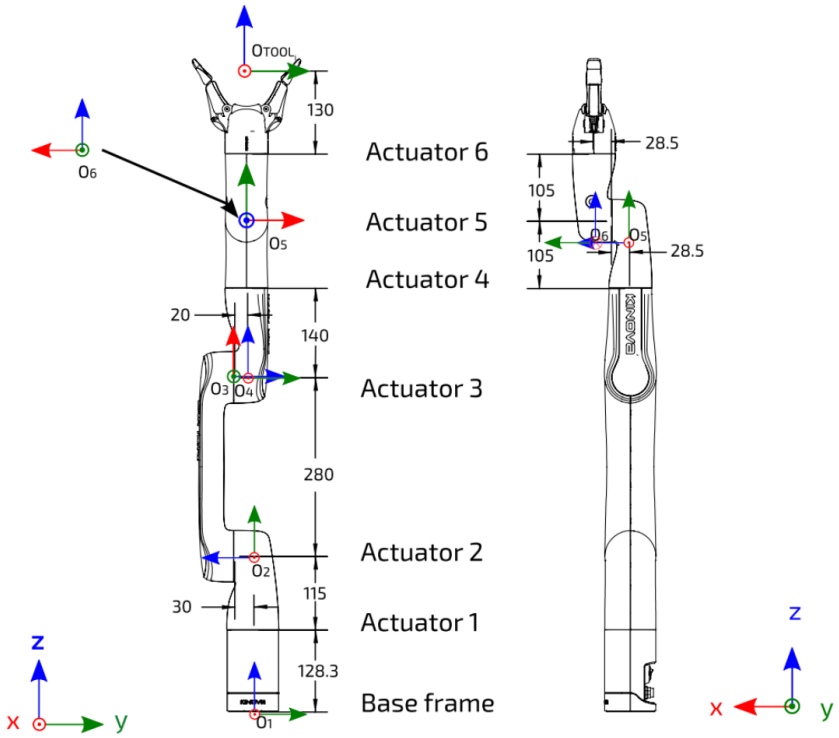



**Figure A.2:** *KINOVA Gen3 Lite* link dimensions and joint frames.

## A.2 Inverse Kinematics

The *KINOVA Gen3 Lite* has 6 DoF, so it has a maximum of 16 possible solutions for the IK problem. A 6-DoF manipulator is the most simple manipulator where is possible to compute a desired pose (position+orientation) using IK. Any manipulator with less than 6 DoF can not compute a specific orientation of the end-effector.

*Zohour et al. (2021)* [27] propose a polynomial solution to the IK problem of the *KINOVA Gen3 Lite* robot. The work of this paper was used in this thesis to compute a first non-tuned solution in order to achieve a solution close to the correct one, as it will be explained in more detail later.

To compute the IK for *KINOVA Gen3 Lite*, the ROS *Kortex* package provides a function to calculate it directly using a IK solver based on gradient descent. Since this solver uses a gradient descent-based method, it has to take into account the current pose of the end-effector (from the values of the angle of the joints), which makes this solver a bad approach in cases where the end-effector is still a long way from the desired pose and the manipulator has few DoF. This fact leads us to many cases where this robot gives errors because it cannot find a solution, despite its existence, such as mentioned before.

The solution found to this problem is to firstly find an approximation of the desired solution using another numerical solver, and then complement it with the solver provided in the ROS *Kortex* package

to fine-tune the final solution. Afer this first approximation, the gradient descent-based solver works much better, as the initial pose is already close to the desired pose.

This is where the work of *Zohour et al. (2021)* [27] enters in this thesis: the solver proposed in this work is the one used to perform the first approximation. This algorithm also has the advantage of providing all possible solutions for the IK problem, with the optimum solution being the one that minimizes the error between the goal pose and the resulting pose.

The first step that this method use to obtain a solution to the IK problem of the *KINOVA Gen3 Lite* is to reduce the number of unknowns, currently 6 joint angles $\theta_i$, to only one ($\theta_1$), therefore reducing the problem to a univariate polynomial equation that can be solved eventually. The remaining joint variables are computed by back substitution. Thus, an unique set of joint angles is obtained for each root of the univariate equation.

The results of this work prove a similarity between the solution obtained by this solver and the ones obtained using the robot provided solver and *MoveIt!*. In many cases the solution obtained by this work has some considerable error, but it is good enough to get an approximation and then after that use the robot solver for the fine-tuning process. This approach allows us to prevent the limitations of both solvers and with that achieve a resulted pose with negligible error.

## A.3 Reachability map

For *KINOVA Gen3 Lite*, the process consists of performing IK for each pose, where the polynomial solver is performed first and then the robot solver. If the manipulator's end-effector is not in the desired pose or the polynomial solver does not find a solution, then the manipulator will only try to perform IK with the robot solver. If the manipulator is able to place its end-effector in the desired pose in the first procedure, then the reachability map will have a value of 1, if it only succeeds in the second procedure it returns 2, if it fails with any of them it returns 0 (that is, it is not possible to place the end-effector in the desired pose).

## A.4 X-Y-Z Fixed-angle notation

The numerical IK solver used only for *KINOVA Gen3 Lite* uses the X-Y-Z Fixed-angle notation to represent the desired orientation. The X-Y-Z Fixed-angle notation consists on rotating the object with fixed axis (always with the "world reference frame"). Figure A.3 shows an illustration of this rotation convention.

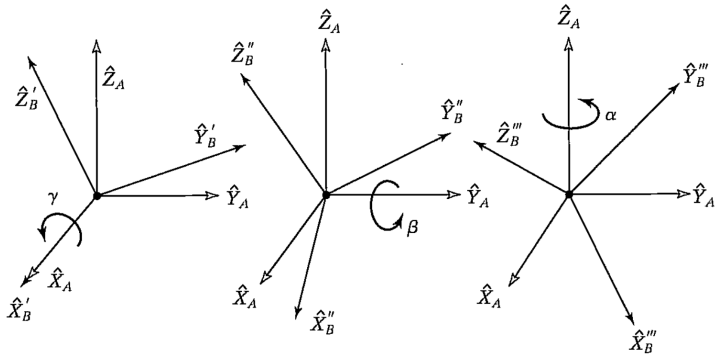



**Figure A.3:** X-Y-Z Fixed-angle notation.

A X-Y-Z Fixed-angle notation has the following parameters:

$$\begin{aligned} \theta &= \sqrt{\alpha^2 + \beta^2 + \gamma^2} \\ u_x &= \frac{\alpha}{\theta} \\ u_y &= \frac{\beta}{\theta} \\ u_z &= \frac{\gamma}{\theta} \end{aligned} \quad . \tag{A.1}$$

The transformation to rotation matrix is then computed as follows:

$$R = \begin{bmatrix} u_x^2 \cdot C + c & u_x \cdot u_y \cdot C - u_z \cdot s & u_x \cdot u_z \cdot C + u_y \cdot s \\ u_y \cdot u_x \cdot C + u_z \cdot s & u_y^2 \cdot C + c & u_y \cdot u_z \cdot C - u_x \cdot s \\ u_z \cdot u_x \cdot C - u_y \cdot s & u_z \cdot u_y \cdot C + u_x \cdot s & u_z^2 \cdot C + c \end{bmatrix} \quad , \tag{A.2}$$

where $C = (1 - \cos\theta)$, $c = \cos\theta$ and $s = \sin\theta$. The conversion of a rotation matrix to X-Y-Z fixed-angle is computed as follows:

$$\begin{aligned} \theta &= \cos^{-1}\left(\frac{tr(R) - 1}{2}\right) \\ \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} &= \frac{1}{2\sin\theta} \cdot \begin{bmatrix} r32 - r23 \\ r13 - r31 \\ r21 - r12 \end{bmatrix} \cdot \theta \end{aligned} \quad , \tag{A.3}$$

where $tr(R) = r11 + r22 + r33$. In special cases, when $\sin\theta = 0$, [$\alpha$, $\beta$, $\gamma$] is given by:

$$\begin{aligned}\alpha &= \sqrt{\frac{r11+1}{2}} \cdot \theta \\ \beta &= \frac{r12+r21}{4 \cdot \alpha} \cdot \theta^2 \\ \gamma &= \frac{r13+r31}{4 \cdot \alpha} \cdot \theta^2\end{aligned} \quad . \tag{A.4}$$

# B

# Proofs

## B.1 Equation (3.4)

Since $\pi(s) \equiv a^*$, the state value $V^\pi$ may be written as $V^\pi = R + \gamma \times P_{a^*} \times V^\pi$. Thus,

$$V^\pi = (I - \gamma \times P_{a^*})^{-1} \times R \quad . \tag{B.1}$$

Substituting Equation (3.2) into Equation (3.3), we see that $\pi(s) \equiv a^*$ is optimal iif:

$$\begin{aligned}
& a^* \equiv \pi(s) \in \underset{a \in A}{\operatorname{argmax}} \sum_{s'} P_{sa}(s') \times V^\pi(s') && , \forall s \in S \\
& \Leftrightarrow \sum_{s'} P_{sa^*}(s') \times V^\pi(s') \geqslant \sum_{s'} P_{sa}(s') \times V^\pi(s') && , \forall s \in S, a \in A \\
& \Leftrightarrow P_{a^*} \times V^\pi \succeq P_a \times V^\pi && , \forall a \in A \setminus a^* \\
& \Leftrightarrow P_{a^*} \times (I - \gamma \times P_{a^*})^{-1} \times R \succeq P_a \times (I - \gamma \times P_{a^*})^{-1} \times R && , \forall a \in A \setminus a^* \\
& \Leftrightarrow (P_{a^*} - P_a) \times (I - \gamma \times P_{a^*})^{-1} \times R \succeq 0 && , \forall a \in A \setminus a^*
\end{aligned}$$

## B.2 Equation (3.6)

In this section I will demonstrate that:

$$\sum_{s \in S} \quad Q^\pi(s, a^*) - \max_{a \in A \setminus a^*} Q^\pi(s, a) \quad ,$$

is equivalent to:

$$\gamma \times \sum_{s \in S} \quad \min_{a \in A \setminus a^*} (P_{sa^*} - P_{sa}) \times (I - \gamma \times P_{a^*})^{-1} \times R \quad .$$

Substituting the Equation (3.2) into the first equation and given that $a_1$ is the second action with best action value for state $s$, we see that:

$$\begin{aligned}
& \sum_{s \in S} \quad Q^\pi(s, a^*) - Q^\pi(s, a_1) \quad , \\
& \Leftrightarrow \sum_{s \in S} \sum_{s'} \quad (\gamma \times P_{sa^*}(s') \times V^\pi(s') - \gamma \times P_{sa_1}(s') \times V^\pi(s')) \quad , \\
& \Leftrightarrow \sum_{s \in S} \quad \gamma \times (P_{sa^*} - P_{sa_1}) \times V^\pi \quad , \\
& \Leftrightarrow \sum_{s \in S} \quad (P_{sa^*} - P_{sa_1}) \times \left(\gamma \, (I - \gamma \times P_{a^*})^{-1} \times R\right) \quad , \\
& \Leftrightarrow \gamma \times \sum_{s \in S} \quad (P_{sa^*} - P_{sa_1}) \times (I - \gamma \times P_{a^*})^{-1} \times R \quad , \\
& \Leftrightarrow \gamma \times \sum_{s \in S} \quad \min_{a \in A \setminus a^*} (P_{sa^*} - P_{sa}) \times (I - \gamma \times P_{a^*})^{-1} \times R \quad .
\end{aligned}$$

Since we want to maximize this expression, we can cut the $\gamma$ scalar, as it will not make any difference in the maximization.