UNIVERSIDADE DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

UNIVERSITÉ PIERRE ET MARIE CURIE

High-level planning of dexterous in-hand manipulation using a robotic hand

Urbain PRIEUR

Orientadores *(Supervisor)*: **Doutor** *(Doctor)* **Véronique PERDEREAU**

Doutor *(Doctor)* **Alexandre BERNARDINO**

Tese especialmente elaborada para obtenção do Grau de Doutor em Engenharia Electrotécnica et de Computadores

Thesis specifically prepared to obtain the PhD Degree in Electrical and Computer Engineering

Tese Provisória

Draft

Setembro 2013

September 2013

Abstract

This work considers a robot equipped with an anthropomorphic hand and aims at providing it with efficient autonomous in-hand manipulation skills. While fine in-hand action planning algorithms have interesting state-of-the-art solutions, we built a competitive high-level control layer to plan the complete in-hand manipulation activity. Our solution generates a sequence of subgoals from an initial to a final configuration provided by the task, thus decomposing in-hand manipulation into simple transitions that can be easily planned by the low-level algorithms.

We use a Markov decision process (MDP) to generate the sequence, taking into account the object influence and the desired final subgoal. We use a simple state representation for the sugoals: canonical grasp types from a taxonomy, enabling fast and on-line computation. The transitions between grasp types are modelled as probabilities of success. The simple formulation of the sequence leaves the complete configurations and transitions to be planned by the low-level layer, which can ask for a different subgoal path if required.

The MDP can generate the appropriate behaviour if the in-hand action skills of the robot are known. They can be learnt by self-exploration of the robot if possible. Otherwise, the behaviour can be directly learnt from human demonstration. We boost the learning process using an empirical guess of the transition probabilities and an active learning algorithm. We implemented our solution on a real platform. The planning of in-hand manipulation relies on the grasp sequence generated which probability of success is used as an insight of the task achievability for the initial grasp choice.

Resumo

Este trabalho considera o problema da manipulação dextra autónoma em robots equipados com mãos antropomórficas. Enquanto a maioria dos trabalhos de estado-da-arte se focam o planeamento fino de ações, o trabalho presente desenvolve um algoritmo competitivo para o planeamento de alto nível de sequências de manipulação dextra, um problema que tem sido menos abordado na literatura. A partir de uma condição inicial e de um objectivo para a tarefa, a solução proposta gera uma sequência de sub-objectivos que decompõem o problema da manipulação dextra em transições mais simples que podem ser planeadas mais facilmente pelos algoritmos de baixo nível. As técnicas utilizadas baseiam-se em processos de decisão de Markov (MDP) para gerar a sequência, tendo em conta o objecto manipulado (a sua forma) e a tarefa desejada (o objectivo final). É utilizada uma representação de estado baseada em preensões canónicas dos objetos de acordo com uma taxonomia baseada em modelos humanos. As transições entre preensões canónicas são modeladas com probabilidades de sucesso. Esta representação simples permite efetuar o planeamento de forma muito rápida, deixando os detalhes da configuração e transição para os controladores de baixo nível, que poderão requisitar em tempo-real outra sequência se alguma das ações for infactível. O planeamento com o MDP's consegue gerar um comportamento desejável se as capacidades de manipulação de baixo nível do robot forem conhecidas. Estas poderão também ser aprendidas por auto-exploração ou através de demonstrações humanas. Neste trabalho é feita uma aprendizagem incremental onde, numa primeira fase é criada um modelo inicial através de conhecimento humano e depois é aplicado um algoritmo de aprendizagem ativa. A solução foi implementada numa plataforma real. A probabilidade de sucesso da sequência gerada permite priorizar soluções, o que é utilizado para determinar de forma eficiente o tipo de agarre inicial a aplicar no objecto, em função da tarefa desejada, propondo assim uma solução para um problema ainda em aberto na comunidade científica.

Résumé

Jusqu'ici, les solutions apportées à la problématique de manipulation dextre robotique par une main anthropomorphe se concentraient sur la planification bas niveau des différents types de mouvements fins. Aussi, une solution complète de planification, prenant en compte l'étendue de ces différentes actions, reste à développer. Tel est le fondement de ce travail : pour une tache définie, caractérisée par des configurations initiale et finale, une séquence de configurations intermédiaires est générée : la totalité de l'activité complexe se trouve donc décomposée en une succession de transitions simples à planifier par le bas niveau.

Afin de générer cette séquence, influencée par l'objet et la configuration finale, on utilise un processus de décision markovien (MDP). Les configurations intermédiaires sont représentées par des types de saisies tirés d'une taxonomie existante, les transitions sont formulées par des probabilités de succès : si ces approximations assurent la rapidité de l'algorithme, elles requièrent du bas niveau la planification des configurations complètes et le détail des transitions. En cas d'impossibilité d'exécution, des séquences alternatives peuvent être proposées.

Le MDP doit connaître les capacités du robot à réaliser les transitions, par auto-apprentissage, ou par observation de mouvements humains. Nous utilisons un processus d'apprentissage actif initialisé par une estimation empirique des probabilités de succès des transitions. La solution a été implémentée avec succès sur un robot réel, prenant un rôle central dans la commande de la manipulation fine, jusqu'à propager l'influence de la tâche sur la saisie initiale.

Acknowledgements

The work presented here is the result of a few years of work, and the outcome of this great period of my life is larger than this manuscript only. I learnt a lot, as a scientist and as a human being, while meeting a lot of great people from all over the world, improving my English and opening my mind.

I worked in very good conditions in two very charming cities: Lisbon and Paris, and my gratitude goes first to my advisors for making this possible: Véronique for offering me to join the HANDLE project and her careful support and guidance, and Alex for his warm welcome in Lisbon, his scientific advice and his availability. The feedback and meticulous comments from both of them greatly enhanced the quality of this manuscript.

I thank the two laboratories for welcoming me: Vislab, IST, Lisbon and its director, José Santos-Victor, and ISIR, UPMC, Paris, and its director, Philippe Bidaud.

I want to express my sincere gratitude to each jury member of this thesis: Raja Chatila from ISIR, Paris and Ruben Martinez-Cantin from CUD, Zaragoza. I would like to offer my special thanks to Thierry Siméon from LAAS, Toulouse, and Abderrahmane Kheddar from LIRMM, Montpellier for reviewing this manuscript.

My co-workers from the Assist-Handle team in ISIR have been a very great humane and scientific support throughout this whole work, and I want to thank each one of them. I am in debt to Guillaume Walck who gave all the help he could besides his admirable work on the Handle platform. I deeply thank Juan Antonio Corrales Ramón and David Flavigné for the coffee breaks and the delightful talks we had in our office. I am grateful to Lan Pham and Kien N'Guyen for their exotic contribution to the atmosphere of the team. My great appreciation goes to Ella for her refreshing presence, to Michel for his scientific contribution and to Brigitte for the spice she added in the team.

I deeply enjoyed my few stays in Vislab, working there has been a blessing. The credit for it goes to the senior researchers as well as the younger members, among them: Jonas, Ricardo, Matteo, Giovanni, Filipe, Rui, Nuno, Plinio, Feng, Duarte and Sebastien.

English proofreading provided by Maddy and Tamara has been precious and I thank them for it and also for being so valuable family members.

The next paragraph is in French, as it has been written for French speaking people.

Cette thèse conclut de longues études, et je souhaite partager ce succès avec tout ceux qui m'ont accompagné sur ce chemin. Ma mère et mon père m'ont élevé, soutenu et avec mes frères et ma soeur m'ont enrichi, motivé et aidé à poursuivre ce que j'aime. Il m'est impossible de leur en être assez reconnaissant. Enfin, pour l'étendue de ce qu'ils m'apportent, j'adresse un très large et très profond remerciement à mes amis dont le nombre et la qualité sont deux de mes plus grandes chances. Je ne cite que les principaux: Manu, Clément, Albin, Antoine, Gilles, Jérémie, Ivan, Aurore, Romain, Florent, HA, Nico, et spécialement Nelly pour son soutien final.

Contents

Chapter 1

Introduction

Manipulation is one of the challenges in modern robotics. To increase their utility in society and industry, robots must be provided with advanced manipulation abilities. We start with a philosophical view of the core topics related to robot manipulation. Then we explain the objectives of this work, its funding and the organisation of the manuscript.

1.1 Robotics

Humans, in order to survive, have to work. In early ages, this work was mainly about fulfilling the basic needs (shelter, water, food). In the developed civilizations that have been built since, this work can take a very large range of activities, fulfilling basic and secondary needs: chop down a tree, crop wheat or grapes, kill an animal, cook, lift and move light or heavy loads, dig the ground, machine steel parts, cut vegetables, explore an area, check somebody's vital signs or collect and process data.

To do this, humans move, manipulate objects with their hands or use a tool. When the task is beyond their capabilities, humans build a machine and a human being drives it (lifting loads, precision machining, ...). When the task is in a risky environment or out of reach (Mars, radioactive, ...), in the same way, the machine is remotely controlled. With robotics, humans try to build a machine with autonomy so that it does not need a driver any more, but only high-level orders. This can be done by building machines specific to a single task. This can also be done by building a robot that uses tools or drives other machines, thus substituting a human being in any process. Such a robot, to have the ability to use any human tool or machine, would probably be human like. Here lies the quest of the humanoid robot: robots able to substitute humans in activities they cannot or do not want to do.

Some activities will never be done by somebody else than a human, like arts, as it involves creativity, expression, beauty, and finds motivation within human nature. An artist can use robots for art, but a robot will not initiate artistic activity. Some activities should not be done by somebody else than a human: strategy/politics, law, war. The opposite would generate strong ethical issues.

Until now, in an industrial context, robots are able to surpass humans for specific tasks in strength, speed, precision and working hours. Their cost is high, and the related investment is justified only under precise conditions that makes the robot cheaper than human operators.

- The task has to be repetitive, because a robot is designed for a single task and usually carries one tool.
- The environment has to be adapted to the robot.

The adaptation of the environment is an answer to a key problem in robotics, the need to know the environment of the robot. This means knowing the positions and sizes of obstacles and objects involved in the task. The mainly used solution for now is to always have the same environment, and write this information in the robot memory. It drastically limits the robots versatility. Other solutions, more costly and less reliable use sensors (cameras, laser range devices) able to detect the environment space and obstacles, or an object (by contact detection). Failures are usually hard to detect, prone to generate heavy damages and need a human operator to be solved. In an industrial context, it means that the production process has to be specifically designed to be automated to meet the strict robot specifications. Industrial processes must be decomposed into successive repetitive tasks, it implies dedicated areas, tools and parts adapted to the robot capabilities. Then, a long and precise robots tuning is required, together with specific watch and maintenance while operating.

To illustrate the increase of versatility of a machine, a good example is a modern computer numerical control (CNC) machine tool. With a single tool, a milling cutter for example, the machine can only perform milling. If it is able to use several tools (milling cutter, drill bit), and has the associated intelligence, this robot becomes more versatile and more useful. However, what it does is always machining. A versatile robot should be able to use a large set of tools.

This is profitable only for high scale production lines, where operations are performed in an identical way thousands of times whereas lines issuing a low number of products mainly consist of human operators who are more versatile and autonomous. It explains why, for now, robots are widely used in high production industries (car production, hi-fi devices, ...) where production scale justifies the investment in robots.

Cost is an issue for the spread of robotics in a human world. But building cheap machines as complex as robots is no piece of cake. Some efforts are made in this way, such as the "10$ Robot Design Challenge" [1]. The target of this challenge is to design and build a cheap robot that could be used to teach robotics in developing countries.

High-cost robots can find a way to counterbalance the investment they require by proposing a wide range of possible activities, suitable for the evolutions of the company. It implies that versatility is one of the next challenges for robots to spread into the industry. It means, on the one hand, being able to detect the variation in the environment, and on the other hand, being able to act on the environment in numerous ways. This is especially true in manipulation, where robots often have an end-effector especially designed for the manipulated object.

It is also a challenge to fit robots into our homes, to assist humans in their household duties or assist disabled people in everyday life. In houses and flats, the environment can be adapted in some ways, but the tasks are many and varied, making versatility a major issue.

According to the Willow Garage CEO, Steve Cousins [16], the development of robots in everyday environment (not industrial robots) is likely to be a stairway. Firstly, robots will move automatically. This robotic activity is called navigation and addresses fields as path planning, trajectory generation, obstacle detection and avoidance. It is useful for surveillance robots or telepresence robots ("go and transmit"). In a second step, robots will be able to move and act. About moving, robotic navigation is currently providing good solutions and keeps improving, whereas the "act" step is still at an early stage. This is robotics' next challenge, where manipulation is the key.

1.2 Manipulation

Robots among humans have to perform tasks. If the task is to only use a specific tool: "go there and write", the robots embed the tool and use it properly. But for manipulating objects of the environment, often designed for human hands, or having a wide task range using different tools, also often designed for human hands, a humanoid hand seems the most appropriate end-effector. It gives the robot the same capabilities as a human, as far as intelligence and control are also at the same level. Tasks as diversified as the following would be possible for such a robot:

- Take an object, bring it and give it to somebody (give me a beer bottle).
- Pick an object, place it somewhere else (tidy a kitchen).
- Act on an environment object (switch off the light, turn off the heater, close the gas valve).
- Use tool (open the beer can, take a pen and write, cut a tomato, use the vacuum cleaner, clean the fridge).
- Interact with humans (touch, limb motion, tactile interaction, sign language).

Among these tasks, few of them need that the robot grabs the object, moves it and releases it. Some tools only need to be grasped in a specific way to be used. Other tools need to be manipulated within the hand. It is obvious for objects like a pen or a remote control. Such reconfiguration of the hand on the object is called in-hand manipulation. Manipulation without in-hand manipulation considers the problem of grasping, exposed in section 2.1.2, and moving the object. The in-hand dexterous manipulation problem is rather considered in this work.

Human hand is the result of evolution. It built (and destroyed) entire civilizations, proving its wide abilities. A robot with such abilities would be useful in a house, in a hospital, an industry, a restaurant, or anywhere where humans can do things. A

human-like hand on a robot, able to perform human-like actions, would give robots the ability to take on work as a human being. These versatile robots would take a crucial role in our society. Most robots nowadays have a single arm and tool, being designed to execute single tasks. Manipulation robots often have a specific effector, or a simple gripper, or for example the Barrett Hand [2], all unable to perform all of the aforementioned tasks at a human-like level.

If we assume that the human hand is the result of biological evolution, it is then optimal in this world, with a human nervous system as controller, when performing tasks useful for humans. The environment where robots would be acting (homes, factories, tools, objects) have been designed for human beings, and thus the human hand should be the best end-effector for robots. Another point of view would be that the human hand design may be far from the final stage of evolution, and thus may not be optimal. Building robotic hands enables to look for a new design of hand different from the human hand (for example, with two thumbs and four fingers, more phalanxes, ...). Such hands might prove more efficient than the human hand. We consider that the human hand has proven its abilities and our work concerns human-like robotic hands. Reaching human-like manipulation capabilities would be an outstanding progress in robotics and would bring important benefits to society. The use of a human-like robotic hand enables us to use experience from humans and observation of human in-hand manipulation which is a great help.

It is interesting to observe that some humans have different hands, either from birth (with 6 fingers), or after losing a phalanx or a finger. These humans are still able to perform outstanding dexterous manipulation, from a robotic point of view. Even after loosing both hands, a human being is able to use its feet to perform better manipulation than any existing robot. Even if a human foot has better capabilities than any robotic hand, maybe not in motor skills, but in sensing abilities, this fact highlights that the key to in-hand manipulation is not the hardware (the device) but the control system, that is in humans, the brain and nervous system, provided that the hardware presents enough motor and sensing abilities. Human manipulation control relies on a learning process, and so does our work.

1.3 Learning

The human control system uses learning. During childhood, a human being learns and improves its sensorimotor skills. This learning process includes in-hand manipulation and continues the whole life, especially when encountering a new tool or a new activity. Despite its very high efficiency exceeding any robot control, human brain needs to spend time learning to perform in-hand manipulation. Using a similar learning strategy by having a robot able to improve its manipulation skills by learning from experience is a great inspiration in this work.

A robot can learn while performing a task, optimizing its movements and its knowledge about itself and the environment. Once the task achievement is efficient, it is possible to transmit the knowledge of the robot (to teach) instantaneously to other robots (that learn), as it only consists in copying data. Minor tuning would be

then needed for similar robots, as small differences always remain (friction, weight, precision in size of mechanical parts). A heavier tuning would be required for robots that are different but some knowledge would still be valid, when addressing environment or general concepts of the task. For humans, training is also a long and heavy process but this direct transfer of knowledge is not available. Teaching makes training sometimes faster, but this teaching has to be performed entirely by each human, like in a classroom, wasting time and energy.

In this paragraph, the following example of task is considered: throwing a ball. This task can be represented in high level concepts such as: the goal of the task, the objects involved and the criteria to assess performance (distance, height of the pitch, energy consumed). The low level concepts are the trajectory of the ball and the trajectory of the joints. One robot can try repeatedly to throw the ball, and tune its movement to reach the maximum throwing distance. Then, it can give the movement to another similar robot. A robot with a different design could not use the same joint trajectories. But it can use the ball trajectory. If the first robot learns the relation between the trajectory of the ball and the throwing distance, it could give it to the robot with a different design, boosting its learning process.

Another example can be taken from the science-fiction movie *The Matrix*. Trinity, a character, needs to fly a chopper but do not know how. The skill is downloaded to her brain, enabling her to fly it instantly.

In the same manner, if a robot needs to manipulate a new object, or needs a new skill, it could download the knowledge and experience. This knowledge/data can be centralized (in a database, or in an electronic cloud accessible to multiple robots as RoboEarth [5]) to be downloaded by compatible robots, updated whenever new experiences are available and corrected when mistakes are found. One robot could change its role on a production line just by changing his program or only download the skills it needs when it needs them. A production line made of different robots can be arranged in different manners, use different tools and produce different products easily.

Such robots, gaining in intelligence, will be able to make decisions. Decisions are made on sensing (seeing, hearing) and understanding the environment and context. A decision can be quick but rough, similar to a reflex. It can also be slow and smart, like a strategy. Both in robots and humans, a trade-off lies between the time used to make the decision and the relevance of the decision concerning the goal to achieve.

Having a fast response behaviour depending on events is key for stressful, critical situations when there is "no time to think" (sports, combat). In humans, this explains the heavy training to drill procedures in case of emergency or critical situations, or to improve the reaction time of the body by acquiring reflexes. A repetitive training builds reflexes possible to rely on in stressful situations, when adrenaline boosts physical power while blocking constructive thinking, preventing the human from building a strategy.

For a robot however, strategy building would be very fast. While learning by trial and error the robots collect the action strategies that best suit to particular tasks and scenarios. When faced with the same circumstances, robots can recycle the learned plans and execute them automatically, boosting the planning speed. Once learned,

these automated motor programs can be easily stored in the memory of a computer. Robotic strategy can involve performing the task virtually to check the outcome of a behaviour (in simulation), even though this is usually very time consuming. For strategic decisions, as for reflex behaviour, learning could make robots gain more knowledge automatically to boost the decision making process in speed and quality of the result.

Sensing together with learnt knowledge could lead to a quick but strategic decision of action. A procedure given to a robot would be followed strictly without the need to think. In some cases, robots came out to be superior to human beings. It is obvious when playing Tetris, or speed chess, for example. Once trained, robots are better than humans. But this is only possible in "perfect" worlds, like a game, where everything is discrete, with known rules, easy to model and can be stored in a limited amount of memory. These performances have to be extended to a more complex world to enable robots to behave efficiently in a real environment.

To put this section in a nutshell, learning is the solution for humans to be able to tackle complex control problems. It appears to be an interesting, if not mandatory solution for complex robotic control. Benefits of learning in robotics is that the knowledge gained by learning can be transferred instantly to another robot. But any learning process involves a preliminary time spent on collecting knowledge before the robot becomes efficient and available to fulfill its role.

1.4 Our work

The usefulness of robotics in our society has been emphasized in 1.1. To build really useful robots, dexterous manipulation is a key challenge (1.2). It is a mandatory skill for robots to be integrated in human environments, acting on the world where humans live. Good manipulation abilities would give robots the versatility they lack for now and justify their certainly high cost. Using any tool a human can use requires to have a similar end-effector as a human, implying a humanoid hand.

Manipulation is not only grasping an object and moving it. Dexterous manipulation means to be able to use the object, and to move it within the hand. This particular aspect of manipulation is called in-hand manipulation. Planning manipulation considering in-hand manipulation forms the complete problem we address. When considering some tools, like a pen, a knife, a screwdriver, a spray or a remote control, it appears obvious that their use is efficient only when performing in-hand manipulation. Following these observations, this work aims at improving dexterous in-hand manipulation of a robotic hand. The existence of a robotic hand is assumed. Making the parallel with humans, we highlighted that the control system is the main challenge for manipulation in 1.2. We therefore developed a solution for the control of robotic dexterous in-hand manipulation (chapter 4).

Human control is a lot more powerful than robotic control. Conceptualization of the activity, recognition of the object, improvisation are examples of abilities where humans surpass robots by far. Even for humans, it takes several years of training before manipulation gets well controlled. It proves the complexity of manipulation.

Thus, standard robotic control strategies are doomed to fail on this problem. This observation is supported by the study of existing methods used for manipulation in 2.2.

The use of learning for our solution appears unavoidable to get a fast enough planner. It is the human way of achieving efficient manipulation. The learning phase might be long and heavy to perform, but it is a knowledge that can be stored and duplicated among robots. In our case, learning can be performed by the robot self-experience, or, using its human-like design, the robot can learn from human movement observation.

Our interest is to approach the control problem from the top, by understanding how to break down a manipulation activity. Our solution generates autonomously a plan according to this break down, which is called by roboticists high-level planning (precise definition in 2.2). Standard control algorithms can then be used once the main problem is broke down into smaller ones. This is more detailed in chapter 3. We will test our work on a real robotic manipulation platform.

This manuscript is entitled "High-level planning of dexterous manipulation using a robotic hand", which sums up what has been explained in this section and describes accurately our solution.

1.5 The HANDLE project

The research presented in this thesis has been supported by the HANDLE project, which has received funding from the European Community under the grant agreement ICT 231640. The project addresses theme 2 : Cognitive Systems, Interaction and Robotics of the EU Seventh Framework Programme.

The full title of the project is *Developmental pathway towards autonomy and dexterity in robot in-hand manipulation*. It is a large scale project involving nine partners all around Europe. It has been coordinated by the University Pierre and Marie Curie of Paris, and took place from February 2009 to February 2013 with a duration of 4 years.

Figure 1.1: HANDLE project logo.

The project aimed at observing and analysing human in-hand manipulation in order to reproduce these skills on a robot. Multiple scientific and technological aspects received contributions: perception capabilities, hardware and software, planning and control techniques or high-level reasoning about actions.

Following this guideline, the work presented here is human inspired, making us analyse human manipulation to build the control architecture. Work from the project

provides data from observation of manipulation by humans. Moreover, our work is implemented on a robot using an anthropomorphic hand. This platform (Fig. 1.2) is the demonstrator of the HANDLE project, upgraded all along the project, and receiving the work of several partners. The integration effort performed within the project was large and successful, the different modules are working altogether to perform in-hand manipulation. The work described in this manuscript takes a central role in the control framework, as detailed in 5.2. The HANDLE platform is one of the best manipulation platforms in the world right now.

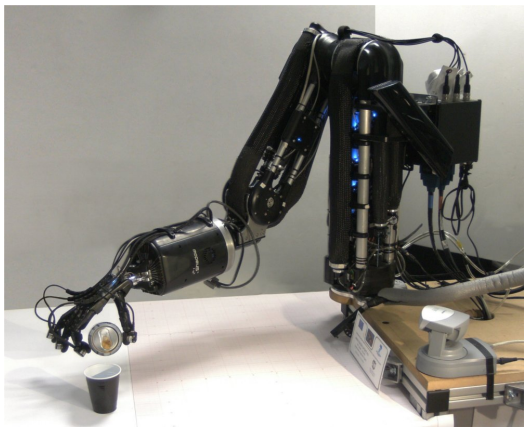


Figure 1.2: HANDLE project demonstration platform.

The two main partners involved in the work presented in this thesis are:

- University Pierre and Marie Curie (UPMC), Paris, France. The team belongs to the ISIR laboratory and is specialized in manipulation, planning and control.
- Instituto Superior Tecnico (IST), Lisbon, Portugal. The team, called Vislab, is specialized in vision, learning and manipulation.

As a result, this Ph.D. thesis has been conducted under a cooperation agreement between the two universities and co-supervised by Véronique Perdereau from UPMC and Alexandre Bernardino from IST. The student worked mainly in UPMC, Paris, leading to numerous Franco-Portuguese video conferences. He spent time in IST, Lisbon, through successive stays:

- Three months from September to December 2010.
- One month in summer 2011.
- Two months from January to march 2012.
- One month in September 2012.

1.6 This manuscript

This work proposes a solution to in-hand dexterous manipulation control using learning to tackle the problem. Chapter 2 presents the state-of-the-art of robotic in-hand manipulation planning, underlining that a high-level approach is required. Our solution uses Markov Decision Processes to encode the behaviour of our robot. This technique is introduced in section 2.3. Then, basis of machine learning are exposed to introduce how we can use it in our solution (section 2.4).

Chapter 3 justifies the choice to develop a high-level algorithm. The rough sketch of our solution is drawn. Key choices of our work are presented and justified while our strategy is explained. A precise overview of in-hand manipulation is given.

Chapter 4 consists in the development of our solution. A first model was developed and proved strongly limited, as it cannot learn. A second model using Markov Decision Processes is built, giving better possibilities.

Chapter 5 presents the results of our solution in planning and learning (section 5.1). It also describes its implementation on the experimental platform (section 5.2.2). The conclusion chapter ends the manuscript giving an insight of further work.

Chapter 2

State-of-the-art

The core of this thesis is the design of an efficient solution for dexterous in-hand manipulation planning. The main focus of this chapter is to present the state-of-the-art in this field. First, the context of manipulation is reminded in section 2.1, before focusing on in-hand manipulation planning in section 2.2. The existing solutions for planning in-hand manipulation, partially (section 2.2.2) or entirely (section 2.2.3), are explained. Our solution aims at overcoming the limitations of these methods and go beyond the state-of-the-art. The development of this solution is presented in chapter 3 and chapter 4.

This solution uses a decision making technique, the Markov Decision Process, which has to be explained. This is the aim of section 2.3. As presented in section 1.4, we intend to make our robot learn to perform manipulation. Machine learning techniques are introduced in section 2.4 to show how learning can be integrated to our solution. Specific applications of learning on manipulation are presented in section 2.4.2, while learning techniques on Markov Decision Processes are explained in section 2.4.3.

2.1 Robotic manipulation

The field of our work is robotic manipulation. In section 2.1.1, we present this problem and highlight how in-hand manipulation is integrated into it. Then, in sections 2.1.2 and 2.1.3, activities belonging to manipulation and complementary to in-hand manipulation are presented. This whole section introduces key information for in-hand manipulation planning and is a springboard for the state-of-the-art of in-hand manipulation (section 2.2).

2.1.1 Overview of manipulation

The different phases of a typical human manipulation activity are shown in Fig. 2.1. Reaching the object only requires to move the arm and hand towards the object, possibly pre-shaping the hand in prevision of the grasp. Grasping the objects starts the real manipulation activity. This phase does not involve in-hand manipulation. Once

the object is grasped, a possible regrasping of the object requires in-hand manipulation in order for the hand to be in the appropriate position to perform the given task. The object is then replaced after a possible regrasping and released.

Figure 2.1: Phases during a complete manipulation activity.

Human-like robotic manipulation follows the same outline. The solution we aim at concerns in-hand manipulation. State-of-the-art techniques for planning it are presented in section 2.2. In the following sections, existing solutions for planning grasping (2.1.2) and moving the arm (2.1.3) are explained. Such solutions do not consider in-hand manipulation and therefore do not tackle the complete problem.

The appropriate solution is likely to be a combination of different complementary techniques. For this reason, we might have to integrate these methods or similar ones into a complete solution for robotic manipulation.

2.1.2 Grasping

For some applications like when the task is only pick-and-place, the need for manipulation boils down to grasping the object. This is useful, considering the number of industrial applications (packing, sorting, ...) but limited. The problem is to make the robotic hand grasp the object, this activity is therefore called grasping. For dexterous manipulation, grasping is totally relevant, and represents a part of the problem. Planning grasping consists in finding the appropriate fingers' configurations on the object: the contact points, the forces to apply, and the movements to make the hand reach this configuration. For a complete pick-and-place task, the object is moved, placed in the desired position and eventually released. A lot of work has been done to solve this problem as presented further in this section.

Grasping is achieved by the hand, applying contact forces on the object, located at contact points or contact regions. Most of the time these forces are applied using fingertips, but they can also be applied by regions of the palm or of the phalanxes. These forces are contact forces, but resulting from friction, torques can also be generated, as well as tangential forces, and consequently, a wrench is exchanged between the hand and the object. This has been expressed in a mathematical way through grasp quality metrics.

Grasp quality metrics

A grasp quality metric is a value related to a grasp, given its parameters, which assesses how good this grasp is. Efficient planning of grasping requires a grasp metric,

to compare different grasp candidates and choose the best one, as well as to observe if a grasp has been correctly executed.

Most of the metrics derive from the metric proposed in [25], which quantifies the concept of Force Closure grasp introduced by N'Guyen in [55]. A force-closure grasp is a grasp that can resist any wrench applied to the object, given that the intensity of the wrench is under a threshold. The higher the threshold, the higher the metric. Thus, metrics are focused on stability, and are used for planning grasping in [14], [28], [7], [10], [31], [41], [43].

An illustration of the force closure concept is presented in Fig. 2.2, taken from [55]. w_1, w_2 and w_3 represent forces that can be applied to the object without friction. They are normal to the surface at the contact point. f_l, f_2 and f_3 are the directions of the forces, that have to satisfy two conditions to ensure force closure: two of the three directions f_l, f_2, f_3 are independent, and a strictly positive combination of the three directions is zero: $\alpha f_1 + \beta f_2 + \gamma f_3 = 0$.

Such a metric has been made continuous even when force closure is not satisfied, in [71].

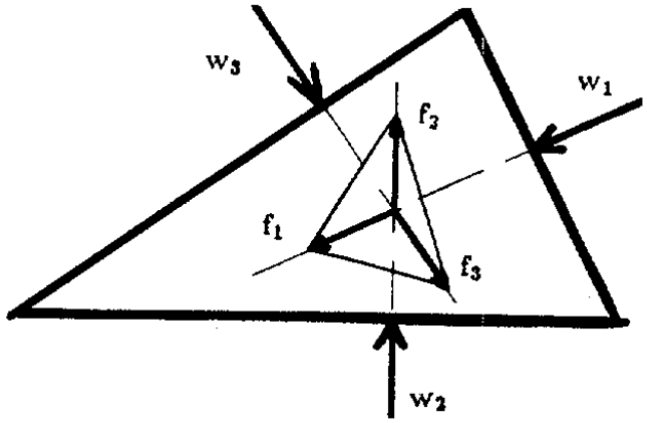


Figure 2.2: A geometrical view of force-direction closure (from [55]).

Planning grasping

Planning grasping is usually done by automatically finding grasping points for the fingers and the appropriate hand configuration (position, internal parameters...). A gripper has 2 fingers, the Barrett Hand has 3 fingers, a humanoid hand 5 fingers. From possible grasping points, a common approach consists in using an algorithm that generates a list of grasps, and the best one is chosen using a grasp metric that gives a quality measurement of the grasp.

The main problem is dimensionality, and a lot of works are using techniques to reduce the number of parameters, by simplifying the number of variables characterizing the grasp. As a lot of grasps can be generated, some works try to limit this amount. Borst et al. [10] uses arbitrary hard coded rules to quickly generate a few grasps and compare them using a grasp metric. Lozano-Perez et al. [41] checks only the force

closure as a criteria. Geidenstam, in [28], uses a 3D box approximation to compute possible grasp points. The best ones are then chosen using a quality metric. A neural network is trained to relate experimental grasp success to this quality metric.

Grasp configuration can also be searched in a lower dimension subspace using a simpler representation of grasps by finding synergies. Synergies consist in coupling command variables to reduce the dimension of the space of search. Different variables are changed "in synergy" by a single parameter. This has been observed in human manipulation in [58]. This work realized that most grasps performed by humans have similar patterns, implying that a reduced number of parameters could describe grasping, and then command it. In [27], the authors propose a very rich model for grasping. Doing so, they show that very few postural synergies can generate force closure grasps. Bicchi in [9] gives the most complete work on synergies for grasping. It reviews different techniques using synergies for grasp generation or describing the hand state. It also introduces force distribution control by the hand using synergies.

The use of synergies has also been extended to in-hand manipulation planning, as explained in 2.2.2.

Such reduced set of parameters is also used in [14], allowing to consider hand posture subspaces. Synergies are defined using the concept of eigengrasp, which is a fundamental grasp type. Each eigengrasp describes a range of configurations using a single parameter. Two eigengrasps are chosen, creating a two dimensions subspace in which the search is made. Then, the grasp is optimized by iteration, using a quality metric. The definition of eigengrasps is related to the works on lists of grasp types, called taxonomies, presented in 3.5.2. These taxonomies result from works observing human most used grasps.

In [14], the authors use a two-step strategy like in [7] and [31]. A first rough grasp is calculated and refined or optimized in a second step. Berenson [7] generates a low dimension preshape considering a cost function that is fast to compute. Harada [31] uses two convex (polygon) models for the object and for the hand grasping capability. This gives a "grasp style", a rough grasp description. The grasping posture is then precisely computed by randomly sampling the whole configuration space and selecting the grasp using a force closure criterion. Other "two-step" works reduce the search space for grasps in the same way. The rough grasp is chosen among a few known grasp types of a taxonomy. Grasp types from Cutkosky's taxonomy [19], in [43], are associated to shapes of objects. The object detection gives the grasp type. This grasp is tried in several configurations, and the best one is chosen using a feasibility test and a grasp metric.

The object is detected using vision, or known and located using vision [41]. The model is sometimes simplified to reduce dimensionality. Miller [43] uses a predefined simple shape set, and N'Guyen [56] models the object as polyhedrons. Saxena [60], on the other hand, does not model the object. Grasping points are learnt directly from two images of the object (or more). By applying filters on the image, a probabilistic model is fed and afterwards generates quite efficient grasping points to the robot.

Computing a grasp is made complex by the high dimensionality of the hand configuration space, and the high cost of validating a candidate grasp by collision-checking and testing for force-closure or grasp metric. Computationally, all stages are heavy.

The generation of the high-dimension hand posture, the collision checking, and the grasp quality metric calculation are all time consuming. All these works only consider the grasping problem, just picking up, moving and placing the object. It is not complex in-hand manipulation.

2.1.3 Moving the arm

The arm has to be moved at nearly every stage of manipulation, while approaching the hand of the object, lifting it, placing it, and possibly during in-hand manipulation. Moving the arm is a control problem that has been receiving continuous contributions for decades now. Industrial robotic arms are widely used for various tasks. The same techniques are used to control the movements of an arm for dexterous manipulation. These techniques also apply to the control of a single finger of the hand, which can be considered as a standard manipulator. This is explained with more details in section 2.2.2.

A robotic arm is made of a number of successive elements called links, considered as solids. These links have joints between each of them. These joints usually have one rotational degree-of-freedom. The configuration of the robot can be formulated by the list of all its joint angles (joint space), or by the position of its end-effector in the environment (Cartesian space).

Planning the target configuration of the arm

The problem is simply formulated. The arm lies in an initial position and its end-effector (the last link, usually bearing the tool/hand) has to reach a known target in position and orientation. When the environment presents obstacles, the end-effector target position has to be reached while avoiding contact between any links of the arm and the obstacle. The target position of the arm is known in the Cartesian (operational) space. First, the corresponding joint angles (articular space) must be computed. Switching from the joint angles to the Cartesian position of the end-effector is called *Forward Kinematics*. The standard model used for the forward kinematics is the Denavit-Hartenberg model. It is used to build a matrix that, multiplying joint angles, gives the Cartesian position.

The other way around, *Inverse Kinematics*, is what we need. Joints angles have to be computed from the desired end-effector position. It is much more complex due to singularities (no solutions) or ambiguities (multiple solutions). The first step is to build the forward kinematics. The formulation of the Denavit-Hartenberg matrix model is differentiated, giving the Jacobian Matrix. Then, by inverting this matrix, we obtain an equation giving the speed of the joints from the speed of the Cartesian variables. The key problem lies in the inversion of the Jacobian Matrix, which may not be invertible. Pseudo-inverse or transpose can be used, each method having its drawbacks. No efficient overall solution has been developed, and this subject is still receiving contributions from the robotic researchers. Details of all this procedure can be found in reference books [62] or [17].

This is a local technique. From the initial configuration, sequential movements are calculated using the local linearization obtained by the Jacobian matrix. Other non analytical local methods can avoid using *Inverse Kinematics*. At each time step, a set of small movements is executed, believed to be getting the system closer to the final configuration. Iteratively, the system comes close and finally reaches the desired state, knowing the joint angles. To calculate the set of small movements, the technique of potential fields gives good results. This can be simulated and smoothed offline to return a final configuration. [73] proposes planning the movement of a manipulating arm using potential fields. It uses an attractive well to drive the end effector to a desired position. Obstacles are avoided using a superquadric potential repulsive field (Fig. 2.3).

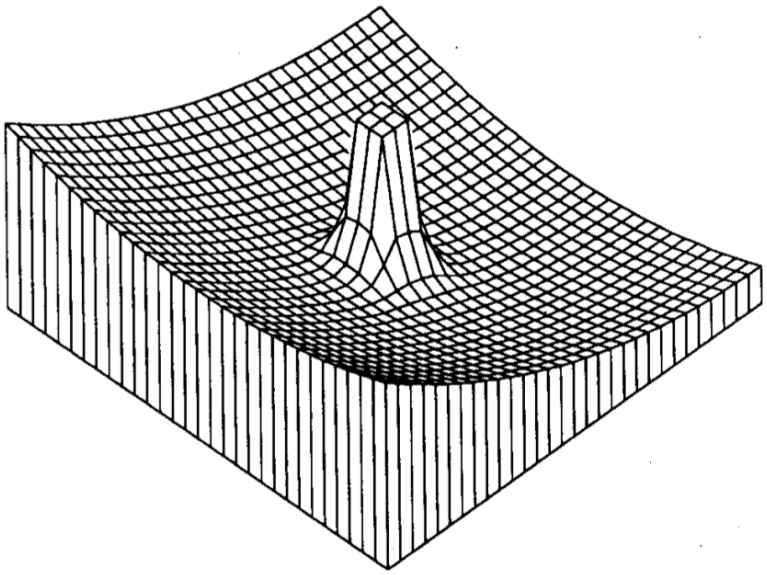


Figure 2.3: Repulsive potential added to an attractive well (from [73]).

Planning the trajectories of the arm

When the target configuration is planned using inverse kinematics, trajectories have to be generated. Once the final position of the arm is known, planning a trajectory is finding a continuous path of positions from the initial position to the final position. Trajectories can be generated reasoning in the velocities space. The trajectory has to be achievable by the robot given its abilities in terms of speeds, forces, torques and then accelerations.

Local techniques can be used to avoid trajectories generation. In this case, the robot can perform the movements calculated at each time step and then build its own trajectory along the movement. However, the resulting trajectory cannot be optimized regarding speed and acceleration, the movement is slow and sometimes jerky, as it is generated step-by-step.

Different techniques can be used to generate trajectories. This planning problem has already received a fair amount of contributions. The purpose of the section is not to give an exhaustive list of these techniques. The simplest one is standard interpolation between the initial position and the final position for each joint. The time used for the movement is sampled and each joint moves by the same fraction of the whole movement on each time step. Improvements can be required for the starting and stopping phases of the movement, when the acceleration peak is higher. Smaller speed variation can smooth the acceleration and avoid overload on the motors.

Trajectories can be generated using functions, or primitives (such as dynamic motion primitives [38]), with parameters that can be tuned to optimize different variables among which:

- Acceleration to lower the load on the actuators.
- Energy to lower the power consumption of the robot.
- Jerk (derivative of the acceleration with respect to time) to lower the shocks.

Then the trajectories have to be followed by the robot. This is a problem called control.

Control of the arm movement

Once the target position is known and the trajectories are generated, they are sent as set points to the control algorithm, which uses the actuators to make the joints reach the desired positions. This problem is the field of the control theory.

A common solution is a closed-loop servoing with PID controllers (proportionnal integral derivative) appropriately tuned. These controllers are widely known for controlling a variable y and make it stick to a reference value c. The block diagram of a standard PID controller is shown in Fig. 2.4. First, the controller computes the error e, which is the difference between the current value of the variable and the reference value. Three parameters: K_P, K_I and K_D are used and have to be tuned depending on the system. The error is multiplied by K_P (proportionnal), the integral of the error with respect to time is multiplied by K_I (integral) and the derivative of the error with respect to time is multiplied by K_D (derivative). These three products are summed and given as command v to the system.

Our work focuses on planning and not on control. Consequently, no further details are given on control techniques.

2.1.4 Conclusion about robotic manipulation

This section gives an overview of manipulation as a succession of different phases. Complementary phases not involving in-hand manipulation are explained along with existing solutions for them. This allows to circumscribe the use of in-hand manipulation planning.

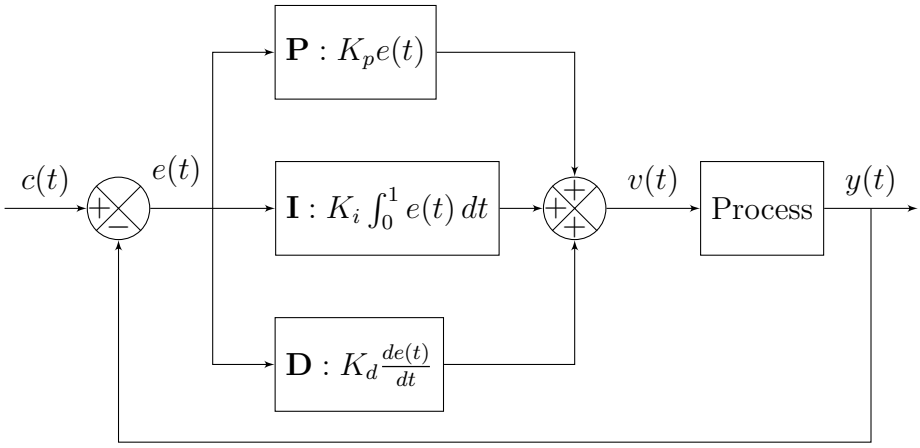


Figure 2.4: Block diagram of a PID controller.

Moving the arm is a problem that composes the preliminary phase of manipulation: reaching the object. It is also required to move the object once it has been grasped. Such techniques have to be combined to our planning solution.

Grasping the object is also a preliminary phase of manipulation. The concept of grasp and stability of a grasp are key for in-hand manipulation. We use it when building our solution in chapter 3 and 4.

We intend to build a solution for in-hand manipulation planning. The next part presents the state-of-the-art for this problem.

2.2 In-hand manipulation planning

The work presented here concerns the planning of movements to make a multi-fingered robotic hand execute in-hand manipulation of an object. In-hand manipulation planning consists in finding trajectories of the hand joints from an initial configuration of the whole system (hand, object and environement) to a final configuration. Movements of the arm can also be considered, but are relevant only for specific cases. In-hand movements are required for using objects efficiently either to perform specific actions with the object or to regrasp an object because of a different initial pose or a problem of accessibility. In the previous part (2.1), an overview of robotic manipulation introduces the core state-of-the-art of existing techniques to solve our problem: in-hand dexterous manipulation planning. This is the topic of this part.

The problem of in-hand manipulation is presented from a general point of view in section 2.2.1. The distinction is made between low-level planning that deals with small movements and elementary actions of manipulation, and high-level planning that approaches the problem from the top and tries to find the appropriate combination of low-level actions to reach the final configuration. Existing solutions for low-level planning are presented in 2.2.2 and those for high-level planning in 2.2.3.

2.2.1 Overview of in-hand manipulation

Theoretically, planning in-hand manipulation forms a high dimension problem. The arm and the hand are directly controlled. Using the example of a humanoid hand: the Shadow hand, the system forms a set of 30 degrees-of-freedom: 24 for the hand and 6 for the arm (a humanoid arm). The object is moved, changing its pose in the hand, adding 6 parameters to the problem when the object is considered rigid. Performing in-hand manipulation is formulated as reaching a target configuration from an initial configuration. Configurations can be related to the hand, the object, or both. Planning in-hand manipulation consists in the generation of every movement of the hand to make the hand and object system reach the final configuration.

When performing manipulation, different movements of the fingers can be made: sliding, rolling, finger repositioning (called finger gaiting), etc. It implies that the system encounters phenomenons like dry or viscous friction, stick/slip friction, contact making or contact losing, which are complex, highly non linear or even discrete. These actions are listed in [23] and shown in Fig. 2.5. Each action has an associate group of fingers involved (Th: thumb, In: first finger, D3: middle finger, D4: ring finger, D5: little finger). The original caption of the figure is : *Digits employed in specific movement patterns and their functional groupings. Each box represents one or more digits; if more than one they move together in the same way. Linking boxes by simple lines indicates reciprocal action between groups defined by boxes: linking by arrows represents sequential action.* We choose to name these elementary in-hand actions: atomic actions. It avoids confusion in the rest of this manuscript.

A more recent work [11] proposed a different classification of atomic actions. It considers sliding and rolling but not finger gaiting. This list is shown in Fig. 2.6. This representation is based on the movement of the object implied by the action. In-hand actions are listed by rotations and translations along the different hand coordinate axes. This approach appears more complete than in [23], where atomic actions are listed only from observation.

If in-hand manipulation is decomposed in that way, then planning consists in finding a combination among the great variety of possible atomic actions to make the fingers cooperatively execute the movements that better suit the object characteristics (size, shape) and the chosen task to execute. The details of each atomic action (finger movements, amplitudes, forces) also have to be planned.

In the following state-of-the-art of in-hand manipulation planning, we make the distinction between low-level and high-level planning. Low-level planning regroups the techniques for planning each of the atomic in-hand actions. High-level planning, on the other hand, considers the whole problem. Its purpose is to find the combination of atomic actions, or at least to decompose the problem into a sequence of configurations. These configurations should be sufficiently close to enable the low-level techniques to plan the transfer from one to another.

In-hand manipulation movements can be teleoperated. In this solution, the hand is remotely controlled by a human operator through a physical interface (ex : Cyberglove [3] shown in Fig. 2.7). The system is not autonomous, as all the planning is performed by the operator. Another solution is replaying movements that have

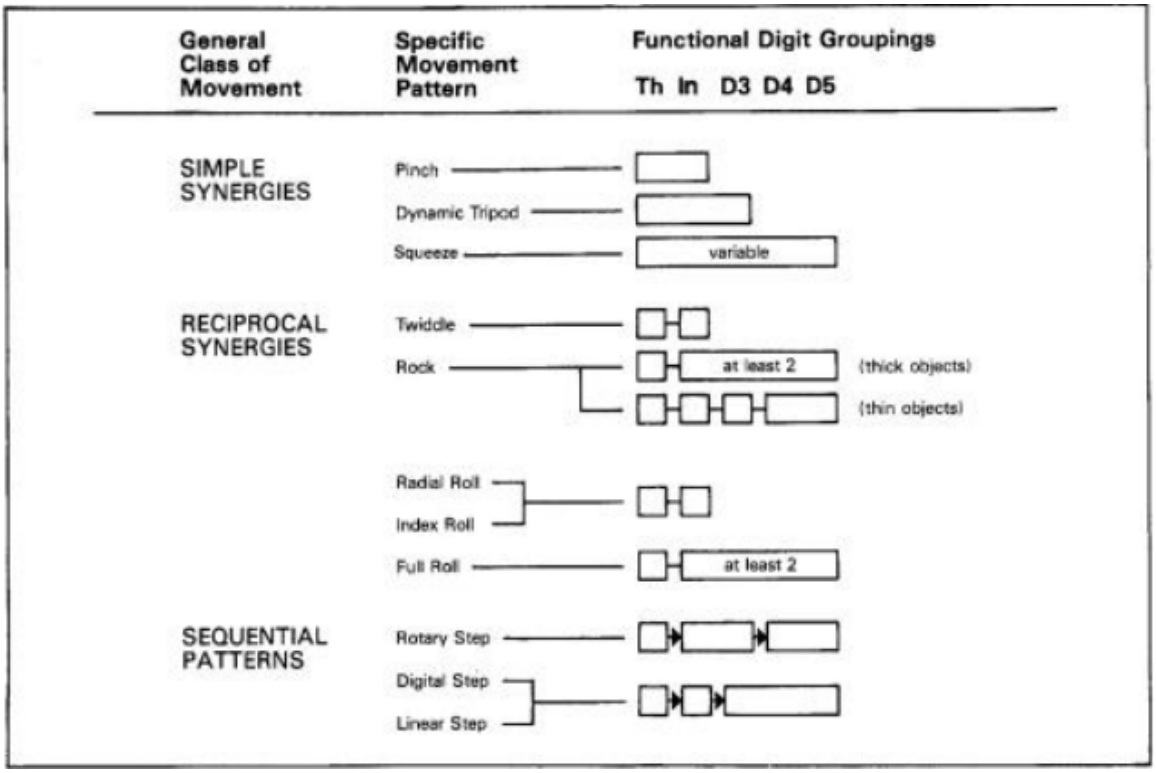


Figure 2.5: List of possible in-hand actions (from J.M. Elliott and K.J. Connolly in [23]).

been pre-programmed using an analogical or digital interface (joystick, computer). The resulting movements are not human-like movements, and even if the system is autonomous, its versatility is very limited as it cannot adapt to new contexts (object, task, environment).

A useful robotic hand should autonomously decide what to do with a given object, provided the high-level objective (task) is known, and execute human-like movements, while adapting to the world context in real time. This is what is aimed at in this work.

2.2.2 Low-level planning of in-hand manipulation

In this section, we review the main techniques for planning in-hand manipulation atomic actions (as defined in section 2.2.1). This planning is done considering directly what can be called the low-level variables: joint angles and joint torques. Low-level algorithms are sometimes called "local planners", and are used to plan and execute the fine movements of the hands following a path given by the high-level planner, sometimes called the "global planner".

Low-level planning methods usually apply to a single atomic action listed in [23] or [11]. In [40], the author groups together in-hand actions in four categories : coordinated manipulation, rolling motion, sliding motion and finger relocation.

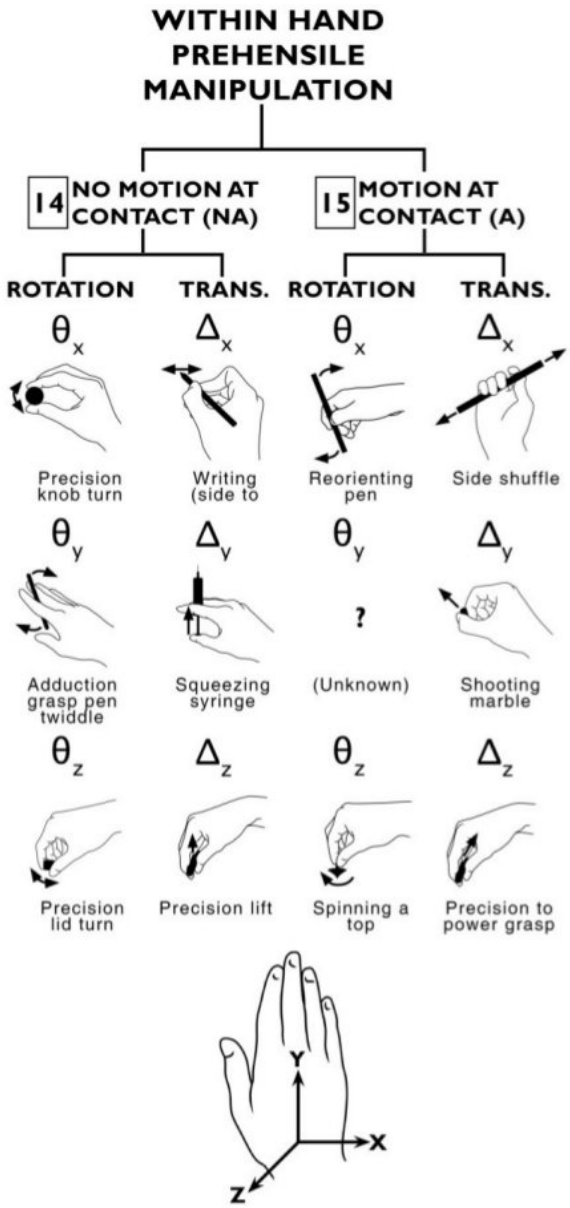


Figure 2.6: List of possible in-hand movements (from [11]).

First, the atomic action of finger gaiting (finger relocation) is presented. Then, explanations on whole hand actions (rolling and sliding) are given. Afterwards, the use of synergies for in-hand actions is detailed before a short presentation about control.

Moving a finger: finger gaiting

Finger relocation, commonly known as finger gaiting, might be the simplest atomic action to perform for a robotic hand. We start by considering this case to introduce standard low-level planning techniques.

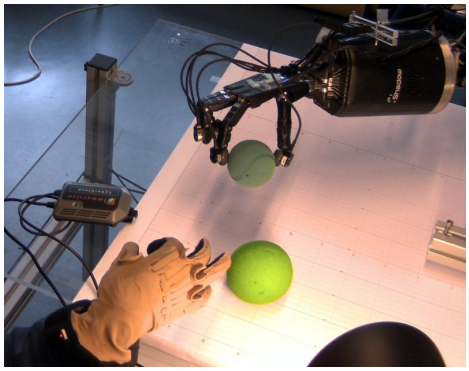


Figure 2.7: The Cyberglove, an interesting device to teleoperate in-hand manipulation movements. On this picture, it is used to control a Shadow electric hand.

The problem boils down to moving a single standard manipulator : the finger. A humanoid finger is an under-actuated manipulator with 4 degrees-of-freedom (dofs), 2 of them being coupled. Techniques are well known for moving a manipulator and have been presented in section 2.1.3.

The target position of the fingertip is known in the Cartesian (operational) space. First, the corresponding joint angles (articulation space) are computed using *Inverse Kinematics* as described in 2.1.3. Once the final configuration is known, trajectories have to be generated. This part is detailed further in this section, along with the control solutions for following trajectories.

It is also possible to use local techniques, as potential fields, like Volpe did in [73]. These methods, however, do not guarantee to reach the final configuration as they can be trapped in deadlock positions. This explains why they have not been used much by scientists in recent manipulation planning works.

Whole hand actions: rolling and sliding

Whole hand actions involve parallel movements of multiple fingers, implying more complex methods.

In [52], the theory of manipulation of a single serial manipulator is extended to hands with multiple fingers (multiple manipulators). A mathematic formulation is set up to allow deterministic control of manipulation. In [40], the authors follow the same strategy, and provide an extensive formulation of the dexterous manipulation problem for a robot hand, directly on the complete configuration space. The complexity of the resulting algorithms illustrate how hard a complete analytical planning is.

When planning the whole hand manipulation movements, usual solutions generate joint trajectories [15], [12]. The desired configuration can be formulated as the whole hand-object state, or only the object position. In this case, a second planner has to generate the contact points and the position of the fingers, either only at the final configuration, or directly along the whole movement [15]. Trajectories can be either discrete or continuous, at this point. Then, control techniques are used to make the

robot's joints follow the trajectory. Using tactile sensing, the fingers movements can stop when contacting the object.

[12] proposes a local planner able to generate pure rolling and pure sliding motions. The algorithm checks if the next hand-plus-object configuration is reachable through dexterous movements. The motions are divided in time steps. Movements are directly generated using velocities. A random search of the relative control space is performed for each contact, randomly generating relative velocities for both a pure rolling motion and a pure sliding motion. Velocities that reach the subgoal are selected for execution. The control of an ellipsoidal object with 3 fingers requires more than a minute to compute the solutions (with a state-of-the-art computer at this time). That work does not consider finger gaiting. The whole chart of the algorithm is shown in Fig. 2.8. The local planner is drawn inside the larger rectangle. Random velocity controls are iteratively generated. The resulting grasp and object configuration are computed. If both are valid, the motions of the fingers are validated to be executed.

The work published in [15] proposes a state-of-the-art local planner for rolling movements. It is a geometric planner, as it generates a trajectory for the object, and then calculates the subsequent finger positions at each time step, using a local planner. From the initial position, the positions of the contact points are inferred from the previous contact points, considering possible rolling. The joint angles are then found using the inverse kinematics model. This process is repeated every time step until the target is reached. The fingers and the object are modelled by triangle meshes, which are commonly used to model any shape. This model is consequently more versatile than the usual spherical models or parametrized shapes used to model the object and the fingertips. The precision of the mesh may vary, allowing a trade-off between accuracy and speed of calculation.

In [79] and [80], a local planner is used to check the ability of the system to link successive intermediate configurations. Like in [12], both rolling and sliding are considered. The inverse dynamic formulation is used and solved using Montana's equations [46] (formulating the torques or forces given the desired velocities of the joints). This technique does not consider finger gaiting and collisions, and yet, because of inverse dynamics complexity, is less efficient.

These solutions usually need to generate a trajectory between two configurations, which is detailed further in this section.

Synergies

Controlling the hand using synergies to lower the dimension of the problem has been presented in section 2.1.2 about grasping planning. This concept has been extended to in-hand manipulation planning.

Work in [26] is inspired by the analysis of human actuation of the hand. It shows that single muscles actuate a set of articulations of the hand. It means that humans use synergies to control the hand. These synergies are implied by the hardware, that is, how the muscles and tendons are built. In [26], the author proposes a synergy model of a robotic hand based on synergies observed in humans. Using this model,

an additional layer of synergies between the defined parameters is added, enabling the system to perform human-like grasps.

Trajectory generation

The generation of trajectories (for the object, as well as for the fingers), is done using the same solutions as for a manipulator arm (section 2.1.3). Multiple techniques can be used. The simplest one is standard interpolation. Dynamic motion primitives (DMP) is an elegant solution to generate smooth trajectories with very few parameters. In [38], DMPs are used to perform hand robotic writing.

Control

Once the trajectories are known, their values are sent as set points to the control algorithm, having the actuators make the joints reach the desired position. A common solution is a closed-loop servoing with PID controllers appropriately tuned. These techniques belong to the control theory and have been presented in section 2.1.3.

When performing in-hand actions, the hand must be controlled to ensure stability of the grasp. Control algorithms have to be running in the background, whether the robot is performing finger gaiting only, or is rolling the object at the fingertips. The object is maintained by contact of the fingers through friction. Friction is induced by the forces applied at contact points. These forces have to work against each other so that their sum is null, and the object is kept still. The stability of the grasp, its resistance to perturbations, depends on these forces. Force control is then necessary.

Conclusion

All the techniques mentioned in this section are key to solving in-hand manipulation. But they were designed for efficiently planning and executing low-level movements only, not planning a complete in-hand manipulation activity. No technique addresses every type of atomic actions. Planning a sequence of different atomic actions, or choosing among different atomic action types is needed, answering questions like: how many fingers should be used? Do fingers need to be relocated? To which position? Should some fingers not be used? Sliding or rolling motion?

These solutions are incomplete. To solve this problem, one can top these methods with a high-level planning, able to choose the appropriate sequence of atomic in-hand actions. Some methods are presented in the next section.

2.2.3 High-level planning of in-hand manipulation

High-level planning takes the whole in-hand manipulation problem into account. It does not focus on a few atomic in-hand actions. From an initial configuration, the appropriate sequence of atomic actions has to be generated to reach the target configuration. In this section, we will explore existing techniques aiming at this goal.

Solutions for planning high-level in-hand manipulation are divided into three subsections. The first one, 2.2.3, presents hybrid techniques that combines planning

discrete and continuous aspects of the problem. The second part, 2.2.3, addresses graph search techniques that need to be combined with low-level planning algorithms. The third section, 2.2.3, regroups probabilistic techniques that try to plan both the low-level and high-level concept at the same time.

Hybrid framework

The hand-plus-object manipulation planning is represented as a hybrid system combining continuous as well as discrete aspects. One existing hybrid representation is a stratified system. In [29] [74] [32], the whole configuration space is decomposed into strata, each one containing only configurations where specific fingers are in contact. The main difficulty is to determine links between strata that are necessary to resolve any manipulation problem. Another hybrid representation is an hybrid automaton [75] that divides a manipulation task into a sequence of sub-manipulations described by continuous variables and separated by discrete transitions. In all these works the contact positions are considered as fixed, which allow neither rolling nor sliding motions at fingertips.

Some hybrid solutions use two layers of planning: a global planner is in charge of discrete aspects, like the type of action to choose, or the sequence of intermediate configurations of the object. Then, a local planner is used to compute the precise movements of the fingers.

Work presented in [32] addresses finger relocation planning. Even if rolling and sliding are not considered, relocation of fingers involves choosing different discrete actions. The author proposes a stratified system that can be considered as the combination of a global and a local planner. Formulations from [29] are used, but symbolic computations are replaced by numerical computations. A desired trajectory of the object is divided into subsegments. At each time step, the contact points are inferred from the initial configuration. The planner checks if the fingers stay in their workspace along the trajectory, and relocate them when a boundary is reached.

In [76] and [77], a global planner considers discrete choices of sub-manipulations, using a hybrid automaton. Sub-manipulations correspond to continuous movements, and are allowed a time interval. Sub-manipulations can describe different actions: manipulation (object is moved by the fingers) or finger gaiting. The local planner executes movements using velocities as variables, and a switching controller can change the manipulation mode (manipulation or finger gaiting). In [75], the method is upgraded, replacing the switching controller with a probabilistic planner.

In most of the works on this subject, in-hand manipulation tasks are decomposed in a two-level hierarchy: the high-level planner and the low-level planner. The low-level focuses on the continuous control of the manipulation physical aspects (finger motions, contacts, forces) in order to achieve desired hand-object configurations. The high-level decomposes a manipulation task into discrete primitive actions, and focuses on the rules for composing these actions to execute elaborate tasks. Due to the complexity of in-hand manipulation, it is essential to choose a suitable set of primitive actions. A too abstract set would demand complex low-level techniques, whereas very

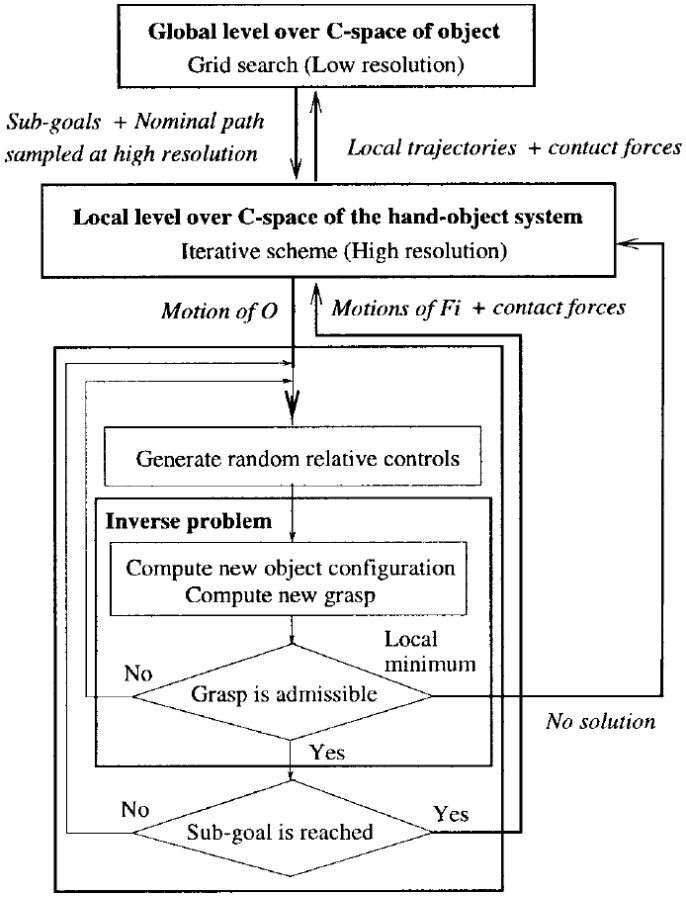


Figure 2.8: A hybrid framework from [12] : an overview of the two-level planning algorithm for the reconfiguration problem.

elementary, and then numerous, primitive actions would increase the complexity of the high-level.

Graph search techniques

Dominant approaches to high-level in-hand manipulation use graph search techniques: a graph is first built and a path is searched in the graph to link initial and final configurations. The graph can represent the feasible transitions between either hand states (finger configurations) or object states [30] [70], [81]. Both require an additional low-level planner to check if a transition can be performed according to the constraints imposed by the object and by the fingers, respectively. The graph is constructed in a preliminary phase, and contains the nodes relevant to a specific task, which severely limits its versatility and adaptability.

[81] presents a framework for planning dexterous manipulation. It uses the concept of canonical grasps (CG), that are, for a given hand and object, all the possible grasps. It defines all the CGs by linking topological features of the hand and object. The Grasp Transfer Graph (GTG) is built. Using this graph, high-level manipula-

tion means finding a path in this graph. No corresponding low-level solutions are presented.

In [12] is presented a solution combining a local planner and a global planner for the resolution of the dexterous manipulation problem (reused in [70]). The global planner, or high-level planner, uses a graph search technique. The configuration space is decomposed and an A* algorithm is used to find a path between the initial and final configurations in the obtained graph.

Probabilistic methods

In another approach for in-hand manipulation planning, a path among the whole set of control parameters is built to reach a final configuration. Probabilistic techniques are used to plan in-hand manipulation from both high-level and low-level aspects. Probabilistic path planning methods are applied to the hand considered as a whole system, randomly sampling hand-plus-object configurations to build a tree-like graph in a high-dimensional configuration space. Proposed solutions are all variants of the classical PRM or RRT methods [39] with some modifications aimed at reducing the search space when determining the path [59], [78], [75].

A resulting movement generated in simulation is shown in Fig. 2.9 (from [59]). The considered hand has four fingers with conical fingertips implying that contacts are points. Sliding is not considered. The system can either move fingers in contact with the object (rolling) or relocate a finger. The initial configuration is shown in the first frame, and the final configuration is reached in the last frame. A sequence of in-hand movements and finger gaiting actions is generated and effectively reaches the final configuration. A transition from frame 2 to frame 4 looks feasible, in that case frame 3 can be identified as a useless intermediate configuration.

These methods are very time consuming. They are very dependent on the accuracy of the object model and on the knowledge of the environment. The probabilistic nature of the solution generates useless movements, the system seems to be babbling. The generated hand motion does not look natural, and is far from being optimal. But for now, this work is the closest to a complete planning and execution of in-hand manipulation.

2.2.4 Conclusion about in-hand manipulation planning

Along this presentation of state-of-the-art solutions for in-hand manipulation planning, both low-level and high-level planning techniques have been presented. The concept of atomic actions, which are the elementary in-hand actions that can be performed by the hand has been introduced.

It appears that the low-level techniques provide satisfying results for specific limited range movements. Different planning algorithms exist for the different atomic actions. Some are well known and efficient. Others are still at an early stage of development but are already usable.

Such conclusion does not apply to high-level planning. It is noticeable that this problem did not receive a lot of contribution. Among the presented solutions, ran-

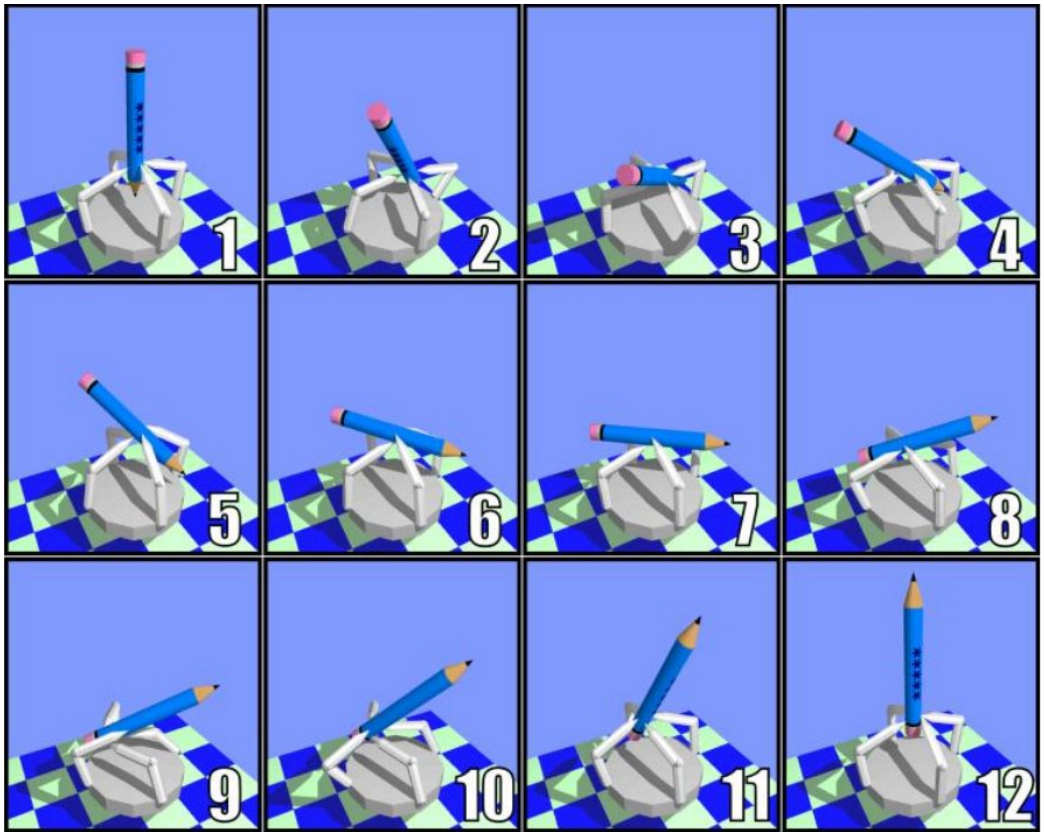


Figure 2.9: Planning of in-hand manipulation using probabilistic path planning (from [59]). Regrasping sequence of a pencil.

dom sampling techniques are doomed to be too time consuming because of the large dimension of the system. Other techniques did not get any implementation on a real robot. Hybrid techniques combine different complementary solutions for high-level planning and low-level planning. They never consider the complete list of atomic actions. Graph search techniques only consider high-level planning and model the evolution of the system as transitions between discrete hand and/or object states. In particular, the problems of selecting the best initial grasp to accomplish a task, and learning dexterous in-hand manipulation from humans and self-exploration have been mostly unaddressed in the literature.

This presentation of the state-of-the-art solutions justifies our work on a high-level planning algorithm for robotic in-hand manipulation. We aim at considering every atomic action and plan the appropriate combination to reach the final configuration. We also aim at learning as much as possible the strategies employed by humans in performing in-hand manipulation tasks. Our solution will follow the principle of graph search techniques using specific decision algorithm (chapter 4).

2.3 Markov Decision Processes

The aim of the present work is to plan high-level dexterous manipulation actions. A choice among a list of actions has to be done autonomously to reach a final configuration. As we have seen before, there are several techniques able to tackle this problem. However, in real scenarios, the outcome of actions is not guaranteed. This leads us to use a probabilistic technique to model manipulation and make decisions. A Markov Decision Process (MDP) is a way to model decision processes under uncertainty. It is an interesting tool that fits our need, as it will appear in this section. MDP are a specific case of Bayesian Networks (BN), which model processes of uncertain outcome. This part describes these tools, starting with BNs, and then MDPs. But first, a quick review of the notations we use in this manuscript is given.

2.3.1 Notations

Every equation in this manuscript follows the rules given in this section.

- Capital letters are used for variables. We consider a variable V.
- Small letters represent instantiations of variables. V can either be $V = v_1$ or $V = v_2$.
- Sets of variables or values are noted by calligraphic letters. A discrete variable V has a set of possible values: $\mathcal{V} = \{v_1, v_2, v_3, v_4\}$.

This is specifically important when writing conditional probabilities. We consider a discrete variable S_t (state at time t) depending on another discrete variable A_t (action at time t). $\mathcal{S} = \{s_1, s_2, s_3\}$ and $\mathcal{A} = \{a_1, a_2, a_3\}$ are the set of possible values for S_t and A_t, respectively.

- $P(S_t|A_t)$ gives the probabilities for S_t to be equal to s_1,s_2 or s_3 for each value of A_t: a_1, a_2 and a_3. It is a table of 9 probability values.
- $P(S_t|a_2)$ gives the probabilities for S_t to be equal to s_1,s_2 or s_3 when $A_t = a_2$. It is a table of 3 probability values.
- $P(S_t = s_1|a_3)$ gives the probability for S_t to be equal to s_1 when $A_t = a_3$. It is a single probability value: $P(S_t = s_1|a_3) = 0.6$ for example.
- $P(S_t = s_1|A_t)$ gives the probability for S_t to be equal to s_1 for each value of A_t: a_1, a_2 and a_3. It is a table of 3 probability values.

2.3.2 Bayesian Networks

Overview

The name Bayesian Network comes from Thomas Bayes (18th century), who formulated the main bases of Bayesian statistics. BN themselves have first been mentioned

in [57]. A BN is a set of variables, along with probabilistic dependencies between these variables. Here are more details about variables and dependencies.

- Variables : Let $\mathcal{B}$ be the set of variables. A variable B_i belongs to $\mathcal{B}$, and has a value included in $\mathcal{B}_i$. Variables can be discrete or continuous.

- Dependency : A variable B_i can either depend on one or more parent nodes. Let $\mathcal{P}a_i$ be the set of parents of B_i: $\mathcal{P}a_i = \{\text{Pa}_{ij}\}$, $j \in \{1, ..., n\}$. B_i can be independent of any other variables, in that case : $\text{Pa}_i = \emptyset$. The value of B_i depends on the value of each parent Pa_{ij}. Dependency between discrete variables are usually represented as a multinomial distribution. For each combination of values of the parents, B_i has a probability distribution over $\mathcal{B}_i$, that sums to 1: $\sum_{b_i \in \mathcal{B}_i} P(b_i|\text{Pa}_i) = 1$.

These relationships can be depicted in a graph describing dependencies among the set of variables. It is a directed acyclic graph:

- A graph: made of nodes and edges. Nodes are variables, and edges are dependencies.

- Directed: edges are directed, which means dependencies are only one way.

- Acyclic: means that the graph does not have any loop. From any node, when following the directed edges, it is impossible to reach this node again.

To explain more clearly what a BN is, we provide here an example: the sunstroke. Let us consider that a headache comes from a sunstroke and only depends on the sunshine and the hat worn by the subject. The associated graph is shown in Fig. 2.10. This process has three discrete variables:

- *Sunshine*: can be true or false. The weather is sunny or is not.

- *Hat*: this variable is the hat worn by the subject. It can either be a cap, a sombrero or no hat at all.

- *Headache*: can be true or false, the subject has a headache or not.

Each of the dependencies shown as edges in the graph is probabilistic. The process is uncertain. The probability table of each variable depends on its parents:

- *Sunshine* is independent assuming a random day of the year, it does not have any parent. Its probability distribution is simply the fraction of sunny over cloudy days of the year, in average.

Sunshine	
True	False
0.7	0.3

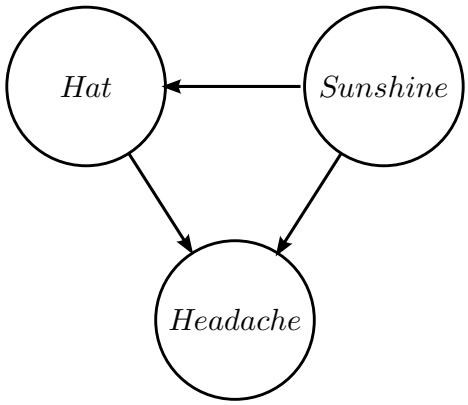


Figure 2.10: Graphical representation using a BN of an imaginary process of sunstroke. Dependencies between the sunshine, the hat worn and the headache are represented.

- *Hat* depends on the sunshine, as the subject is more likely to wear a hat when it is sunny. An hypothetical hat probability distribution is:

Sunshine	*Hat*		
	No hat	Cap	Sombrero
True	0.2	0.5	0.3
False	0.7	0.2	0.1

- *Headache*: has two parents. It depends on the hat worn, and on the sunshine, resulting in this probability distribution, as an example:

Sunshine	*Hat*	*Headache*	
		True	False
True	No hat	0.8	0.2
True	Cap	0.6	0.4
True	Sombrero	0.3	0.7
False	No hat	0.1	0.9
False	Cap	0.2	0.8
False	Sombrero	0.05	0.95

In a BN, variables can also be continuous, either all of them, or mixed with discrete variables (Fig. 2.11). In our example, we could consider the sunshine as a continuous variable corresponding to the solar irradiation, or any other index, normalized between 0 and 1: $Sunshine \in [0, 1]$. For a fixed value of Hat, the probability of $Headache$ could be defined as a continuous function of $Sunshine$:

As an all-continuous variable problem BN, our example could be designed this way:

- *Sunshine* : solar irradiation, or any other index, normalized between 0 and 1: $Sunshine \in [0, 1]$.

$Sunshine$	Hat	$Headache$	
		True	False
$\in [0, 1]$	No hat	$0.1 + 0.7 \cdot Sunshine$	$0.9 - 0.7 \cdot Sunshine$
	Cap	$0.2 + 0.4 \cdot Sunshine$	$0.8 - 0.4 \cdot Sunshine$
	Sombrero	$0.05 + 0.25 \cdot Sunshine$	$0.95 - 0.25 \cdot Sunshine$

Figure 2.11: Hybrid probabilistic distributions (discrete and continuous).

- Hat: the surface covered by the hat, with a maximum of 1 square meter.
- $Headache$: an arbitrary index of pain, normalized between 0 and 1: $Headache \in [0, 1]$.

In that case, as $Headache$ is continuous, the probability distribution cannot be multinomial. It can be, for example, a beta distribution with parameters depending on $Sunshine$ and Hat.

Now that we know what a BN is, the solving techniques and the applications can be presented.

Solving BN

A BN, as set in the example, can be used to determine the probability of a variable, given additional knowledge about the other variables. Methods have been developed to compute probabilities within a BN in any way, which is called inference, as unknown variable distributions are inferred from known/observed variables. The key rule that allows to do so is the Bayes' rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.1}$$

In this equation (2.1), $P(A|B)$ is read: *probability of A given B*. It is the probability distribution of A when B is known. This can be used in both ways: inferring the probable values of a child node depending on its observed parents, or inferring a parent's most probable value depending on the observation of its children's variables. This is done using Bayes' rule and methods derived from it. These methods are efficient and allow to compute probabilities in large BNs. These techniques can be exact or approximate. Dominant exact solution is message passing algorithms. We can cite junction tree and belief propagation proposed by Pearl in [57]. In [37], the author presents these algorithms as formulation of the sum-product algorithm. Exact methods have a complexity that is exponential in the treewidth of the network. To lower the complexity, approximate inference algorithms have been developed. They use Markov Chains, Monte Carlo inference or variational belief propagation among other techniques.

A very useful tool for building and solving BNs has been developed by Kevin Murphy [51]: the Bayesian Network Toolbox for MatLab®. It allows to build BN quickly, and to use solving algorithms, already coded as Matlab functions.

Application

BNs are used to depict processes with multiple variables, with probabilistic dependencies between each of them.

Building a Bayesian Network consists in a number of steps. First, the variables have to be identified. Then, the dependency between each of them is found, thus giving the structure of the BN. This step allocates a set of parents to each variable. This is usually given by an expert, or results from the nature of the problem. Eventually, the probabilistic distribution of every variable depending on its parents is given. These distributions can come from statistical observations, from expert knowledge, or be calculated through a learning process. Concerning learning, even the structure of the BN can be learnt, such as in [48].

Applications are very wide. A BN suits obviously well diagnostical activities, like medical diagnosis, failure and risk analysis, and any model-based diagnosis. It is also used for classifying data, like in natural language understanding. For robotics, it has been used for planning behaviours and even motions ([72], [6], [67]). It provides a way of modelling uncertainty, which is key to making robots able to act in the world, as real environment is always uncertain, or at least its perception is uncertain or incomplete. Actions are then variables, and their outcome is uncertain, stochastic.

2.3.3 Markov Decision Processes

MDPs are specific BNs. They are used to model sequential decision-making problems where an agent interacts with an uncertain system. A MDP models a process as a simple BN, and gives tools to help the decision system choose the appropriate behaviour. This section is mainly inspired from [65] and [68].

Overview

An MDP is a stochastic model. It describes, in probabilistic terms, the evolution of a system described by a state during successive time steps. At each time step, a reward is given to the agent depending on its state and an action is chosen by the agent to reach the state of the next time step. The process is a BN that can be represented graphically as in Fig. 2.12. At each time step t, there are three nodes: one with variable S_t as state, one with variable A_t as action, and one with variable R_t as reward. The number of time steps is called the duration D of the MDP.

A discrete MDP has a set $\mathcal{S}$ of possible states and a set $\mathcal{A}$ of possible actions. MDPs with finite sets $\mathcal{S}$ and $\mathcal{A}$ are called "countable MDPs". This is the case considered in this work. $\mathbf{P}$ is the probabilistic transition model. It assigns to each state-action pair $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$ a probability distribution over $\mathcal{S}$ for the state at the next time step S_{t+1}. In other words, $\mathbf{P}$ gives, for each possible next state, the probability of assuming a particular value s_{t+1} given the previous state s_t and the chosen action a_t. This can be noted by $P(s_{t+1}|s_t, a_t)$. Therefore $\mathbf{P}$ encodes the evolution of the state of an agent according to the actions taken and the dynamics of the environment.

In addition to this, at each time step, the agent receives a reward r_t. The reward reflects the positive or negative outcome of being in state s_t (Fig. 2.12).

Formally, an MDP is denoted as the following 4-tuple: $(\mathcal{S}, \mathcal{A}, \mathbf{P}, R)$.

- $\mathcal{S}$ is the set of possible states. At each time step t, the state of the agent is represented by a random variable S_t that can have values in $\mathcal{S}$.
- $\mathcal{A}$ is the set of available actions. A random variable A_t represents the action taken at time t. a_t is an instantiation of A_t.
- $\mathbf{P}$ is a probabilistic transition model. It models the time evolution of the state according to the executed action. It is usually represented as a probability distribution over next states S_{t+1} given the current state S_t and action A_t: $P(S_{t+1}|S_t, A_t)$.
- R is a reward signal collected by the agent at each time step. It depends on the MDP state, the action or both. In the case we consider, illustrated in Fig. 2.12, the reward $R_t(s_t) \in \mathbb{R}$ depends only on the state.

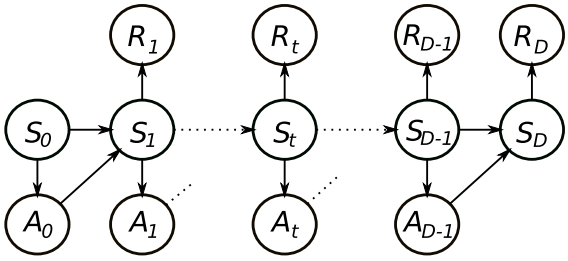


Figure 2.12: A general MDP of duration D, with a reward signal depending on the state.

Solving MDPs

Solving an MDP is finding which action sequence should be chosen in order to maximize the obtained rewards. The rules deciding which actions to take are called a policy, written π, and correspond to the behaviour of the system. Some policies can be seen as a mapping from states to actions: in a given state s, action a should be taken. A policy can be deterministic: $\pi : \mathcal{S} \to \mathcal{A}$. In that case, following π means that at any time t, the action a_t is selected using $a_t = \pi(s_t)$. A policy can also be stochastic. In that case, π maps states to distributions over the action space: $\pi(A_t|S_t)$. If a policy, deterministic or stochastic, is followed in an MDP, then the action A_t depends only on the state S_t, and the whole MDP behaves as a BN. The policy that maximizes the reward obtained by the agent during its lifetime is called the optimal policy π^*.

As rewards are obtained at each time step, a variable *Ret* is created, called the return, that accumulates each reward as a discounted sum:

$$Ret = \sum_{t=0}^{\infty} \gamma^t R_t \tag{2.2}$$

γ is the discount factor needed to assure convergence of the series in the infinite duration case. If $\gamma < 1$ then rewards far in the future weight exponentially less than the reward received at the first stage. When $\gamma = 1$, the MDP is called undiscounted and can only be used in the finite duration case, otherwise it may not converge. Considering a finite duration MDP (duration D) with $\gamma = 1$, the return becomes:

$$Ret = \sum_{t=0}^{D} R_t \tag{2.3}$$

An optimal behaviour is defined as getting the highest possible return (or expected return) for each initial state. The simple but brute way of finding an optimal behaviour in an MDP is to compute all possible returns by an exhaustive search over the action sequences. Then, the ones that give the highest possible return for each initial state are chosen. But due to the high number of policies, this is nearly always computationally intractable.

To avoid such a burden, value functions are used. They are fast to compute, and are helpful to find the optimal behaviour. A value function is noted V, and corresponds to a policy. It is the expected return of the policy. It is defined for each possible state $s \in \mathcal{S}$. A value function associated to a policy π is written V^π. Here we assume an MDP starting at time 0 and evolving into the future. In finite duration MDPs the value of a state is time varying. In infinite duration, the value of a state does not depend in time and can be computed defining an arbitrary time 0 and computing the limit of the expected discounted reward. In this case we have:

$$V^\pi(s) = E(Ret|s_0 = s; \pi) = E(r_0 + \gamma r_1 + \gamma^2 r_2 + ...|s_0 = s; \pi) \tag{2.4}$$

$$V^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|\pi(s), s) E(r_1 + \gamma r_2 + \gamma^2 r_3 + ...|s_0 = s; \pi) \tag{2.5}$$

$$V^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|\pi(s), s) V^\pi(s') \tag{2.6}$$

This equation (2.6) shows the recursive property of the value function.

The optimal value, $V^*(s)$, of state $s \in \mathcal{S}$ gives the highest achievable expected return when the process is started from state s. A behaviour that always drives the MDP to the optimal values in all states is optimal. So, when searching for the optimal

behaviour, only the optimal value should be considered. Using equation (2.6), the optimal value function is written:

$$V^*(s) = \max_a(R(s) + \gamma \sum_{s'} P(s'|a, s)V^*(s')) \tag{2.7}$$

$$\pi^*(s) = \max_a(R(s) + \gamma \sum_{s'} P(s'|a, s)V^*(s')) \tag{2.8}$$

As an alternative to the value function V, one may define the Q-function, called the action value function. It is the expected return given an initial state s_0 and an initial chosen action a_0:

$$Q^\pi(a, s) = P(r_0 + \gamma r_1 + \gamma^2 r_2 + ...|s_0 = s, a_0 = a; \pi) \tag{2.9}$$

$$Q^\pi(a, s) = R(s) + \gamma \sum_{s'} P(s'|a, s)V^\pi(s') \tag{2.10}$$

$$V^\pi(s) = Q^\pi(\pi(s), s) \tag{2.11}$$

$$\pi^*(s) = \max_a Q^*(a, s) \tag{2.12}$$

Solving the MDP is finding an optimal policy. Different techniques have been developed. If $V^*(s)$ or $Q^*(a, s)$ can be computed, the optimal policy can directly be derived using equations 2.8 or 2.12. In the case where the value function, or action-value function cannot be directly computed, it is a common solution to start with a suboptimal policy and iteratively improve it until the optimal policy is reached. These techniques are called value-iteration algorithms and policy-iteration algorithms. Other techniques use Bayesian inference, for more details, please refer to [69]. Learning can also be used to find the policy, as explained in 2.4.3.

But to use these standard techniques, every element of the MDP has to be known. This is not always the case. The core part of some decision problems consists in giving values to this model. Learning can be used for this, and not only to find the policies, as explained in 2.4.3. In Fig. (2.13) are presented some ways of using MDPs, whether the model is known or not. Experience can provide learning data for the model, to search the policy directly or through V or Q. Policies can also be computed directly from the model, or using value function/Q-function.

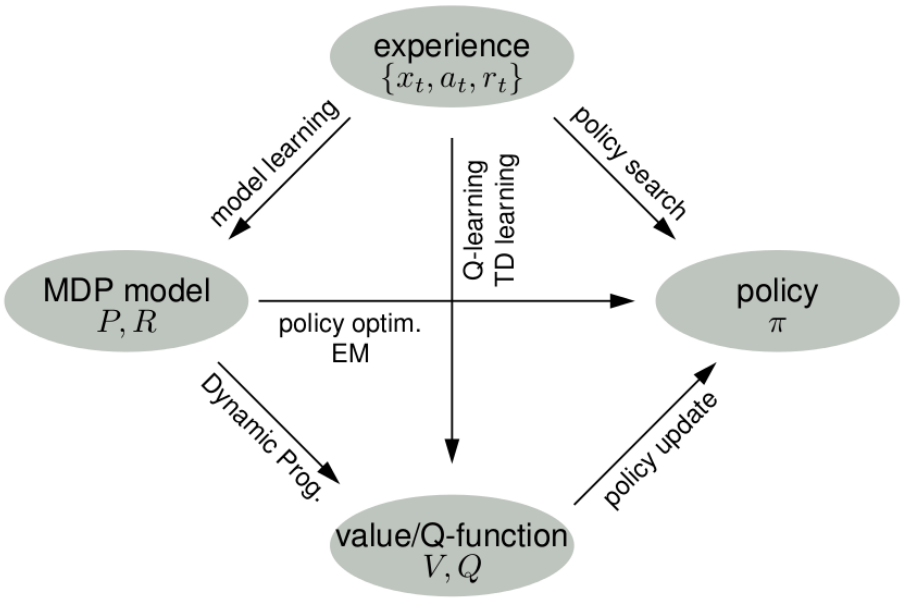


Figure 2.13: Different ways of using MDPs (source: [69]).

Conclusion about MDPs

This section gave an overview of MDPs. This tool is commonly used for decision making. This solution for high-level planning of manipulation seems to have a potential. The fact that it is a probabilistic model allows to describe the uncertain outcome of manipulation actions. It enables the use of learning to improve the accuracy of the model (see section 2.4.3 for more details). We will use it for planning in-hand manipulation in chapter 4.

2.4 Machine learning

The field of machine learning contains the techniques making an intelligent system learn from experience, demonstrations, reinforcement, etc. This learning activity can be performed in different ways explained in the next section 2.4.1. Learning has already been applied to some extent to manipulation, which is described in section 2.4.2. Finally, in section 2.4.3 are explained with more details the learning techniques especially applied to Markov Decision Processes. More detail on machine learning can be found in [33] and [44], from which parts of this section are inspired.

2.4.1 Learning: overview

Machine learning is a field of artificial intelligence, and consists in building methods to make a machine autonomously evolve through learning. A definition given by Tom Mitchell ([45]) is: "improving performance at some task through experience".

What a machine is

A machine is typically defined as a system that generates ouputs depending on inputs. It can be fully computational, dealing only with information (a data classifier, a game playing algorithm...), or more physical (mechanic, electronic, mechatronic, pneumatic, ...) with an embedded intelligence that makes decisions (decision system) and controls the system.

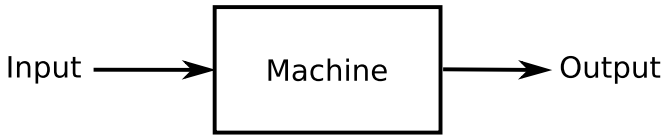


Figure 2.14: Simple diagram of a machine.

The machine is the mapping between inputs and outputs (Fig. 2.14). What can evolve in a machine are the internal parameters belonging to this mapping. Anything that can change can be modelled as a variable. Variables can be continuous values, discrete (integer, boolean), be used as coefficients, a table size, or even an identifier in a list. Even choosing between two complex behaviours can be implemented as tuning a discrete parameter that labels which behaviour is used. A machine usually comprises several parameters that can be learnt. Here are some examples to illustrate the definition of a machine:

- A writing recognition software takes information as input. This information is an image coded by a set of pixel values. The output is the letter written on this image.
- A weather forecast algorithm takes as input the meteorological present data and gives as output the weather prediction: probability of raining, sun cover, temperature, etc.
- A robotic arm throwing a ball at a target. The input is the target position, the ball weight, or any relevant parameter. The output is the effective hit point of the ball.

What machine learning is

How the parameters of the machine are changed is the core of the learning system. Random values of the parameters can be tried and their effect analysed. The best set is kept, as in genetic algorithms. Or the effect of the change of the parameters can be observed to try to make the system evolve in a positive way. All this depends a lot on the machine and on the learning system architecture.

The evolution through experience of the system is designed to reach objectives. The output of the machine is assessed as good, bad, or using any criteria reflecting the desired characteristic, or a distance to the optimal tuning. The learning process

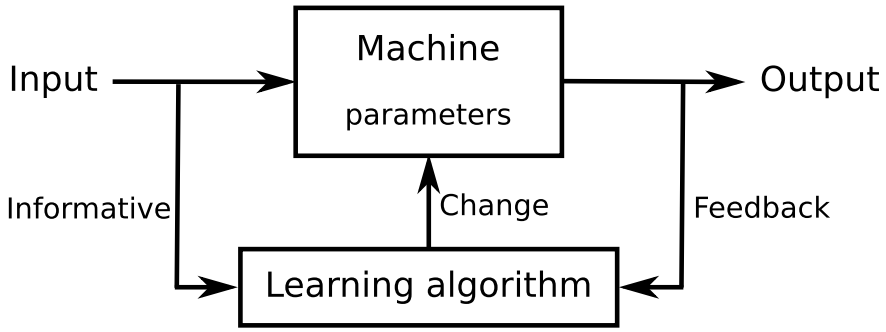


Figure 2.15: Diagram of a machine with a learning process.

is usually trying to reach an optimal set of values for the parameters, but sometimes tries to maximize or minimize an output effect, and the associated criterion. The quality measure of the parameters is a feedback calculated on the outcomes of the machine, through their effects or directly (Fig. 2.15). All these good or bad trials form the experience of the machine.

As an example, for the robotic arm throwing a ball, the decision system gives orders to the arm: movement trajectory, joint velocities, etc. The feedback for learning can be the hit point of the ball or the whole trajectory of the ball, depending on what is the robot able to observe. For example, in [36], the robot iCub is learning to shoot a bow.

The experience of the machine is a knowledge that the robot can build on its own, performing trials; this is robot self experience. Learning in this way is called reinforcement learning. Another possibility is that this knowledge is provided by an external database, through a set of already made examples. These data are provided by what is called an expert (Fig. 2.16). These data can be obtained from simulations, from another robot, or given by a human expert. In that case, it is called supervised learning. The output of the machine is compared to the desired output provided by the expert.

When the learning algorithm can decide which input it takes, then it can select the input that would generate the most valuable experience. It is then called active learning. It is feasible in both reinforcement learning and supervised learning, and is especially interesting in cases where generating the expert data is costly, as it boosts the speed of the learning process.

Advantages and drawbacks of machine learning

Learning presents a clear advantage when the controlled system and/or the environment influencing it are very complex. The solution to compute directly the optimal behaviour is then intractable: the effect of actions or the environment influencing the actions are impossible to model. Learning is simplifying the problem because the machine usually has parameters of fewer dimensions than all the real inputs of the

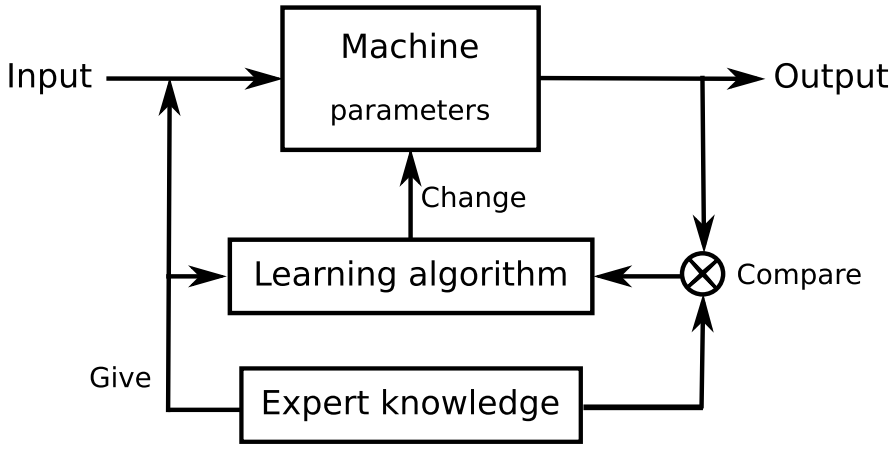


Figure 2.16: Simple diagram of a machine with a supervised learning process.

system: i.e. all the influencing parameters. Learning obtains the maximum of the decision system possibilities in a tractable way.

Current applications of learning are wide: database mining, speech recognition, text analysis, motor control (walking, grasping), vision (object recognition, image segmentation), etc.

An interesting summary of the actual use of machine learning is given by Mitchell in [45]:

What we know about machine learning:

- It is very efficient to draw conclusions from available data and adapt to it, such as with decision trees or neural networks.
- It works to reduce dimensionality of a problem. A large set of variables can be reduced to a set of fewer parameters, and learning is used to find the translation between the two sets. This can be applied to techniques like principal component analysis (PCA) or other compression algorithms.
- Active learning, choosing the query when acquiring learning data, greatly improves the speed of the learning process.
- Reinforcement learning (RL) is seen as a delayed feedback. The reward is obtained after the end of the action. Algorithms for RL give good results: Temporal difference learning, Q learning, policy iteration.

Machine learning has little to say about:

- Learning from instruction (lectures, discussion) is not effective, whereas humans can learn in a classroom, or be explained the activity without actually doing it or seeing it done. Data provided that way are indirectly linked to the activity, and robots are not able to use them.

- A lot of phenomenons encountered in human learning are not yet properly exploited for robot learning: motivation, forgetting, curiosity, fear, boredom, ...
- Learning by humans can be deliberate or unconscious. Some skills are developed without having the human subject think about it, whereas others need the human being to be focused on learning every detail. This difference between implicit (unconscious) and explicit (deliberate) learning does not exists among robots.

Human analogy

By performing machine learning on a robot, the analogy with human learning appears. A human learns from his/her own experience by training but is also able to learn by observing others, learning by imitation. He/she can also learn by receiving information from a teacher. These concepts are widely used for machine learning. Deeper analogies have been highlighted, for example in [61]. In this work, an analogy between the training error in a reinforcement learning process and dopamine activity in the human brain is observed. The dopamine circulation in specific neurons is shown acting similarly as a reward signal of a well-known learning technique: temporal difference algorithm (TD-learning). Fig. 2.17 shows the dopamine activity of a human subject trained to a positive event, the reward, with a prediction signal possibly occurring slightly before the reward. In three different cases, the dopamine activity response along time (from left to right) is given. In the first case, the reward occurs without a prediction signal. The dopamine activity rises just after the reward, looking like a positive feedback for the learning process. In the second case, the prediction happens, triggering a rise in the dopamine activity just after the prediction. The reward occurs, as predicted, but this time no rise in dopamine activity is observed after the reward. A positive feedback seems to be given, linked to the prediction. In the third case, the prediction happens and triggers a dopamine positive response, as in the second case. But this time, the reward does not occur, the prediction is false. Soon after the non-occurrence of the reward, a drop in dopamine activity is observed. This drop seems to counter balance the false positive feedback following the incorrect prediction. It could be a negative learning feedback or a cancellation of the positive feedback. This matches the operation of a temporal difference learning algorithm.

2.4.2 Learning in manipulation

The complexity of robotic manipulation motivated scientists to use learning techniques to solve the planning of such activity. Due to the scarcity of the work in in-hand manipulation, we enlarge this presentation to standard manipulation and grasping.

In [22], grasp affordances, i.e. possible ways of grasping an object in a stable way, are learnt by the robot self experience. The grasp affordances are modelled using a probability density function: grasp densities are computed on the 3D model of the object. These probabilities are initialized and then refined through experience: a

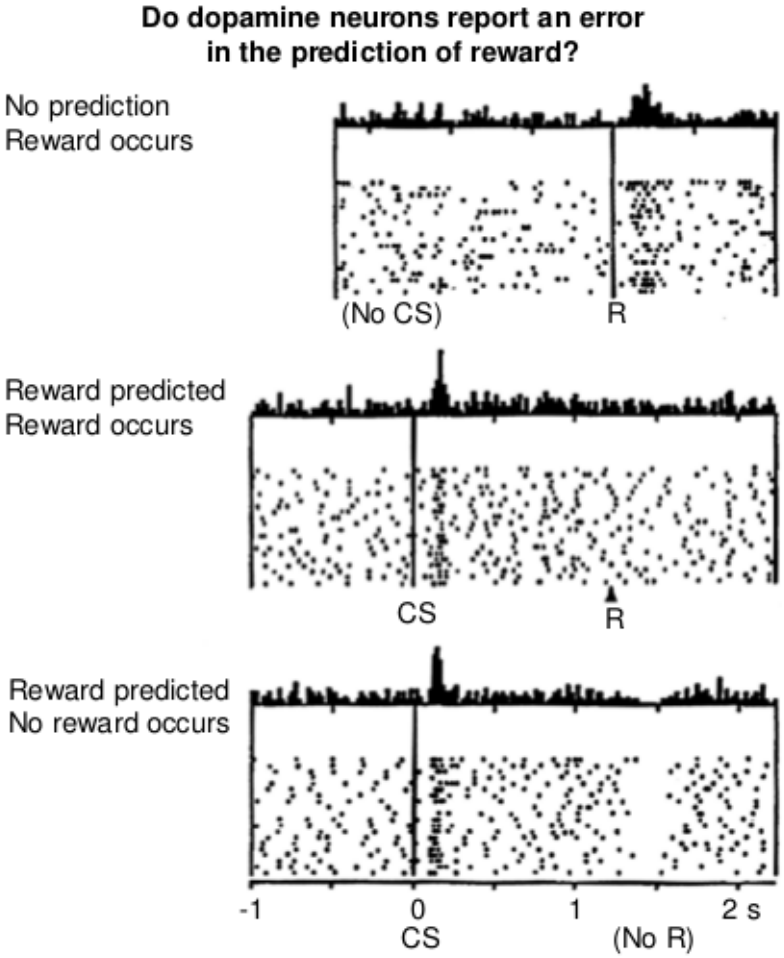


Figure 2.17: Dopamine activity response to events matches a machine learning reward signal (from [61]).

robot executes grasps drawn randomly on an object, and recalculates the probability densities depending on the outcome of the try. This method gives interesting results using the Barrett Hand (3 fingers). The model is initialized using manual initialization from expert knowledge. The speed of learning largely depends on this initialization. This method is discriminative learning, as only the outcome of the process is observed. The relation between the grasp parameters and the outcome are not learnt. Moreover, the grasp of an object should be chosen by the agent considering the task.

In [60], grasping points are learnt directly from vision using only a few images (2 is the minimum). These images are divided into small parts and filters are applied to each part without reconstructing or using an object model. A probabilistic model is built with maximum a posteriori (MAP) inference, to relate the probability of success of each image part to be a grasping point. Montesano takes a similar approach in [47] but uses the whole image of the object. Using kernels, the probability distribution of graspability is built. This learning technique propagates knowledge by local extrapolation. This work has been completed in [49] by making the learning active,

choosing to explore grasping in parts of the object more likely to be graspable, according to previous data. In these works, algorithms are learning from the analysis of the outcome of the grasp on a real robot. This can also be done in simulation. The learning process can be made automatic, saving time. This could also be made using a grasp metric as a reward indicator. But as Geidenstam highlights in [28], grasp quality metric and grasp success do not always match. In that work, a neural network learns to relate the quality metric to the real success of the grasp.

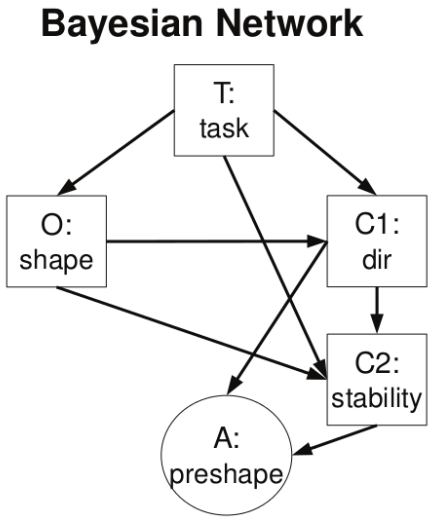


Figure 2.18: The Bayesian Network used in [63].

In [21], grasps are learnt for training. A robot autonomously makes successive attempts to grasp an object. Depending on the outcome of the grasp (success or failure), a complex representation of grasp affordances (graspability of the object) is built by learning. Some experiments have been conducted with a 2 finger gripper and give satisfying results. Trials are made autonomously, however, it is still a very time consuming process (forty to sixty seconds for a trial). The learning algorithm shows a few limitations as it takes a lot of trials to converge, depending on the initialization of the model.

Going further, in [63], possible grasps of an object are learnt as in [21]. But in this work, the grasp configuration and object are also related to the task performed. The method is decomposed as follows: a large number of grasps on several objects are generated in simulation. The learnt grasp parameters are the parameters of the preshape of the grasp with a Barett Hand (3 fingers). Then, the relationship between objects, grasps and tasks is learnt from expert knowledge (manual labelling) through supervised learning. The probabilistic relations are modelled using a Bayesian network (Fig. 2.18), which dependencies are learnt. In that work, the robot does not try to perform the grasp to learn.

Taking into account the dependency to the task is relevant for smart robot interacting efficiently with their environment. These interactions can also be learnt directly, without using algorithms specifically designed for manipulation. In [50], unsupervised learning is used to learn the effect of actions, also seen as tasks. The

dependencies between each of the parameters of the problem are not hard-coded. The dependencies modelled as a Bayesian Network (BN) are learnt. It means that the structure of the BN itself is learnt. Two resulting BNs are shown in Fig. 2.19. The original caption is : *(a) An example of the previous model using color, shape and size information for the object features and motion and contact information as effects. (b) Generic model where the nodes represent the actions A, the object features available to the robot F (1)...F (n) and the effects obtained through the actions E(1)...E(m).*

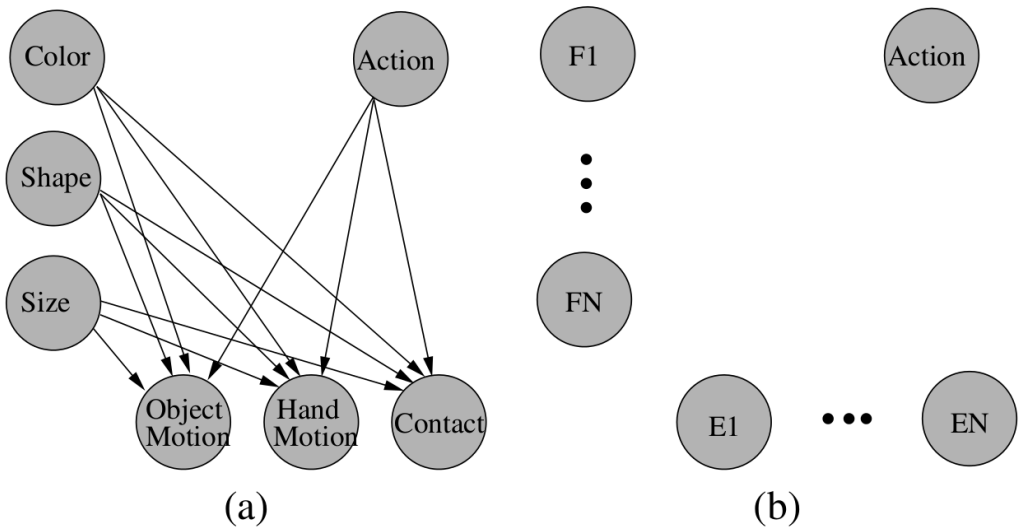


Figure 2.19: Original caption: Bayesian network model to represent the affordances (from [50]).

Learning is considered a useful approach to achieve complex activity. All the works described in this section prove that learning improves manipulation activities in robots. Such techniques can therefore improve in-hand manipulation, which is even more complex.

2.4.3 Learning in Markov Decision Processes

Bayesian networks in general are well suited for learning. [34] gives a good overview for learning in BN. [66] proposes an algorithm for active learning in BNs. MDPs are specific BN, and learning also applies very well. Specific techniques have been developed, taking advantage of the particular formalism of MDPs. Sources of this section are [65] and [69]. More details about reinforcement learning can be found in these publications.

MDPs have been introduced in section 2.3. An MDP models a sequential stochastic process where actions are chosen by an agent. It is used to compute an efficient decision system. Solving an MDP is finding the optimal behaviour, called the optimal policy $\pi^*(S)$. This policy gives the action to take depending on the state of the

system. When the model $P(S_{t+1}|A_t, S_t)$ is known, techniques listed in section 2.3.3 are used to find the optimal control policy π^*.

The state-action model $\mathbf{P}$ can be learnt from experience (Fig. 2.13). Reinforcement learning (RL) techniques are used in this case. The robot tries to perform the action and the outcome is observed. These data are used to calculate the probabilistic distributions forming $\mathbf{P}$. When the state action probability $\mathbf{P} = P(S_{t+1}|A_t, S_t)$ is not a priori observable, a solution is to learn directly the policy. When the policy is modelled as a BN, then learning techniques used for BN can be used to learn it.

Different types of RL are possible:

- Model-based reinforcement learning: the state-action model $\mathbf{P}$ is learnt, and then used with classical MDP solving techniques to find a policy.
- Model-free reinforcement learning: the value function V is learnt directly, without using $\mathbf{P}$. Main algorithms for this are temporal difference (TD-learning) and Q-learning.
- Policy search: a number of policies are assessed using learning. The search for the optimal policy is made directly in the space of policies.

Taking the learning in a different way, inverse reinforcement learning (IRL) consists in learning from an observed optimal behaviour (policy) and finding the reward structure that leads to this optimal behaviour. For more details, please refer to [54].

2.4.4 Conclusion about machine learning

In this final section is explained how machine learning can be applied to an MDP. Using these techniques would allow us to have a planning algorithm for manipulation able to refine through experience, just as humans do. If the robot is able to perform trials by itself, it can learn directly the state-action model $\mathbf{P}$. Otherwise, depending on what information is available, the optimal policy π^* or the value function V can be learnt. It appears that learning is already a solution used for robotic manipulation. It gives good results and allows to tackle the complexity of the problem.

2.5 Conclusion of the chapter

This chapter presented the state-of-the-art of robotic in-hand manipulation. We want to build a planning algorithm for in-hand manipulation using a robotic hand. We saw that in-hand manipulation can be divided into atomic actions, that correspond to in-hand movements of the object or finger relocation (finger gaiting). Good solutions exist to plan each of these atomic actions. However, complete planning of manipulation means to be able to choose among these different atomic actions and find the appropriate sequence to reach the final configuration. Existing manipulation planning solutions that try to solve the whole problem are doomed to be very time consuming. Some high-level approaches propose discrete planning by finding a path through successive configurations. This appears very interesting. However, we note that the

consideration of the task and the surrounding collision context for the selection of the initial grasp is mostly absent. We consider this a key aspect in the planning on complex manipulation actions, beyond simple pick-and-place, and will show that the proposed approach is able to tackle these problems.

For discrete planning, we present two promising tools in section 2.3, BNs and MDPs. These models allow to learn the appropriate behavior from self experience or demonstration. As highlighted in section 1.4, learning appears to be a key solution for performing in-hand manipulation. We will propose algorithms to both learn from humans and from robot self-exploration.

The next step is to figure out how to model manipulation using a high-level layer of planning. Using what has been presented in this chapter, the next chapter describes which choices have to be made for modelling manipulation from a high-level point of view and what are the implied hypothesis. The following chapter (4) details the use of BNs and MDPs for in-hand manipulation planning and the learning techniques used to learn the models. In chapter 5 we describe experiments executed to support the methods proposed in this thesis.

Chapter 3

High level control of in-hand manipulation: our strategy

The state-of-the-art of in-hand manipulation has been presented. It highlighted that in-hand manipulation can be divided into atomic actions, for which satisfying planning algorithms exist. Our solution should be able to plan choosing among these different actions.

In this chapter, we present the approach we have chosen to develop a high-level layer of control for in-hand manipulation. This layer is part of a larger solution to plan the complete activity of in-hand manipulation. It constitutes the core work of this thesis.

3.1 Formulation of in-hand manipulation

In this section, more details are given about in-hand manipulation. We explain what the problem looks like from the planning point of view. First, the variables of in-hand manipulation are listed. Then, details are given on how these variables are acquired on a real robot. Finally, the need for a high-level discrete planner dealing with discrete variables is justified.

3.1.1 Variables of in-hand manipulation

As presented in section 2.2, planning in-hand manipulation is far from being solved. No complete solution has been developed yet. The complexity of in-hand manipulation relies in the large size of the state vector, making intractable searches in such a state space. Indeed, a complete state of in-hand manipulation activity is represented by:

- Arm and hand joint angles (24 degrees-of-freedom for a state-of-the-art humanoid hand, the Shadow hand), joint velocities, accelerations, ...
- Object position (6 parameters)

- Object model (the number of parameters depends on the model used)
- Contact points position (3 parameters for each contact point)
- Forces/Torques at contact points (6 parameters at each contact point)

This represents a number of parameters around 40 at the very least.

3.1.2 Origin of the variables of in-hand manipulation

Considering a manipulation robot equipped with a robotic hand, the hardware acquiring the variables is detailed in the next paragraphs. All these parameters are acquired in different ways. An in-hand manipulation platform consists in different modules detailed as follows:

Proprioception

The arm and the hand are equipped with proprioceptive sensors, indicating the state of the arm and hand, i.e. the position of every segment, in the case of rigid hands. Most of the time, this information is given by angles sensors on every joints.

Vision

In order to recognize the object and its position, the robot needs an exteroceptive sensor. The state-of-the-art devices for such sensors are cameras, and a combination of devices enabling to detect the range. The range can be detected by different techniques: stereovision (2 or more cameras) or an infra-red projector and a camera. The infra-red projection of a pattern is observed by the infra-red camera and the depth computed from the deformation of the pattern on the surface. The Kinect® is a common device used to observe in-hand manipulation scenes, including object detection.

Tactile sensing

Tactile sensors can provide information about the contact of the object on the hand. For fine manipulation, such sensors should be installed at fingertips. The information can be simple, just differentiating if there is contact or not at a fingertip. More advanced sensors provide the force and torque applied on the fingertip, and the contact point. They are called 6D sensors as this measurement refers to 3 components of forces and 3 components of torques. Ideally, to enable human-like manipulation, this contact detection should also be available on all the fingers and on the palm. This is attempted at by the development of tactile skins.

3.1.3 Discrete high-level planning variables

Now let us set up the problem of planning in-hand manipulation. A solution for continuous planning and control of manipulation would be a fully continuous method, driving the system from the initial state to the final one. But dimensionality is very high with 24 dofs for the hand, 5 for the arm, 6 for the object, etc. Classical planning methods have been presented in 2.2.2 and their limitations show that this approach is doomed by the time required for computation.

While observing what happens during in-hand manipulation, a question is raised: are the effects all continuous? Physical variables are always continuous, like trajectories, velocities, forces and torques. But at our level of control, phenomenons like a finger making or stopping contact with an object are sudden events. Friction also implies strong discontinuities when a finger is sliding or rolling. Going from static friction to friction on a sliding movement is a discrete change of behaviour, either fully sliding (1st order model friction) or "stick and slip" sliding.

Then, an implied question lies in: is it possible to deal with discrete phenomenons (or discontinuous phenomenons) with continuous planning? A discrete planner intuitively appears more likely to deal efficiently with the discrete aspects of in-hand manipulation. Another justification for this choice is that continuous planning would be intractable because of the large dimension of the problem.

Let us now formulate an in-hand manipulation activity following the description found in [23] and described in section 2.1 (Fig. 2.5). This description is illustrated in Fig. 3.1. From an initial configuration to a final configuration, in-hand manipulation consists in a sequence of in-hand atomic actions of manipulation. Different actions can have the same types or not. Between two actions, our system (the robot, its hand and the object) is in an intermediate configuration. These configurations are added in Fig. 3.2.

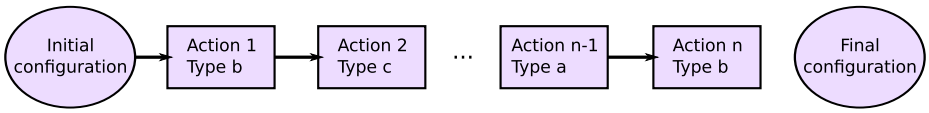


Figure 3.1: In-hand manipulation decomposed into atomic actions.

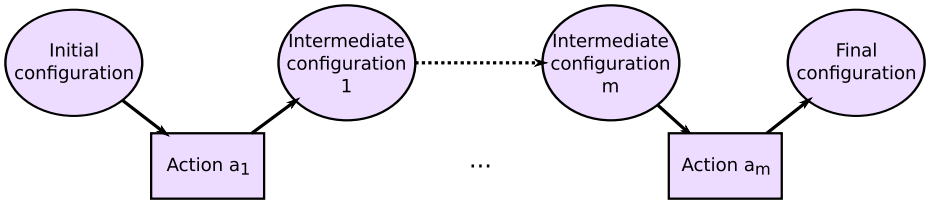


Figure 3.2: In-hand manipulation with intermediate configurations.

Low-level variables of manipulation are continuous, while high-level considerations reveal discrete choices of atomic actions. It implies that in-hand manipulation planning needs both a discrete planning solution and a continuous planning solution. This

is a common approach in state-of-the-art work presented in section 2.2. These solutions should be combined to tackle the main problem of in-hand manipulation planning dimensionality. What appears naturally when analysing in-hand manipulation decomposition as in Fig. 3.2 is a stratified planning system. Two strata, also called layers, communicate with each other to generate a plan for the given manipulation task (reaching the final configuration). A first layer, the high-level layer, generates the sequence of intermediate configurations. And a second layer, the low-level layer, is in charge of planning every atomic action linking two intermediate configurations. In Fig.(3.3), the limit between the high-level layer and the low-level layer is drawn.

As explained in section 2.2, a planning architecture decomposed into hierarchical layers has been introduced in previous works. This is a way of simplifying the planning problem, dividing it into different layers, as it is simpler to deal with a portion of the problem than to deal with the whole system. This layer is presented in section 3.2. Communication between layers is then a subsequent problem addressed in 3.3 and 3.4.

3.2 Overview of our solution

In this work, we aim at developing a high-level planning layer for in-hand manipulation. We keep a complete vision of in-hand manipulation planning, as our solution is designed to be integrated on a real robot. The complete framework consists of different hierarchical layers communicating with each others. The low-level layer generates fine movements, using continuous planning techniques. As presented in section 2.2 and at the beginning of this chapter, the main limitations of these techniques are that they are unable to deal with the whole panel of different actions existing in in-hand manipulation. Our strategy to tackle in-hand manipulation difficulties is to set up a high-level layer defining a sequence of key configurations or states. From this sequence, the low-level planner only has to deal with reaching the next subgoal of the sequence. This is illustrated in Fig. (3.3). From the high-level layer point of view, an action makes a transition between two intermediate configurations and requires a combination of one or more elementary in-hand actions.

The search for a path in the subgoal space has to be fast. The subgoals have to be simplifications of the state, otherwise, the dimensionality of the problem would stay the same. Thus, the states of the sequences are likely to be discrete ones. Choosing to have such a simplification seems mandatory for planning reasonably fast. The lowest layers deal with the continuous variables but only on a highly reduced problem. That is what allows to tackle dimensionality.

The high-level layer should provide the subgoal sequence so that the low-level layer can plan and execute the transitions between subgoals. This implies that the subgoal sequence planner knows if the low-level algorithms can perform these transitions. A transition between two subgoals depends on a large number of parameters. Some of these parameters are constant during a manipulation task:

- The system itself, its geometry, maximum forces, contact surface: the hardware.

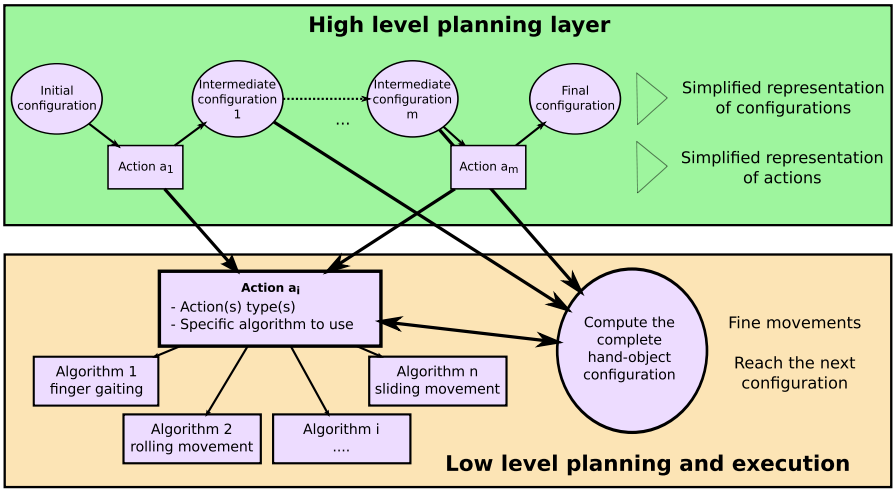


Figure 3.3: Boundary between the high-level layer and the low-level layer of our solution.

- The system's low level of planning: the software.
- The object: geometry, weight, contact surface.
- Task related constraints.
- Other objects in the environment.
- The collision context.

Other parameters are variable:

- The current subgoal.
- The next subgoal.

There are too many parameters to have a quick computation. Some parameters are even sometimes impossible to acquire. At our simplified level, transitions between states have to be simplified (more details in section 3.6), just like the fact that the state has to be simplified (section 3.5).

3.3 Link with the higher-level layer

In this section, we discuss what the highest level of planning of a manipulator robot is. The link between our layer and the higher-level layer results from this study.

For now, manipulator robots used in the industry are performing very specific tasks. They are even designed for nearly one single task. They are not adaptive to unexpected events. The high-level order that they receive is: "here are the parameters of the one and only task, do it". In this work, we consider an advanced manipulator

robot. It is intended to be multi-task, with a high degree of adaptability. We have to define here how the tasks are defined, how they are transmitted to the robot, how the mission of the robot is defined.

Our layer is at an intermediate level. Higher-level layers deal with non-physical concepts, ideas, words, human representations of reality. This requires a great degree of conceptualisation, and a knowledge hard to acquire. The planning layer of highest level is "what should the robot do", "what is the appropriate task to perform now?". This deals with desires of the human beings served by the robot, or when acting without human order, the task to perform comes from the environment analysis and the programmed behaviour of the robot.

Having a robot able to detect and reconstruct its own environment, with the ability to automatically infer the most appropriate task still belongs to science fiction. Some works are in progress about automatic conceptualization of the environment using artificial intelligence and cognitive sciences. In [64], the author presents a recent work about automatically understanding an activity. An activity tree is built from video observations of an actor performing activities. The tree is built using grammar rules as in language but applied to activity description. A key work using such an approach is the minimalist program by Chomsky [13] that tries to reveal an universal generative grammar for language. This would mean these rules underlying beneath language are the way that humans conceptualize their environment. An extensive analysis of this work can be found in [82].

The grammar in [64] has one simple rule that is: an activity consists of:

- Using a tool to bring two objects together, resulting in a new object or a tool.
- Using a tool together with a single object, resulting in a transformed object or a tool.

This framework is used to recognize simple activities made of simple actions. Focus is set on manipulative actions and activities like: "making a sandwich" activity, made of the actions "spread" and "slice" applied to the objects: "bagel", "ham", "cheese", "knife". This work is limited to activities made of sequential actions manipulating objects. More complex, abstract actions, like "playing soccer", cannot be detected by these techniques. Higher level reasoning techniques have to be used. The author suggests stochastic context free grammars for this.

So far, in-hand manipulation planning didn't meet this field. The solution for state-of-the-art robots is to have the task definition hard-coded. The robot can then detect and choose which task has to be done automatically. We start our work just at this point: the task to perform is known, and its details are also known. Explanations about which information of the task is used are given in the next chapter (4).

3.4 Link with the low-level layer

Our layer has been presented, its limit toward high-level has been drawn, foreshadowing the task definition and the orders received. Now let us present the hierarchical

relationship linking our layer and the low-level layer. We detail what is provided to the lower level layer and which information is exchanged between these two layers.

Following the work presented in section 2.2.2, we understand how close to the low level our layer has to go. Different techniques for low-level planning and control exist but cannot deal with the complexity of in-hand manipulation (section 2.2). A lot of effort has been made and work is still in progress on continuous lower level planning and control: trajectories, movements, DMPs, control, synergies, force/position, force/vision, etc. The gap between the highest level described before in section 3.3 and these techniques has to be filled. Consequently, we aim at building a high-level of planning at least up to the point where the low-level problem becomes computable.

We assume the existence of a low-level layer able to perform different types of actions, making the system evolve from a stable configuration to another stable configuration. The aim of our layer is to provide a sequence of stable configurations, so that the low-level is able to perform each of the transitions in the sequence. The subgoal sequence is the main transmission from our layer to the low-level layer. The subgoal definition forms a key step in our work, and is introduced in section 3.5 (Fig. 3.4).

In our layer, the problem is simplified, and consequently some information gets lost. Thus, the subgoal sequence given to the low-level planner might contain transitions that, for any reason (unknown context, inaccuracy, ...), are not feasible. The low level planner then fails to plan the transition. In this case, the low-level planner can inform the high-level layer and cause the generation of a different subgoal sequence. We call this operation "replanning". It has to be very fast to provide an alternative way on-the-fly. The information about the failure allows our layer to refine its inaccurate knowledge on the ability of the low-level layer. This implies appropriate learning techniques that are explained in chapter 4.

3.5 State representation

The subgoal sequence is the outcome of our intermediate planning algorithm. The subgoals are simplifications of complete states. The simplified states, called states as from now, should be simple enough to enable a fast planning or replanning, and precise enough to be efficient and provide a feasible path to the system.

A solution could be to use synergies (subsection 2.2.2). The full state variables would be coupled to a small set of synergy variables and the whole system controlled by a small number of variables. In order to successively reach high level subgoals, the low-level of planning can still use all the variables, decoupled, between the subgoals. However, synergies are better fitted for continuous planning, and not for our path search involving discrete aspects.

In this section, we present possible discrete state simplifications. First, we explain what a grasp is and how we can define a discrete parameter describing it in subsections 3.5.1 and 3.5.2. Then, we present how we can simply describe the hand and object positions. It introduces the state we will use to plan the sequence.

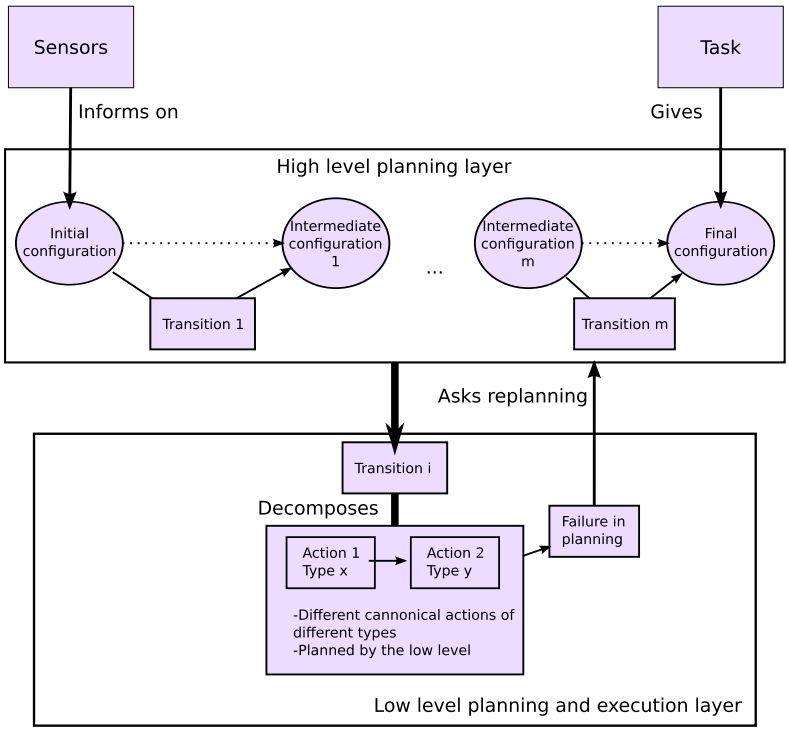


Figure 3.4: Information exchanged between the mid and low layers.

3.5.1 Grasps

Subgoals should be stable configurations so that the system can converge easily towards a given subgoal. A stable configuration also allows the system to remain in this configuration while waiting for an event, like the end of a computation or the completion of a task by another robot. A stable configuration, in the field of in-hand manipulation, is obviously a grasp. Let us define precisely what a grasp is. This completes the introduction about grasping planning given in section 2.1.2.

A grasp is a configuration where the hand holds the object in a stable manner.

- Holds: the object is kept in the same position relatively to the hand.
- Stable: perturbation forces/torques can be applied without moving the object relatively to the hand/changing the grasp/moving the contact points, in a range corresponding to the hardware limitations.

A grasp is achieved by contacts between the hand (finger, palm) and the object. Forces are applied by the hand, through contact points or contact areas, so that the forces, using friction or not, sum to 0, keeping the object motionless relatively to the hand. Perturbation forces can be applied and, with friction forces modified to counter them, the sum of the forces remains 0.

As introduced in section 2.1.2, geometric criteria have been proposed by researchers in robotic manipulation to characterize stability. Parameters characterizing a grasp are the following:

- Contact points/contact zones. Can be defined relatively to the hand or to the object. A point position needs 3 parameters. Relatively to the hand, contact points can be defined as parts of the hand: palm, fingertips, first phalanxes.
- Forces applied on these contacts (3 or 6 parameters).

Scientists have identified different grasp types used by human beings, as presented in section 3.5.2. The grasp type can be used as a discrete simplified state. In that case, information about the position of the hand and the object is missing to describe the whole system. Including this information in the state is discussed in section 3.5.3.

3.5.2 Grasp taxonomies

Humans naturally use different typical grasps, or grasp types, after a long learning process, using the large and still unknown capacities of the brain and human sensory motor control. These grasps have been identified by observing human manipulation activities and taxonomies of grasp types have been set up. These taxonomies are a list of grasp types with criteria defining each grasp type.

The next paragraphs present a few examples of taxonomies.

From Napier, 1956

In [53], the author reviews what parameters could by used to classify grasps in grasp types. He gives a large weight to the task to which the grasp is applied. This work was published in a medical journal and is totally decoupled from robotics and planning. After a deep analysis, the author ends up defining two main grasp types: precision grasps and power grasps, also called precision grip and power grip (Fig. 3.5). From the author's words, here are the conclusions of this analysis:

- *The prehensile movements of the hand as a whole are analysed from both an anatomical and a functional viewpoint.*
- *It is shown that movements of the hand consist of two basic patterns of movements which are termed precision grip and power grip.*
- *In precision grip the object is pinched between the flexor aspects (the manipulating side of the phalanxes - writer's note) of the fingers and that of the opposing thumb.*
- *In power grip the object is held as in a clamp between the flexed fingers and the palm, counter pressure being applied by the thumb lying more or less in the plane of the palm.*
- *These two patterns appear to cover the whole range of prehensile activity of the human hand.*

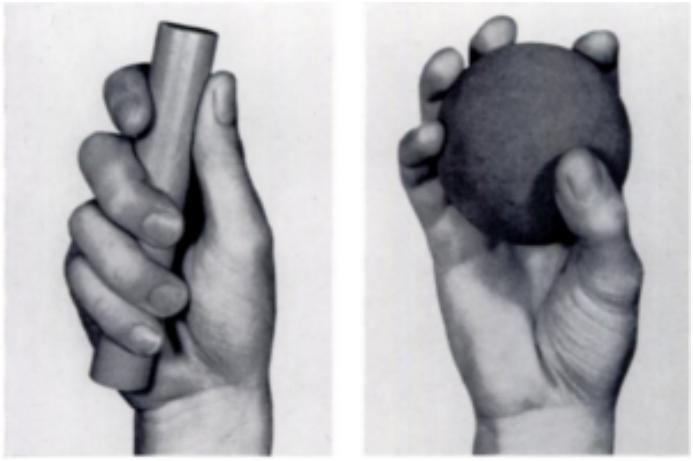


Figure 3.5: Illustration of the two defined grasp types (from [53]).

From Lyons, 1985

Lyons takes the problem into robotics. In [42], the author defines three grasp types (Fig. 3.6) in order to use this simplification into planning dexterous movements:

- Encompass grasp: corresponds to the power grip defined in [53].
- Lateral grasp: when the hand works in the same manner as a simple gripper. The thumb and the index, or more fingers, contact the object on opposed sides, but without using the fingertips.
- Precision grasp: the fingertips hold the object.

From Iberall, 1987

After comparing previous taxonomies, the author in [35] introduces two new parameters to characterize grasps:

- Opposition Type: the surface of the hand to which the object is opposed: palm opposition, pad opposition (finger pad) or side opposition (the side of at least one finger)
- Virtual Finger assignments: the grasp is seen as 2 or 3 virtual fingers grasping the object. The virtual fingers can be the palm, the thumb, a finger or a group of fingers.

These two parameters are illustrated in Fig. (3.7).

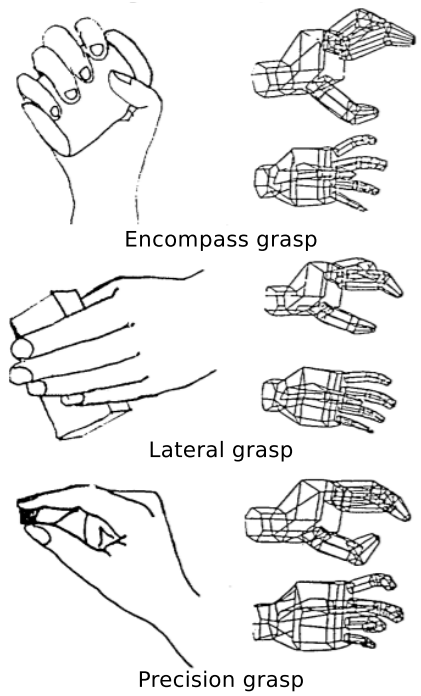


Figure 3.6: The three defined grasp types (from [42]).

From Cutkosky, 1989

Cutkosky [18] offers a taxonomy taken from the observation of human machinist during manufacturing tasks. This enables to have clues in designing robotic hands with useful abilities. The study also proves the interest of methods like expert systems to characterize grasps and grasp quality measures derived from an analytic model.

The author also lists grasp attributes:

- Sensitivity
- Precision
- Dexterity
- Stability
- Security

These attributes are mapped to grasp analytical measures.

In common use, grasp security is related to stability, but is most closely associated with resistance to slipping. This could be extended to risks, using tools like FMECA (Failure Modes, Effects and Criticality Analysis).

From Feix, 2009

In [24], the purpose of the work is to identify the most useful grasps in order to simplify, when it is possible, the design of prosthesis and more widely anthropomorphic

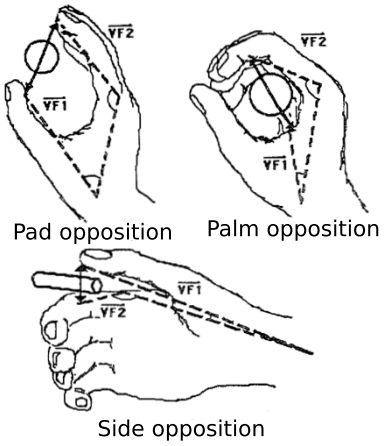


Figure 3.7: Basic oppositions that make up human prehension. Illustration of the opposition type parameter and the virtual finger assignments VF1 and VF2 (from [35]).

hands. It compiles a lot of already existing taxonomies, including [53], [42], [18] and [35].

The criteria characterizing a grasp type are taken from [35] and applied to every grasp encountered in the considered literature, opposition type and virtual finger assignments (section 3.5.2). Each grasp is given a name and an identification number. Each grasp type class receives the quality of "power", "intermediate" or "precision", according to the definition in [53]. The state of the thumb, adducted or abducted, is also given.

147 grasps from previous works were listed and grouped into 45 different grasping patterns. In this list, 33 proper grasp types were extracted, forming a very rich grasp taxonomy. Some of these grasp types have the same characteristics, look very similar, and only differ because of the shape of the object. It is then possible to merge these grasps and reduce the set to 17 grasp types.

Our choice and conclusions

The taxonomies presented here provide sets of characterized grasp types. The grasp type of the system appears to be very interesting as a discrete variable of the state representation, describing the hand and the object configuration. Any discrete grasp type is a very simple representation parameter. Consequently, it misses some information, as expected in our attempt to simplify the state. All these missing details like the exact joint angles have to be found by the lower level.

The taxonomy suggested by Feix in [24] appears to be the most advanced one. For the planning of robotic manipulation, our choice is to keep the large set of 33 grasp types (which is fully given in appendix A). The reason is that even if some grasp types are similar, and are changed via the object shape, planning the execution of these

grasps might be different. We use this grasp type set for manipulation planning, as explained in chapter 4.

3.5.3 Hand object relative pose

Grasp type describes the hand and object state. As seen in Fig. (3.8), the same grasp type can be applied to an object in various positions. Information about the hand and object position is lacking from the grasp type representation. In-hand manipulation actions are likely to be different when the hand holds the object with the same grasp, but in different positions.

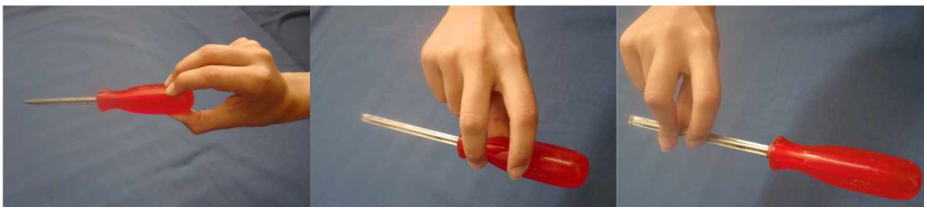


Figure 3.8: Illustration of the same grasp type on the same object, with different positions.

What is relevant for in-hand manipulation is the relative pose between the hand and the object. We consider that the absolute pose of the hand and object does not affect manipulation. Obstacles would not influence in-hand manipulation in a regular environment. Gravity, however, can influence some specific in-hand actions. We choose to ignore this influence and not to weigh down our model for an uncertain gain in accuracy. This approximation should not prevent the robot from performing efficient in-hand manipulation.

To describe the relative pose between the hand and the object, 6 parameters are required, 3 position coordinates and 3 angular coordinates as for any position between two solids. These continuous parameters can be made discrete in order to have a tractable model.

3.6 Influencing parameters

We have to discuss which parameters will influence the subgoal sequence generation. Whatever state is chosen in our work, the generated subgoal sequence should be the "best" sequence. "Best" needs an appropriate formulation, according to what our system aims at. The target is efficient in-hand manipulation by the robot, which translates in:

1. The successful proceeding of the task, which means reaching the final configuration and satisfying task related constraints (keep the object straight, low acceleration, ...). This is linked to the ability of the low level to plan and execute the actions implied by the transition, and thus, implied by the sequence.

2. Energy efficiency, a secondary consideration. It may be used to optimize different solutions with equal satisfaction of the first point.

3. Comfort, a secondary consideration too. This applies to humans, but not obvious to apply to robots.

Our generated subgoal sequence will be the one best achieving the chosen criteria. Due to the complexity of in-hand manipulation, focusing on the first criterion is a realistic limitation. But the other criteria can be kept in mind and used later. The aim of the sequence generation is to have the easiest sequence to perform, regarding both subgoals themselves and in-hand manipulation actions (transitions) linking the subgoals.

Next, we give an overview of the parameters and try to establish an exhaustive list. Then, we detail the influence of the manipulated object, which can itself be described by several parameters.

3.6.1 Overview

In this section, we introduce parameters influencing the ease to perform a sequence. First, parameters influencing the subgoals themselves in a sequence are considered. The grasp type is used as an example of a subgoal. On the one hand, it enables us to reason on a concrete subgoal, and on the other hand, we use this parameter as a subgoal representation in our solution. Secondly, the influence of transitions between subgoals is detailed.

Grasps

During human manipulation, the choice of the grasp depends on a lot of parameters. Some grasps will be used preferentially depending on the context. Considerations about these dependencies in previous works are focused on grasping only, not especially on in-hand manipulation.

Napier [53] lists the following parameters:

- Shape of the object
- Size of the object
- Miscellaneous factors linked to the object: weight, texture, temperature, wetness/dryness
- Fear, distaste and hunger, which are rather linked to the task, or even the human subject. For robots, this would translate into risks. This is related to "security" in [18] (see 3.5.2).
- Influence of the intended activity.

In [42], the author confirms that the grasp is concerned with the object characteristics and also with the intended object usage (task). Each of three grasp types are then associated to a given set of influence parameters:

- Task influence: firmness or no firmness required, precision or no precision required.
- Object influence: shape (flat or round) and size (small, long, large).

Transitions

In our problem, we consider a whole in-hand manipulation activity, with both grasps and transitions. The transition from a subgoal to another has to be modelled. The actions and their ease of accomplishment have to be assessed. Current work is missing on the subject. Here, we offer our own vision of transition ease, which determines the choice of these actions in an in-hand manipulation activity.

Both for humans and robots, dexterous manipulation actions depend on a large diversity of factors that we sort in two main groups:

- Intrinsic factors: biomechanical constraints, the hand geometry and capabilities, the energy effort implied to execute the action and comfort criteria. The object: shape, size, friction. The possible actions and their limitations. For robots, it is the low-level planner.
- Extrinsic factors such as real-time contingencies, obstacles and task constraints (example: manipulating a mug full of liquid).

The hand geometry and capabilities are considered fixed; the same hand will manipulate all the time. The object, however, might change from one activity to another. Extrinsic factors are more likely to change, even during an activity. The way to model these parameters is explained when we build our solution in section 4.1.

3.6.2 Object influence

We see from previous works that the object is one of the most influential parameters in in-hand manipulation. High-level planning of in-hand manipulation depends on the object properties. A system that generates subgoal sequences is affected by the type of object encountered. We have to decide how to model the objects, so that the chosen parameters are the ones that really influence the transitions in robotic in-hand manipulation.

In their everyday life, human beings use a wide range of different objects that we aim at covering with our method. These objects can be tools (hammer, screwdriver, spoon, knife, key), liquid containers (bottle, can, mug, spray), user interfaces (remote control, keyboard, mouse), food (fruits, vegetables, sandwich, bread, dishes), information containers (book, CD, USB key), natural objects (stick, stone) or even mobile parts of a fixed object (switch, button, lever, wheel).

Parameters that characterize an object are numerous:

- Object shape, object size (can be both included in a 3D model of the object).
- Weight.

- Friction.
- Specific graspable or ungraspable parts (handle, blade).

Object shape and size We consider that shape and size are the main parameters influencing in-hand manipulation. It is the geometry of the object, that determines where contact points can be placed, if force closure criterion will be achieved or if the fingers can be moved around the object.

Friction Friction is commonly described by using friction coefficients. It is a simplification of a complex physical phenomenon influenced, for example, by roughness or mechanical strength of the material. But this simplification is widely used in robotics and robotic manipulation. Theory of friction using friction coefficients was first published by Charles-Augustin de Coulomb by the end of the eighteenth century, but its original identification might have been by Guillaume Amontons a century earlier or by Leonardo da Vinci two centuries earlier.

The static friction (dry friction) law sets that the limit of the tangential resistance between two solids depends linearly on the normal force applied by a solid on the other one with the static friction coefficient μ_s. When dry friction is assumed, no movement between the two solids happens. The dry friction coefficient does not vary much for usual materials. It drops for wet materials, and in this case, the influence of friction should be considered, as manipulation behaviour would be changed. Considering dry objects and thus avoiding to take dry friction into account appears to be a reasonable first approach.

Kinetic friction (or dynamic friction) is a force that opposes the movement of two solids in contact. It is proportional to the normal force applied between the two solids. The coefficient of kinetic friction μ_k relates the tangential friction to the normal force. Kinetic friction is important for sliding movements of fingers on the object, but μ_k does not vary significantly among usual materials, and thus its influence can be neglected.

Weight Weight, when too high, may prevent some transitions' achievements. It is very complex to model as it will depend on where the hand grasps the object, on the hand and object position relatively to gravity, on the hand maximum force at fingertips and on the number and quality of contacts. We decide to neglect the weight influence in high-level planning. Our high-level planning is not designed to be perfect and never fail. Low-level planning, using force sensors, will check the feasibility of transitions and thus avoid to try transitions that are impossible because of weight. The low-level layer will deal with weight more easily. When a transition is recognized impossible, the low-level planner asks the high-level layer to generate a new path. The high-level drives the low-level in paths that are the most likely to succeed, and low-level checks these paths to ensure the feasibility. This strategy gives only one or a few paths to be checked by the low-level, drastically reducing the dimensionality of the low-level planning problem.

Specific graspable parts Specific graspable parts is a human concept when designing an object. A graspable part, because of its size and shape, is easier to manipulate, and the robot should choose by itself to grasp this part because it is indeed easier. We choose not to take specific graspable parts into account in our transition model, allowing the robot to choose by itself if the specific part makes manipulation easier, or choose another part if it reveals easier to manipulate. Specific parts designed to avoid to grasp hot/slippery/dangerous parts of objects are not linked to the object but to the task and should be taken into account this way, not through an object-related influence.

3.7 Chapter conclusion

This chapter has presented details on in-hand manipulation and its planning. The study highlights the need of a high-level layer that would generate a sequence of subgoals, given very simple high-level informations like orders. The low-level of planning and execution is then able to follow this path with a very high success probability, and reach the target task without being trapped in a deadlock position. The subgoal representation has to be defined and the parameters influencing the sequence generation have to be detailed. These are choices made when building our solution, which is the topic of the next chapter.

Chapter 4

Development of our solution

The objective of this thesis is the development of a high-level layer of planning for in-hand manipulation. This chapter presents how we achieved this layer. Due to the very high dimensionality of the problem, in-hand manipulation is very complex. To the best of our knowledge, no complete solution has been proposed yet. To tackle in-hand manipulation, our strategy is to set up a high-level layer generating a sequence of key configurations, or states (chapter 3). The state (configuration) and the contextual parameters (object, constraints) have to be defined. Following a hierarchical approach to the problem, the high-level state and context variables represent abstractions (or simplifications) of the problem so that it becomes tractable. As defined in section 3.1.3, these parameters are likely to be discrete ones. The level of abstraction should be such that the high-level planning can compute efficiently a sequence of subgoals for the problem and the generated subgoal sequence forms a feasible path that the low-level of planning and execution is able to follow step-by-step. Transitions between states represent abstraction of the in-hand atomic actions linking subgoals. They are planned by the low-level of planning and control.

Because it is extremely hard to obtain analytical models of atomic in-hand manipulation actions, we intend to make the robot learn the desired behaviour. During its lifetime, the robot will collect information about the best way to perform the tasks, either by human teaching or by self exploration. It is consistent with a probabilistic modelling allowing progressive inclusion of knowledge as introduced in section 2.3 and 2.4. A greater effort is required in modelling and learning but this approach allows the system to quickly sketch a realistic plan of actions in run-time. Low-level planners/controllers will just have to check for a limited number of promising subgoal transitions.

In this chapter, we are building a subgoal sequence generator for in-hand manipulation high-level planning. We use a Markov decision process (MDP), which is a probabilistic model and therefore deals with uncertainty. The theory of the model is presented in sections 4.1, 4.1.5 and 4.5. The influence of the object is taken into account (section 4.2). We initialize our model from empirical reasoning (section 4.3) and then refine the model using self-exploration (section 4.4) and human demonstration (section 4.6).

4.1 Overview

As explained, our work is a high-level planning system, which consists in finding a sequence of subgoals from an initial configuration to a final configuration. The generated sequence should be the most likely to succeed in reaching the final configuration. The solution we consider in this work is a probabilistic model, a Markov Decision Process (MDP, see section 2.3). We use it to describe the sequence of subgoals and the actions taken to successively reach the next subgoal. This section describes this model (section 4.1.1) and the related choices. An important hypothesis of this work is the choice of the state representation detailed in section 4.1.2.

4.1.1 Markov decision process

A Markov Decision Process (MDP) is a Bayesian Network that models the evolution in time of the state of an agent according to the actions taken and the dynamics of the environment. At each time step the agent receives a reward reflecting the successful (or not) accomplishment of the task (Fig. 4.1).

We built this model taking the influence of the object into account. In our work we consider a discrete set of objects that are typical of everyday use. Generalization to any shape can be done in a following stage, using dimensions of the objects of the set. This generalization is possible using discrete descriptors as defined in [42], two continuous parameters approximating objects as cylinders, or 6 parameters using an object model with superquadrics. We use a set of 9 discrete objects. This choice is detailed in section 4.2. In general, a set of M object types is defined:

$$\mathcal{O} = o_m,\ m \in \{1, \cdots, M\} \tag{4.1}$$

An MDP is denoted as a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathbf{P}, R)$ that, in our case, is defined as:

- $\mathcal{S}$ is the set of possible states. At each time step t, the state of the agent is represented by a random variable S_t that can have values in $\mathcal{S}$. As presented in section 4.1.2, the set of possible states includes the canonical grasp types g_i and a null state ϕ representing failures, infeasible or non-human like transitions. Thus, $\mathcal{S} = \mathcal{G} \cup \phi$ with:

$$\mathcal{G} = \{g_1, \cdots, g_N, \phi\} \tag{4.2}$$

and $N = 33$ as in [24].

- $\mathcal{A}$ is the set of available actions. Transitions between grasp types are produced by the robot using a single or a combination of atomic in-hand actions. An action in an MDP is an attempt to make a transition, in other words to reach a next grasp type from a previous grasp type. Let us consider the action set:

$$\mathcal{A} = \{a_{ij}\},\ i, j \in \{1, \cdots, N\} \tag{4.3}$$

The discrete variable A_t represents the action taken at time t.

- **P** is the probabilistic transition model. In our case, it depends on the object being manipulated. It is composed of the probability distributions over the next state S_{t+1} when an action A_t is taken at a current state S_t using an object O, written $P(S_{t+1}|S_t, A_t, O)$.
- R is a reward signal collected by the agent at each time step. It may depend on the MDP state, the action or both.

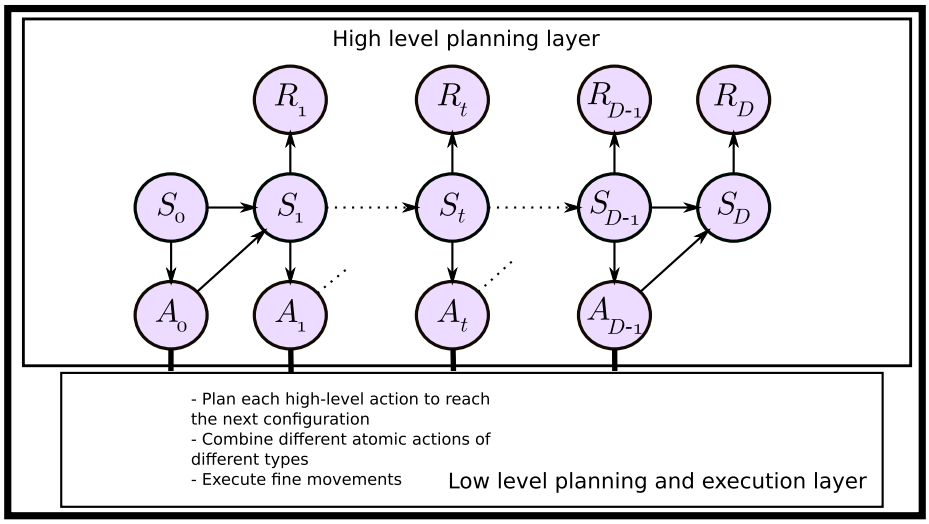


Figure 4.1: An MDP modelling a grasp sequence of duration D with a reward signal.

We are now reformulating in-hand manipulation using the MDP. The robot goes from a grasp type to another using actions. The outcome of an action taken when the robot is in a given state/grasp type is stochastic, it is the probabilistic transition model **P** that encodes the evolution of the system. At the beginning of an in-hand manipulation activity, the object is grasped using the initial grasp type $g_{init} \in \mathcal{G}$. The robot is the agent, it chooses the action performed at each time step. Its goal is to obtain the reward by reaching the final grasp type $g_{final} \in \mathcal{G}$. To reach g_{final}, the robot might have to go through intermediate grasp types and perform successive actions to follow a grasp type sequence, as shown in Fig. 4.2.

The object does not change during a manipulation task. It is a parameter influencing **P**. To keep the presentation simple, we sometimes consider an object specific MDP as in Fig. 4.1. In that case, we will have M independent MDPs.

The duration D of the MDP has to be known because of the static structure of any MDP, $t \in \{0, \cdots, D\}$. We decide to fix the duration of the MDP to 9. It means that the sequences cannot have more than 10 grasps (a duration of 9 corresponds to a sequence of 10 grasp types). Observations of human manipulation revealed that grasp type sequences are never longer than this. The duration is just a parameter to change in the algorithm, it can be changed easily if we need to. Shorter sequences fill the missing steps by nature, using actions that keep the system under the same state with an unconditional success, as illustrated in Fig. 4.3. The grasp sequence

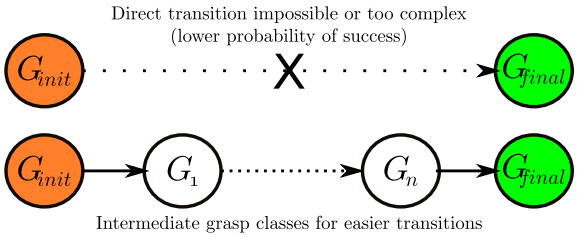


Figure 4.2: Grasp sequence may require more or less intermediate grasps.

illustrated is shorter than the length of the MDP. Actions keeping the system under the same state fulfil the remaining time steps, only obtaining the reward in the last time step. This strategy is also used by [72]. We consider that 10 grasp types in a sequence is a reasonable number. It will be confirmed by the observations made in section 4.3.

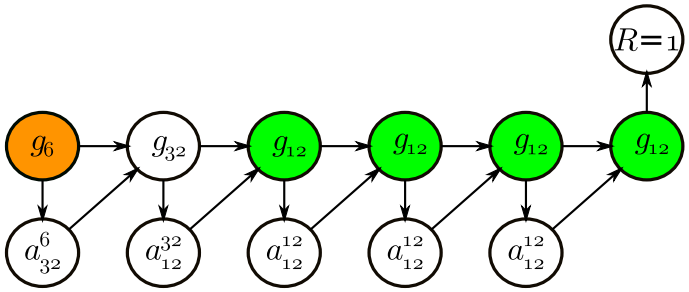


Figure 4.3: MDP with a sequence shorter than the MDP's duration.

In the next sections, we describe each constitutive elements of the MDP with more details: the state (section 4.1.2), the actions (section 4.1.3), the probabilistic transition model (section 4.1.4) and the reward (section 4.1.5).

4.1.2 State

The state is the main constitutive element of our model. We choose very simple states as a first approach: the grasp type introduced as a possible state representation in chapter 3. We do not include in the state the relative position of the hand and object introduced in section 3.5.3. This will surely make our model less accurate. However, as a first approach to validate our method, we choose to proceed using the simplest possible state. It greatly reduces the dimension of the problem. The inclusion of the hand/object pose should be done in a later stage.

A grasp type is a discrete canonical grasp, a way of grasping an object, with characteristics differentiating the different grasp types. The set of grasp types we consider is taken from the taxonomy described in [24]. The set includes 33 grasp types and includes power, precision and intermediate grasps involving a variable number of digits and contact surfaces (tips, pulp or lateral surfaces of the fingers). The list of

grasp types is presented in appendix A. The null state, or failure state ϕ is added to this set to describe the outcome of a failure in the execution of an action.

Thus, the grasp types are simply defined by numbered labels. Our model is the representation of manipulation actions by a sequence of stable grasp configurations linked by local movements making the hand-plus-object reach the next grasp configuration. The set of grasp types has been given in Eq. (4.3) and is completely detailed in appendix A.

We consider that 33 grasp types is a sufficient number of grasps to generate various possible paths to reach a final configuration. To our high-level solution point of view, grasp types are pure discrete entities, but for low-level planning each grasp type has one or more degrees-of-freedom that allow the hand to adapt to the particular object shape through sensory feedback from the low-level layer.

When our solution generates the sequence, the initial configuration of the hand is known, and is usually the current grasp, or no grasp at all. In that case, each initial grasp in a set of possible grasps is considered. More details about this possibility are given in section 4.1.5. From this initial configuration, the initial grasp type has to be identified in the taxonomy. In the same manner, the final configuration depends on the task that has to be accomplished. This final configuration should also be translated into a grasp type, called the task grasp type or the final grasp type.

4.1.3 Actions

The presented method aims at generating the most feasible grasp sequence, with only the first and final grasps known. One or more intermediate grasps may be necessary to achieve the sequence and several different paths may lead to the same goal. However we want to choose the path that is the most likely to succeed. We consider a grasp sequence as shown in Fig. 4.2. Transitions between grasp types are produced by the robot actions taken at each time step of the MDP.

Action a_{ij} represents the atomic in-hand action or the combination of atomic in-hand actions that drive the hand configuration from g_i to g_j. Due to our particular action model, all information is condensed in the transitions and the action does not need to be encoded explicitly to generate the sequence. However, the low-level planning layer needs the mapping between the action a_{ij} and the in-hand atomic actions required to perform the transition from g_i to g_j. The relation between the MDP action and the in-hand atomic actions needs a specific learning process which is not part of the work presented in this manuscript.

An action a_{ij} should only be applied if the hand is in configuration g_i and represents the attempt to reach grasp type g_j. Depending on the grasp types and manipulated objects, these transitions may be hard or easy, comfortable or uncomfortable, efficient or not in terms of energy, prone to failures or not, both in human manipulation and in robot manipulation. The outcome of an action is the next state S_{t+1}, it depends on which action is taken and on the current state S_t. This is encoded in the probabilistic transition model explained in section 4.1.4.

4.1.4 Probabilistic transition model

The above definitions configure an object dependent MDP. **P** holds the ability of the robot or the human being to perform every transition given the context. **P** consists in the probabilities describing the next state S_{t+1} given the action A_t taken at time t and the state S_t at time t. It is the key element of the MDP, it encodes how the system evolves along time given the actions chosen by the robot. For each object in our set, the probabilistic transition model will have different values for the probabilities of the transitions. Given the reasonably low number of object types, this improvement does not enlarge the complexity of the problem in an intractable way. We use multinomial distributions to encode the probabilistic transition model **P**. Considering that O is the object type, the probabilities can be written:

$$P(S_{t+1}|S_t, A_t, O) \tag{4.4}$$

As presented in chapter 3, the transitions depend on several factors either intrinsic or extrinsic. Intrinsic factors are linked to the hand, the object, the abilities of the robot. These factors do not change during a manipulation activity. Extrinsic factors, on the other hand, are likely to change during a manipulation task, such as obstacles and task constraints. In this work, we aim at setting up a model of the intrinsic factors. The extrinsic factors are considered as influencing the task, and then the final grasp type g_{final}. They also influence the low-level layer in its execution, for example when the force that can be applied on the object has to be below a given maximum value. The fact that those factors are jointly represented is not a limiting factor for designing a high-level planner for a human-like robotic system. It is the combination of the numerous factors that matter to achieve the overall optimality of human behaviour and there is no need for the individuation of the factors. It is only at a later stage when the robot actually executes actions that individual factors become observable and can be sorted out.

Theoretically, the dimension of **P** is the total number of probabilities. It is equal to the product of the number of possible states, the number of possible actions, the number of possible next states and the number of objects, then

$N \times (N-1) \times (N-1) \times N \times M = 34 \times 1089 \times 34 \times 9 = 11,329,956$. Due to the nature of our problem, there are several constraints on the probability transition model that limit the dimensionality of the problem.

An action a_{ij} is coherent only if it is used when the system is in state g_i. Moreover, it can only reach state g_j or fail, leading to state ϕ. Because of this, we have, $\forall k \neq i$ and $\forall l \neq j$:

$$P(S_{t+1} = g_j|S_t = g_i, A_t = a_{kl}, O) = 0 \tag{4.5}$$

As in the above case, $k \neq i$ or $l \neq j$, infeasible actions lead with certainty to the null state:

$$P(S_{t+1} = \phi|S_t = g_i, A_t = a_{kl}, O) = 1 \tag{4.6}$$

The null state ϕ is an absorbing state, i.e., when the system is in state ϕ, it will stay in that state for the rest of the sequence. It means that we consider that the system cannot recover from a failure. If a transition in the sequence fails, the sequence is considered as a failure and no reward is received. $\forall i, j, k$:

$$P(S_{t+1} = \phi | S_t = \phi, A_t = a_{kl}, O) = 1 \tag{4.7}$$

$$P(S_{t+1} = g_i | S_t = \phi, A_t = a_{kl}, O) = 0 \tag{4.8}$$

An action that is feasible and leads to the same state cannot fail:

$$P(S_{t+1} = g_i | S_t = g_i, A_t = a_{ii}, O) = 1 \tag{4.9}$$

$$P(S_{t+1} = \phi | S_t = g_i, A_t = a_{ii}, O) = 0 \tag{4.10}$$

Finally, feasible actions can have two outcomes, either the transition succeeds or it fails. Failure is taken *latu sensu*, i.e., the object can fall, the next state is unstable or the transition leads to a non desired state. This is represented as:

$$P(S_{t+1} = g_j | S_t = g_i, A_t = a_{ij}, O = o_m) = p_{ijm} \tag{4.11}$$
$$P(S_{t+1} = \phi | S_t = g_i, A_t = a_{ii}, O = o_m) = 1 - p_{ijm} \tag{4.12}$$

The values of p_{ijm} are thus the important parameters to define in our model. They encode the success likelihood of a transition for a particular object. There are at most $(N-1) \times (N-1) \times M = 33 \times 33 \times 9 = 9,801$ such parameters, with N the size of the grasp type set and M the size of the object set. Learning from scratch all these parameters would require a very long process, therefore we initialize them using human empirical knowledge. It is described in section 4.3.

A probability p_{ijm} is the conditional probability of success of an action a_{ij} when it is consistent with the current state: $S_t = g_i$. In other words, it is the probability to reach grasp type g_j at step $t+1$ given the grasp g_i in the previous stage t and the object o_m. The key part of the problem is to find the values of all p_{ijm}.

We take an incremental approach. In a first stage, we average out the influence of the object and create an object independent model, $P(S_{t+1}|S_t, A_t)$, to sort out impossible grasp transitions (zero probability) that will never be considered in the learning and planning processes. In a second step, we will add the influence of the object.

Let us consider an object independent MDP. Parameters of the probabilistic transition model are just p_{ij} as defined in Eq. (4.13). They can be stored in a matrix called the transition matrix described in Eq. (4.14).

$$p_{ij} = P(S_{t+1} = g_j | S_t = g_i, A_t = a_{ij}) \tag{4.13}$$

$$T_{transition} = \begin{bmatrix} 0 & p_{1,2} & p_{1,3} & \dots \\ p_{2,1} & 0 & p_{2,3} & \dots \\ p_{3,1} & p_{3,2} & 0 & \dots \\ p_{4,1} & p_{4,2} & p_{4,3} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{4.14}$$

Many transitions are impossible due to hand kinematic constraints. The transition matrix will thus be sparse and can be represented with a graph (Fig. 4.4) analogous to the Grasp Transition Graph (GTG) in [81]. Each transition has an associated probability. On the illustration, the graph is undirected meaning that transitions can be made both ways. Generating a grasp type sequence can be seen as finding a path in this graph from the initial grasp type to the final grasp type.

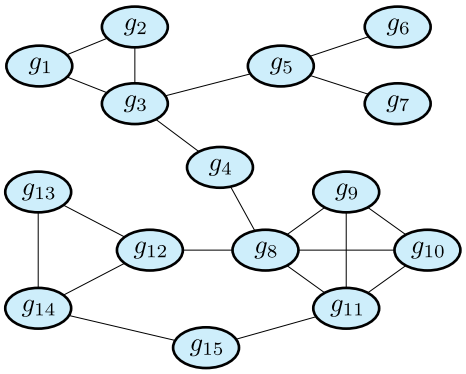


Figure 4.4: Example of a transition graph.

In order to compute the parameters, $33 \times (33 - 1)$ values p_{ij} must be calculated. To limit the amount of data to collect, an initialization is performed as a first step based on an empirical analysis of the individual grasp transitions. The influence of the object will be added. In a subsequent step, the parameters will be finely tuned by additional training data. This is explained with more details in 4.3, 4.4 and 4.6.

4.1.5 Reward

An in-hand manipulation task consists in the application of forces and velocities on a certain object. Its performance depends on the way the hand grasps the object. For instance, the use of a spoon to stir coffee is more efficient when the spoon is held with a tip pinch; the use of a pen to write is more efficient with a writing tripod type of grasp. In this work, we concentrate on in-hand action sequences required to achieve a desired grasp type on the object. Starting from an initial grasp type $g_{init} = g_0$, our objective is to determine the most feasible sequence of intermediate grasps to achieve the task, reaching $g_{final} = g_f$. This goal oriented behaviour can be encoded in an MDP by defining a reward function structure that promotes the achievement of the task. As previously mentioned, without any further constraints on the task, it can be

implemented by a reward function that assigns value 1 to the goal state at the end of the sequence and 0 to all other states ($R_D = 1$ if and only if $S_D = g_f$).

Assuming an arbitrary sequence of actions $\mathbf{A} = \{A_0, \cdots, A_{D-1}\}$, the expected reward is given by:

$$\bar{R} = E\left[R(S_D)\right] = P(S_D = g_f | g_o, \mathbf{A}, O) \tag{4.15}$$

A sequence of actions corresponds to a policy. Let π be the policy associated to $\mathbf{A}$. Once the reward structure is defined and if the probability transition values (p_{ijm}) are known, the solution of the MDP is obtained by choosing actions that lead to high values of the expected reward $\bar{R}$. The recursive nature of $\bar{R}$ enables it to be more adequately represented through the definition of a value function for each state $V^\pi(g_o)$ representing the expected reward obtained when starting at an arbitrary state g_o.

$$\begin{aligned} V^\pi(g_o) &= P(S_D = g_f | S_0 = g_o, \mathbf{A}, O) \\ &= \sum_k P(S_1 = g_k | S_0 = g_o, A_0 = a_{ij}, O) V^\pi(g_k) \end{aligned}$$

Because of the constraints on the state-action transitions defined in Eq. (4.5)-(4.10), that is, action a_{ij} only succeeds in the transition from g_i to g_j with probability p_{ijm}, the only action sequences that return rewards are the ones that form chains starting at the initial state and terminating at the goal state. Therefore, we define a feasible chain, for initial state g_o and final state g_f, as the one with $A_0 = a_{ok}$, $A_{D-1} = a_{lf}$ and whose end state of A_t is the initial state of A_{t+1}. Constraining the action sequence to a feasible chain, the sum in the previous equation disappears:

$$V^\pi(g_o) = P(S_1 = g_k | S_0 = g_o, A_0 = a_{ok}, O = o_m) V(g_k) = p_{okm} V(g_k) \tag{4.16}$$

Unfolding the recursion until the goal state is reached, we can verify that the state value function is given by the transition probabilities along the chain:

$$\begin{aligned} V^\pi(g_o) &= P(S_1 = g_k | S_0 = g_o, A_0 = a_{ok}, O) \\ &\times \prod_{t=1}^{D-2} P(S_{t+1} = g_j | S_t = g_i, A_t = a_{ij}, O) \\ &\times P(S_D = g_f | S_{D-1} = g_l, A_{D-1} = a_{lf}, O) \\ &= p_{okm} \times \prod_{t=1}^{D-2} (p_{t(t+1)m}) \times p_{lfm} \end{aligned}$$

Thus, given the feasible chain of actions $\mathbf{A} = \{A_0, \cdots, A_{D-1}\}$ of a policy π, we can compute the expected reward of the chain by simply multiplying the p_{ijm} probabilities

along the chain. We can compute all the possible chains of actions and find the one with the highest expected reward. It will give the grasp type sequence, the solution of our algorithm.

We can find the solution by finding the maximum expected reward in order to compute the optimal value function V^*. The optimal policy is the sequence of actions that achieves the optimal value in all states (section 2.3).

The reward guides the system towards the task grasp. The solution we build aims at fulfilling the task. From an initial grasp, we are able to quickly sketch a path to the final grasp type related to the task, and assess the feasibility of the transition. In the beginning of a manipulation activity, the object is usually not grasped. The initial grasp has to be generated for the object to be grasped, using grasping techniques (section 2.1.2). Our solution can provide an insight of the achievability of the task and influence the initial grasp choice. Taking into account the task achievability is a very interesting feature for grasping algorithms. The overall behaviour of the robot would be greatly improved by avoiding to grasp the object with a grasp type making the fulfillment of the task very complex and prone to failures. This feature has been implemented in the whole control framework of our robot. It is presented in section 5.2.2.

4.2 Modelling the object

In this section, we will analyse how we can consider the object and justify the use of a discrete set of objects.

Following the conclusion of section 3.6.2, we only consider shape and size in our model of the object. This information is usually given to the robot via its vision module. A vision module is made of one or more cameras, taking pictures of the environment. Efficient algorithms are able to locate and segment the object. At the vision module level, the shape is represented as a cloud of points (point cloud).

At this point, a solution is to have a database of known objects, the vision module recognizes which object of the database is present and the model stored in the database is then used by our algorithm. Otherwise, the object has to be modelled on the fly. This solution is more versatile as it adapts to any object.

Object characteristics are defined in prototypical shapes of object. A highly accurate model is not required for this high-level of planning, which is not the case when controlling fine movements of fingers dealing with contact and stability of a grasp. In [42], 6 basic shapes are defined as combinations of 2 discrete shapes (flat, rounded) and 3 sizes (small, long, large). Vision algorithms can reconstruct an object and build a parametrized shape fitting the point cloud. Parametrized shapes can be a good model for objects. Among the possible parametrized shapes are:

- Superquadrics using 6 parameters for a nearly complete range of objects.
- Cylinders matching most objects in everyday life with 2 continuous parameters.

A set of 6 continuous parameters highly redinscreases the dimension of our problem. Here, we are still aiming at getting the most simple model of objects and

therefore use the cylinder representation. We aim at representing the object using a limited set of reference objects, and classifying the object into one of those categories.

Figure 4.5: The set of objects used for our work.

First, we analysed the compatibility of the grasp types with the different objects. Some grasp types can be unfeasible with a given object. Then we used these results to classify every object into a few classes.

Analysis of human manipulation for object set definition

We conducted a study from a set of real objects to obtain a representative set of objects. First of all, we observed the influence of the object on the grasp type. From a large set of objects, listed for the HANDLE project, we checked the feasibility of each grasp type on each of these real objects, determining which grasp types are valid for a given object. The 18 objects considered are shown in table 4.1.

The analysis revealed that only a limited number of grasps are possible with each object. Some objects have the same set of possible grasps. This gives a natural classification by putting the objects with the same possible grasp types in the same class. These objects influence possible grasps in the same way but grasp sequences, transitions between grasps, may differ a lot even if possible grasp types are the same. The possible grasps of every analysed object are shown in Fig. 4.6. The objects are regrouped so that they form 8 preliminary classes.

Even though grasp classes are the same among objects of the same class, the transition probabilities can differ. In chapter 3, we stated that in-hand manipulation and consequently transitions mainly depend on the object size and shape. We added to the classification the criterion of shape and size using the representation as cylinders. When encountering a new object, the robot can rely only on vision to plan in-hand manipulation. Vision gives only the shape and size of the object. Thus, it should be allowed to distinguish object classes through size.

We checked if objects of the same preliminary class have similar sizes and shapes using a cylinder model with two parameters: diameter and length. The sizes of each of the 18 objects considered are given in Fig. 4.2. Given the dimension differences among the preliminary class *Pen, Spoon, Marker, Test tube, Ladle, Screwdriver, Hammer*,

Pen
spoon
orange
mug (body)
mug (handle)
Ladle
bottle
Hammer
Can
Remote control/phone
Screwdriver
Spray
Rubik's cube
test tube
Sponge
Key
Book
Marker (pen)

Table 4.1: The 18 objects considered.

Object	Possible grasps																						
Key	2	6	7	8	9	14	15	16	20	22	23	24	25	33									
Mug handle	6	7	8	9	25	30	31																
Pen, marker	2	3	4	5	6	7	8	9	13	14	15	16	17	20	22	23	24	25	27	29	31	32	33
Spoon, Test tube	2	3	4	5	6	7	8	9	13	14	15	16	17	20	22	23	24	25	27	29	31	32	33
Ladle	2	3	4	5	6	7	8	9	13	14	15	16	17	20	22	23	24	25	27	29	31	32	33
Screw driver	2	3	4	5	6	7	8	9	13	14	15	16	17	20	22	23	24	25	27	29	31	32	33
Mug body	1	4	6	7	8	9	11	12	13	14	16	17	20	22	26	27	28	31	33				
Phone	2	3	4	5	6	7	8	9	13	14	15	16	17	20	22	23	24	25	27	31	33		
Orange	4	9	11	13	14	15	16	22	25	26	27	28	31	33									
Can	1	4	6	7	8	9	13	14	17	20	22	25	27	31	33								
Bottle	1	4	6	7	8	9	13	14	17	20	22	25	27	31	33								
Book	4	6	7	8	9	10	12	13	14	15	16	17	18	20	22	23	25	29	30	32	33		

Figure 4.6: 8 classes of objects depending on the possible grasp types of each object

we split them into two classes, depending only on dimensions. The other preliminary classes show a consistency in size and shape and are kept unchanged. The result is a set of 9 object classes shown in Fig. 4.3.

After an analysis of most of the objects encountered in everyday life, we managed to regroup all of these objects in 9 classes using parameters of cylinder models. This forms a set of 9 object classes, which representative objects are shown in Fig.4.5. Objects of the same classes do not have to be absolutely similar in shape and size. Being part of the same object class means that they are manipulated in the same way with the same probability of success when performing transitions between grasp types.

Object	Diameter	Length
Pen	1	15
spoon	1	9
orange	7	7
Mug body	7	10
Mug handle	1	4
Ladle	2	15
bottle	9	25
Hammer	2	25
Can	7	12
Remote control/phone	5	14
Screwdriver	2	10
Spray	5	13
Rubik's cube	6	6
test tube	2	17
Sponge	10	2
Key	1	6
Book	15	2
Marker	2	15

Table 4.2: The dimensions of each object, in centimeters.

Key
Mug handle
Pen, Spoon, Test tube
Ladle, Marker, Screwdriver, Hammer
Mug body
Phone, Remote control
Orange, Rubik's cube
Book, Sponge
Bottle, Can, Spray

Table 4.3: The 9 classes of objects used.

Object	Mean Diameter	Variance Diameter	Mean Length	Variance Length
Key	7.8	4.16	58.0	106.0
Mug handle	13.67	10.89	56.67	155.56
Pen	14.99	15.51	130.28	209.37
Ladle	22.83	28.37	214.05	1174.66
Mug body	74.17	53.47	105.33	610.56
Phone	40.0	66.67	133.34	2422.22
Orange	63.75	33.13	63.75	33.13
Book	135.0	250.0	14.63	26.59
Bottle	58.67	309.5	140.34	1127.0

Table 4.4: Table of the Gaussians parameters for each class of our object classification.

Classifying unknown objects

Encountering new objects means that we have to classify them according to our list of objects. We use a simple representation of objects using the cylindrical model of objects. This representation implies only two parameters : diameter and length. If a point cloud, a mesh or a parametrized shape is given by the vision modules, modelling it as a cylinder will not require a great effort.

We use a Naive Bayes Classifier (NBC). This classifier uses a Gaussian distribution on each parameter for each object type. The product of the probabilities for diameter and length gives the probability that the object encountered belongs to the class. The class with the highest probability is chosen.

To compute the parameters of the NBC, we selected a few real objects of each class and computed a Gaussian distribution over each parameter. Gaussian distributions are defined by a mean value and a variance. The results of this analysis are presented in table 4.4.

All the Gaussian distributions have been drawn on the same graph in Fig. 4.7: the two horizontal axes are the diameter (x) and the length (y) in millimeters. The vertical axis (z) is the probability (unscaled) of each class. The resulting area of detection has been calculated and the results are shown on a histogram in Fig. (4.8). This shows that every class has an area of recognition and can get the highest probability given that the object has the appropriate dimensions.

4.3 Initialization via empirical guess

In the MDP set up in section 4.1.1, we needed to find the probabilities, noted p_{ijm}, forming the probabilistic transition model $\mathbf{P}$. Transition probabilities will not exhaustively be validated through numerous tests related to hand mechanics, object, task and environment, but initialized from human empirical knowledge and refined through learning from human trials. The parameters of the probabilistic model are

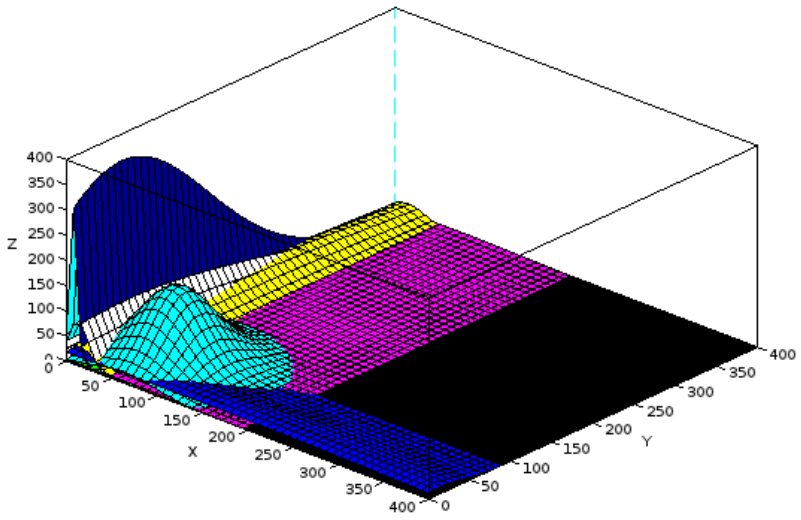


Figure 4.7: Gaussians of the NBC drawn on the same graph.

set by an empirical analysis of every transition, in order to assess its chance of success. This analysis is based on the observation of human in-hand manipulation movements.

We are presenting here how we obtained these values in the first section 4.3.1. The second section 4.3.2 gives the first observations made from these results. The third section 4.3.3 extends the probabilities to initialize an object dependent model.

4.3.1 Estimation of the object independent model

First, we did not consider the object influence. The probabilities, instead of being p_{ijm}, are p_{ij}. To give coarse values to p_{ij}, we first classified the transitions in terms of difficulty or probability of success. A human expert considered every transition from grasp g_i to grasp g_j and assessed the difficulty using three categories: possible and easy (category a), possible but complex (category h) or impossible (category N).

As this model is object independent, we had to avoid the influence of the object. Thus, we did not consider any specific object. The difficulty of a transition was assessed on multiple objects and the easiest transition determined the class of the transition. This method avoided restricting the possibilities. For example, if a transition could be performed with a complex combination of actions with object A and could be performed with an easy combination of actions with object B, the transition was classified as easy so that no unwanted limitation appeared.

To evaluate the difficulty of a transition, the number and type of sub-actions composing it were considered. We refered to the atomic in-hand actions defined in section 2.2.1. The rules chosen for the classification process are simple and intuitive.

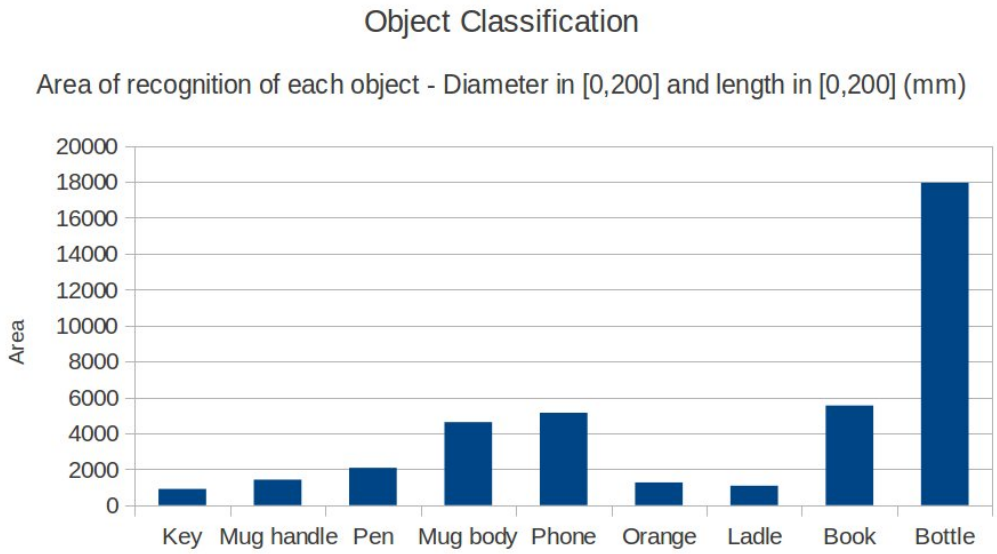


Figure 4.8: Areas of recognition of the 9 classes of objects using the NBC.

To convert this classification into probabilities, we chose to give a sensible probability of success to categories of difficulty. A transition is:

- Impossible: if no direct transition is possible, i.e. any tentative of transition from grasp g_i to grasp g_j uses an intermediate grasp (as defined in the grasp type list), like on the example shown on Fig. 4.9. During the transition, if the object has no contact with the hand, even briefly, the transition is also considered impossible, as it is not considered as in-hand manipulation.

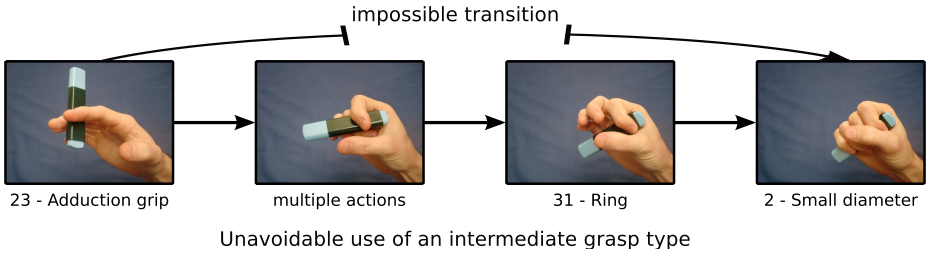


Figure 4.9: Example of an impossible transition.

- Easy: if the direct transition from grasp g_i to grasp g_j uses a single atomic action (example on Fig. 4.10).
- Complex: if the direct transition from grasp g_i to grasp g_j uses a combination of two atomic actions or more, like on the example shown on Fig. 4.11. During the transition, if the object is in contact with the hand on one plane surface, even briefly, and then in an unstable state, the transition is considered complex.

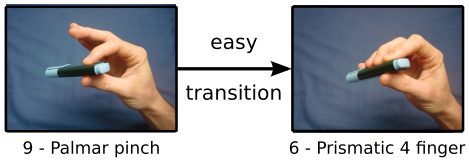


Figure 4.10: Example of an easy transition.

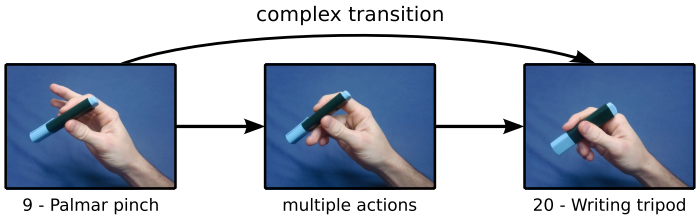


Figure 4.11: Example of a complex transition.

The results from this empirical classification are presented in appendix B. From the taxonomy in [24], grasps 19 (distal type) and 21 (tripod variation) address very specific objects (scissors and chopsticks), they were not taken into account in the initialization, every probability of success involving these grasps was set to 0. Once every transition difficulty had been assessed using this process, the next step was to convert this classification into probabilities. The easy transitions will be used more often than difficult ones. We chose to attribute a fixed probability index to each category of difficulty as follows:

- Impossible: probability of success $p_{ij} = 0$
- Easy: probability of success $p_{ij} = 0.8$
- Complex: probability of success $p_{ij} = 0.2$

redThese values are fixed and arbitrary. The ratio between the probability of success of an easy transition and the probability of success of a complex transition can be understood as the number of easy transitions above which a complex transition would be preferable. We choose that 6 easy transitions are equivalent to a complex transition. We state that an easy transition should have a probability between 0.5 and 1, and a complex transition should have a probability of success between 0 and 0.5. The values we used (0.8 and 0.2) satisfy all these requirements.

These values are just rough estimations. Using them should allow us to generate a plausible though inaccurate model of in-hand manipulation. The learning process will finely tune these probabilities in a later stage of the work (sections 4.4 and 4.6).

4.3.2 Key grasp identification

This section presents some interesting observations from the object independent probabilistic transition model resulting from the empirical initialization explained in section 4.3. We conducted a study to identify grasps likely to be used very frequently for in-hand manipulation, that we call “key grasps”.

We performed the expert guess of grasp transition probabilities of success, independently from the object. This gives a Grasp Transition Graph shown in Fig. 4.12. The transition probabilities p_{ij} are the edges of the graph and grasp types are the nodes. As noticed previously, grasps 19 (distal type) and 21 (tripod variation) are not taken into account.

On this Grasp Transition Graph can be observed an interesting fact: four groups of grasps have been identified.

- Group A: each grasp from Group A is connected to grasps from every other groups.
- Group B: grasps from Group B are all connected to each other and only connected with grasps of Group A.
- Group C: grasps from Group C are not connected to grasps from other groups, except from Group A.
- Group D: grasps from Group D are not connected to grasps from other groups, except from Group A.

On the transition graph, grasps from group A appear to be more connected than grasps from other groups. They have to be used as intermediate grasps for any sequence between two grasps of different groups and are more likely to be "key grasps". The graph illustrates the fact that some grasps may be more used for in-hand manipulation than others. A simple study was conducted to confirm it. We used two considerations, the linkability of each grasp and the estimation of grasp frequencies.

- Linkability: Once the empirical values of possible transitions have been assessed, some grasps appear to be more connected, more linkable than others. The linkability is, for each grasp, the number of possible next grasps weighted by the probability of success of transitions $P(G_t|G_{t-1})$. In this way, easy transitions will count more than complex ones.
- Estimation of the grasp frequency: It tries to reflect the frequency of use of every grasp, regardless of the object or the task. To estimate this frequency, an algorithm of random grasp sequences, according to transition probabilities, has been run: from an initial grasp, a next grasp is chosen among the possible ones following the p_{ij} values. From this next grasp, the following grasp is chosen in the same way, and so on. The simulation gives a succession of grasps with probable transitions between each other. A count of occurrences of each grasp is made. It can be translated into the estimation of a grasp frequency by normalizing it over the total number of grasps observed. The algorithm is run during $10,000$ grasps. At this point, grasp frequencies are stabilized.

Once the linkability and the estimation of frequency are obtained, we normalize the indexes and sum them up for each grasp. The results are presented in a bar chart in Fig. 4.13. The normalization is simple, we multiply the linkability indexes of each

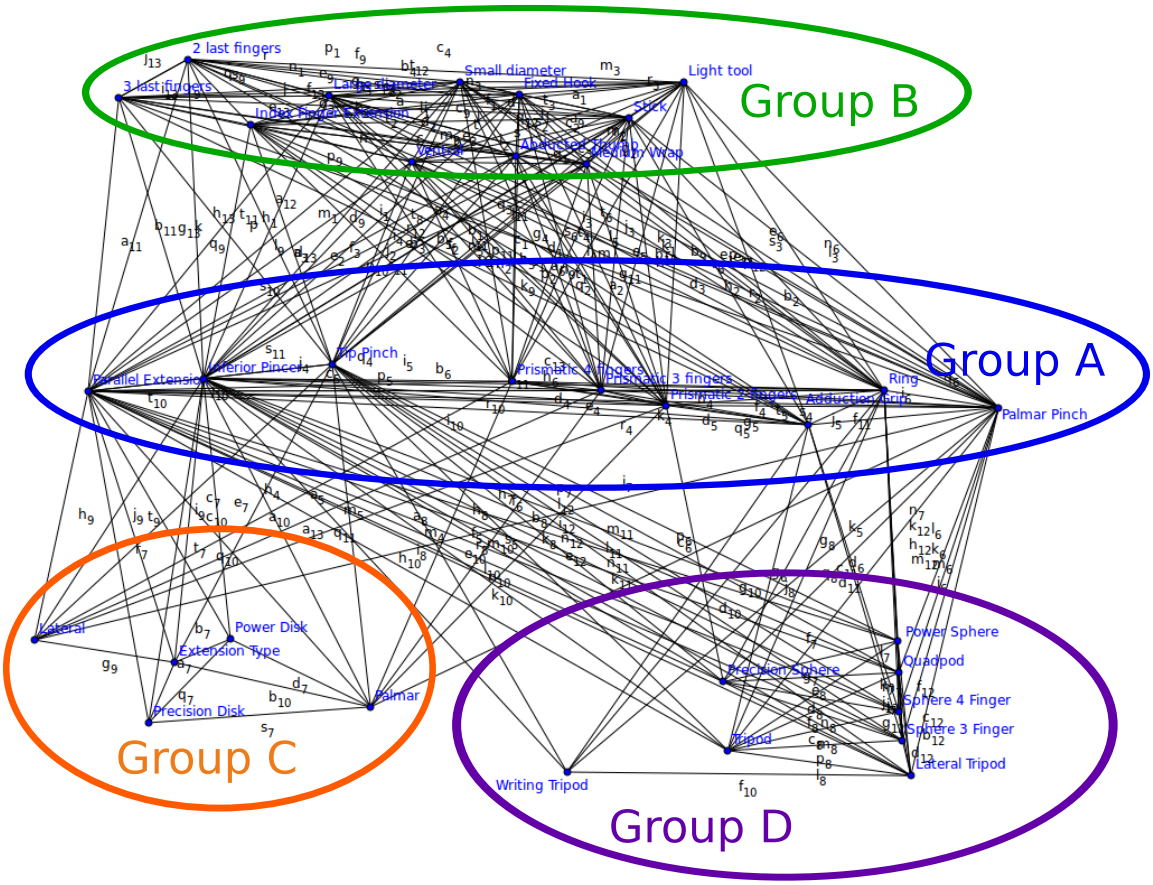


Figure 4.12: Object independent Grasp Transition Graph resulting from the empirical analysis. Each transition has an associated probability.

grasp type by a factor so that the sum of all linkability indexes is 100. We do the same for the estimations of grasp frequency. Then, the two indexes are comparable.

According to this study, key grasps appear to be: 9 (Palmar Pinch), 22 (Parallel Extension), 24 (Tip Pinch), 31 (Ring) and 33 (Inferior Pincer) (Fig. 4.14). Except for grasp 22, these key grasps are 2 fingers grasps, which allows a lot of in-hand movements (rotations, rolling, actions with the three other fingers). This is intuitively consistent with their identification as key grasps and reinforces the validity of the empirical analysis. Such conclusions should be observed on human behaviour to be validated. It would require a huge number of trials to average the results for all tasks and objects (or at least a large number of them). We choose not to look deeper in this study, this section content is just additional observations not used in further work but giving a view on the results of the empirical analysis conducted.

4.3.3 Extension to the object dependent model

The initialization of the probabilistic transition model presented in section 4.3.1 is object independent. We aim at taking into account the object influence to generate the subgoal sequences. We use the set of 9 objects defined in section 4.2, representative

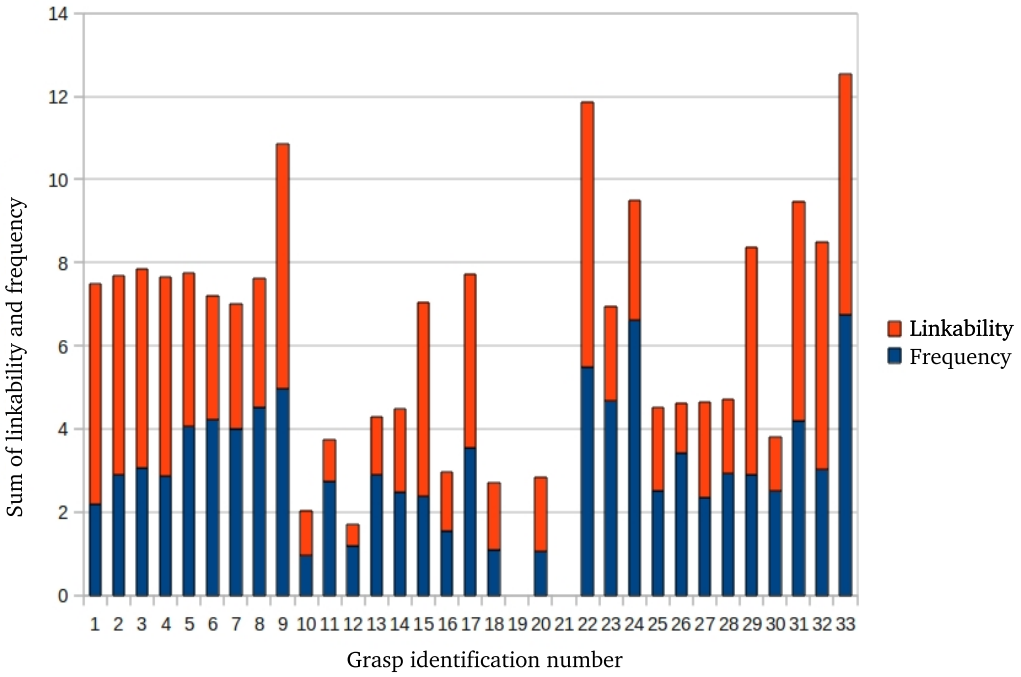


Figure 4.13: This bar chart shows the comparison of linkability and grasp frequency between grasps and allows to identify the "key grasps".

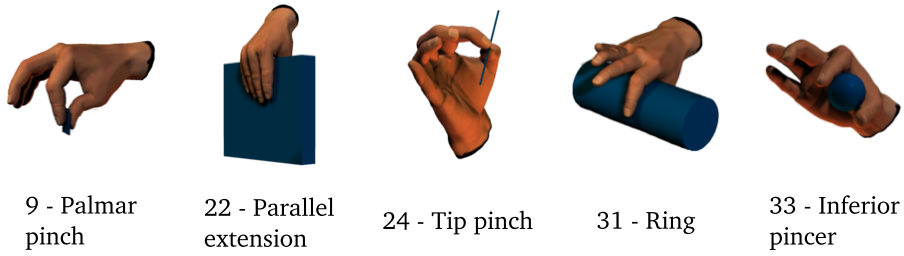


Figure 4.14: Illustration of the "key grasps".

of the objects encountered in everyday human life. The analysis of the possible grasp types for each object of the set was presented in Fig. 4.6. Some grasp types are not feasible for a given object.

The extension of the object independent model obtained from empirical observation of transitions to an object dependent model is simple. For each object o_m, we initialize the object dependent model with the object independent model: $p_{ijm} = p_{ij}$, $\forall m, i, j$. Then, if one of the grasp types involved in the transition, g_i or g_j, is not compatible with the object o_m according to the analysis presented in section 4.2, then the probability of success is set to 0: $p_{ijm} = 0$.

Except for the grasp type/object compatibility consideration, the probabilities of transitions are the same for the different objects after the initialization. The learning process will take into account the influence of the object and tune the probabilities differently from an object to another.

4.4 Learning by self-exploration

The ideal way to generate the grasp sequences is to know the probabilistic transition model **P**. Then, just by adapting the reward, standard techniques would generate the optimal policy. The probabilistic transition model **P** has been roughly initialized (section 4.3), it should be refined by learning, using data obtained from the robot own experience. The parameters characterizing our model, p_ijm are learnt. The aim is to make them reach optimal values, or converge toward values that enable to generate the appropriated sequences.

By learning **P**, the robot explores its abilities and learns what it can do, the outcome of its actions, just as a human child does. During the learning process, the robot tries to perform actions to go from a grasp type g_i to a grasp type g_j using object o_m. This transition has a probability of success p_{ijm}. By observing its own success and failures, the robot is able to refine the value of p_{ijm}.

Formulating it using the MDP, an action a_{ij} corresponds to a transition from a grasp type g_i to another g_j. This action can succeed and lead to the targeted next grasp g_j, or fail and result in the null state ϕ. It is possible to consider that an action can lead to another state. We do not take this possibility into account, to keep our method simple. Then, the only possible outcomes of an action a_{ij} are success by reaching g_j or failure by reaching ϕ. Then, the probability of success and the probability of failure sums to 1. If during learning, we notice that the robot reaches a different grasp regularly, we will consider improving our algorithm and consider this possibility. The probabilities of success and failure are written as:

$$P(S_{t+1} = g_j | S_t = g_i, A_t = a_{ij}, O = o_m) = p_{ijm}$$

$$P(S_{t+1} = \phi | S_t = g_i, A_t = a_{ij}, O = o_m) = 1 - p_{ijm}$$

The principle of the learning process is simple, we count the number of times the transition was attempted: Na_{ijm}. The number of successes Ns_{ijm} and the number of failures ($Na_{ijm} - Ns_{ijm}$) are recorded. The ratio of the number of successes over the number of attempts gives the probability of success of the transition.

$$p_{ijm} = \frac{Ns_{ijm}}{Na_{ijm}} \tag{4.17}$$

We have to integrate the prior knowledge provided by the empirical analysis presented in section 4.3. We can achieve this using an imaginary number of occurrence ha_{ijm} equivalent to a number of attempts of the action a_{ijm}. The number of imaginary observed successes hs_{ijm} is calculated from the probability of success given by the empirical analysis. Let us call this probability ${p_{ijm}}^{initialization}$:

$$hs_{ijm} = ha_{ijm} {p_{ijm}}^{initialization} \tag{4.18}$$

$$p_{ijm} = \frac{hs_{ijm} + Ns_{ijm}}{ha_{ijm} + Na_{ijm}} \tag{4.19}$$

We can update the probabilities of $\mathbf{P}$ with Eq. (4.19), choosing the importance of the empirical knowledge compared to the knowledge obtained from learning.

To perform the learning process, it is mandatory that the robot has a low-level of control able to plan and execute the transitions. It means that for a given action a_{ij} using an object o_m, the robot knows which atomic in-hand actions it should use and is able to plan and execute them. The involved grasp types g_i and g_j are not fully detailed grasps. The robot should be able to generate complete grasp representations associated with g_i and g_j.

The learning process can be performed using a human operator to analyse the outcomes of the action attempted by the robot. But the process can be made automatic. If the robot is able to detect the current grasp type autonomously from the proprioceptive information and its sensing abilities, then it is able to attempt a transition, detect the outcome automatically and assess if g_j has been reached. Going even further, if the robot is able to perform a grasp of the object using the initial grasp type g_i on its own, the learning process can be made fully automatic. The robot can try to perform a transition, assess the outcome, update $\mathbf{P}$, and start again. The empirical analysis has been made considering human-like abilities of the robot. Some probabilities can be manually set to 0 if the robot is known to be unable to perform these transitions.

Until the last phase of the HANDLE project, the low-level planning layer of the robot was under development. Learning the probabilistic transition model $\mathbf{P}$ was impossible. We had to find another possibility to learn the behaviour of the robot. An alternative solution would be to learn from human demonstration instead of using the robot. A human being could be asked to perform each transition, and the number of successes and failures could be counted. A human being is able to perform the transitions between different grasp types. The probabilistic transition model obtained with this method would not be perfectly adapted to the capabilities of the robot, even if the hand is antropomorphic as explained in section 5.1.2. But this is not the main problem to learn from demonstration. A human being, when manipulating, is nearly optimal. Its learning process has seen an incredible amount of data and consequently, it hardly ever fails. When the transition is difficult or impossible, the subject uses intuitively an intermediate configuration, implying that the observed transition is not the one that has been demanded. This means that we cannot learn in a straightforward manner the probability of success of transitions from humans. What we observe from humans is exactly what the subject intended to do, in other words, we observe the optimal policy. However, policy can be learnt from human demonstration. This is the topic of the next section (4.5).

4.5 Stochastic policy

The policy is the solution of the MDP, it gives the sequence of actions. If we had a proper probabilistic transition model $\mathbf{P}$, we could be able to generate the policy on the fly, using the value function. As explained in section 4.4, we cannot learn $\mathbf{P}$, so we use learning from human demonstration. In section 4.5.1, we formulate the

representation of the policy, and in section 4.6 is detailed the process used to learn the policy from human demonstration.

4.5.1 Definition of the policy

For MDPs, it is common to represent the decision making process as a policy function that chooses actions according to the current state. The rationale for the adoption of a stochastic policy model (probability distribution over actions) instead of a deterministic one (single choice on a given state) has to do with the fact that humans have a certain variability in the choice of actions due to a multitude of factors (tiredness, stress, attention level, etc.). It highlights that several sequences are possible and feasible for the same manipulation context. Considering our robot, the low-level abilities is only assessed by probabilities of success of transitions. This approximation may cause the generation of non-optimal or unfeasible paths. Then, we need to keep the information about other possible solutions, which demands a probabilistic representation of the policy. Under probabilistic representations, active learning techniques can be used to design experiments that maximize the expected model improvement through learning (section 4.6.2).

Given the object (O) and task (T), the policy is written as $\pi^{OT}(A_t, S_t)$ and represents a probability distribution over actions at each state. This reflects the probability that an agent performs some action A_t at state S_t, which depends on the current object and task:

$$\pi^{OT}(A_t, S_t) = P(A_t|S_t, O, T) \tag{4.20}$$

The state value recursion, under a particular policy, is now an expected value over the action distribution induced by the policy :

$$V^{\pi^{OT}}(g_o) = \sum_k \pi^{OT}(a_{ok}, g_o)$$
$$\times P(S_1 = g_k|S_0 = g_o, A_0 = a_{ok}, O)V^{\pi^{OT}}(g_k)$$

As Eq. (4.20) suggests, the policy can be encoded as a Bayesian network $P(A_t|S_t, T, O)$ (Fig. 4.15). A_t is the action at time t, S_t is the state at time t, O is the object and $T = g_f$ is the task. The BN represents the fact that the action node depends on the parent nodes which are the current state, the task (goal state) and the manipulated object. This consists in a multinomial table with a set of weights:

$$q_{ijfm} = P(A_t = a_{ij}|S_t = g_i, T = g_f, O = o_m) \tag{4.21}$$

4.5.2 Optimal Policies

Optimal policies are the ones that maximize the expected reward. At each time step, the agent should choose an action that leads to a sequence of states with maximal

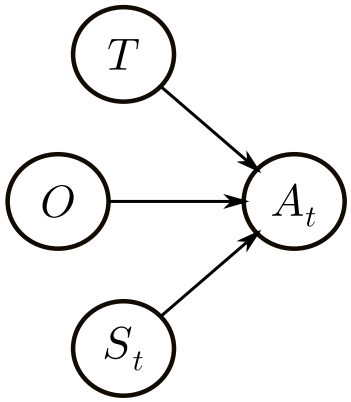


Figure 4.15: Bayesian Network modeling the policy of the MDP.

pay off, i.e., maximum state value:

$$a_{ok} = \text{argmax}_k P(S_1 = g_k | S_0 = g_o, A_0 = a_{ok}, O) V^{\pi^{OT}}(g_k)$$

This is the so-called greedy approach that actually leads to the optimal policy π^{OT^*} and optimal state value function V^{OT^*} [65]. Let us define the optimal action-state value function $Q^*(A, S)$ representing the expected reward of choosing action A in state S:

$$Q^*(A_0 = a_{ok}, S_0 = g_o) =$$
$$P(S_1 = g_k | S_0 = g_o, A_0 = a_{ok}, O) \times \max\left(V^{OT^*}(g_k)\right)$$

The optimal policy should choose actions according to the values of Q^{OT^*}. In a stochastic setting, the highest the value of $Q^{OT^*}(A, S)$ is, the highest the probability of choosing action A in state S should be. In this work, we choose:

$$\pi^{OT^*}(A, S) \sim Q^{OT^*}(A, S) \tag{4.22}$$

The next section will present how we gave values to the policy using the empirical initialization of the probabilistic transition model $\mathbf{P}$ presented in section 4.3. We will initialize the policy in the next section and then refine it using a learning process (section 4.6).

4.6 Learning the policy from humans

Proper learning of $\mathbf{P}$ would consist in having the robot trying to perform each transition, as explained in section 4.4. This is possible only with a real robot. Learning from the robot was not possible until the very end of the HANDLE project, when all the low-level algorithms were implemented for the final demonstration.

A simulation of the robot running on a computer could be used for learning. The feasibility of this option needs to be investigated first. A whole simulation of

the robot, with multiple contact points, all the dynamic effect, friction, deformable contacts and the low-level algorithms might imply a very long time of computation. This depends on the precision that is required to simulate in-hand manipulation realistically. This process could be used if the low-level algorithms are implemented, which was not done until the end of the project. The set up of the whole system's simulation represents a large working effort. The main benefit would be that the trials could be performed automatically on the computer, making the samples much easier to collect than on the real robot. We decided to make our system learn from human demonstration instead. Learning $\mathbf{P}$ in a straightforward manner is impossible, so we directly learn the policy that can be observed in human in-hand manipulation (Fig. 4.16). Considering dimensions here, we realize that learning $\mathbf{P}$ is more efficient than learning the policy as the size of $\mathbf{P}$ is $9 \times 33 \times 33 = 9,801$ whereas the size of the full policy is $9 \times 33 \times 33 \times 33 = 323,433$.

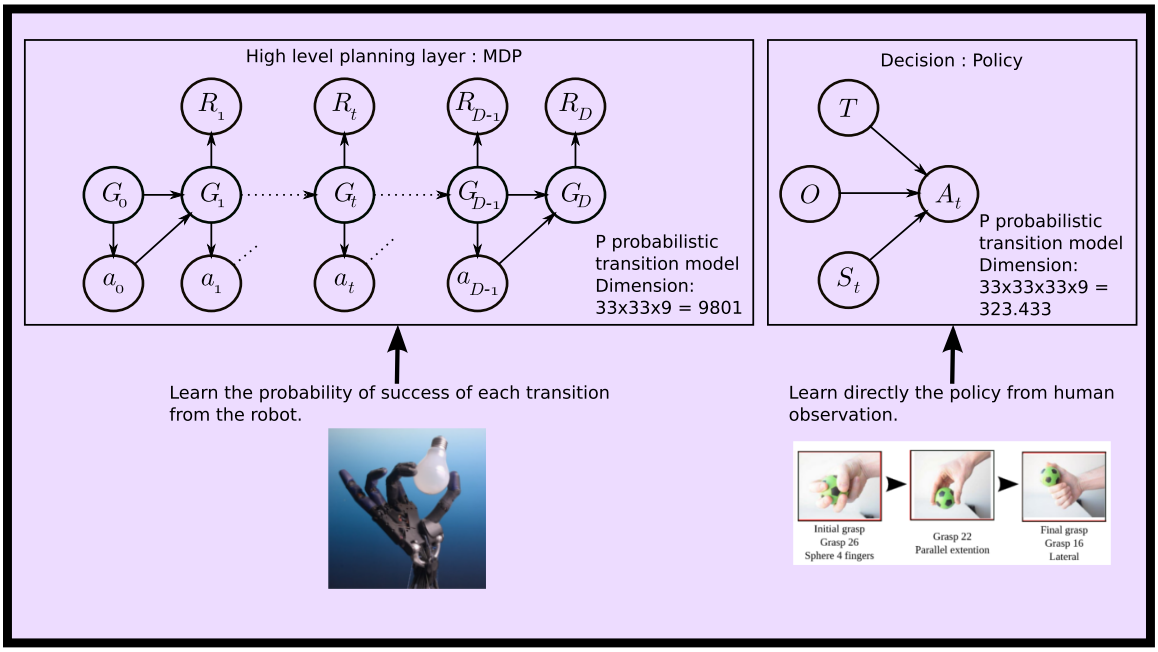


Figure 4.16: Illustration of learning $\mathbf{P}$ or the policy.

4.6.1 Policy initialization

In section 4.3, a probability of success of every transition was assessed through an empirical analysis. With these data, we are able to build a rough probabilistic transition model $\mathbf{P}$. Depending on the object, some probabilities are set to 0: the probabilities of transitions involving grasps incompatible with the object. This is a way to avoid learning all values from scratch. Afterwards, the values have to be tuned finely, by learning it from human recordings.

Using these initialized probabilities, we can accordingly generate a policy that will be used as an initialization to the policy learning process. The policy is made of

multinomials written in Eq. 4.21: $q_{ijfm} = P(A_t = a_{ij}|S_t = g_i, T = g_{final}, O = o_m)$. For every set of parents (S_t, T, O), we try every feasible chain of actions corresponding to Eq. (4.16). Several sequences are found. Only keeping the one with the highest expected reward is possible but would disregard solutions of high-but-not-the-best probability of success. So we compute the value $Q^{OT*}(A, S)$ as in Eq. (4.22), for each set of parents, using the MDP with the values of p_{ijm} obtained in section 4.3. We update q_{ijfm} as $Q^{OT*}(a_{ij}, s_i)$. To have a multinomial distribution, these values are normalized over A_t so that $\sum_j (q_{ijfm}) = 1$.

Empirical estimation of $\mathbf{P} : p_{ijm}$;
for *each set of parents* $(S_t = g_i, T = g_f, O = o_m)$ **do**
  Compute every possible sequences using initialized $\mathbf{P}$;
  for *each first action* a_{ij} **do**
    Compute $Q^{mf*}(a_{ij}, g_i)$;
  end
  Normalization: $q_{ijfm} = \frac{Q^{mf*}(a_{ij}, g_i)}{\sum_k Q^{mf*}(a_{ik}, g_i)}$
end

Algorithm 1: Initialization of the policy.

The policy is used to generate the sequence of subgoals. It provides the best action to choose at each time step with state $S_t = g_i$, it is the action a_{ij} with the highest $Q^{OT*}(a_{ij}, s_i) = q_{ijfm}$. Assuming the success of the action, $S_{t+1} = g_j$, we choose the action in the next time step in the same way. The sequence of actions generated is the one that maximizes the expected reward.

We do not know the length of the sequence and then the length of the MDP. But as the action that keeps the same grasp has a probability of success equal to one, if the goal can be reached in fewer steps than the length of the MDP, this solution is shown with stationary actions in it, as explained in section 4.1.1. This gives several equivalent solutions that are considered as a single solution.

The policy obtained by the work presented in this section is used as initial policy, and refined using learning from the observation of human policy following the learning process explained in the next section (4.6.2).

4.6.2 Policy updating

To update the policy probabilities with experiments, we used multinomial updates with Dirichlet priors [34]. The method consists in counting how many times an action a_{ij} has been observed: $N_{ij}(S_t, T, O)$ in a set of $N_i(S_t, T, O)$ when in state S_t, with task T and object O. A set of positive hyperparameters $h_{ij}(S_t, T, O)$ ($h_i(S_t, T, O) = \sum_j h_{ij}(S_t, T, O)$) defines the prior knowledge on the process, in our case, the initialization: $h_{ij}(g_i, g_f, o_m) = q_{ijfm} \cdot h_i(g_i, g_f, o_m)$. Intuitively, $h_i(S_t, T, O)$ represents the number of imaginary cases in which each set of parents (S_t, T, O) has been observed and can be used by the human expert to encode its certainty on the process. If the certainty is high, this number should be high, otherwise low. The probability of a transition is given by:

$$q_{ijfm} = P(a_{ij}|g_i, g_f, o_m) = \frac{h_{ij}(g_i, g_f, o_m) + N_{ij}(g_i, g_f, o_m)}{h_i(g_i, g_f, o_m) + N_i(g_i, g_f, o_m)} \tag{4.23}$$

We can update the policy with Eq. (4.23), choosing the importance of the empirical knowledge compared to the knowledge obtained from learning by setting the number of imaginary samples corresponding to the initialization $h_i(S_t, T, O),\ \forall i$.

Active learning

As introduced in section 2.4, we can do better than update the policy with random recordings of human in-hand manipulation. As it is possible to choose which samples are recorded from human subjects, we can select the trials that would bring the most valuable knowledge to our probabilistic distribution.

An interesting method to do this is presented in [66]. A function representing the probable reduction of risk of loss of information is calculated for each set of parents (a query), and the one reducing the risk the most is selected and sampled in the next experiment.

We used a similar technique but with a different, much simpler function. The probable risk of loss of information in [66] is complex to intuitively analyse. Our previous knowledge is obtained from an empirical analysis, which we assume is sufficient to give the appropriate policy for obvious cases where one solution is much more likely to succeed than the others. Learning is preferably used to finely tune the policy in the cases where it is unclear whether an appropriate solution should be chosen. The concept of entropy describes this uncertainty. For every set of parents, we compute the entropy $H(g_i, g_f, o_m)$ over feasible actions. Feasible actions are actions a_{ij} with $q_{ijfm} > 0$. Let us define this set of actions $\mathcal{F}_{ifm} = j_1, j_2, ..., j_{f_a}, j_k \in \mathcal{G}$. f_a is the size of the set of possible actions.

$$H(g_i, g_f, o_m) = \sum_{j \in \mathcal{F}_{ifm}} -(q_{ijfm}) \cdot \ln(q_{ijfm}) \tag{4.24}$$

To make the entropies comparable between sets of possible actions of different sizes, we normalize the entropies using the entropy of the uniform distribution of each size and obtain, after simplification, comparable entropy parameters.

For the uniform distribution $q_{ijfm}^{uniform} = q_u,\ \forall j \in \mathcal{F}$. And we know that $\sum_j q_{ijfm}^{uniform} = 1$, then $q_u = \frac{1}{f_a}$. As a consequence:

$$\begin{aligned} H_{uniform}(g_i, g_f, o_m) &= \sum_{j \in \mathcal{F}_{ifm}} -(q_u) \cdot \ln(q_u) \\ &= f_a \cdot (-q_u) \cdot \ln(q_u) \\ &= f_a \cdot (-\frac{1}{f_a}) \cdot \ln(\frac{1}{f_a}) \\ &= -\ln(\frac{1}{f_a}) \end{aligned}$$

$$
\begin{aligned}
H'(g_i, g_f, o_m) &= \frac{H(g_i, g_f, o_m)}{H_{uniform}(g_i, g_f, o_m)} \\
&= -\frac{H(g_i, g_f, o_m)}{\ln(1/f_a)} \\
&= \sum_{j \in \mathcal{F}_{ifm}} q_{ijfm} \cdot \frac{\ln(q_{ijfm})}{\ln(1/f_a)}
\end{aligned}
$$

$$
H'(g_i, g_f, o_m) = -\sum_{j \in \mathcal{F}_{ifm}} q_{ijfm} \cdot \ln(q_{ijfm}) \ln(f_a) \tag{4.25}
$$

The sets of parents of the highest entropy parameters H' are the next samples to record from human in-hand manipulation.

4.7 Chapter conclusion

We have proposed a model for the high-level generation of human-like in-hand manipulation for a dexterous robotic hand using an MDP. Actions are represented as transitions between canonical grasp types from human studies. With this formulation, the grasp sequence generated depends on the task and on the object. Initial and final grasps are provided to the algorithm which generates automatically the optimal grasp sequence that, from the initial grasp, drives the hand to the final one. The identification numbers and names of grasps are taken from [24]. The method can adapt to a very large set of objects and works for any task as all it needs is an initial and a final grasp types given by a higher hierarchical level. The initial grasp can be empty, in which case the algorithm can rank the best initial grasps to accomplish the task.

The probabilistic transition model $\mathbf{P}$ encodes the evolution of the system depending on the action, the current grasp type and the object. An empirical estimation of the transition probabilities forming $\mathbf{P}$ has been done. We conducted the analysis of the possible grasps for each object, extending the validity of the probabilities to the whole set $\mathcal{O}$. However, $\mathbf{P}$ could not be refined by the robot through learning by self-exploration. We had to learn from human demonstration, but as humans nearly always succeed, we can only observe the sequence of actions, and then the policy. We presented a probabilistic formulation of the policy that is initialized using our first guess of $\mathbf{P}$ and then refined by learning from human demonstration. Using this policy, we can generate the sequence of grasp types, using it as a subgoal path for in-hand manipulation planning.

Once the policy is known and stored, the optimal sequence is found in less than a hundredth of a second with a standard computer (Intel®Core™2 CPU with 2.0Gio RAM), which is fast enough to plan and replan grasp sequences online during an

in-hand manipulation action. Using a simple model without object dependency, the grasp sequence generation can be performed on-line. The obtained sequences look natural when performed by a human being but this point still needs to be validated rigorously once the complete model is available.

In case the object is not grasped yet, the initial grasp of our algorithm can be a list of candidate initial grasps. The reachability of the final grasp can be used in this case to influence the initial grasping of the object.

An example of a result obtained from the grasp sequence generator is illustrated in Fig. 4.17 with initial grasp type 1-Large diameter and final grasp type 27-Quadpod. Another example of sequence is illustrated in Fig. 4.18. The initial grasp type is 17-Index finger extension and the final grasp type is 18-Extension type.

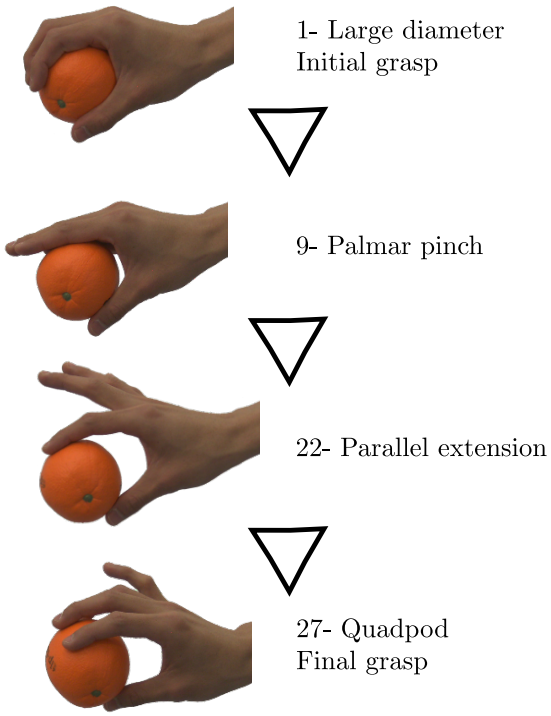


Figure 4.17: Grasp sequence generated automatically from grasp 1 to 27.

The implementation of the solution presented in this chapter has been successfully performed and is explained in chapter 5.

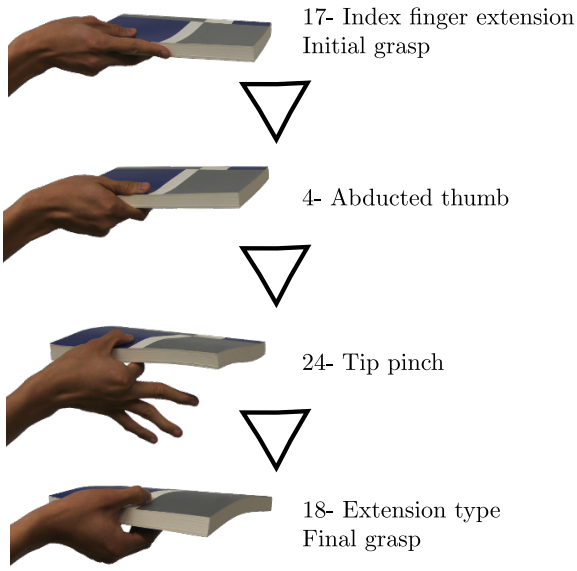


Figure 4.18: A grasp sequence generated automatically from grasp 17 to 18.

Chapter 5

Experiments and results

This chapter presents the implemented high-level layer of control for in-hand manipulation and the results obtained with experiments on simulations and on the real robotic platform. The theory of this layer has been detailed in chapter 4: we use a Markov Decision Process to generate a grasp sequence between a given initial grasp and a final grasp (task grasp). The sequence generated is the one with the highest probability of success. Policies are solutions of Markov Decision Processes and compute which transition should be taken (which next grasp should be reached) given the context: state (current grasp), task (final grasp), and object.

The first section 5.1 of this chapter addresses how we performed the learning process to initialise the parameters of our model. We generated initial policies from a human assessment of every state transition probability. These are rough policies that were then refined by a learning process, first from human in-hand manipulation demonstration and then from robot self-exploration. The human demonstration process is explained along with the validation of the corresponding algorithms in section 5.1.1. Once the high-level layer of control has been implemented, we built a primitive algorithm to enable the robot learn from self-experience. This part of our work is presented in section 5.1.2.

We implemented our high-level planning solution on the platform of the HANDLE project. This is the topic of section 5.2. The first section 5.2.1 gives a description of the whole platform and its complete control framework. Information in this section is necessary to understand the implementation, which complete presentation is provided in section 5.2.2. Eventually, a complete demonstration with the HANDLE platform is presented in section 5.2.3 explaining with an example the gain brought by our high-level layer of planning.

5.1 Learning skills

The theoretical underpinnings of the learning process were presented in section 4.6. In the present section, we first focus on learning the policies from human demonstration. We can do this either by observing random human demonstrations or by using an active learning technique. In a nutshell, with active learning we are able to choose

which samples we should ask humans to demonstrate. We try to select the ones that would bring the most important knowledge to our policies depending on the current knowledge of the robot. We calculate the entropy of all possible trials and choose the ones which entropy is the highest.

We were able to implement our solution on a real robot. This is presented in section 5.2. Then, we had the possibility to make the robot learn from self-experience, as detailed in section 5.1.2.

5.1.1 Learning the policies from humans

We presented the learning algorithm in section 4.6. In-hand manipulation is modelled as a MDP. The behaviour of the robot is a policy, and depends on the task, the object and the current grasp. Policies are what we are able to learn from human demonstrations.

The policies are updated from observations of transitions. For each set of parent variables (*task*, *object*, and *current grasp*), the number of occurrences of every transition is stored. A transition observed is equivalent to an action in our MDP formulation. As we look at human beings performing in-hand manipulation, the transitions used are considered as successful executions of the corresponding actions. This occurrence model can be easily converted into a multinomial distribution using equation 4.23.

As we can choose which sample we want to observe from a human being, we can choose the samples expected to bring the most valuable information. This is the active learning technique described in section 4.6.2. We calculate the entropy of each set of parents, normalized over the entropy of the uniform distribution (equation 4.25). We choose the set of parents of highest entropy, corresponding to a case for which the action to take is less obvious.

The proposed learning algorithm is validated further in this section.

Human recordings

The experimental protocol is described in Fig. 5.1. For a given object, we choose 30 trials of the highest entropy. These trials were performed by 5 human subjects and recorded using a camera. The recordings are watched and analyzed by an expert, who manually recognize the grasp sequence that was used and write the sequence into a file. These grasp sequences are given as data to the learning algorithm that updates the policy with every transition observed.

How we effectively made the human subject perform the trials is shown in Fig. 5.2 and Fig. 5.3. We used pictures of the object grasped by a human hand with a specific grasp type to inform the subject of the desired initial or final grasp. The pictures were displayed on a screen. Subjects were asked to grasp the object with a given grasp task. Once it was done, the final grasp was shown on the screen, and the subject had to reach this grasp type from the current grasp.

Subjects were told:

- They could use only one hand.

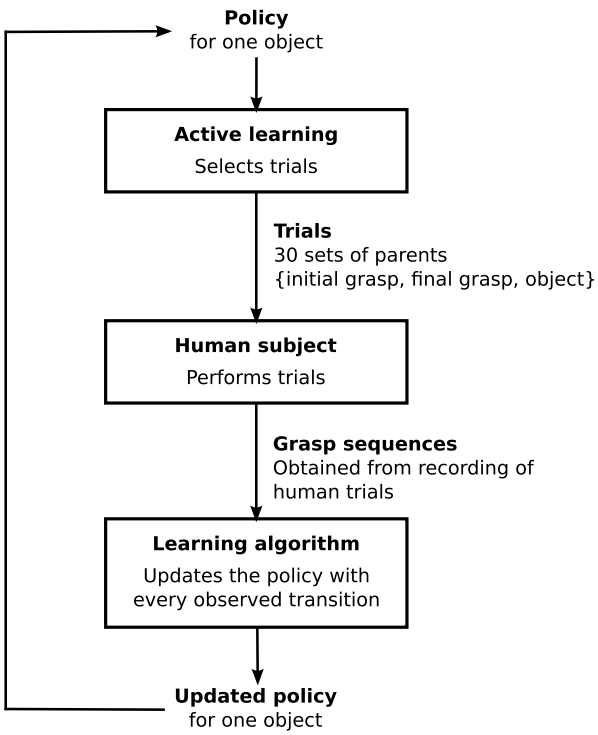


Figure 5.1: Diagram explaining the protocol to learn a policy from human demonstration.

- They could hold the object wherever they wanted given that the grasp type was correct. Nevertheless, it is probable that the position of the grasp on the picture influences the position of the grasp done by the human.

The protocol to acquire data from human demonstration is the following:

- An object is chosen.
- The 30 sets of parents with the highest entropy are selected.
- For each set of parents (each trial), the human subject is given the object. The initial grasp is shown and the subject performs it. Then, the task grasp is shown and the subject reaches it from the initial configuration.
- The manipulation by the subject is recorded on video.
- The sequence of grasps is manually recognized and data are stored in order to be used by the algorithm.
- Learning is performed using these data, or part of them.

5 human subjects participated in the experiment and performed 30 trials each. These experiments required a significant amount of time. The heaviest part of the

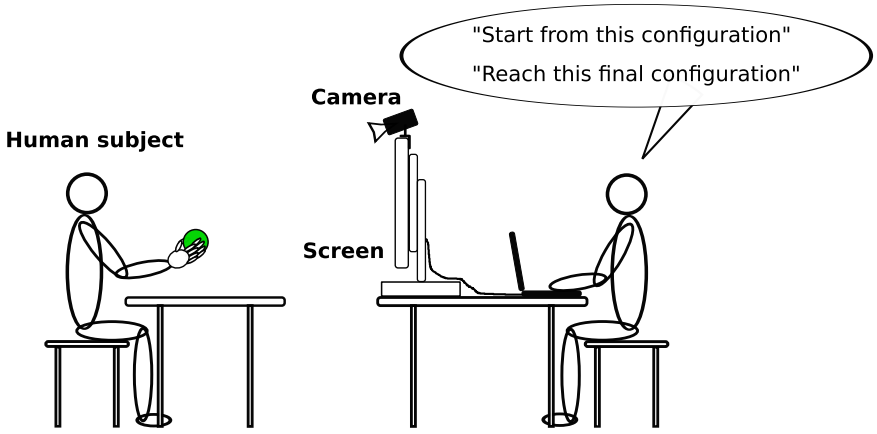


Figure 5.2: Illustration of the context of the recordings when acquiring data from humans.

process was the manual annotation of the videos to extract the grasp sequence. Annotating a video takes approximately four times the video length, which is about the time of the experiment. So, 80% of the time was taken by this annotation activity. Data from humans is very heavy to acquire. That justifies the validation of our algorithm on a simulated human further in this section.

Programming

All the programs related to the work presented in this thesis are written in Python. They were developed separately from the platform before any integration. Python has the benefit to allow a quick and efficient programming and to be integrated easily on the platform. Several functions have been implemented: functions to initialize the policy from the empirical MDP, functions to save and reload policies, and functions to learn. All these functions are presented in a library with an associated documentation. All the initialization, analysis of the algorithm were performed using only this library and were done out of the whole control framework.

Policies are stored in files. This allows to reuse previously learnt policies. The format of the text file is specific but functions are provided in Python to load policies as a table of probabilities. Each policy file corresponds to a policy for a single object. The whole policy, depending on objects, should be obtained by rebuilding the full table from each table for each object. Policies are used to generate grasp sequences by programs integrated in the control system of the robot. The files of policies are therefore stored as hard data in the control framework.

Results of learning on humans

The algorithm to learn from human demonstration was implemented and tested to observe whether the learning was effective or not. The best way would be to perform the complete learning algorithm from human observations, as it is designed to.

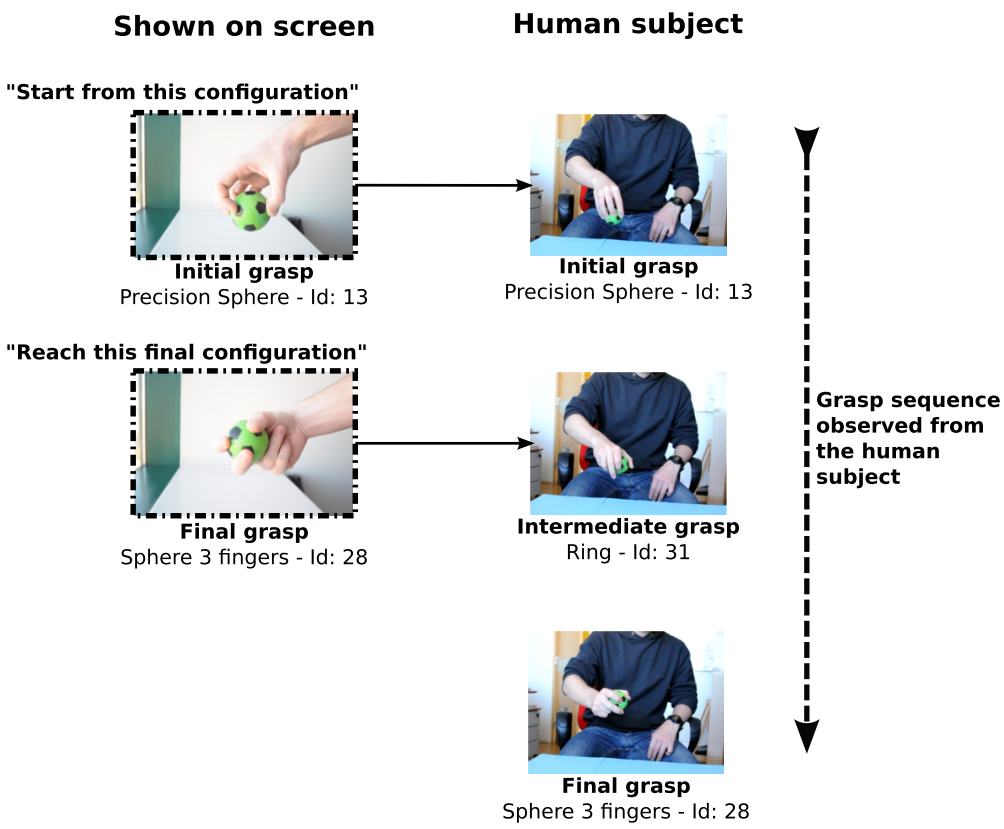


Figure 5.3: Illustration of the acquisition of human data.

However, as the acquisition of data from humans is a heavy process, we made these observations on a single object, the ball, for proof of concept.

We performed the active learning technique described in section 4.6. The effects of the learning on the policy is a good estimation of the quality of the initialization. If learning changed the policy in a radical way, the initialization would be questionable. The multinomial hyperparameters in section 4.6 were chosen as: $h_i(S_t, T, O) = 30$, $\forall i$ ($h_i(S_t, T, O)$ is the value of the imaginary number of samples associated to the empirical knowledge). We used the 30 sets of parents of highest entropy as explained in section 4.6.2, and recorded movements of 5 human subjects for these sets. After learning from these samples, the entropy of every set of parents has been updated and the variation of the entropy before and after learning ($\Delta_{entropy}$) observed.

These results are shown in Fig. 5.4. The blue values are before learning the selected sample, and the yellow values are after learning them. The red values are the differences of entropy. More than 20 sets of parents were modified as every transition in a sequence modifies its corresponding set of parents.

The entropy is decreased by the learning process for most of the sets of parents. It means that for these sets, the choice made by the policy is confirmed by the observations. This shows that the initialized policy is coherent with human behaviour, though not perfect, allowing us to use it as it is for planning in-hand manipulation.

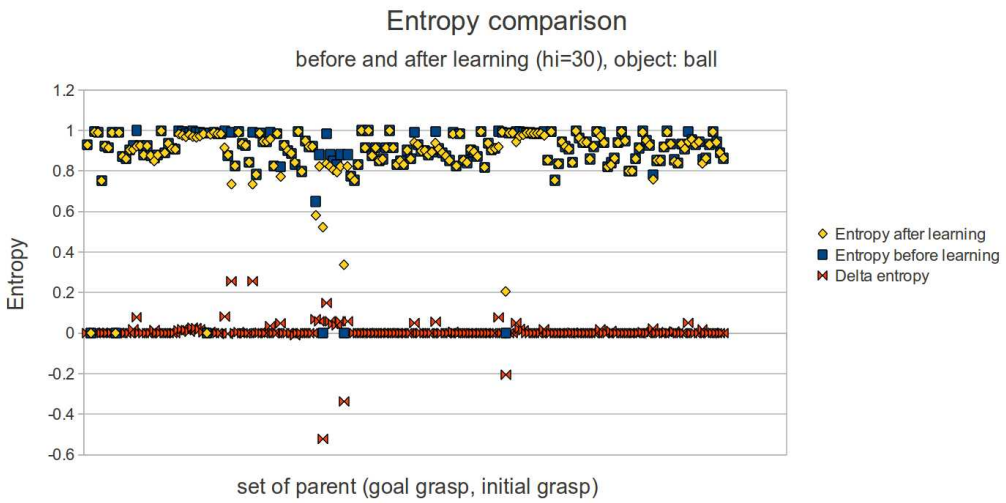


Figure 5.4: Histogram of the entropy of every set of parents (missions) for the object "ball", using the initialization policy, before and after learning.

The entropy is increased for three sets which entropy was null after initialization. These sets previously had only one possible next action. The learning process showed other possible actions, increasing the entropy but giving the system new actions to consider. It highlights that some learning is profitable to refine the policies and that our algorithm is able to correct imprecisions in the initialization and can reinstate transitions considered impossible. The sets of parents with a null variation of entropy are sets that were not selected for the trials. As an example to illustrate the effect of learning, we present one of the few sequences which generation has been modified by the learning process. The sequence we consider is generated from grasp type 26, *Sphere 4 fingers*, to grasp type 16, *Lateral*. Before learning, our algorithm generated only a single intermediate grasp, as presented in Fig. 5.5, using the policy obtained from the human expert estimation of transition probabilities of success. The learning process modified the policy, as the human subjects used a different sequence. Then, after learning, the sequence generated by our algorithm is different, using two intermediate grasps, as shown in Fig. 5.6.

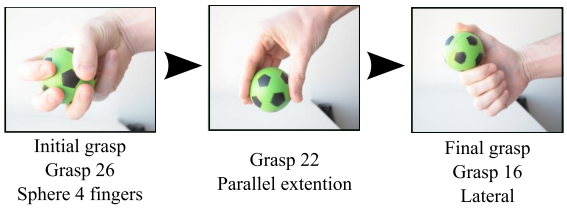


Figure 5.5: A generated sequence for a specific object from an initial grasp type and a goal grasp type, using the initialization policy.

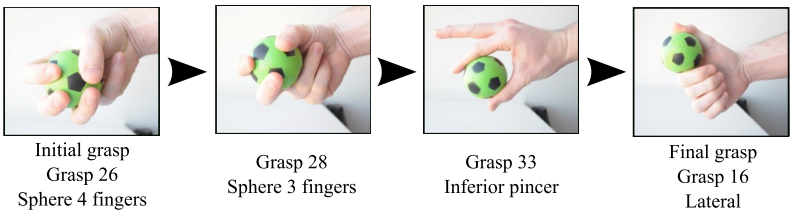


Figure 5.6: A generated sequence for the same specific object, goal grasp type and initial grasp type as shown in Fig.(5.5) but using the policy after learning.

Results of learning using a simulated human

The topic of this section is to observe how our algorithm operates learning on a large amount of data, which is a time consuming and expensive process. Instead of obtaining it from human demonstration, we built a simulated human in order to quickly and automatically get data and evaluate our algorithm.

To monitor the learning progress, we can observe different things. We can detect when the learning process converges and reaches its final state, then the learning progress can be inferred with respect to these values. We can also monitor the difference between the teacher model and the learner model: it is the distance between the simulated human policy and the policy improved by learning.

First, we defined an artificial human MDP. We used the MDP initialized through the assessment of every transition probability of success, and we applied a Gaussian noise to each transition probability, noted p_{ijm} in section 4.1.4. The variance was arbitrarily defined and fixed to 0.02 for probabilities between 0 and 1. The noisy values were truncated so that they did not go out of bounds $[0, 1]$. Then, we used this new probabilistic transition model to rebuild the corresponding policy as we did from the empirical knowledge in section 4.6.1. We obtained a different policy, still consistent with our empirical analysis. This policy can be compared to the initialized one using a measure of distance.

Using the simulated human, we can observe the efficiency of our algorithm in the case of a behaviour strictly modelled as an MDP. We were able to draw samples automatically and instantly whereas observing samples from humans is very time consuming. To simplify the analysis of our algorithm, we were dealing here with the policy of a single object: the "*Orange*".

To measure distances between policies, we considered the Euclidean distance and the Bhattacharyya distance. The equations are given for two standard distributions T and U.

- The Euclidean distance is the standard distance measurement typically used for distance in multi-dimensional spaces:

$$d(T, U) = \sqrt{\sum_i (t_i - u_i)^2}$$

- The Bhattacharyya distance was introduced in the 40s by A. Bhattacharya [8], an indian statistician. This distance is designed to measure the similarity of two probability distributions, either continuous or discrete:

$$D_{BC}(T,U) = -ln(\sum_i \sqrt{t_i \cdot u_i})$$

The initial distance between the initialized policy and the simulated human model policy is 3.17 for the Euclidean distance and 23.43 for the Bhattacharyya distance.

Learning on the simulated human has been done using different options. These options will be reminded when presenting the results:

- Batch (random) learning versus active learning. The active learning is the technique presented in section 4.6 where the trials of highest entropy are supposed to bring the most valuable data. Random learning, or batch learning, just picks random trials. To assess the gain offered by active learning, we will try both and compare the results.

- Set size: single sample or more. At each time step, a number of trials are drawn. In case of a random choice of samples, this does not change anything. 100 loops of sets of 10 samples is equivalent to 1000 loops of one set of a single sample. But when using active learning, a large set will imply that low entropy samples can be selected. A smaller set means that the algorithm will focus on high entropy samples until their entropy goes below other samples.

- Single transition or multiple transitions. We can either observe only the first transition of the sequence or observe the whole sequence of transitions. The first transition is the one for which the entropy is calculated in the case of active learning. The next ones are implied by the sequence, not chosen at all. They can be considered as randomly chosen. We will observe both single and multi transition learning. Learning from multiple transitions should make an active learning process more similar to a random learning.

- Stochastic or greedy behaviour. Using the policy, the behaviour of the simulated human can be either stochastic or greedy. A greedy behaviour means that the simulated human will choose the most likely action from the policy. A stochastic behaviour means that the simulated human will make a random choice following the probabilistic distribution of the policy. A greedy behaviour does not allow to observe the low probabilities of the policy.

We monitored the distance between the learned and the simulated human (known) policies to evaluate if the learned policy, during and after learning, was coming closer to the simulated human policy. We monitored the entropy to observe how it evolved during learning given the parameters of learning.

First, we observed the evolution of the standard algorithms. In Fig. 5.7 and Fig. 5.8, the distance between the learnt policy and the simulated human policy is measured while the system receives a very high number of samples. The learning

algorithms used are batch learning and active learning, with 800 sets of 100 samples. All the transitions of a sequence are learnt and the behaviour of the system is stochastic. Bhattacharyya distances are shown in Fig. 5.7 and Euclidean distances are shown in Fig. 5.8. They behave in the same way. The learning process converges towards 0 for both active and batch learning, which proves that both algorithms are working properly. In addition, they appear to learn at the same speed in this case. However, no conclusion can be drawn on the comparison of the two algorithms as the set size should influence the efficiency of the active learning.

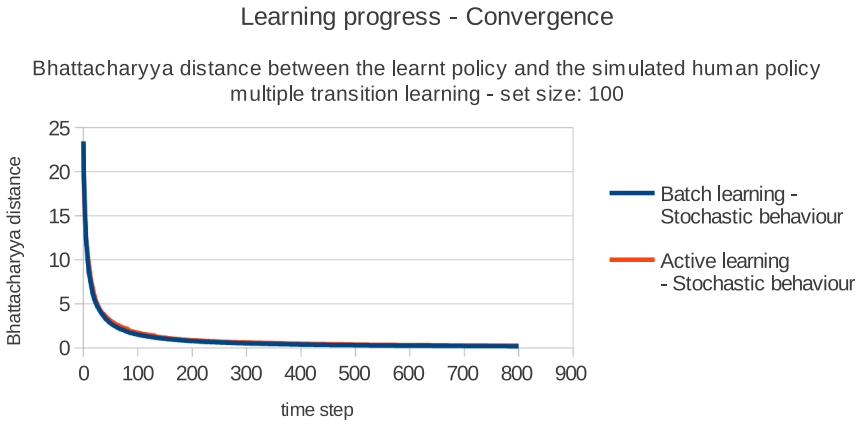


Figure 5.7: Learning from a simulated human: convergence of batch learning and active learning using the Bhattacharyya distance.

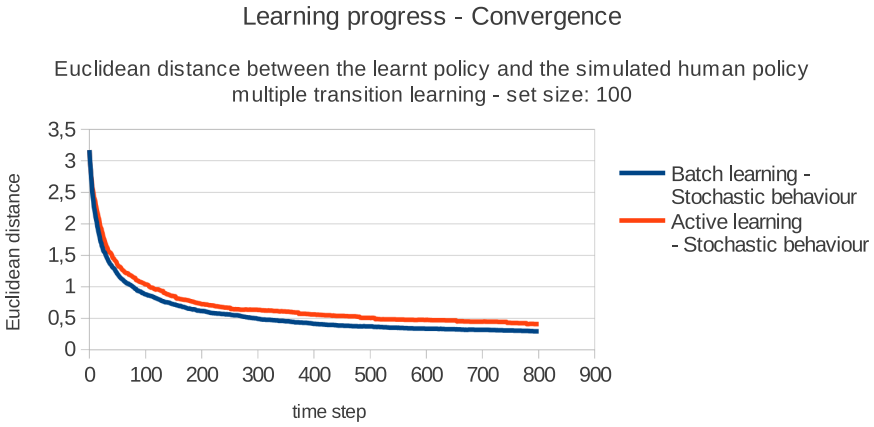


Figure 5.8: Learning from a simulated human: convergence of batch learning and active learning using the Euclidean distance.

From now on, in order to simplify our presentation, we will present only the Bhattacharyya distance. Along this analysis, we measured the Euclidean distance as well and it gave equivalent results to the ones obtained using the Bhattacharyya

distance. The Bhattacharyya distance is theoretically more appropriate for distance between probability distributions than the Euclidean distance.

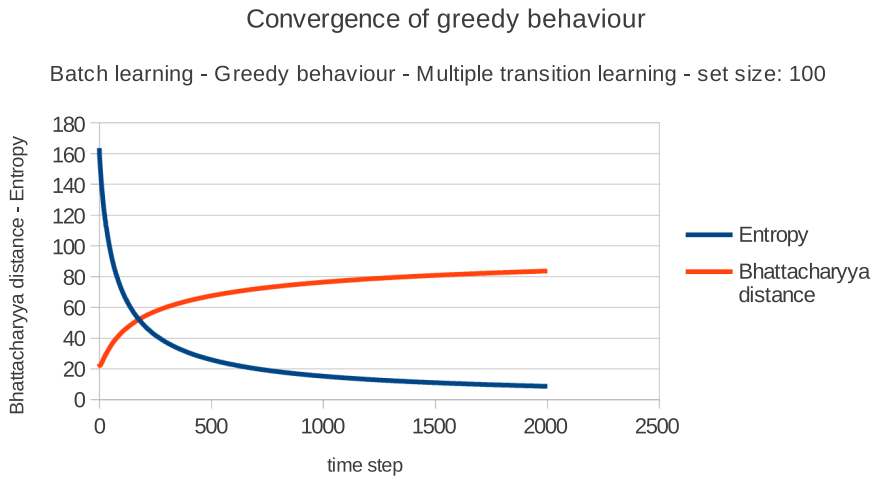


Figure 5.9: Learning from a simulated human: teacher greedy behaviour.

In Fig. 5.9, we can observe the Bhattacharyya distance for a batch learning algorithm observing all the transitions of the sequences, but in this case the behaviour of the model is greedy. We can observe that the distance to the simulated human policy grows so the learning algorithm does not bring the policy closer to the simulated human policy. It appears logical that greedy behaviour learning cannot lead to the efficient learning of a probabilistic model as the system never encounters cases with low probability. It then assumes that these cases have a null probability, which is false. However, the distance seems to converge. We also monitored the total entropy along the learning process (sum of the entropy of every possible set of parents), it is shown in the graph. The total entropy of the simulated human policy is $163,58$. For our learnt policy, the entropy decreases and converges toward 0, implying that the learnt policy converges towards a different policy with a null total entropy. The policy observed by the system is the greedy policy, which can be computed from the simulated human stochastic policy by assigning a probability of 1 for the highest transition probability of a set of parents, and 0 to the lower transition probabilities of the same set. This policy has a null total entropy, the learnt policy converges towards it. The conclusion of the analysis is that if humans adopt a greedy policy in in-hand manipulation, we cannot observe the underlying variability and thus, the ability to learn a stochastic policy. In the sequence, we assume humans follow a stochastic behaviour.

We conducted a few simulations to study the influence of the sample set size for active learning. In Fig. 5.10, the Bhattacharyya distance between the learnt policy and the teacher policy (simulated human) is shown for an active learning algorithm with stochastic behaviour. Only the first transition of a sequence is observed, emphasizing the effects of the active learning technique. Different sizes of learning sets

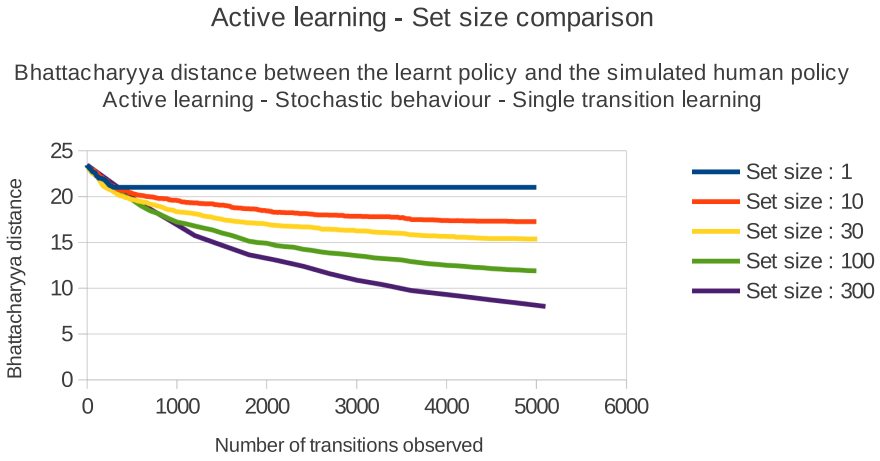


Figure 5.10: Learning from a simulated human: active learning with different set sizes.

are visualized. To be able to compare the learning speed, we count the number of observed transitions as it reflects the amount of data given to the learning algorithm. The maximum set size is the maximum number of different sets of parents. This number is around 350 for the object we used (orange). During the first few hundreds transitions, the algorithm learns nearly at the same speed for every set sizes. Then, the speed of learning changes for the different set sizes, and after $5,000$ observed transitions, each set size leads to a different distance to the teacher's policy. The larger the set size, the less the distance. We can also observe that the learning process saturates for a single transition set around 200 observed transitions. A detailed analysis revealed that the learning process gets stuck on a sample. It keeps asking for this sample and observing transitions from it but the entropy never decreases. The reason is that the learning process reached the entropy of the simulated human policy for this sample, consequently, the entropy did not decrease. As the sample is the one with the highest entropy, the learning process focuses on it and the learning process saturates. We encountered the same saturation phenomenon with a set size of 10, around $10,000$ observed transitions (Fig. 5.11). The learning process gets stuck on 10 samples in the same way.

In Fig. 5.11, we performed the same simulation but adding batch learning to it. We removed active learning with a set size of 1 for a clearer chart. Batch learning algorithm learns faster than active learning. It means that selecting the samples randomly is more efficient than choosing the samples of high entropy. A very large set size is closer to a batch learning behaviour. It is important to notice that the batch learning algorithm can be stopped at around 50,000 observed transitions, as it has reached its final value (5% at $48,700$ observed transitions).

In Fig. 5.12 are shown the same data as in Fig. 5.11 but zoomed on the first 1,000 steps. We can see that until 300 transitions observed, all the algorithms go at

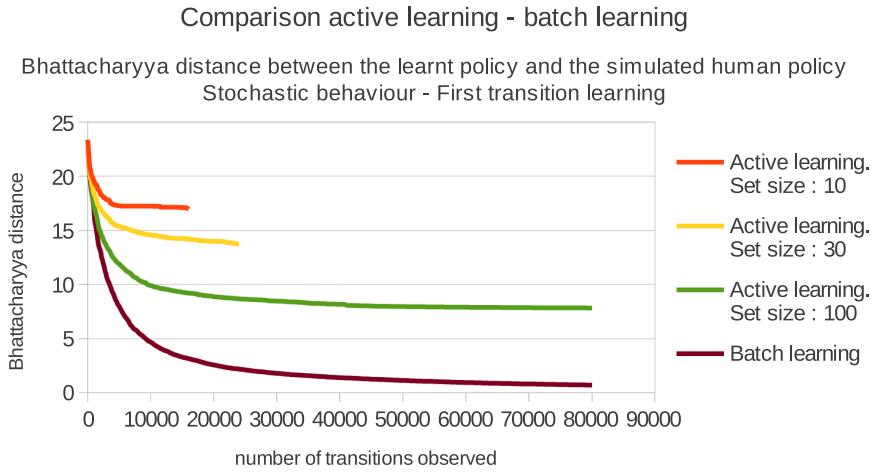


Figure 5.11: Learning from a simulated human: comparison between active learning and batch learning: 80,000 observed transitions.

the same speed. A zoom on the first 300 transitions shows that batch learning is a bit slower than active learning at the beginning of the process.

We performed a few tests learning every transition in the sequences, which corresponds to observation of data from real humans. The results are shown in Fig. 5.13. As expected, the greedy behaviour diverges. The active learning behaviour using a single set size saturates as in Fig. 5.10. The same algorithm with a set size of 30 gives much better results. On the first $2,000$ observed transitions, it learns faster than batch learning algorithm. This is especially visible on the enlargement on the 500 first transitions. Active learning algorithm is proved to be faster than batch learning on the first $2,000$ transitions when using a large set size. A set size of 30 represents approximately 10% of the maximum set size. This observation made us use a set size of 30 for the human recordings presented previously in this section. However, in the long run the gain brought by the use of the active learning is not significant. We do not greatly shorten the time needed for learning. The disappointing performances of the active learning algorithm can be explained. As active learning focuses on entropy and not on distance reducing, the active learning algorithm either saturates, or goes slower and slower compared to batch learning.

We monitored the entropy while performing the different learning algorithms. Results are shown in Fig. 5.14. Every learning technique with a stochastic behaviour stabilizes at a high entropy. Given that the simulated human model has an entropy of 163.58, the policies learnt from this model keep a high entropy. The active learning algorithm, when it does not saturate because of set size, leads to a slightly lower entropy, as it can be expected from this algorithm focusing on lowering the entropy. Greedy behaviour gives very different results concerning entropy. The entropy for this algorithm converges towards 0, as previously explained and shown in Fig. 5.9.

We can draw some conclusions from this section. First, if humans use greedy policies then we cannot learn the stochastic aspects, we should adapt our model.

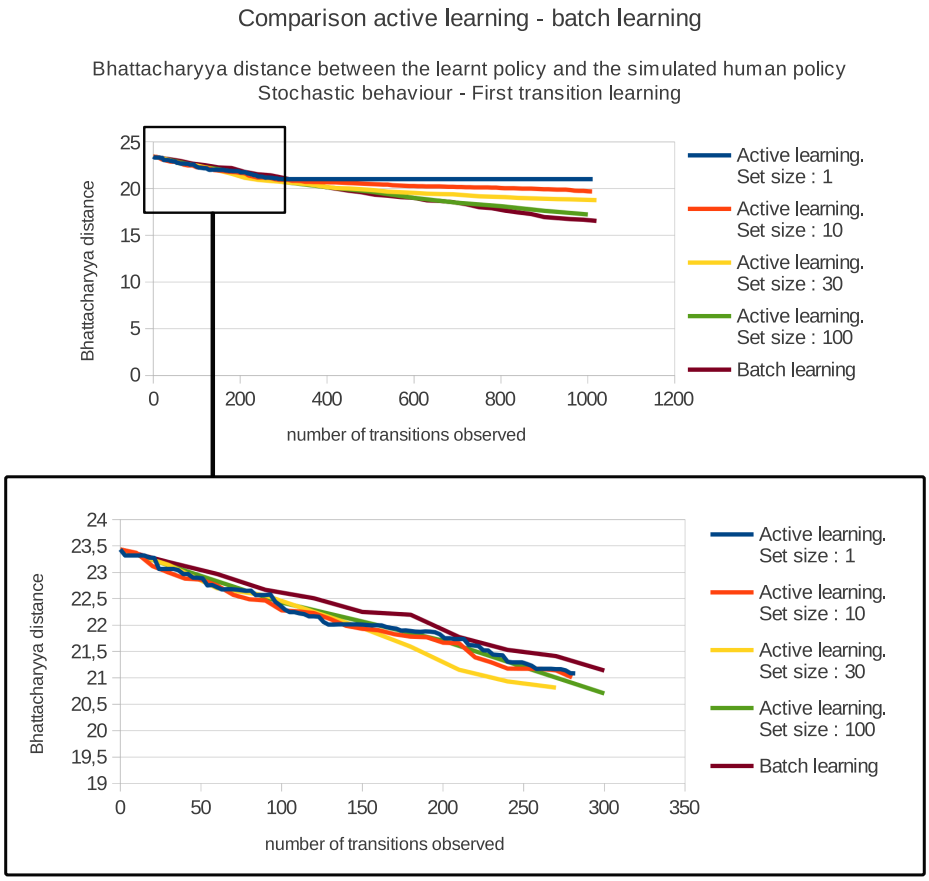


Figure 5.12: Learning from a simulated human: comparison between active learning and batch learning: 1,000 and 300 observed transitions.

Secondly, active learning by entropy minimization can improve learning rates on the initial stages, depending on the set size, but the learning process gets stuck in local extrema in the long run. An appropriate set size is around 10% of the total number of possible samples. It justifies the set size of 30 used for human recordings in section 5.1.1. To take advantages of these active learning benefits, we can use the entropy minimization in the first stage and then use a batch learning algorithm. The recordings of human behaviour we performed belong to the first stage of learning, when the use of entropy minimization is relevant. We recorded different human subjects performing manipulation. For the same samples, the actions used were slightly different from one subject to another. This is consistent with the observation of a stochastic behaviour, validating the use of a probabilistic model for the policy instead of a greedy approach.

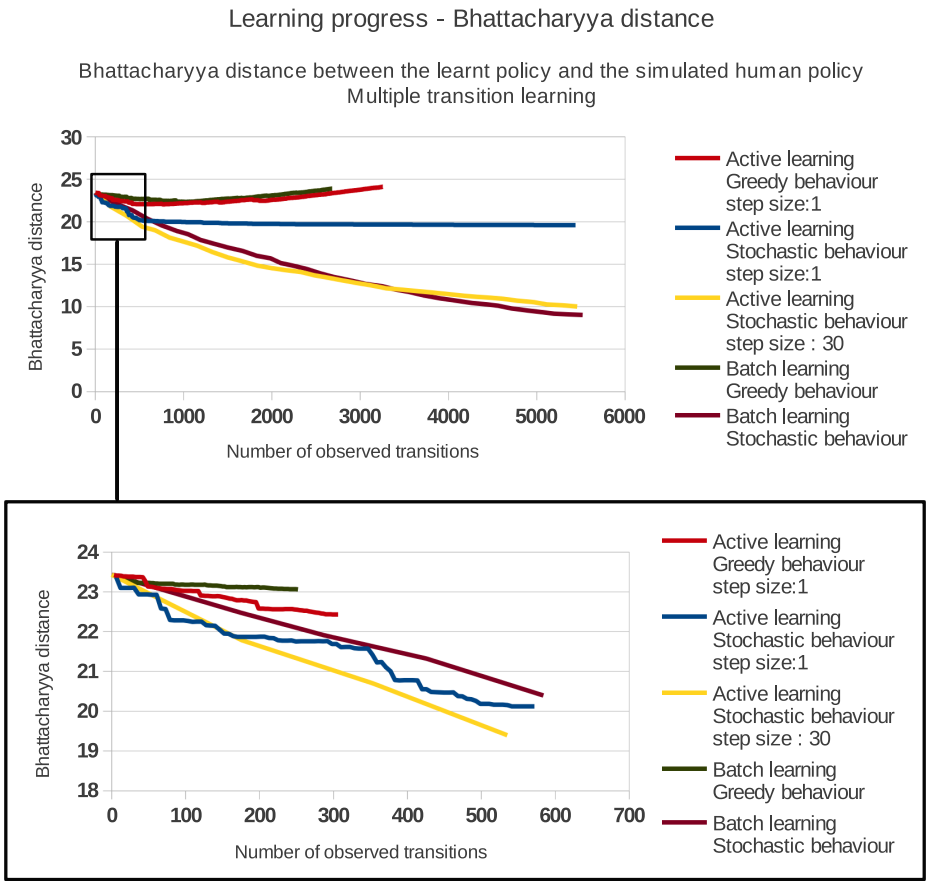


Figure 5.13: Learning from a simulated human: multiple transition learning.

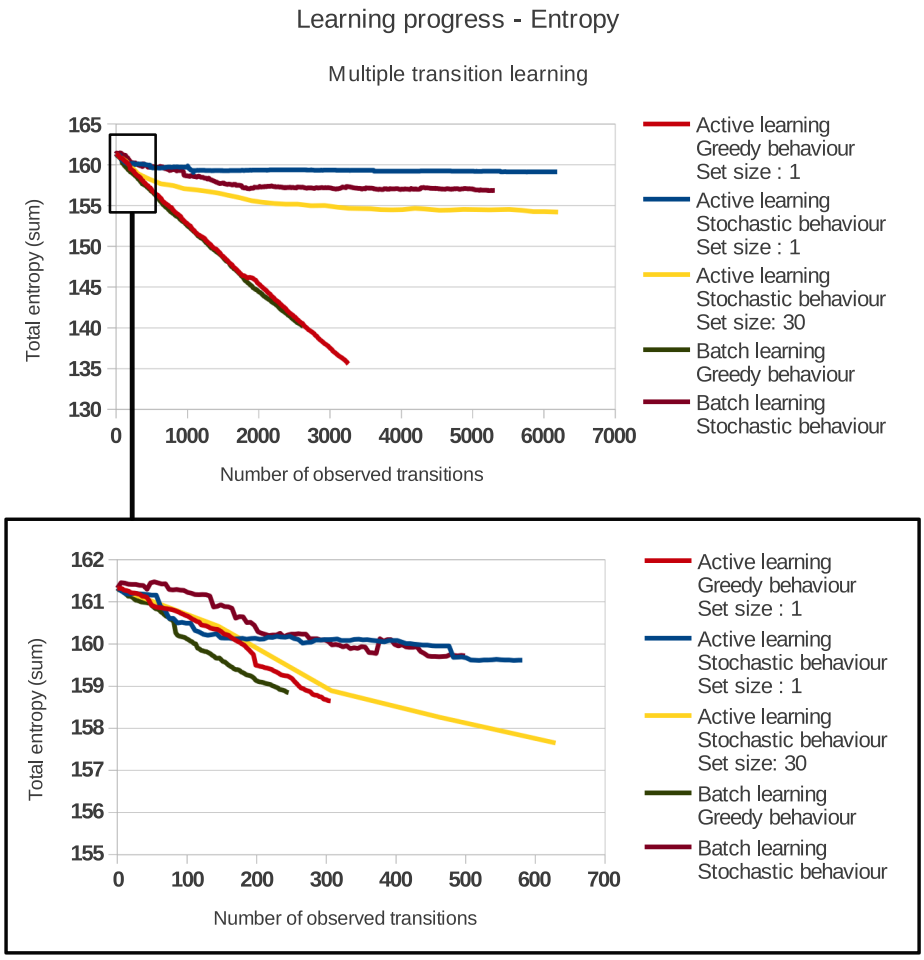


Figure 5.14: Learning from a simulated human: entropy.

5.1.2 Learning from robot self-experience

We were able to implement our solution on a real robot, as it will be explained in section 5.2. The robot has its own capabilities of low-level in-hand atomic actions, which are described further in this section. When performing in-hand manipulation with the robot, we have the possibility to make it learn from its self-experience.

Having the robot directly improve its knowledge from analysing its own behaviour is more accurate than learning from humans, as the robot and the humans, despite the anthropomorphism of the hand and the control system, have different capabilities. These different capabilities have multiple origins. On the one hand, the robotic hand and the human hand have fundamental differences coming from the design of the robotic hand. This point is discussed in the next paragraph. On the other hand, the capabilities of the robotic hand are limited by the low-level algorithms developed for performing the atomic in-hand actions, as explained later in this section.

Human likelihood

The state-of-the-art robotic hand we use in this work is a Shadow electric hand. Its capabilities are very large, and it looks like a human hand in terms of geometry. But differences still exist. The ranges are not exactly human-like, as shown in Fig. 5.15, neither is the power of the hand. The maximum force at fingertips is between 300g and 500g, whereas it is between around 2 to 5kg for a human hand.

The sensing abilities are below human ones. The human skin is a continuous sensor with an impressive accuracy, whereas the robotic hand has accurate sensors on the fingertips only. The human joints are a bit elastic, adding more compliance.

The materials of the fingertips and the contact zones of the palm in the robotic hand are different types of rubber. Rubber provides a high contact friction, intuitively close to the human skin, but somewhat different. The characteristics of the materials, especially deformation resistance, are quite different and have an impact in the manipulation experience.

Some specific configurations are only feasible with the Shadow hand (Fig. 5.16) but this case is unusual, compared to cases where the human hand surpasses the Shadow hand's capabilities. Anyway, these differences are just specific cases, when the limits of both hands are near. The overall geometry and possible movements are very similar between the human hand and the Shadow hand, ensuring us that the human-inspired methods we use are relevant. However, the small differences listed here make the learning from robot self-experience more accurate than learning from humans.

Platform capabilities for atomic actions

Now we discuss the real in-hand atomic action capabilities of the robot. A transition, as defined in chapter 3 is made of atomic in-hand actions. The robot is currently able to perform a limited set of in-hand manipulation atomic actions. It does not have human-like capabilities. This made us adapt our algorithm as explained in this

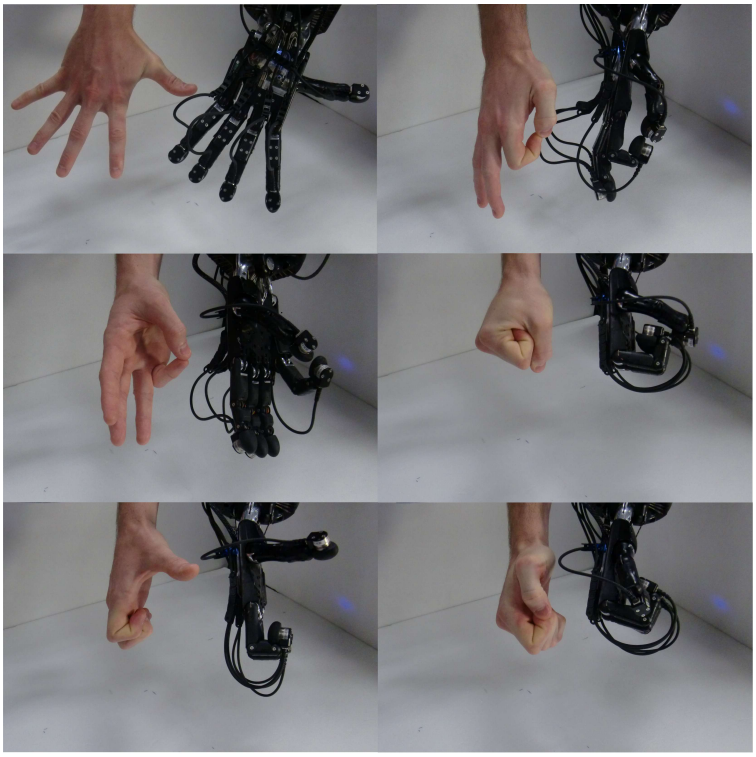


Figure 5.15: Shadow hand and human hand comparison. Human hand joint ranges appear to be slightly larger than the joints of the Shadow hand. Some configurations are possible for the human hand but the Shadow hand can only come close to these.

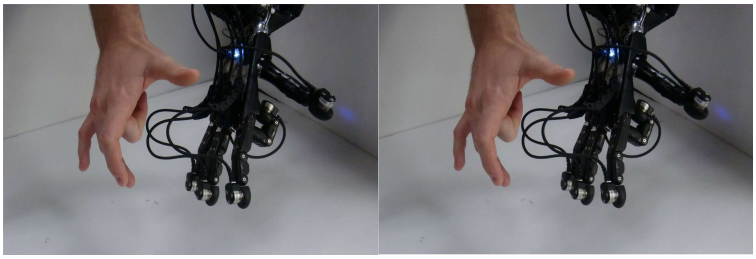


Figure 5.16: Shadow hand and human hand comparison. Here, a specific configuration is shown where the Shadow hand can fully bend the middle finger while keeping the other fingers straight opened. It is impossible for the human hand, the other fingers, especially the ring finger, getting interference from the position of the middle finger because they are actuated using a common tendon.

section. In the next paragraph, we are listing the atomic actions that the robot is able to do.

The transitions from a grasp to another can be composed of one or more actions. Planning which actions have to be performed, and in which order and with which movement, is done by the low-level module (*Execute Grasp Transitions* in Fig. 5.26). Our solution does not autonomously choose the atomic action sequence. For now, a primitive planning has been set up for this. The atomic action sequence for each

transition is hard-coded in the database. The hard-coded sequences of atomic actions for each transition have been empirically fixed from observation of human movements and abilities of the platform. This means that this part of in-hand manipulation planning is not fully autonomous. Some parameters of the actions are autonomously adapted to the size of the object, by reacting on contact for example.

Four algorithms can be called from the low-level main module *Execute Grasp Transitions*. Each node performs a single or more elementary actions. For now, the robot is able to perform the following actions :

- *Synergy movements sequence* : this node is able to open or close the hand using synergy parameters. It is also able to perform finger gaiting still using synergy parameters.
- *Geometry planner movements* : the movements planned by this node are movements without a change in the grasp. They are still elementary actions. The translation movement of the object along the axis normal to the palm (called $\vec{y}$) is feasible as rotations of the object along any of the three axis $\vec{x}$, $\vec{y}$ or $\vec{z}$. These movements imply moving every finger with great coordination to move the object without losing contact at any contact point. Contact is supposed to be rolling only, without sliding. These movements only apply to precision grasps.
- *Gravity rotation* : this specific elementary action concerns the *Palmar Pinch* grasp. It can be seen as a rotation of the object along the axis linking the two contact points. But unlike in *Geometry planner movements*, the object is sliding on the fingers. Gravity keeps the object in a position, while the hand moves around it, generating sliding.
- *Finger gaiting* : this node plans the movement of a single finger considered as a standard robotic manipulator.

This sums to a total of 7 atomic actions, one of them is opening or closing the hand, and two of them can be classified as the same type of atomic action (in-hand rotation). According to previous works ([23] and [11]) on human manipulation, 11 in-hand movements are listed, using different classifications. [11] considers sliding and rolling. Finger gaiting has to be added. It highlights that the low-level layer is incomplete in terms of atomic action planning and execution. In its current state, the robots' abilities regarding control of in-hand manipulation atomic actions are well below the human ones.

Adapting to the real robot's capabilities

As the learning algorithm had been implemented on the real robot platform, it became possible to develop an algorithm making the robot learn from self-experience. This is the topic of this section.

The robot learns from its attempts to perform transitions between grasp types. It should know when it succeeds or fails a transition. Counting these trials, the number

of successes and failures of each transition is the starting point of the algorithm. This data is stored in the database integrated with the whole control framework. The database also stores details for each transition. A key information is the sequence of elementary actions to perform.

As explained in chapter 4, the ideal method for generating subgoal sequences would be to use directly the MDP model. This implies learning the probabilistic transition model $\mathbf{P}$. This is only possible if the robot learns by itself its ability to perform transitions, depending on the influencing parameters. Our work has been integrated to the HANDLE platform. We then introduced a primitive learning algorithm, adapting the policies to real transition abilities of the robot. This learning algorithm was directly modifying the policies. This was mandatory. Indeed, due to a lack of capabilities of the robot, some transitions do not even have a low-level planning algorithm. These transitions are doomed to fail and thus, we had to adapt.

When the robot performed these transitions, it sometimes succeeded, and sometimes failed. These trials are stored in the associated database. As a primitive way to include this knowledge, this data has been used as if it was observed human trials, in standard supervised learning of the policy. A success would be seen as an observed human trial, and increase the count $h_{ij}(g_i, g_f, o_m)$. A failure would decrease the count $h_{ij}(g_i, g_f, o_m)$. This count should not be negative, so if it reaches 0, failures will not have any effect on it.

This algorithm is very rough, and does not take all the profit of the data. If the state transition model $\mathbf{P}$ is directly learnt, the knowledge ability of the transition would be generalized to every possible tasks. $\mathbf{P}$ does not depend on the task, that is why its dimension is lower than the associate policy.

5.1.3 Conclusion about learning

Learning is a natural solution to the complexity of dexterous manipulation problem. In an efficient approach, the robot should learn its own abilities by self-exploration and then be able to plan an optimal subgoal path when manipulating objects. We implemented a primitive algorithm on the HANDLE platform to do it.

We also developed an alternative learning method by making the robot learn from human demonstration. The policies used by humans can be observed and learnt so that the robot can store and imitate human behaviour. The relevance of this technique relies on the fact that the considered robotic hand is close enough to human hands. We highlighted that the design of the Shadow hand has some differences with the human hands, mainly regarding sensing abilities. Otherwise, the parallel between the two hands is relevant. Nevertheless, capabilities of atomic transitions are far below human ones because of the low-level planning algorithms. This justifies that learning from robot's self experience is a better option than learning from human demonstration. We developed a primitive solution for doing so.

However, learning from humans is not pointless. It makes a good initialization to the policies. Consequently, a very accurate improvement by learning is not required, as a second learning process, by the robot's self-experience has to be done. We started

by learning from humans as we were able to perform this learning when the whole platform was not available yet.

The learning algorithm on the simulated human proved its viability. Active learning is only interesting in the beginning of the learning process, but the gain obtained this way is not significant.

5.2 Implementation

Our solution has been implemented on the HANDLE platform. Our algorithms have been integrated into the control framework. This is the topic of this part. First, a complete presentation of the platform is given in section 5.2.1. Secondly, we explain the details of the implementation in section 5.2.2. Eventually, in section 5.2.3, the operation of our algorithms is explained through the demonstration of complete manipulation tasks on the platform.

5.2.1 The experimental platform

Our algorithm generates sequences of subgoals so that the low-level of planning and execution for in-hand manipulation can perform easy transitions, without having to consider the whole complexity of the task. The type of subgoals chosen are grasp types as explained in chapter 4. We use the probability of success of transitions from a grasp to another, i.e., the low-level capability of performing the in-hand manipulation actions forming the transition, to provide values for the probabilistic transition model of a Markov decision process. Consequently, our algorithm receives the final grasp as input, that is, the grasp to reach according to the task. The initial, or current grasp, is supposed to be known from the low-level layer. The subgoal sequence is then provided to the low level. These links are shown in Fig. 5.17.

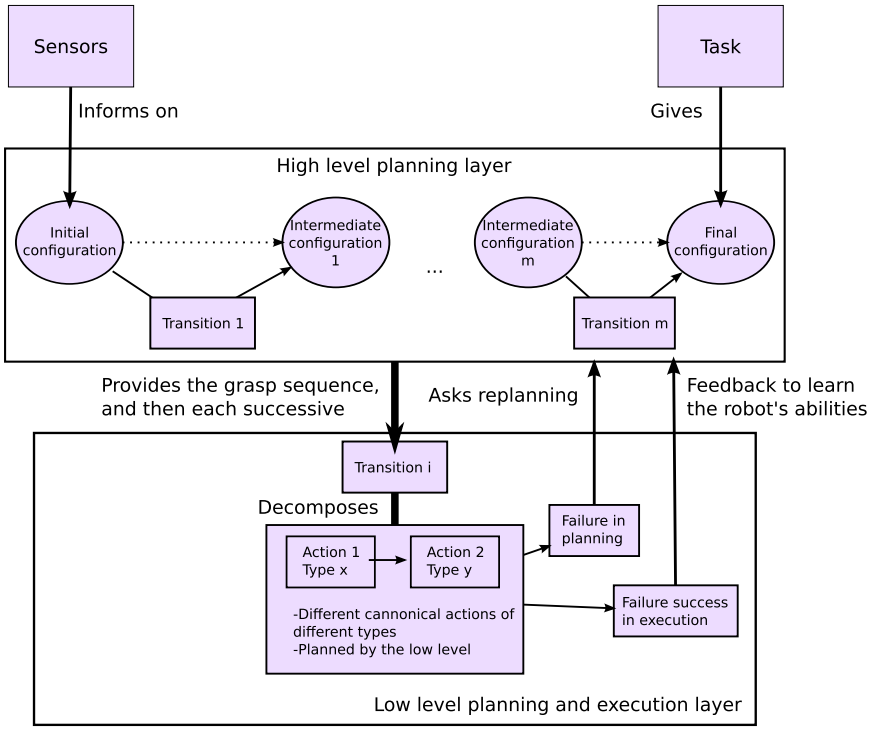


Figure 5.17: Links between our layer and the higher and lower levels of planning and execution.

The hand considered in our work is supposed to be anthropomorphic and we aim at human-like capabilities. Using the most human-like hardware and software is consequently the normal course. We implemented our layer on the HANDLE platform, which is described in the next section, understanding why our method is well suited for the platform, and understanding the limitations of the implementation.

Details of the platform

The arm The HANDLE platform mainly consists of a Shadow EDC Motor Hand from Shadow Robot Company mounted on the Shadow Biomorphic Arm. The arm has 4 degrees-of-freedom (dofs) not considering the wrist (2 dofs), which is part of the hand. It is a pneumatic arm. Two air muscles work antagonistically to actuate each joint. A fully humanoid arm would have 5 dofs, plus the 2 dofs in the wrist. The platform lacks a dof in the arm to be exactly human-like. This does not influence in-hand manipulation capabilities as every action is done by articulations of the hand. However, it causes limitation in initial grasp achievability and object transportation with the arm.

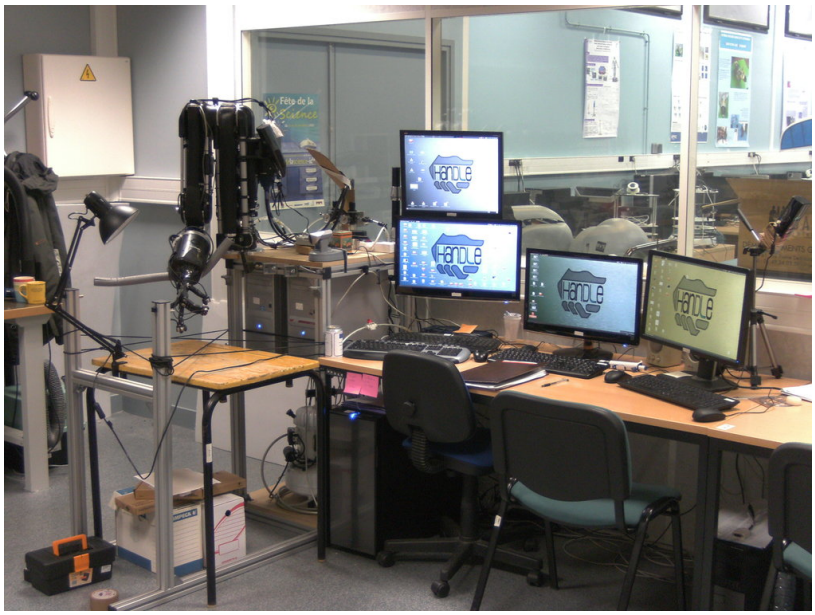


Figure 5.18: The experimental platform: the Shadow hand and arm, mounted on a support structure. The different computers controlling it are visible.

The hand The Shadow hand (Fig. 5.19) has five fingers in an anthropomorphic configuration. The thumb is opposable and more agile than the other fingers, as in a human hand. The kinematic representation of the hand is shown in Fig. 5.20. Table 5.21 gives the range of motion of each joint. This hand is actuated by electric DC motors. These motors are pulling on tendons that transmit the movement to the joints. Each joint of the hand and the arm is equipped with an angle sensor. Force sensors measure the traction in each tendon, which approximates the corresponding

torque at each joint. The hand has 24 degrees-of-freedom including the wrist. The two joints of the last phalanxes of each finger, except the thumb, are coupled with a ratio 1 : 1. This brings the number of actuators (motors) to 20.

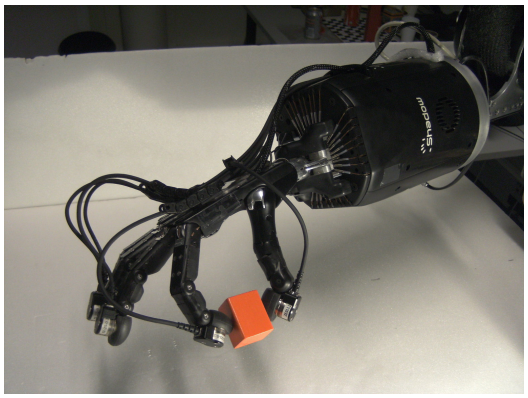


Figure 5.19: The Shadow hand.

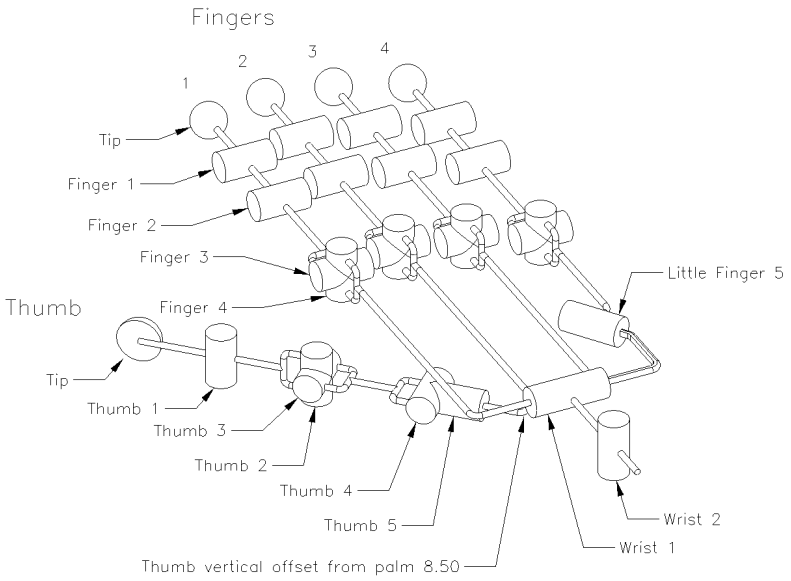


Figure 5.20: Shadow hand range of joints.

Sensing abilities The Shadow hand on the HANDLE platform is not a standard one as far as its sensing abilities are concerned. It has been upgraded by work from the HANDLE project. It is equipped with ellipsoid fingertips including ATI Nano17 sensors, which allows to extract contact data. These sensors and the special fingertips provide a very accurate measure of the contact points on the fingertips as well as a measure of efforts. It provides forces in three dimensions at the contact point (one normal to the contact surface and two tangential) and also the torque around the normal axis of the contact surface. The contact location is measured to a precision

Joint(s)	Min (deg)	Max (deg)
FF1, MF1, RF1, LF1	0	90
FF2, MF2, RF2, LF2	0	90
FF3, MF3, RF3, LF3	0	90
FF4, MF4, RF4, LF4	-25	25
LF5	0	40
TH1	-10	90
TH2	-30	30
TH3	-15	15
TH4	0	75
TH5	-60	60
WR1	-40	20
WR2	-30	10

Figure 5.21: Table of the range of each Shadow hand joint.

of 0.3 mm. Efforts are detected with a precision around 0.8 Nmm for torque, 0.04 N for normal force and 0.05 N for tangent forces (friction).

Figure 5.22: ATI Nano17 sensors with ellipsoid rubber fingertip.

An artificial skin was under development within the HANDLE project but this work did not get to an integrated prototype. For now, a commercial pad on the palm detects contact. It uses a matrix of capacitive sensors, developed by PPS [4]. It is shown in Fig. (5.23).

Vision The vision hardware is composed of a Kinect device by Microsoft (Fig. 5.24). It gives a color image and a depth map of the environment. This can be translated into a 3D point cloud, which is used to recognize the environment. With this information, several software modules are combined to get information about the object. Using

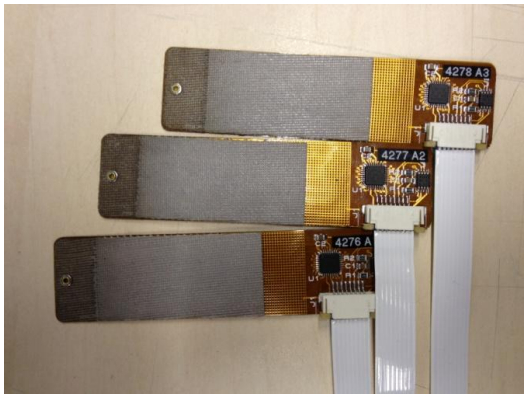


Figure 5.23: PPS palm pad.

the whole 3D point cloud, the object is segmented, extracted as a cluster, and can be either recognized from known objects in a database, or recontructed as a 3D mesh. The field of view of the Kinect allows to observe a manipulation environment, as highlighted in Fig. 5.25. A multi-core computer is fully dedicated to run the vision algorithms.

Figure 5.24: The Kinect is mounted on the side of the shoulder.

ROS

The control framework described in this thesis is multi-layered. The planning layers have been described, but other layers are needed, for vision, sensing, etc. The HAN-

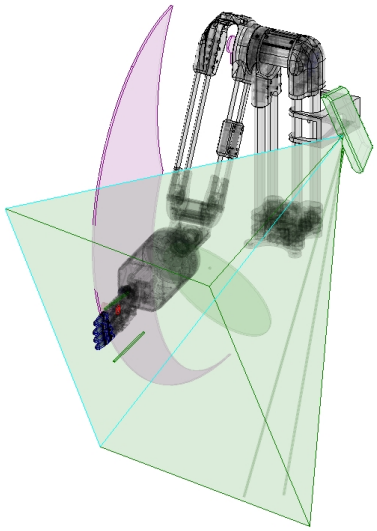


Figure 5.25: The Kinect's range of vision.

DLE platform is controlled using ROS (Robot Operating System). This software was launched by Willow Garage, which maintains its core development. A user community is developing and maintaining additional features. ROS works as an operating system, designed to run different processes, providing message-passing between them. These processes, called **nodes**, can communicate with external devices, or with the user, through computer user interfaces.

ROS is designed to allow the control of robots of all kinds, provided that the appropriate nodes are developed. This can be done by Willow Garage itself, or any member of the ROS user community. The existing nodes are grouped into Packages and Stacks, and then made available for anyone in repositories.

ROS is licensed under an open source BSD license. This open source development is consistent with the displayed primary goal of ROS, according to Willow Garage: "to support code reuse in robotics research and development". Along with existing programs, developing tools are also provided.

The software allows running on multiple computers, which is very useful. For example, a single computer may be required to run vision modules that are computationally very heavy. Nodes can be coded in different languages: Python, C++ and Lisp. Its framework, with independent nodes connected to each other, makes it well suited for large control systems, such as the one needed for in-hand manipulation.

Control Framework

Using ROS, the control framework of the whole platform was developed. Each partner of the HANDLE project programmed nodes based on their scientific work. The purpose of this section is not to describe this work, but to provide an overview of the control framework. This is required to understand how the work presented in this thesis was implemented.

To have an efficient development of nodes by different developers, the framework was designed first and faced minor corrections along the implementation. The type and aim of each message to be passed between nodes were also fixed very early.

Flowchart The final control framework is shown in Fig. 5.26. Colours show the purpose of each set of nodes.

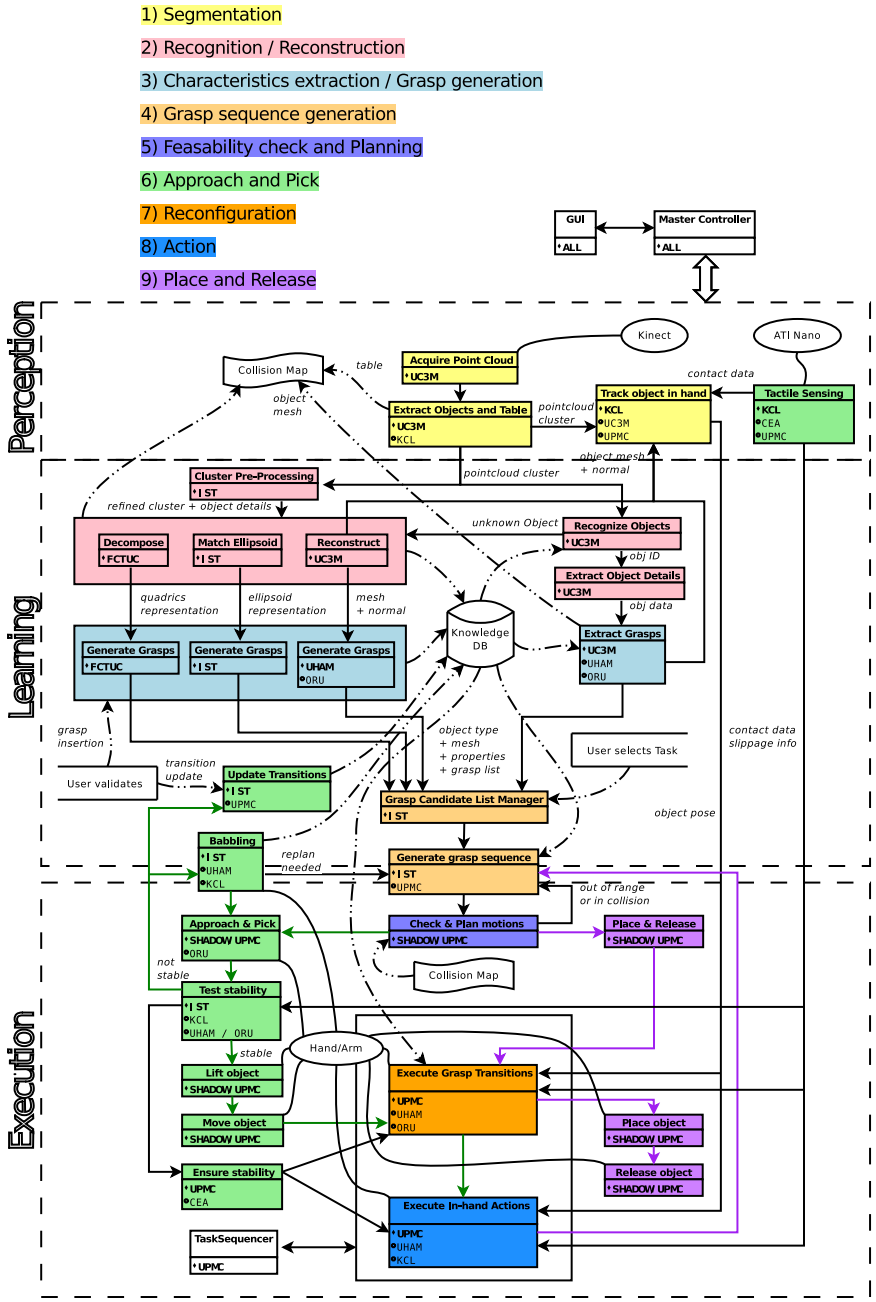


Figure 5.26: Flowchart of the HANDLE manipulation control framework.

The node at the top, called *Master Controller*, is the main node. It triggers the other required nodes at each step of manipulation. It also drives the user interface (GUI) displaying major information to the user and interacting with him.

The chart has to be read from top to bottom. Yellow nodes are the first modules to be called. They are the vision modules. The environment (object, obstacles) is acquired. Then, the corresponding 3D point clouds are given to the next modules: the pink nodes. Their role is to translate the point cloud into an object model. The modules on the left reconstruct the object into an object model (either a quadric, ellipsoid or mesh model). The modules on the right try to recognize the object in the database of known objects.

Then, light blue nodes generate possible initial grasps on the object. The module *Extract Grasps* outputs the grasps associated with the known object, in case it has been recognized. Modules on the left generate grasps automatically from the appropriate object model.

All the generated grasps are given to the *Grasp Candidate List Manager*. They are complete grasp configurations, not only grasp types. The role of the two pale orange nodes: *Grasp Candidate List Manager* and *Generate grasp sequence* is explained with more details in section 5.2.2, as it is the implementation of the work presented here. In a few words, this module validates the initial grasp and generates the grasp sequence, which is transmitted to the module *Check and Plan motions*.

The other modules (deep blue, green, purple and bright orange) concern the low-level of planning and control for both the hand and the arm. The feasibility of the initial grasp is checked by *Check and Plan motions*. If the initial grasp is considered not feasible, another initial grasp is requested to light orange modules. If the grasp is feasible, it is performed and the object is picked, using the upper green module. The grasp, if not stable, is improved by trying to execute the same grasp with a few minor variations. By tracking the evolution of a grasp metric (section 2.1.2), the robot is able to move toward a stable grasp. This process is called babbling. The object is then lifted and moved, according to the task.

The two modules *Execute Grasp Transitions* and *Execute in-hand Actions* are performing low-level planning and control of in-hand manipulation. This includes getting the sequence of in-hand elementary actions to perform each transition of the grasp sequence. How these modules work is described in the section (5.1.2).

Database The database, that is represented in the middle of the flowchart in Fig. 5.26 stores all the useful information for manipulation.

As explained in section 5.1.2, the database stores information for each transition. Some key data are the sequence of elementary actions to perform with primitives of movement. The experience of the robot is also stored : the number of attempts and the number of successful achievements of the transition. This allows the robot to learn its own abilities as presented in section 5.1.2.

The task is manually given by the user. The *Master Controller* asks for it after the object has been acquired. The details of each task are stored in the database, each major step and configuration. The task grasp type, relevant for our algorithm, is

stored here. Additional information for manipulation is stored in the database. The list of known objects is a good example. It can be updated depending on the objects encountered during manipulation.

5.2.2 Implementation of the high-level planning algorithms

The platform has been presented in section 5.2.1 and the details of our algorithm in chapter 4. In section 5.1.1 how we improved the policies from human demonstration was detailed. These policies are available for our programs. In this section is presented the last step of our work: the implementation of our solution on the real robot.

Context

Before integrating our work to the HANDLE manipulation stack, all the code was available in Python, in the form of a library. We implemented the ROS node: *Generate grasp sequence* (Fig. 5.27) and integrated it to the whole control framework of the platform. This has been done without a major effort. ROS is designed to make the integration of external code easy. Wrapping our programs to make them usable with ROS was pretty straight-forward. This aspect of ROS enables it to easily connect with other robotic operating software.

The input and output formats of the messages have been defined consulting the other partners. The main work was to implement the different functions of our node. The functions have been coded as ROS services. Services have to be called by another node, in this case the *Master Controller*. The query is then sent, containing the input information for our program. Our called service returns the desired output to the *Master Controller* that is then able to trigger the next node.

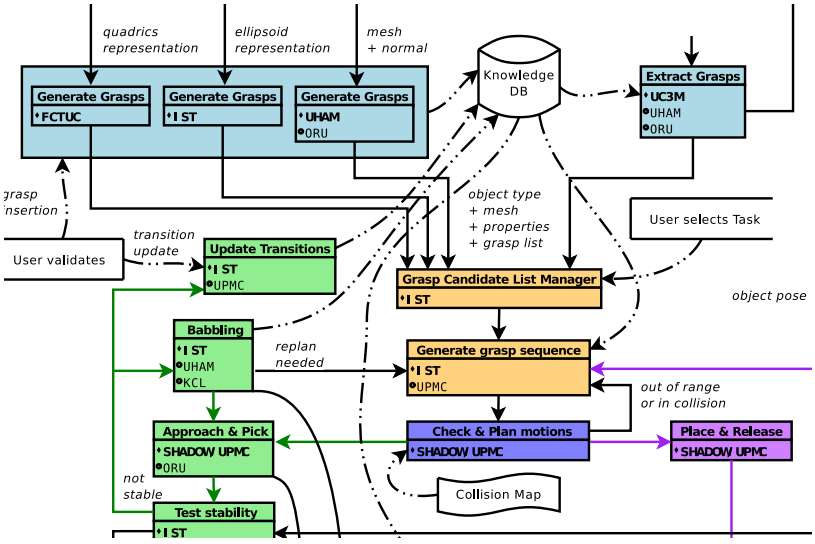


Figure 5.27: Flowchart of the HANDLE manipulation control framework: detail of the *Generate grasp sequence* node.

Input messages

The *Master Controller* sends a query message to one of our services. This message contains the information our program needs to generate a sequence from these data: the object, the initial grasp and the task grasp.

- The object message type: from the vision module and the object recognition modules, we receive information about the object: type of shape, dimensions, etc. However, it does not correspond to the classification we use. We had to translate these data given about the object into one of the classes used by our algorithm. This was done using the Naive Bayes Classifier presented in section 4.2.

- Initial Grasp configuration: some modules generate grasp configurations, all the joint angles, forces, synergies and grasp types, according to the classification we use, described in section 4.1.1. The grasp type of a grasp configuration is known from the beginning as the grasp generator algorithm builds a grasp from a chosen grasp type. So, an autonomous grasp type classifier is not required here. It is valid either for a candidate initial grasp or an initial grasp already executed.

- Task grasp type: the desired grasp defined by the task that has been ordered to the robot. A specific database stores task details with associated grasp type, or details allowing to generate a grasp and the subsequent grasp type. For example, writing with a pen is a task that requires a specific grasp type named "writing tripod", while finely rotating a button requires a pinch grasp to be able to perform precise movements. The grasp type related to the task and other details about the task could be automatically acquired after learning from human activity observation. This implies rebuilding human tasks and activities, and requires conceptualization. This field of research connects to cognitive sciences, as briefly presented in section 3.3, and this whole part of robotics is not addressed in this work, neither in the HANDLE project. The grasp type of the task is consequently hard coded in the database along with other task details. The task is asked by the *Master Controller* to the user, who selects one in the existing list.

The output messages depend on the service. They are detailed afterwards in each service description.

Service generating grasp sequences

This service is called *Grasp sequence generation*. This is the standard function. It results from the main goal of our method: planning high-level in-hand manipulation. As input, it is given an initial grasp and a final grasp. It generates a sequence of grasp types. So, when the system is under a given grasp type, it asks for a sequence to reach the final grasp. It is also used in case of unexpected events. The system then lies in an unexpected grasp type. It needs to replan a new sequence to reach the

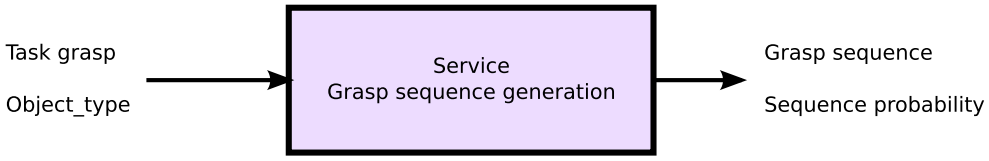


Figure 5.28: Function 1 : service *Grasp sequence generation.*

final grasp type as the previous sequence is outdated. This is called replanning. As our method is nearly instantaneous, it is able to perform replanning on-the-fly.

The output message in this case is a grasp sequence: a vector of grasp types.

Service ranking initial grasps

This function is more used than the previous function. During manipulation, at the beginning, the object is not grasped, the robot has to choose an initial grasp to get the object in hand and then perform in-hand manipulation. The initial grasp has to be achievable given the reachability of the object (range, obstacles, kinematics). We call this constraint "achievability". The grasp also has to be stable, which can be evaluated by a grasp quality metric. Modules in the framework (light blue on the chart Fig. 5.27) generate possible initial grasps. Using a measure of grasp quality, the best grasp is selected. It is what was usually done in existing works. The low level of planning and control are able to check the feasibility of the grasp. But our algorithm is able to give additional information about the ability of the system to reach the task grasp from a candidate initial grasp. This is done by considering the candidate grasp type and generating the corresponding sequence to reach the task grasp. Then, the probability of success of this sequence can be assessed. Each of these transitions has a probability of success taken from the state-transition probability model $\mathbf{P}$ of the MDP. The product of each of the transition probabilities gives the probability of success of the sequence. For each initial grasp candidate, we then have a probability of success to reach the task grasp. This task relevance measurement allows us to choose the initial grasp not only with achievability and grasp quality considerations but also using the influence of the task.

The choice of the initial grasp should take into account the influence of all the previously mentioned constraints: achievability, grasp quality and task relevance. Checking the achievability of a grasp by the low-level modules is very time consuming as it has to deal with a lot of parameters, planning the complete movement of the hand and arm. We are forced to select the best grasps given the other parameters and then test these selected grasps, starting from the best. The first tested grasp considered achievable is then performed.

We still need to merge grasp quality and task relevance considerations. Different methods are possible :

- A few feasible grasps considered stable enough are selected. This means that there is a threshold on the grasp metric stating that a grasp is stable. Then,

these grasps are ranked considering their task relevance. The most relevant, the one with the highest probability of success to reach the task grasp, is selected.

- A few grasp types considered task relevant are selected. This means that there is a threshold on the probability of success of a sequence. Then, these grasp types are used to generate grasp configurations, which are ranked considering their grasp quality metric. The one with the highest grasp metric is selected. In both methods, the ranking can be done at first, and then, starting from the highest ranked grasp, the other metric is checked, and the first grasp above the threshold is selected.

- We can also design a metric that combines both grasp quality and task relevance. The main problem for designing this metric is to make the grasp quality metric and the task relevance assessment comparable. Once this is done, the global metric can be the weighted sum of the two metrics (achievability and task relevance). The weights should be chosen to tune the importance of the two criteria.

Figure 5.29: Function 2: service *Initial list ranking*.

We choose to operate using the first method. This leads to a service called *Initial grasp list ranking* illustrated in Fig. 5.29. Many possible grasp configurations are generated in very high numbers and ranked using their quality metric. The best ones are selected. This is the role of the node *Grasp Candidate List Manager*. It receives grasp configurations from the different grasp generation modules. Some modules provide more relevant grasps than others. For example, modules using object recognition, drawing known grasps from the database, are taken with higher consideration than those generated from the object reconstruction. Among the generated grasps, *Grasp Candidate List Manager* selects a given number of grasps. It chooses the best ones, giving priority to those drawn from the database.

The *Master Controller* sends this list as a query to our *Generate grasp sequence* using the first service. The service returns a ranked list of grasp configurations. And the first one is sent for achievability test to the low-level modules. Some configurations given to our *Generate grasp sequence* might have the same grasp type. In this case, the one with the highest grasp quality metric is put first.

The output message is then a vector of ranked grasp configurations. For each line, the grasp configuration is given with its task relevance measure and the corresponding sequence (a vector of grasp types) so that the system does not have to rebuild it.

5.2.3 Demonstrations

To illustrate the whole operation of the platform and its control framework, two demonstrations are shown. The description of every step gives an example of the practical relevance of what was developed in this thesis.

Coke can scenario

The coke can scenario consists in manipulating a can and performing a given task (manually chosen by the user). The task description is hard coded in the database: pouring liquid from the can in the other container. Task related informations like relative positions of the two objects when pouring, rotation angles of the can when pouring or grasp type associated to the task are all described in the database.

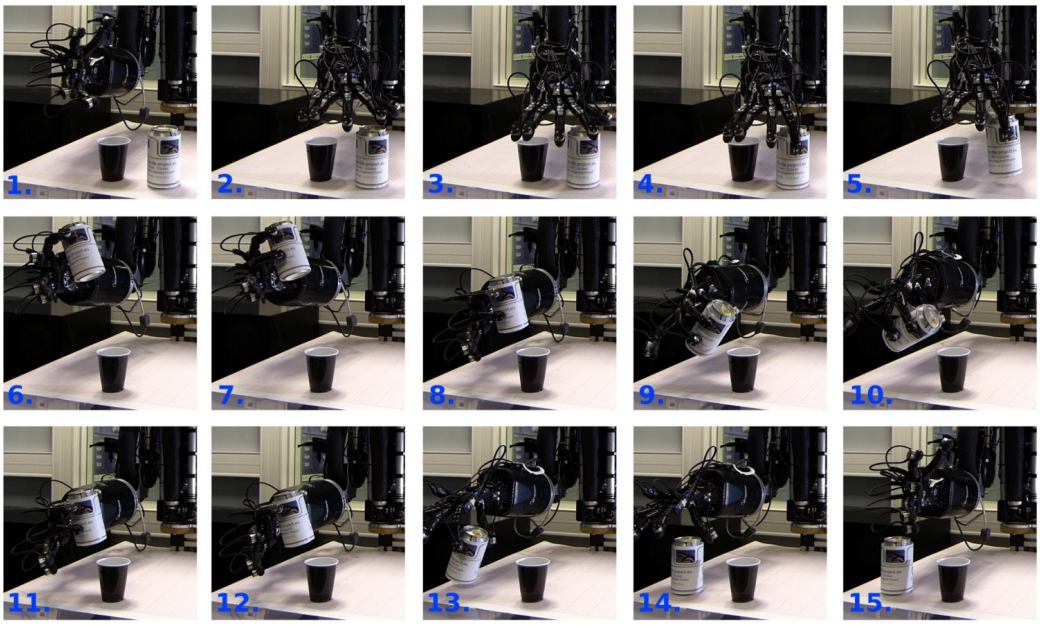


Figure 5.30: Illustration of a demonstration with the HANDLE platform. Manipulation of a coke can to achieve pouring into another container is shown step-by-step.

The different steps of the demonstration are shown in Fig. 5.30. In the beginning, the robot is at step 1. The hand is in a random starting position. The environment is observed using the Kinect and vision modules. After extracting the table, the point clouds of the two objects are segmented. The coke can is recognized. The user chooses which object to use for the task.

The user manually enters the task after being asked through the user interface. The task grasp type is automatically extracted from the database by the *Master Controller*. It is grasp type number 8: Prismatic 2 finger (Fig. 5.31). A number of stable initial grasps are generated from the database as well as autonomously computed from the point cloud of the object. All these stable grasp configurations are gathered by the *Grasp Candidate List Manager*. These initial grasp configurations are ranked according to their task relevance, the probability of success of the sequence

Figure 5.31: Task grasp type for pouring: number 8, prismatic 2 finger.

implied by the initial grasp type to reach the task grasp type. No direct stable grasp configuration using grasp type 8 (the task grasp type) has been generated, as the object cannot be reached from the side because the other container is in the way. Otherwise, it would have been chosen, as grasping initially with the task grasp type had the highest chance of success.

The first initial grasp in the list is number 9: Palmar Pinch. The sequence to reach the task grasp is direct: $\{9, 8\}$. The *Check and Plan motions* modules check the feasibility of the grasp. The result is that it is feasible from top but not from the side due to the obstacle. Consequently, the robot starts to perform the initial grasp. The robot gets the hand in a preshape close to the object (steps 2 and 3) and then closes the fingers. Once in contact, the actual grasp quality metric is calculated and if needed, the grasp is improved by babbling (step 4). The object is then lifted (step 5).

Figure 5.32: First initial grasp of the list: number 9, Palmar Pinch.

The first transition, from grasp type 6 to grasp type 9 is planned. The transition details are stored in the database: first, the hand turns around the object using a gravity rotation, and then, the middle finger is closed on the object. This is performed in steps 6 and 7 by the low level algorithms. In steps 8, 9, 10 and 11, the task is performed, using the wrist joint.

Using the grasp sequence generator, the sequence of grasp types to reach the release grasp (grasp 9) is generated. It is exactly the inverse of the sequence to reach

the task grasp. The middle finger is removed from the object (step 12), a gravity rotation is performed (step 14) and the object is released. The hand goes back to a standard position (step 14).

Pipette scenario

In the pipette scenario, the hand fetches a pipette and uses it to put a few drops in a target. The accomplishement of this task is shown step-by-step in Fig. 5.33.

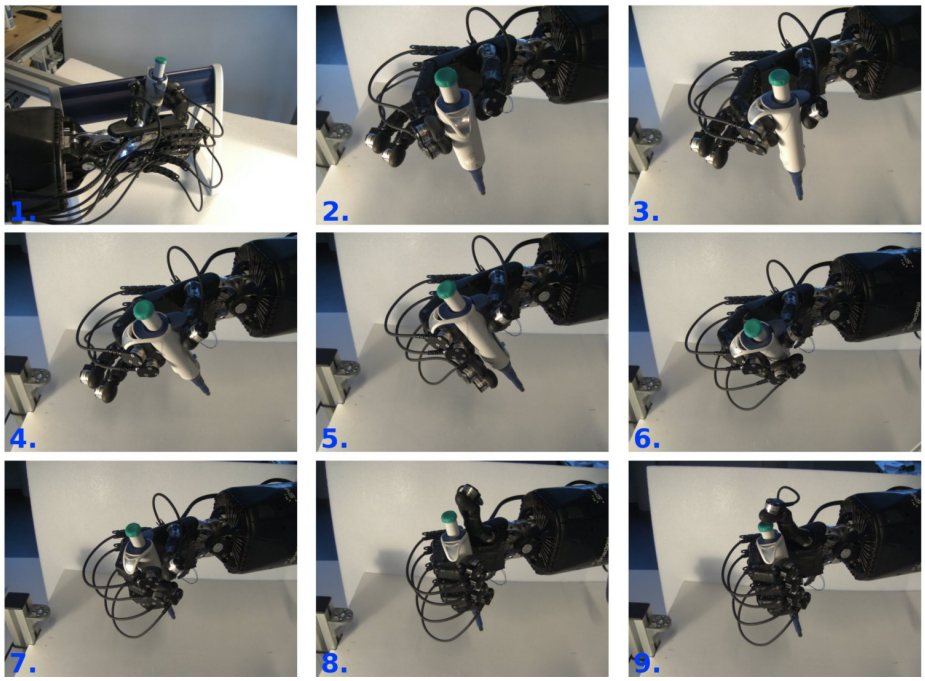


Figure 5.33: Step-by-step illustration of a second demonstration by the HANDLE platform with a pipette.

The pipette is recognized by the vision module. The user manually enters the task after being asked through the user interface. The task grasp type is extracted from the database. It is number 15, Fixed Hook (Fig. 5.34). In the beginning of the scenario, the pipette is hanging on a support. The task grasp type is not directly achievable because the hand cannot wrap the pipette without colliding with the stand. The *Grasp Candidate List Manager* selects a list of stable grasps that are ranked according to their task relevance. The first initial grasp in the list is number 8 : Prismatic 2 finger. The sequence to reach the task grasp is: 8 (step 1), 6 (steps 2, 3 and 4), 22 (step 5), 2 (step 7), and 15 (step 8).

Even if the pipette moves only a little relatively to the hand, this example involves four transitions and is consequently more complex than the single transition sequence of the coke can scenario. They involve very small movements of the fingers. The decomposition of the whole in-hand manipulation activity into simple steps is clear here. The four transitions are simple to perform, the low-level layer of control is able to plan and execute them efficiently.

Figure 5.34: Task grasp type for the pipette: number 15, Fixed Hook.

5.2.4 Conclusion about the implementation

Our solution has been successfully implemented on the platform. It forms a central part in the control framework. Our algorithm uses the given policies to generate subgoal sequences for in-hand manipulation according to the object. As expected, the subgoal sequences provide simple transitions that are performed successfully using the low-level layer. Some information about the transitions between subgoals are hard-coded. This limitation comes from the fact that we started with a simple model: the state is solely composed by grasp types. Because of this, the link between our algorithm and the current low-level layer of the robot is not complete. To make the system fully autonomous, our solution has to be enhanced. This is discussed in the conclusion of this manuscript.

One of the main utilities of our solution is its ability to choose an initial grasp that is suited to the demanded task. We were able to use the subgoal sequences to assess the task relevance of possible initial grasps and make the robot choose among generated stable grasps the ones that enable it to reach the final configuration more easily, which is remarkable. This is possible because our algorithm is very fast. This speed of computation also allows replanning online the sequence to reach the final grasp configuration when the robot's context requires it.

Chapter 6

Conclusion

In this manuscript, we provided a solution to plan in-hand manipulation. Efficient autonomous in-hand manipulation skills would enable robots to greatly improve their versatility and be able to act in a human environment. They would play a larger role in everyday life by handling any object used by humans. But robotic manipulation using a human-like hand is still at an early stage. While the planning and the execution of the atomic actions composing in-hand manipulation can use various efficient techniques, in-hand manipulation planning is missing competitive high-level solutions. This statement is justified by the state-of-the-art presented in section 2.2. Some high-level solutions exist. However, they do not consider every possible in-hand action, and no complete framework has been tried on a robot yet.

6.1 Outcome of the work

We proposed an architecture based on Markov decision processes (MDP) capable of representing task goals, object dependencies and motor skills in the sequencing of actions to achieve the goals. In particular, we proposed a solution for the selection of the initial grasp of an object, depending on the task and on contextual constraints, a problem still open in the robotic grasp field.

Our solution generates subgoal sequences from an initial configuration to a final one. When the object is not grasped yet, a list of candidate initial grasp configurations is considered. Otherwise, the initial grasp configuration is the current configuration of the system at the time of planning. The final configuration is required by the task. We model the subgoal sequence as an object dependent MDP, and the chosen subgoals are grasp types taken from an existing taxonomy. In-hand manipulation is broken down into successive transitions between grasp types. A transition is modelled as a probability of successfully reaching the next grasp type. This probability depends on the object, the current and the next subgoal. The evolution of the MDP over time is driven by these transition probabilities, and the reward put at the desired goal state. The policy is the solution of the MDP: it indicates the action to be chosen in order to reach the final configuration.

Learning appears mandatory to tackle such a high-dimension problem. That is the working solution for humans who start exploring and learning the affordances of objects since very early ages. Our solution is able to learn from human demonstration and from robot self-exploration. We initialized the transition probabilities of the MDP using an empirical analysis of every possible transition, then we refined our robot's behaviour through a learning process. First, we computed all the policies by solving the empirically initialized MDP for every possible task, represented by the end state. Then, we recorded human in-hand manipulation movements and improved the policies by learning from them. We used active learning to boost the learning process by choosing the samples most likely to provide the most valuable knowledge. We proved the viability of our learning algorithm on a simulated human. However, active learning does not bring significant improvements in the learning speed compared to batch learning. It can boost the learning process only for the first samples, but then it saturates.

Learning from humans is useful as an initialization of the robot's behaviour, but limited. The robotic hand is similar in design to a human hand, but its atomic in-hand action capabilities are far from it. Learning from self-experience is therefore the only way to adapt the algorithm to the robot's real capabilities. Our method using a Markov Decision Process allows it without changing the formulation. We implemented a primitive algorithm to learn from self experience, and it can be improved in future work.

We implemented our solution on the HANDLE platform. It holds a central role in the control framework. The grasp sequence generator algorithm is fast, so we are able to use it to assess the probability of reaching the task configuration from every possible initial grasp. By this mean, our robot chooses the initial configuration among the generated grasps taking into account the task relevance. The speed of computation enables replanning on-line, in case the robot finds itself in an unexpected configuration and needs a new plan. The solution fits well to the HANDLE project control framework and is consistent with our analysis of the state-of-the-art of in-hand manipulation planning methods. Our approach integrates the low-level techniques without much burden. Each of the low-level algorithms is specific to some atomic actions, and no global solution can replace them.

The whole problem is broken down to simple transitions that are easy to perform by the low-level layer. The MDP model is probabilistic, it is adapted to uncertainty in the environment. If the subgoal sequence turns out to be impossible at the low-level planning step because the task impose constraints that are not taken into account in our model, an alternative plan is readily formulated. If the execution fails and the system finds itself in an unexpected configuration, replanning is done quickly online.

From these considerations we are confident that the objectives of this thesis have been reached. Now let us explore which improvements can be made in further work on our solution.

6.2 Further work

6.2.1 Validating the learning process

One thing that was hard to validate during the proceeding of this thesis work was the efficiency of the policies obtained through our learning method. The algorithm generates grasp sequences that appear human-like but we did not perform a complete and strict validation of the quality of the sequences. The learning process should be completed for an object at least. Then, a validation set of samples should be recorded from human demonstration. This operation would require a huge effort to obtain policies which are not perfectly adapted to the robot. Now that a platform is available, it is more interesting to perform learning from self-experience.

We performed an empirical analysis to initialize the learning process in a better way than random initialization. To assess the gain brought by this initialization, we should perform the complete learning process using the empirical analysis with a complete learning process executed from scratch (random initialization), which is certainly more time consuming. This would require even more work while focusing on self-experience is more relevant, as explained in the next paragraph.

The benefits of the active learning technique could also be observed by comparing its performance while learning from humans with a batch learning (randomly choosing the samples) method. The study realized on a simulated human revealed the limited benefits of our entropy minimization strategy. We could try other metrics to optimize our active learning approach and use it on the self-experience learning algorithm.

6.2.2 Learning the probabilistic transition model from self-experience

Learning from humans implies that we consider the robot's abilities close to the human ones. In addition, collecting the samples is very heavy, as it involves a manual annotation of the recordings. We figured out that switching to robot self-experiment is more relevant. We implemented a primitive algorithm learning the policy, but learning the probabilistic transition model from the robot itself is more efficient. The probabilistic transition model has a lower complexity than the policy and then would require less trials to learn it. Moreover, it would be simply more realistic and adapted to the robot's real capabilities. However, the possibility to learn from self-experience came very late on the time course of this thesis and we could not validate it properly. Learning from humans does not have to be very accurate, as it will be used only as an initialization for self-experience learning. It is even worth considering the use of the initialized policies obtained from the empirical analysis of the transitions probabilities.

In the current state of the platform, it is possible to learn the probabilistic transition model. This learning process could be run autonomously. The robot could perform in-hand transitions between subgoals by itself. However, to do it in a completely automatic fashion, it should be able to detect failures: fall of the object or reach of a different grasp type. We would need an automatic detection of grasp types, an in-hand tracker of the object, and a tracker of the hand. A simulated human model

could be used similarly to what we did in 5.1.1 to validate the learning algorithm. This time, the simulated human would be built directly as a MDP.

6.2.3 Enhancing our state representation

The solution we demonstrated needs some hard-coded knowledge about transitions but it can be corrected and made fully autonomous. Our model is simple: we only use grasp types as subgoals. However, this puts a lot of responsibility on the low-level controllers. To illustrate this limitation, we consider a manipulation activity that implies a sequence involving the same grasp type in two different positions on the object. The plan should be done by the low-level because from the point of view of our high-level controller, the same grasp type means the same state. As the state is solely the grasp type, the change in position is not considered as a different state. A sequence with a repeated grasp type as shown in Fig. 6.1 cannot be generated by our current planner.

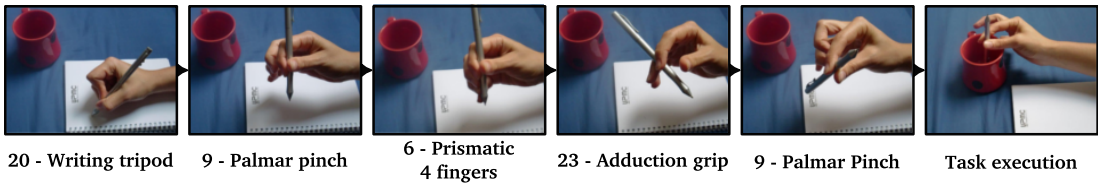


Figure 6.1: A sequence of in-hand manipulation with a repeated grasp type.

This highlights the need to enhance our state representation. As presented in section 3.5, a more accurate model should take into account the hand/object relative position. The state representation of the subgoals would be the combination of the grasp type and the position of the object relatively to the hand. The empirical analysis of transitions should be adapted and completed to be used in this improved solution.

6.2.4 Generating the atomic actions sequence for transitions

Once the subgoals describe in-hand manipulation with enough details (grasp type and hand/object relative position), we could consider associating transitions of the MDP to atomic actions or a sequence of atomic actions. This would bridge the gap with the low-level layer in charge of planning the in-hand atomic actions. For now, the relative hand/object position and the sequence of atomic actions are hard-coded for every transitions between two subgoals.

6.3 Further work in robotic manipulation

6.3.1 Towards a higher level of decision

Our work makes the assumption that the task is known. Hard coded information on the task is required with our method. To make a manipulation robot even more

autonomous, we could connect our solution with methods developed to make the robot characterize, learn and recognize the tasks. This would result in an additional layer, or in a module that provides the task details for our solution. It needs work from the field of neuroscience as presented in section 3.3, together with great improvements in environment recognition to analyse the task while performing it or seeing it being performed.

6.3.2 Development of the low-level capabilities

The platform atomic action planning capabilities have been presented in 5.1.2. The complete set of possible in-hand atomic actions is not implemented yet. The platform can be improved to make the robot more dexterous.

Additionally, the design of the hand can be upgraded. As stated in 5.1.2, the largest gap between a human hand and our robotic hand is its sensing abilities. The integration of a tactile skin would allow more dexterous manipulation by leading to more precise grasp recognition and better atomic action planning.

6.4 Concluding note: ethical issues in future robotics

This work intends to make robotic manipulation more effective. Once this progress is large enough, it will profit to industrial robotics, assistance robotics and any other robot in their ability to replace humans at tasks involving manipulation. This technical and scientific progress would improve the overall productivity of the society. It will reduce the amount of work that humans have to fulfil.

Such a situation has already been witnessed during the Agricultural Revolution. Let us consider the French example. In 1955, 6.2 million people worked in French agriculture, corresponding to 31% of total jobs. In 2000, this figure dropped to 4,8% of the jobs for only 1,3 million people. Meanwhile, the production of French agriculture doubled between 1960 and 2004 without any major changes in the land area dedicated to agriculture ([20]). This improvement was made possible by scientific progress in chemistry, the mechanisation of the tools and an evolution of agricultural habits. The benefits of this revolution are obvious, with the increase of food production, total population and life expectancy. This revolution, already started in the nineteenth century, took place in most developed countries. It made a large workforce available. A great percentage of the population could no longer work in agriculture and was available to work in the industry, which is a factor that participated to the Industrial Revolution.

Let us imagine a Robotic Revolution, with robots taking over a large part of the work in factories, home assistance and any job where robots could replace a human being. This would have two effects, taking place in unknown proportions. The productivity of human workers would grow, followed by the overall production of goods and their consumption. This would increase the environmental crisis we already face. Less humans would be needed to fulfil the material needs of society.

A large part of the population would become unemployed, creating social problems. Avoiding these problems requires to consider, detect and monitor them. Measures should be taken to limit the production and consumption of goods or adapt society to reduce the number of unemployed people. Efforts should be made to accept that a part of the population could live without a proper job and still fulfil their needs, especially regarding esteem. Benefits and drawbacks of these measures should be considered.

These ethical aspects should be kept in mind for the development of robotics. It is the duty of scientists to alert people on the risks deriving from the progress they work on and provide information on it. Then, the citizens through their elected representatives can decide what to do with this progress and which solution should be implemented. Robots are versatile, they can be adapted to the evolution we want for the world, and robotic manipulation is a tool that will be useful in any case.

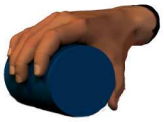
Appendix A

Taxonomy used

The taxonomy of canonical grasp types we use is taken from [24]. Here, we present the taxonomy as provided by its original author. Some abbreviations are used in the table:

- P: Palm
- 1-5: Thumb - little finger
- VF1: Virtual finger 1
- VF2: Virtual finger 2
- VF3: Virtual finger 3, used to opposed a task related force or momentum
- VF3: Virtual finger 3, used to opposed a task related force or momentum
- #: is the number of times the grasp type has been seen in previous taxonomies.

The concepts of opposition type and virtual finger have been explained in 3.5.2 and were first introduced in [35].

Nr.	Name	Picture	Type	Opp. Type	Thumb Pos.	VF1	VF2	VF3	#
1	Large Diameter		Power	Palm	Abd	P	2-5		10
2	Small Diameter		Power	Palm	Abd	P	2-5		3
3	Medium Wrap		Power	Palm	Abd	P	2-5		6
4	Adducted Thumb		Power	Palm	Add	P	2-5	1	2
5	Light Tool		Power	Palm	Add	P	2-5	(1)	2
6	Prismatic 4 Finger		Precision	Pad	Abd	1	2-5		4
7	Prismatic 3 Finger		Precision	Pad	Abd	1	2-4		4
8	Prismatic 2 Finger		Precision	Pad	Abd	1	2-3		2
9	Palmar Pinch		Precision	Pad	Abd	1	2		12

Nr.	Name	Picture	Type	Opp. Type	Thumb Pos.	VF1	VF2	VF3	#
10	Power Disk		Power	Palm	Abd	P	2-5		3
11	Power Sphere		Power	Palm	Abd	P	2-5		8
12	Precision Disk		Precision	Pad	Abd	1	2-5		2+ 2+ 1
13	Precision Sphere		Precision	Pad	Abd	1	2-5		6
14	Tripod		Precision	Pad	Abd	1	2-3		8
15	Fixed Hook		Power	Palm	Add	P	2-5		2
16	Lateral		Inter-mediate	Side	Add	1	2		12
17	Index Finger Extension		Power	Palm	Add	P	3-5	2	4
18	Extension Type		Power	Pad	Abd	1	2-4		1

Nr.	Name	Picture	Type	Opp. Type	Thumb Pos.	VF1	VF2	VF3	#
19	Distal Type		Power	Pad	Abd	1	2-5		2
20	Writing Tripod		Precision	Side	Abd	1	3		6
21	Tripod Variation		Inter-mediate	Side	Abd	1	3-4		1
22	Parallel Extension		Precision	Pad	Add	1	2-5		5
23	Adduction Grip		Inter-mediate	Side	Abd	1	2		3
24	Tip Pinch		Precision	Pad	Abd	1	2		9
25	Lateral Tripod		Inter-mediate	Side	Add	1	3		1
26	Sphere 4 Finger		Power	Pad	Abd	1	2-4		1
27	Quadpod		Precision	Pad	Abd	1	2-4		2

Nr.	Name	Picture	Type	Opp. Type	Thumb Pos.	VF1	VF2	VF3	#
28	Sphere 3 Finger		Power	Pad	Abd	1	2-3		1
29	Stick		Inter-mediate	Side	Add	1	2		1
30	Palmar		Power	Palm	Add	1	2-5		1
31	Ring		Power	Pad	Abd	1	2		1
32	Ventral		Inter-mediate	Side	Add	1	2		1
33	Inferior Pincer		Precision	Pad	Abd	1	2		1

Appendix B

Grasp transition feasability

In this appendix, we present the data resulting from the empirical analysis of the transition probabilities. The transitions are considered reversible, so only half of the table is required. Notations in the table have the following meanings:

- N: the transition is impossible.
- a: the transition is simple.
- h: the transition is complex.

Name	Nr																															
Large diameter	1																															
Small diameter	2	a																														
Medium Wrap	3	a	a																													
Abducted Thumb	4	a	a	a																												
Light tool	5	h	h	h	h																											
Prismatic 4 finger	6	h	h	h	h	h																										
Prismatic 3 finger	7	h	h	h	h	h	a																									
Prismatic 2 finger	8	h	h	h	h	h	a	a																								
Palmar Pinch	9	h	h	h	h	h	a	a	a																							
Power Disk	10	N	N	N	N	N	N	N	N	N																						
Power Sphere	11	N	N	N	N	N	N	N	N	N	N																					
Precision Disk	12	N	N	N	N	N	N	N	N	N	h	N																				
Precision Sphere	13	N	N	N	N	N	N	N	h	h	N	h	N																			
Tripod	14	N	N	N	N	N	N	N	N	a	N	h	N	N																		
Fixed Hook	15	a	a	a	a	a	h	h	h	h	N	N	N	N	N																	
Lateral	16	N	N	N	N	N	h	h	h	h	N	N	N	N	N	N																
Index Finger Extension	17	a	a	a	a	a	h	h	h	h	N	N	N	N	N	h	N															
Extension Type	18	N	N	N	N	N	N	N	N	N	h	N	h	N	N	N	h	N														
Distal Type	19	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N													
Writing Tripod	20	N	N	N	N	N	N	N	N	a	N	N	N	N	N	N	N	N	N	N												
Tripod Variation	21	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N											
Parallel Extension	22	a	h	h	h	N	a	a	a	a	a	h	h	h	h	N	a	N	a	N	N	N										
Adduction Grip	23	N	N	N	N	h	h	h	h	a	N	N	N	N	a	N	N	h	N	N	h	N	h									
Tip Pinch	24	N	h	h	a	h	N	N	N	N	N	h	N	h	h	h	h	h	a	N	h	N	h	N								
Lateral Tripod	25	N	N	N	N	N	N	N	N	a	N	h	N	h	h	N	N	N	N	N	a	N	h	N	h							
Sphere 4 Finger	26	N	N	N	N	N	N	N	N	h	N	h	N	h	h	N	N	N	N	N	N	N	h	h	h	h						
Quadpod	27	N	N	N	N	N	N	N	N	a	N	h	N	a	a	N	N	N	N	N	N	N	h	N	h	a	h					
Sphere 3 Finger	28	N	N	N	N	N	N	N	N	h	N	h	N	h	h	N	N	N	N	N	N	N	h	h	h	h	h	h				
Stick	29	a	a	a	a	a	h	h	h	h	N	N	N	N	N	a	N	a	N	N	N	N	a	h	h	N	N	N	N			
Palmar	30	N	N	N	N	N	h	h	h	h	a	N	h	N	N	N	N	N	h	N	N	N	h	N	h	N	N	N	N	N		
Ring	31	a	a	a	h	h	h	h	h	h	N	h	N	h	h	a	N	h	N	N	a	N	a	h	N	N	h	h	a	a	N	
Ventral	32	a	a	a	a	a	h	h	h	h	N	N	N	N	N	a	N	a	N	N	N	N	a	h	h	N	N	N	N	a	N	a
Inferior Pincer	33	a	h	h	h	h	h	h	h	N	h	h	h	h	h	a	h	a	N	a	N	a	h	h	h	h	h	a	h	h	a	h
3 last fingers	40	a	a	a	a	a	N	N	N	a	N	N	N	N	N	a	N	a	N	N	N	N	h	h	h	N	N	N	N	a	N	N
2 last fingers	41	a	a	a	a	a	N	N	N	a	N	N	N	N	N	a	N	h	N	N	N	N	h	h	h	N	N	N	N	a	N	N
	Nr	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	40	41

Appendix C

Publications

The work presented in this manuscript has been shared with the scientific community in the following publications:

- **International conference with proceedings:**
  Urbain Prieur, Véronique Perdereau, Alexandre Bernardino. Modeling and Planning High-Level In-Hand Manipulation Actions from Human Knowledge and Active Learning from Demonstration. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

- **National conference with proceedings:**
  Urbain Prieur, Véronique Perdereau, Alexandre Bernardino. Génération de séquences de saisies pour la manipulation dextre robotique. *6èmes Journées Nationales de la Robotique Humanoide*, 2011.

- **Poster presentation:**
  Urbain Prieur, Véronique Perdereau, Alexandre Bernardino. Grasp sequence generation for planning robotic in-hand manipulation. In *HANDLE training workshop for young researchers and Ph.D. students*, 2012.

Bibliography

[1] African robotics network (afron). http://robotics-africa.org/. 10 Dollars Robot Design Challenge.

[2] Barrett hand, barrett technology inc. http://www.barrett.com/robot/products-hand.htm.

[3] Cyberglove systems. http://www.cyberglovesystems.com/.

[4] Pressure profile systems, inc. http://www.pressureprofile.com.

[5] Roboearth. http://www.roboearth.org. Cloud robotics infrastructure.

[6] Hagai Attias. Planning by probabilistic inference. In *Proceedings of the 9th International Conference on Artificial Intelligence and Statistics*, 2003.

[7] D. Berenson and S.S. Srinivasa. Grasp synthesis in cluttered environments for dexterous hands. In *8th IEEE-RAS International Conference on Humanoid Robots, 2008. Humanoids 2008.*, pages 189 –196, dec. 2008.

[8] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35:99–109, 1943.

[9] Antonio Bicchi, Marco Gabiccini, and Marco Santello. Modelling natural and artificial hands with synergies. *Phil. Trans. R. Soc. B*, 366:3153–3161, October 2011.

[10] Ch. Borst, M. Fischer, and G. Hirzinger. A fast and robust grasp planner for arbitrary 3d objects. In *IEEE International Conference on Robotics and Automation, 1999.*, volume 3, pages 1890 –1896 vol.3, 1999.

[11] Ian M. Bullock, Raymond R. Ma, and Aaron M. Dollar. A hand-centric classification of human and robot dexterous manipulation. *IEEE TRANSACTIONS ON HAPTICS*, 2013.

[12] Moez Cherif and Kalmal K. Gupta. Planning quasi-static fingertip manipulations for reconfiguring objects. *IEEE Transactions on Robotics and Automation*, 15(5):837–848, 1999.

[13] Noam Chomsky. *The Minimalist Program*. MA:MIT Press, 1995.

[14] Matei T. Ciocarlie and Peter K. Allen. Hand posture subspaces for dexterous robotic grasping. *The International Journal of Robotics Research*, 28(7):851–867, July 2009.

[15] J. A. Corrales Ramon, V. Perdereau, and F. Torres Medina. Multi-fingered robotic hand planner for object reconfiguration through a rolling contact evolution model. In *ICRA Conference Proceedings*, pages 1–6, Karlsruhe, Germany, May 2013. IEEE.

[16] Steve Cousins. Introduction of willow garage. Talk given in ISIR Paris, November 2011.

[17] John J. Craig. *Introduction to Robotics Mechanics and Control*. Pearson Education International, 2004.

[18] M.R. Cutkosky. On grasp choice, grasp models, and the design of hands for manufacturing tasks. *IEEE Transactions on Robotics and Automation*, 5(3):269 –279, jun 1989.

[19] M.R. Cutkosky and P. Wright. Modeling manufacturing grips and correlations with the design of robotic hands. 3:1533–1539, 1986.

[20] Maurice Desriers. L'agriculture française depuis cinquante ans : des petites exploitations familiales aux droits à paiement unique. INSEE website, 2007. L'agriculture, nouveaux défis.

[21] R. Detry, D. Kraft, O. Kroemer, L. Bodenhagen, J. Peters, N. Kruger, and J. Piater. Learning grasp affordance densities. *Paladyn. Journal of Behavioral Robotics*, 2(1):1–17, 2011.

[22] Renaud Detry, Emre Başeski, Mila Popovic, Younes Touati, Norbert Kruger, Oliver Kroemer, Jan Peters, and Justus Piater. Learning continuous grasp affordances by sensorimotor exploration. In Olivier Sigaud and Jan Peters, editors, *From Motor Learning to Interaction Learning in Robots*, pages 451–465. Springer-Verlag, 2010.

[23] J.M. Elliot and K.J Connolly. A classification of manipulative hand movements. *Developmental Medicine & Child Neurology*, 26:283–296, 1984.

[24] T. Feix. The generation of a comprehensive grasp taxonomy. Technical report, KTH Royal Institute of Technology, 2009.

[25] C. Ferrari and J. Canny. Planning optimal grasps. In *IEEE International Conference on Robotics and Automation, 1992.*, pages 2290 –2295 vol.3, may 1992.

[26] David Flavigné and Veronique Perdereau. Representing synergies for dexterous robot manipulation. Poster Presentation at Workshop, February 2012. HANDLE training workshop for young researchers and Ph.D. students.

[27] M. Gabiccini, A. Bicchi, D. Prattichizzo, and M. Malvezzi. On the role of hand synergies in the optimal choice of grasping forces. *Auton. Robots*, 31(2-3):235–252, October 2011.

[28] Sebastian Geidenstam, Kai Huebner, Daniel Banksell, and Danica Kragic. Learning of 2d grasping strategies from box-based 3d object approximations. In *Robotics: Science and Systems*, 2009.

[29] Bill Goodwine. Stratified motion planning with application to robotic finger gaiting. In *Proc. IFAC World Congress*, 1999.

[30] Yisheng Guan and Hong Zhang. Kinematic feasibility analysis of 3-d multifingered grasps. *IEEE Transactions on Robotics and Automation*, 19(3):507–513, 2003.

[31] K. Harada, K. Kaneko, and F. Kanehiro. Fast grasp planning for hand/arm systems based on convex model. In *IEEE International Conference on Robotics and Automation, 2008. ICRA 2008.*, pages 1162 –1168, may 2008.

[32] Istvan Harmati, Bela Lantos, and Shahram Payandeh. On fitted stratified and semi-stratified geometric manipulation planning with fingertip relocations. *The International Journal of Robotics Research*, 21(5-6):489–510, 2002.

[33] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[34] David Heckerman. A tutorial on learning with bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, March 1995.

[35] T. Iberall. The nature of human prehension: Three dextrous hands in one. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 396 – 401, mar 1987.

[36] Petar Kormushev, S. Calinon, R. Saegusa, and G. Metta. Learning the skill of archery by a humanoid robot iCub. In *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, pages 417–423, Nashville, TN, USA, December 2010.

[37] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[38] Tomas Kulvicius, Kejun Ning, Minija Tamosiunaite, and Florentin Worgotter. Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting. *IEEE transactions on robotics*, 28:145–157, 2012.

[39] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Iowa State University, 1998.

[40] Zexiang Li, J.F. Canny, and S.S. Sastry. On motion planning for dexterous manipulation. i. the problem formulation. In *IEEE International Conference on Robotics and Automation, 1989.*, pages 775–780 vol.2, 1989.

[41] T. Lozano-Perez, J. Jones, E. Mazer, P. O'Donnell, W. Grimson, P. Tournassoud, and A. Lanusse. Handey: A robot system that recognizes, plans, and manipulates. In *IEEE International Conference on Robotics and Automation 1987*, volume 4, pages 843 – 849, mar 1987.

[42] D. Lyons. A simple set of grasps for a dextrous hand. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 588 – 593, March 1985.

[43] A.T. Miller, S. Knoop, H.I. Christensen, and P.K. Allen. Automatic grasp planning using shape primitives. In *IEEE International Conference on Robotics and Automation, 2003. ICRA '03.*, volume 2, pages 1824 – 1829 vol.2, sept. 2003.

[44] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[45] Tom Mitchell. Human and machine learning. In *Machine Learning Departement*. Carnegie Mellon University, November 2006.

[46] David J. Montana. The kinematics of contact and grasp. *The Int. Journal of Robotics Research*, 7:17–32, 1988.

[47] L. Montesano and M. Lopes. Learning grasping affordances from local visual descriptors. In *IEEE 8TH International Conference on Development and Learning*, China, 2009.

[48] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor. Learning object affordances: From sensory–motor coordination to imitation. *IEEE Transactions on Robotics*, 24(1):15 –26, February 2008.

[49] Luis Montesano and Manuel Lopes. Active learning of visual descriptors for grasping using non-parametric smoothed beta distributions. *Robotics and Autonomous Systems*, 60:452–462, 2012.

[50] Luis Montesano, Manuel Lopes, Alexandre Bernardino, and José Santos-Victor. Modeling affordances using bayesian networks. In *IEEE - Intelligent Robotic Systems (IROS'07)*, pages 4102 – 4107, USA, 2007.

[51] Kevin Murphy. Bayesian network toolbox. https://code.google.com/p/bnt/.

[52] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[53] J. R. Napier. The prehensile movements of the human hand. *The Journal of bone and joint surgery*, British volume 38-B(4):902–913, 1956.

[54] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[55] V.-D. Nguyen. Constructing force-closure grasps. In *IEEE International Conference on Robotics and Automation 1986.*, volume 3, pages 1368 – 1373, apr 1986.

[56] V.-D. Nguyen. Constructing force-closure grasps in 3d. In *IEEE International Conference on Robotics and Automation 1987.*, volume 4, pages 240 – 245, mar 1987.

[57] Judea Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *7th Conference of the Cognitive Science Society*, pages 329–334. University of California, 1985.

[58] Marco Santello, Martha Flanders, and John F. Soechting. Postural hand synergies for tool use. *The Journal of Neuroscience*, 18:10105–10115, December 1998.

[59] Jean-Philippe Saut, Anis Sahbani, Sahar El-Khoury, and Veronique Perdereau. Dexterous manipulation planning using probabilistic roadmaps in continuous grasp subspaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2907–2912, 2007.

[60] Ashutosh Saxena, Justin Driemeyer, Justin Kearns, and Andrew Y. Ng. Robotic grasping of novel objects. In *Neural Information Processing Systems (NIPS)*, volume 19, 2006.

[61] Wolfram Schultz, Peter Dayan, and P. Read Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, March 1997.

[62] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics Modelling, Planning and Control*. Springer, 2009.

[63] D. Song, K. Huebner, V. Kyrki, and D. Kragic. Learning task constraints in robot grasping. In *4th International Conference on Cognitive Systems (CogSys'10)*, 2010.

[64] Douglas Summers-Stay, Ching L. Teo, Yezhou Yang, Cornelia Fermuller, and Yiannis Aloimonos. Using a minimal action grammar for activity understanding in the real world. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[65] Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Morgan & Claypool, July 2010.

[66] Simon Tong and Daphne Koller. Active learning for parameter estimation in bayesian networks. In *Neural Information Processing Systems 2000*, pages 647–653. NIPS, 2001.

[67] Marc Toussaint. Bayesian inference for motion control and planning. Technical Report 2007-22, Technical University Berlin, 2007. ISSN 1436-9915.

[68] Marc Toussaint. Lecture notes: Markov decision processes. Lecture notes, 2009.

[69] Marc Toussaint, Stefan Harmeling, and Amos Storkey. Probabilistic inference for solving (po)mdps. Technical Report Research Report EDI-INF-RR-0934,, University of Edinburgh, School of Informatics., 2006.

[70] Gabor Vass. *Object Manipulation Planning for Dextrous Robot Systems*. PhD thesis, Budapest University of Technology and Economics, 2005.

[71] Filipe Veiga and Alexandre Bernardino. Towards bayesian grasp optimization with wrench space analysis. In *IROS 2012 Workshop: Beyond Grasping - Modern Approaches for Dynamic Manipulation*, 2012.

[72] D. Verma and R.P.N. Rao. Planning and acting in uncertain environments using probabilistic inference. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2382–2387, 2006.

[73] R. Volpe and P. Khosla. Manipulator control with superquadric artificial potential functions: theory and experiments. *IEEE Transactions on Systems, Man and Cybernetics.*, 20(6):1423–1436, 1990.

[74] Yejun Wie and Bill Goodwine. Stratified motion planning on nonsmooth domains with robotic applications. *IEEE Transactions on Robotics and Automation*, 20(1):128–132, 2004.

[75] Jijie Xu, Tak-Kuen John Koo, and Zexiang Li. Sampling-based finger gaits planning for multifingered robotic hand. *Autonomous Robots*, 28(4):385–402, 2010.

[76] Jijie Xu and Zexiang Li. A kinematic model of finger gaits by multifingered hand as hybrid automaton. *IEEE Transactions on Automation Science and Engineering.*, 5(3):467–479, 2008.

[77] Jijie Xu, Yunjiang Lou, and Zexiang Li. Hybrid automaton: A better model of finger gaits. In *IROS*, pages 4628–4633, 2006.

[78] Masahito Yashima. Manipulation planning for object re-orientation based on randomized techniques. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1245–1251, 2004.

[79] Masahito Yashima and H. Yamaguchi. Dynamic motion planning whole arm grasp systems based on switching contact modes. In *IEEE International Conference on Robotics and Automation, 2002. ICRA '02.*, volume 3, pages 2492–2499, 2002.

[80] Masahito Yashima and H. Yamaguchi. Complementarity formulation for multifingered hand manipulation with rolling and sliding contacts. In *IEEE International Conference on Robotics and Automation, 2003. ICRA '03.*, volume 2, pages 2255–2261 vol.2, 2003.

[81] Hong Zhang, Kazuo Tanie, and Hitoshi Maekawa. Dexterous manipulation planning by grasp transformation. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3055 – 3060, 1996.

[82] Jan-Wouter Zwart. Review article: The minimalist program. *J. Linguistics*, 34:213–226, 1998.