Ecole Doctorale Systèmes

Grasp planning for object manipulation by an autonomous robot

Planification de saisie pour la manipulation d'objets par un robot autonome

THÈSE

Soutenue le 4 juillet 2006

pour l'obtention du

Doctorat de l'Institut National Des Sciences Appliquées de Toulouse

Spécialité: Systèmes Automatiques

par

Efrain Lopez Damian

Jury

Président :	Raja Chatila
Rapporteurs :	Christian Laugier Véronique Perdereau
Examinateur :	Marc Renaud
Directeurs de thèse :	Rachid Alami Daniel Sidobre

Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

Acknowledgment

The work of this thesis has been conducted in the Robotics and Artificial Intelligence at LAAS-CNRS. I would like to thank Raja Chatila head of the group for accepting me in the team and for proposing the research subject.

Of course, this work would not have been possible without the help, the discussions and broad support offered by my advisors, Daniel et Rachid, *Merci !*

I would like to thank Véronique Perdereau, Christian Laugier and Marc Renaud to have accepted to be part of the jury and for all the contributions they have made to enrich this thesis.

This work could not be possible without the financial support from the National Council of Science and Technology of Mexico CONACyT.

Thank you to Sara, Matthieu and Jérôme to mantain *Jido* alive, making possible to perform the experiments in this thesis. Thanks to the Cogniron group, Gaëlle, Felipe and Michel.

Thanks to Joan, Vincent Montreuil, Martial, Olivier. Patrick *vaya tío, vaya pel...s* who makes sure that research advances at the mysterious corner, and the rest of RIA for this pleasant atmosphere of work.

Ehh, oui, gracias, merci au groupe piltrafa : Aurélie, Claudia, Sylvain, Jean-Michel, Akin, Luis, Nacho, mis estimados saltamontes Thierry et Gustavo. Thank you for all the discussions and experiences that we have passed, really without you I could have done a better thesis but immensely less fun.

Thanks to my other friends America, Maggie and Gibran to be so patient and share all those delirious conversations that we had and contributed to make these last years bearable and less dark.

Finally, to my parents, sisters, and my whole family for always be there and for their confidence and support in all my crazy projects.

A mis padres
mis hermanas
mi familia
mis amigos

Contents

List of Figures

List of Tables

Introduction

Robotics as a research field starts at the end of World War II. A teleoperated hand is developed to handle radioactive material, but it can be said that two events mark the beginning of modern robotics. In 1954 the first articulated arm is designed. Based on this scheme, several robotic arms have appeared, the well known PUMA arm and the SCARA arm just to mention some, these robots can perform repetitive tasks with an accurate precision, we can find them in industries generally. The second event is the mobile robot Shakey. This robot is equipped with external sensors to perceive its world, a vision system allows the robot to have a representation of the environment to navigate. A planner called STRIPS was developed to generate a set of actions to accomplish some tasks. The fast and promising results obtained motivated the researchers for a more ambitious goal, make autonomous robots.

Nowadays, robotics research aims to create and to develop an autonomous system capable of taking decisions and to interact with a dynamic environment. The robot has to take into account the constraints imposed by surroundings. Such autonomous robot must rely on a set of functions to carry out tasks successfully.

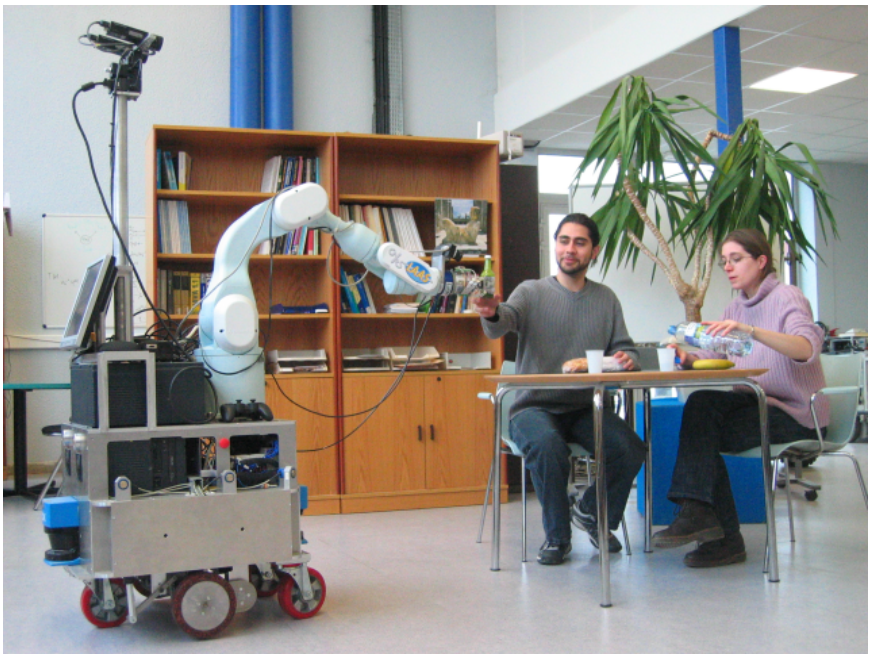


Figure 1 – Hand over task, the planner generates two grasps, one is executed by the robot and the second one tells us that at least the object is graspable, even if we cannot assure that the human will find the same grasp.

The motivation of this work is to develop robots with the capabilites of interaction for service tasks in humain environments. Among these capabilites, we consider object manipulation because it plays an important role allowing the robot to modify its environment or to perform tasks which require an interaction with others entities. Object manipulation deals with the transfer of objects in the environment by grasping them or by pushing them, or by using the grasped object as a tool for a task. In this thesis we propose to solve the object manipulation from a task and motion planning perspective. Being object manipulation planning a sophisticated instance of a more general problem that is motion planning, such instance deals with movable objects, we use recent planning techniques.

1 Interactive Object Manipulation Planning

Interaction can be understood in many ways. For instance, we find cooperative motion where an object is manipulated at the same time by two or more robots [Arechavaleta 04], or interaction in the sense of graphics community. The user takes the role of a high-level planner and only certain parts of the character animation are produced automatically using motion planning algorithms [Kuffner 00a] [Koga 94a] [Kallman 03]. In this thesis, we are interested in interactive object manipulation as motions to allow an object exchange. One of the most representative interaction task using manipulation in human environments is how to plan the motions for a robot to perform hand over of objects between robots and humans as shown in Fig. 1. Several constraints have to be taken into account : how to present the object to the human, the grasps planned must be compatible and collision free to perform the task.

2 Contributions and thesis organization

This thesis is in the context of object manipulation for autonomous robots. The manipulation task requires interaction between the entities involved in the task. As the robot moves in a dynamic human environment, frequently the robot will be constrained to perform tasks that imply the grasp of arbitrary objects. It is impossible to simplify the grasp planning problem by defining a set of grasps on the object or using model-based techniques.

The contribution of this thesis is a manipulation task planner to solve specific but very representative tasks in interactive manipulation delivery tasks. These can be classified into : hand over tasks and pick and place tasks.

Other contributions in the thesis are in the grasp planning problem that constitutes one key component of the task planner. We have implemented a random grasp planner based on the object inertial properties. The inertial axes are used as guide directions for grasp generation. In order to improve grasp planning efficiency in the case of more complex objects, we propose an object decomposition process. The algorithms for the grasp planning were implemented and tested in an experimental robotic mobile manipulator platform. For the tasks that require the interaction and cooperation between entities we plan double grasps on the object to perform the operation based on an object decomposition by planes passing through the object inertial axes. The document is structured in five chapters. The first chapter of this thesis presents a brief state of the art of motion planning algorithms and fundamental notions on robotics and grasping. In chapter 2, we introduce an architecture for interactive manipulation tasks that is used for automatic programming.

A random grasp planner and the grasp generation phase for two different kind of end-effectors : three-contact gripper and an articulated three-fingered hand (Barrett) are presented in chapter 3. The case when objects to be manipulated have a non-convex geometry is presented in chapter 4. Planning results using Move3D planning tool and experimental results using a mobile manipulator are the subject of the final chapter 5.

Chapter 1

Planning for Manipulation : Theory and Algorithms

Manipulation in human world requires that robots can move and interact with the environment by modifying it. This modification can be reflected in the object displacements or using a grasped object as a tool to achieve a task. For example, grasping a pencil to draw something in a notebook. The objects can be made of different materials, they can have several shapes, and they can be soft or rigid. Be that the robot grasps or pushes the object to displace it, friction affects the grasp or motion of the object respectively. Considering that for some tasks the robot will interact with humans, it is desirable that the robot considers some object physical characteristics like texture, and even some associated functions to the model to be used as indicators to know if there are forbidden surfaces where the robot must not place its fingers. This is useful if the robot must grasp and give a knife to a human, or take a hot metallic piece, just to mention some situations. All these topics should be taken into account for a grasp planner. The robot motions needed to perform the task could be planned by using motion planning techniques.

The work presented in this document propose a grasp planner for arbitrary object manipulation tasks. Here we took into consideration rigid objects and we are interested in computing friction grasps on these objects to manipulate them. The reader interested in the mechanical foundations for manipulating by pushing is referred to [Salisbury 85]. We briefly recall some theoretical fundamental concepts and algorithms used in robotics. The theory and algorithmic state of the art is not intended to be exhaustive, we do not describe or mention all the existing algorithms in literature but only those that we found interesting or that have a direct link with our work, this must be seen as a guide for the reader to the different tools in motion planning that can be used as an option to try to solve complex manipulation problems. We start the chapter with the description of the bases of motion planning, making emphasis in recent techniques like probabilistic roadmaps or rapidly-exploring trees to then pass to the geometric formulation of manipulation planning, how the problem is treated and some extensions of the problem for multi-robots. The third part deals with some concepts of grasping, the common contact models used for a grasp and algorithms to find frictionless and friction force-closure grasps with different number of fingers. We continue with methods that use heuristics to search grasps on the object. Finally, in the last sections we mention methods that use human demonstration to learn how to grasp objects and some integrated systems for object manipulation.

1.1 Motion Planning

The motion planning problem has been an active and essential research area over the last years in robotics. The fundamental concepts appeared in the eighties [Latombe 91], one main notion is that of

configuration space CS. Today almost all the motion planners are based on this concept. In the most simple version, the problem is to decide if rigid bodies or a series of rigid bodies (e.g. articulated robot) is capable to move from one initial configuration to a final configuration in an environment with obstacles. There are several motion planners that solve the basic motion planning problem, but nobody have solved the problem in its full generality. In the first part of this section we present the notion of configuration space and motion planning methods.

1.1.1 Configuration Space

The configuration space approach was introduced by Lozano-Perez [Lozano-Perez 83a], transforming the search of a collision free path for a robot moving between obstacles into a search of a path for a point moving in the free zone of the configuration space CS_{free}. To illustrate this concept, we take a planar articulated mechanical system with two degrees of freedom as the robot of Fig. 1.1. The first link is attached to the ground by one of its extremities but it can rotate around this fix point, and the second link is attached to the first with a relative rotation. The position of the system is given by the two joint angles depending on the robot orientation. Let the angle range of motion go from 0 to 360 degrees, the link motion describes a circle. A topological representation of the system is a torus, a point on the torus surface represents a robot configuration given by the joint angle values, the torus surface represents then the configuration space of the system. The robot motions correspond to paths on the torus. The same can be made for more complex mechanisms with n degrees of freedom, incrementing the dimensionality of the CS and the complexity to find a solution. When we take into account the obs-

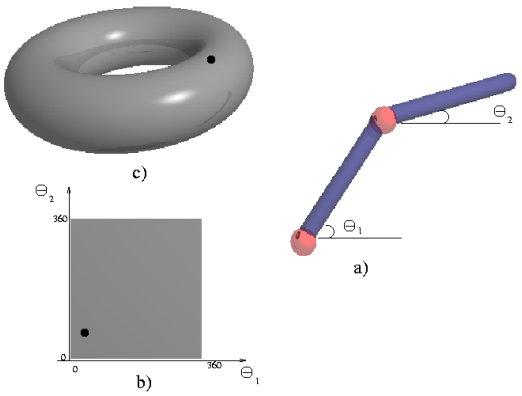


Figure 1.1 – a) Planar two-linked robot, b)Parameterization of CS and c) Configuration space topology

tacles O_i, the CS_{obs} represents the set of robot configurations q_i intersecting obstacles in the workspace, defining regions $CO_i = \{q \in CS \mid R(q) \cap O_i \neq \emptyset\}$, where $R(q)$ represents the volume occupied by the robot at configuration q. The obstacle configuration space is then $CS_{obs} = \bigcup_{i=1}^{n_{obs}} CO_i$, subsequently the free configuration space is $CS_{free} = CS \backslash CS_{obs}$. A free path is then a continuous mapping $\tau = [0, 1] \rightarrow CS_{free}$ connecting two robot configurations.

We can classify the motion planning approaches in three categories : cell decomposition, potential field, and roadmaps. The first method tries to decompose CS_{free} in a certain number of regions, called cells. A connectivity graph is constructed to represent the adjacency of cells, the output is the sequence of cells, a continuous free path can be computed from this sequence. When the dimension of the confi-

guration space is high, this kind of technique is not longer applicable because the high computing time necessary to find a solution [Latombe 91].

The second approach is the potential field, since the robot in CS is a point, a physic analogy is to consider the robot as a particle moving in an artificial force field produced by the goal and the obstacles. The particle is attracted by the positive potential of the goal and repulsed by the negative potential of the objects [Khatib 85]. The gradient of the total potential is taken as the force exerted on the robot. The direction of this force, at every configuration allows to compute the motion direction. The methods based on this approach could fall on local minima, which could make it difficult for their use on global solutions.

The final kind of approach is the roadmap, a graph that captures the connectivity of the free configuration space, the nodes are free configurations and the edges give a path that connects the initial and final configurations. A deterministic technique to solve the problem for the roadmap approach is very difficult and the complexity grows exponentially with respect to the number of the degrees of freedom [Canny 88].

1.1.2 Sampling-Based Algorithms

While the above methods need an explicit representation of obstacles in the CS, the recent methods capture the connectivity of CS_{free} by a graph created from samples of robot configurations in the CS. A good review of these techniques can be found in [Choset 05]. These methods try to connect the samples with local paths (take into account the kinematics constraints of the system) to obtain solutions for motion planning problems.

The main characteristic of these algorithms is the probabilistic completeness (if a solution exists, the algorithm finds it if enough time is given). Due to the nonnecessity of explicit obstacle representation in the CS and that the collision detection is made in the workspace, these techniques are less sensitive to the dimensionality of the configuration space.

These practical algorithms have demonstrated their efficiency to solve high dimensional motion problems. We can classify the probabilistic methods into two groups : the multi-query planners and the single-query planners. Multi-query methods require a preprocessing phase to construct the roadmap, but once the roadmap is done, we can perform a set of queries for the same system. In the other case, the roadmap is generated each time the query is made. The time needed to find a solution is lower, but as it is focused in solving a particular problem, the information cannot be used for another query.

Probabilistic Roadmap Method (PRM)

The simplest sampling-based algorithm is the probabilistic roadmap. A random sampling of the configuration space is done to construct the graph. The edges are paths in CS_{free} linking the nodes, which imposes an efficient collision detection technique CD. Because of the randomized sampling process, the algorithm is weak probabilistic complete. We can find a good description of the PRM method in [Kavraki 96].

The algorithm works in three phases, the *sampling phase* consists in capturing the connectivity of CS_{free} and building the roadmap. Algorithm 1 presents the construction process of the roadmap, where N is the set of nodes and E is the set of edges. A node is the robot collision free configuration q randomly generated, and the edges are created between q and the elements of a set of neighbors N_q if both configurations q and elements q_n belong to the same connected component (subset of nodes where all are linked).

The roadmap allows to advance to the next phase, the *query phase* which tries to find a path in the roadmap by a concatenation of edges to link the initial to the final configuration and give an answer for

the query. Given the random characteristic of the approach, the resulting path can be long and irregular, then a third and final phase is used ; the *smoothness phase*.

There are critical cases where the performance of the algorithm is limited by the arrangements of obstacles, forming narrow passages. With a uniform random sampling, nodes are distributed equally spaced in the configuration space, that makes it difficult to find a path in such circumstances. Some solutions have been proposed for this problem, they allow nodes in the obstacle space and then pull them to the free space [Hsu 98], or they exploit the same idea but in reverse direction, a generation of samples in CS_{free} and push them to the obstacles, called obstacle-based PRM [Amato 98].

Algorithm 1: Construct PRM

```
input  : the robot A, the environment B
output : the roadmap R = (N, E)
begin
    R ← Empty (N ← ∅ and E ← ∅);
    while not StopCondition (nb_nodes < Max_nodes) do
        q ← RANDOM_COLLISIONFREE_CONFIGURATION in CS_free;
        N_q ← SET_CANDIDATE_NEIGHBORS;
        ADDNEWNODE N ← N ∪ (q);
        forall q_n ∈ N_q do
            if SAME_CONNECTED_COMPONENT(q, q_n)
            and FEASIBLE_PATH_CONNECTION(q, q_n) then
                ADDNEWEDGE E ← E ∪ (q, q_n);
                UPDATE_ROADMAP_CONNECTED_COMPONENTS;
            end
        end
    end
end
```

Visibility-PRM

The algorithm uses the CS_{free} structure to construct visibility domains [Nissoux 99] in order to produce small roadmaps. While each collision free configuration is added to the roadmap in the basic PRM, the visibility PRM only keeps configurations which connect two adjacent components of the roadmap. The visibility region of a configuration is defined as :

$$V_{is}(q) = (q' \in CS_{free} | L(q, q') \in CS_{free}) \tag{1.1}$$

where L is any local method that finds a path to connect two configurations q and q', where q is called a *guard* of $V_{is}(q)$. The algorithm constructs a roadmap with two set of nodes : the guards which are nodes that cover a region, and that belong to a same connected component. A guard node cannot be directly connected to another guard node by a local path, the nodes are not visible between them. The *connectors* are nodes generated in the intersection of two visibility regions. The pseudo code is presented in algorithm 2, where iteratively the guard and connector nodes are generated with their respective edges to link them. The guard nodes can show how the CS_{free} is covered with the union of their visibility regions.

Algorithm 2: Visibility PRM

```
input  : the robot A, the environment B
output : the roadmap R = (N, E)
begin
    ntry ← 0 ; Guard ← ∅ ; Connection ← ∅;
    while ntry < M do
        q ← RANDOM_COLLISIONFREE_CONFIGURATION;
        g_vis ← ∅, G_vis ← ∅;
        forall components G_i do
            Found← FALSE;
            repeat
                forall nodes g of G_i do
                    if q belongs to V_is(g) then
                        Found← TRUE;
                        if not g_vis then
                            g_vis ← g; G_vis ← G_i;
                        else
                            ADD_CONFIGURATION_TO_CONNECTION;
                            CREATE_EDGES(q, g) and (q, g_vis);
                            MERGE_COMPONENTS G_vis and G_i;
                        end
                    end
                end
            until Found = TRUE;
            if not g_vis then
                ADD_CONFIGURATION_TO_GUARD;
                ntry ← 0;
            else
                ntry← ntry+1;
            end
        end
    end
end
```

The algorithm iteratively computes the two sets of the roadmap nodes. The nodes $guard$ are elements of subsets G_i, each subset is a connected component of the configuration space. At each iteration, the algorithm randomly produces a collision free configuration q. All components G_i are visited, in consequence all the $guard$ nodes of each G_i are visited to know if they are visible from q. If a $guard$ node is visible, then the node and the component G_i are memorized in g_{vis} and G_{vis}, a second $guard$ node can be visible at the same subset G_i or in another subset G_{i+1}. If such second node is found, then the configuration q is a connector and it is added to the connection set. A merge operation is done with components G_i and G_{vis}. If q is not visible from any component, then it is added to the guard set. If q is only visible from one $guard$ in all components, then q is rejected. The algorithm has the parameter $ntry$ that gives the number of iterations before a node is found. The algorithm stops when the number of iterations is greater than a value M set by the user. The volume of CS_{free} covered by visibility regions becomes probably greater than $(1 - \frac{1}{M})$.

Rapidly-Exploring Random Trees (RRT)

It is an incremental search algorithm. The main idea behind these expanding methods is to create a tree to capture the connectivity of CS_{free} [LaValle 98]. An outline of the RRT-planner can be found in algorithm 3.

The initial configuration q_{ini} is the first node in the tree G. The space is sampled by creating random configurations q_i and connecting them always searching to link them to the nearest node q_n in the tree. The obstacles are not represented explicitly, collision checking is needed for every new configuration. If q_i is in collision, an intermediate free collision configuration q_s is computed in direction of q_i. At some interval of time, the algorithm verifies if the final configuration q_F can be connected to the tree, a path τ is found making a search in the tree from the start node to the node that represents the final configuration.

Algorithm 3: Rapidly-Exploring Random Tree Planner

```
input  : Initial configuration q_ini, final configuration q_F
output : the tree G = (N, E), path τ
begin
    ADDNEWNODE G(N) ← q_ini;
    forall i = 1 to k do
        q_i ←RANDOM_CONFIGURATION;
        q_n ←NEAREST (G(N), q_i);
        q_s ←INTERCONFIGURATION (q_n, q_i);
        if q_s ≠ q_n then
            ADDNEWNODE G(N) ← q_s;
            ADDNEWEDGE G(E) ← (q_n, q_s);
        end
        if q_F CONNECTION q_n then
            RETURN_SOLUTION τ(q_ini, q_F);
        end
    end
end
```

An improvement of the RRT-planner can be made if instead of forming a single tree, a bidirectional search is done by creating two trees, each one taking q_{ini} and q_F as the starting node and progressively approaching each other using some heuristics until connection occurs [LaValle 05] [Kuffner 00b]. An advantage of these algorithms is the nonnecessity of a previous graph construction, however only a single query is possible.

Essential Parts for Motion Planners

As we can see, there are two essential elements for a motion planner : collision detection CD and local methods. In PRM and RRT algorithms every time that a new configuration is generated, a collision detector is executed because these sample-based methods do not need an explicit representation of CS_{free}. Efficient collision detection algorithms are needed to make easier the implementation of sampling-based planners. Basically, the collision detection verifies if a given configuration is colliding with an obstacle or not, the robot and obstacles are described by geometric primitives or polyhedra that occupy a volume in the space, the overlapping between these volumes is a way to detect the collision. Most of the CD, once the geometric models are given, they precompute the convex hull and the hierarchical representation of each model in terms of oriented bounding boxes (OBB). For each pair of objects

whose bounding boxes overlap, the collision detector checks if their convex hulls are intersecting based on the closest feature points. Some examples of collision checkers are SOLID [van den Bergen 99], I-collide [Lin 97], KCD [van Geem 01] and others.

Steering methods are algorithms to compute feasible paths that satisfy kinematic constraints depending on the mechanism between two nodes (configurations) in the roadmap. These methods are commonly called local methods because they are the basic tools to generate local paths. Most of the time, the simplest local method can be a straight line between two configurations, which usually corresponds to a feasible motion for a holonomic robot manipulator or a freeflyer robot. However, some systems have specific constraints in their motions. A classical example is the car-like robot which has non-holomic constraints that do not allow the robot to move at its side directions and then not every path is feasible.

1.2 Manipulation Planning

Manipulation planning is a more complex case of motion planning that considers movable objects. A robot can change the object location by applying an action on that object. There are two common actions that a robot can execute to move an object. The robot can push the object or grasp it. In this thesis we are interested in the second one.

The existence of movable objects in the environment modifies the motion planning problem significantly. For example, given a configuration of static objects called obstacles, it is probable that the planner cannot find a feasible path for a robot between two configurations, in contrast, if some objects are movables, the robot can change its workspace by transferring these objects to other locations to find a path and to attain the goal configuration.

A manipulation plan appears as a sequence of transit and transfer motions. In a transit path the robot moves alone, while in a transfer path the robot moves the object. These two paths belong to different sub-spaces and the problem is to find when to change from one sub-space into another.

The Geometric Problem Formulation

In [Alami 89] [Alami 94] a general geometric formulation to the problem of manipulation planning is presented. The concept of graph manipulation is introduced. The constraints of stable placements (object positions in equilibrium) and stable grasps (the robot immobilize the object) are introduced to solve the problem. The approach consists in transforming the problem of manipulation planning into a problem of finding a path in a certain space. The main idea is to consider a manipulation path as a free-collision path in a combined configuration space defined as the cartesian product $CCS = CR \times CO_1 \times \ldots \times CO_m$, where CR is the robot configuration space and CO_i are the objects configuration spaces.

Two CCS subspaces are constructed from the placement and grasp constraints. The *Grasp* subspace is the space where the robot can move the object without risk of collision with obstacles or other objects. The paths in this subspace are denominated *Transfer* paths. The *Placement* subspace is the space where the robot alone can perform free-collision motion. *Transit* paths are found in this subspace.

The manipulation planning problem is defined as a path search in the different connected components of a subspace defined by the intersection of the *Grasp* and *Placement* subspaces, *Grasp* $\cap$ *Placement* and the other components.

Two main cases can be found in the manipulation planning problem : discrete and continuous. The first case assumes that we have a fixed number of grasps on the object that the robot can perform, and a fixed number of positions in which the object can be placed. *Grasp* $\cap$ *Placement* is composed by a set of discrete configurations, the goal is to search these configurations and to link them by tranfer paths in the *Grasp* subspace and transit paths in the *Placement* subspace. For the case where the planner takes

into account continuous grasps and placements, the topology of *Grasp* ∩ *Placement* has to be found. A solution is proposed in [Siméon 04], where probabilistic roadmaps capture this topology using closed-chain systems [Cortés 02]. *Grasp* ∩ *Placement* is considered as the configuration space of a system composed by the robot and the object placed in a stable position, which forms a closed-chain with the base that supports the object. The manipulation planning problem can generally be solved by constructing a manipulation graph M_g.

Manipulation Graph

The nodes of this graph are the connected components of *Grasp* ∩ *Placement*. The edges can be of two types : transit or transfer edges, then an edge between two nodes tells us if there is a transit or tranfer path linking two configurations of the connected components. If the initial and final configurations of robot and object are in the same M_g connected component, then there is a solution. The manipulation planner output is a sequence of transit and transfer motions. This is possible in the continuous placements and grasps case by applying the reduction property [Alami 94], that says that two configurations that are in the same connected component of *Grasp* ∩ *Placement* can be connected by a manipulation path, and any path can be decomposed into a finite sequence of transit and transfer paths.

One important point in manipulation is how to grasp the object. Most of the manipulation planners simplify the problem by defining *a priori* a finite set of possible grasps on the object or by imposing some simple constraints. The stable placements are described by constraining the object to be placed on horizontal surfaces allowing two translations and one rotation around the vertical axis of the surface. The same constraint is made for the grasps, the end-effector could have contact with two given faces of the object, again two translations are allowed along these faces and one rotation around the face normal. This is an easy way to simplify the problem of grasp planning, given relatively simple objects.

Multi-robots Manipulation

The framework presented in [Alami 89] was extended for a multi-arm cooperation for manipulation tasks [Koga 94b]. An incremental approach is proposed to construct the manipulation graph, where a finite set of grasps G_{set} are defined to be used for the manipulator arms, each grasp in this set is a rigid attachment of the last link of an arm with the object. For each arm, a non-invasive position is defined in such a way that if an arm does not take part in the current motion, the arm must be placed at this non-invasive position. An arm manipulator M_a can regrasp the object only if one of the other arms hold the object while M_a uses another grasp from the available set. Obviously, the selected grasp has to be different from that used to hold the object at this moment.

The approach starts by planning a free-collision path for the object alone using a probabilistic path planner [Barraquand 91]. The object path is defined by a set of adjacent configurations taken from a list. The probabilistic method used was modified in such a way that each inserted configuration in the path can be reached by at least one grasp defined in G_{set}. Once the object path is computed, the arms follow the grasp point trace to keep the same grasp along the path. If a trace cannot be followed, a transit path is computed to allow a re-grasping of the object and continue the object path. Using the same idea of manipulation graph and the inverse kinematic human arm model, in [Koga 94a] the authors solved simple manipulation problems for virtual actors, introducing virtual vision, the actor verifies if the object target is visible and reachable before planning a grasp motion.

A new approach for the multi-robot manipulation problem is proposed in [Gravot 04] [Cambon 05]. The planner is composed by geometric and symbolic parts, using geometric methods to capture the topology of the space and using task planning techniques at the same time. The approach is based on the idea that for solving complex manipulation problems, a pure geometric strategy is not enough and

a task-motion planning hierarchy is not a good solution either because a plan found by the task planner not necessarily can be executed by the motion planner. A symbolic planner called *aSyMov* is developed, where information delivered by the motion planner is used to replan the actions.

This is applicable when the geometry of the object is known, otherwise a different strategy must be found. As we are interested in arbitrary shape unknown objects, we cannot simplify the problem of planning grasps as we have described previously. We need to consider the mechanics of grasping in the planning process, as well as minimum conditions that a grasp should satisfy.

1.3 Grasping

The most common way for a robot to move an object is by grasping it, exerting forces on it. We can see the grasping as the process to constraint the object motion relative to the end-effector by setting contacts. In several manipulation tasks, grasping is a fundamental issue. Some fundamentals notions are presented first.

Grasp statics : only the transmission of forces between the contacts and the object is considered, the location of these contacts are fixed and ignore the possibility that fingers can roll or slide on the object surface.
Contact model : for grasping, there are different contact models between the fingers and the object. A first contact type is the frictionless point of contact. A force can be applied to the object in the normal surface direction. For a hard finger model or point with friction, the contact can resist tangential forces too. A soft finger can apply, besides the force a moment around the normal axis. These three common contact models are shown in Fig. 1.2. Other contact types can be the line contact and the planar contact, for a more complete list see [Salisbury 85].

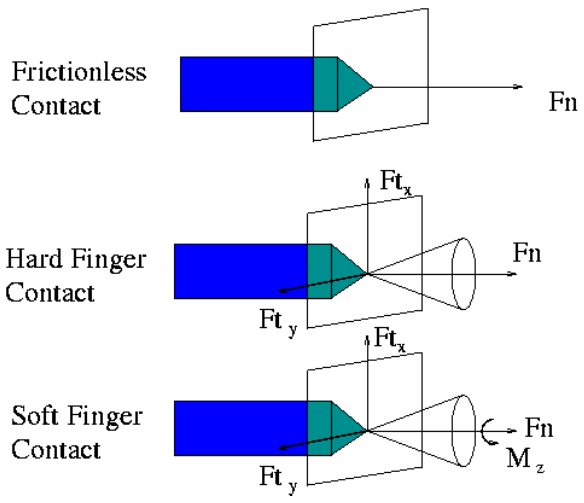


Figure 1.2 – Most common contact models and their associated forces and moments that can be applied

1.3.1 Rigid Body Statics

Considering the forces and moments acting on a rigid body, we can characterize such forces and moments. The motion of such body is determined by the vector sum of forces. A generalized force

applied on a rigid body consists of a linear component, or pure force and an angular component or pure moment acting at a point m. We can represent this generalized force w_m as a vector in R^6, this pair is called a *wrench*.

$$w_m = \begin{bmatrix} f \\ d_m \times f \end{bmatrix} \tag{1.2}$$

The values of the wrench vector depend on the coordinate frame in which the force and the moment are represented. If O is a coordinate frame attached to the rigid body, an object for example, the wrench applied at the origin of O is $w_o = [f\ \tau_o]^T$ and $\tau_o = d_m + o\vec{m} \times f$, where d_m is the distance from the origin to the point m.

1.3.2 Friction Model

As we have seen, some contact models use friction. The most common friction model is the Coulomb model, which is not the best model but it is used for the compromise between simplicity and experimental results. This model tells us how much force a contact can apply in the tangent directions on a surface as function of the applied normal force. The allowed tangential force f_t is proportional to the applied normal force f_n and the constant of proportionality μ is a function of the materials in contact.

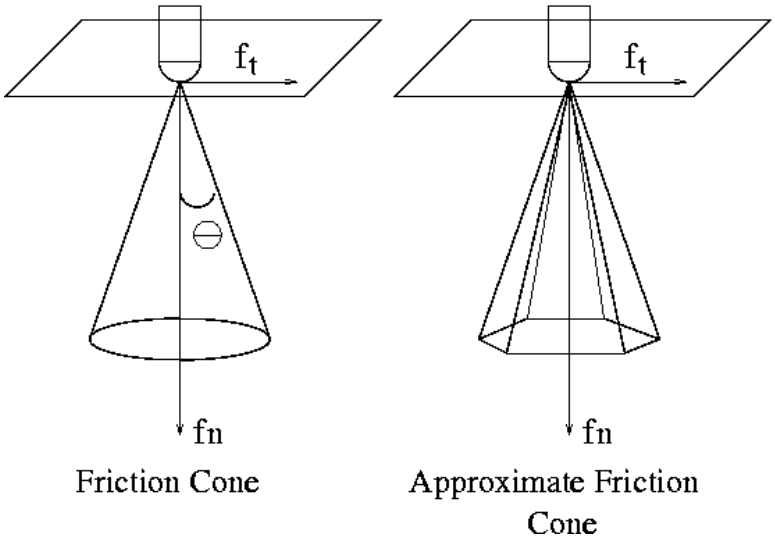


Figure 1.3 – Coulomb friction cone and polyhedral convex cone

The set of forces which can be applied at the contact must lie in a cone centered about the surface normal. This friction cone as we can see in Fig. 1.3 has an angle θ with respect to normal $\theta = \tan^{-1}\mu$.

$$|f_t| \leq \mu |f_n| \tag{1.3}$$

We can see that the friction cone imposes a nonlinear constraint on the force. To simplify the problem, we can approximate the circular cone by a cone generated by a finite number of vectors, constructing a polyhedral convex cone, Fig. 1.3.

Polyhedral Convex Cones

This type of cone occurs when describing the sets of wrenches that characterize rigid body contact. Let v be any vector different from zero in R^n, then the set of vectors $\{kv \mid k \geq 0\}$ describes a ray in

R^n, similar for two vectors non parallel v_1 and v_2, the set of vectors $\{k_1v_1 + k_2v_2 \mid k_1, k_2 \geq 0\}$ makes a planar cone. We can generalize to an arbitrary number of vectors by defining the positive linear span of a set of vectors, $pos(v_i) = \sum k_i v_i | k_i \geq 0$. Any set of vectors constructed in this form is a polyhedral convex cone. The two ways to represent a polyhedral convex cone are by its edges and by its faces. To represent a cone by its edges, we have the positive linear span operator, then given a set of edges e_i, the cone is given by $pos(e_i)$. To represent the cone by its faces, we take the inward pointing normal vector of the planar half space, the cone is given by the intersection of all half spaces.

1.4 Grasp Planning

Grasp planning in its simplest form deals with the problem of finding where to place the fingers on the object surface. Grasp planning is computationally intensive due to the large search space and to the incorporation of geometric and task-oriented constraints. Ideally, a grasp must have five properties : force-closure (fingers capable to resist arbitrary forces/moments), dexterity (how should fingers be configured), equilibrium (how hard the finger squeezes), stability (how to remain unaffected by external disturbances) and dynamic behavior (how soft a grasp should be for a given task) [Shimoga 96].

One desirable characteristic in grasp planning for autonomous robots is the capacity to manipulate unknown objects. For humans the grasps can be divided into two main configurations, power grasps when the whole hand (palm and finger bodies) can be in contact with the object, and precision grasps where only the fingertips make contact.

1.4.1 Number of Fingers

It is desirable that a grasp can be constructed with the smallest number of fingers, which affects the planning of grasps. It is more feasible when it is intended to use the algorithms with a robotic manipulator. In robotics literature we find that Lakshminarayana [Lakshminarayana 78] established minimal bound for the number of fingers needed to achieve force-closure of rigid bodies. Mishra, Schwartz and Sharir [Mishra 87] derived a theoretical number of six fingers required to hold a two-dimensional (2D) object and twelve fingers for three-dimensional (3D) objects in equilibrium for frictionless grasps, mainly used in fixture machining. It is generally accepted that three-hard fingers are enough to have a force-closure grasp for 2D objects and four fingers for 3D objects [Markenscoff 90]. Mirtich and Canny [Mirtich 94] show that if we model the fingers with rounded fingertips and static friction at the contact points, the number of fingers are reduced by one. Thus, any 2D object can be grasped with two fingers and any 3D object with three fingers.

1.4.2 Force-Closure Condition

A desirable notion in grasp planning is that of *force-closure*. In theory, a force-closure grasp has the ability to resist any external force and moment (w_{ext}) by applying appropriate wrenches at n contact points.

$$w_{ext} + \sum_{i=1}^{n} w_i = 0 \tag{1.4}$$

Grasps can be constructed as nonnegative linear combination of wrenches, no sticky fingers apply forces in the negative direction of the normal surface. If we have n wrenches at contact points acting on an object, the total wrench is given by

$$w_{net} = \sum_{i=1}^{n} \alpha_i w_i \mid \alpha_i \geq 0 \; for \; i = 1, ..., n \tag{1.5}$$

A system of n wrenches achieves force-closure when the corresponding total wrench positively spans R^6, this notion borrowed from mechanics was first introduced in robotics by Salisbury [Salisbury 85].

It is important to note that in the literature different notions of force-closure and form-closure (in general the forces applied to some contact points without friction required to immobilized a body) are used. Bicchi [Bicchi 95] considers that in force-closure we have to take into consideration the ability of the grasping mechanism to control some of the contact forces.

Given that a coulomb friction model is taken, the forces f_i must remain in the friction cones to avoid slippage. To simplify the problem we approximate the friction cones by polyhedral convex cones. The grasping forces are represented as

$$f_i = \sum_{j=1}^{n} \alpha_{ij} f c_{ij},\ \alpha_{ij} \geq 0 \tag{1.6}$$

where fc_{ij} represents the j^{th} edge vector of the polyhedral convex cone, the coefficients α_{ij} are non negative constants. As the wrench is given by $w_i = [f_i\ \tau_i]^T = [f_i\ d_i \times f_i]$, substituting equation 1.6 in the wrench we obtain

$$w_i = \sum_{i=1}^{n} \sum_{j=1}^{k} \alpha_{ij} \begin{bmatrix} fc_{ij} \\ d_i \times fc_{ij} \end{bmatrix} \tag{1.7}$$

The net wrench applied to the object throughout the contact points is

$$w_{net} = \sum_{i=1}^{n} \sum_{j=1}^{k} \alpha_{ij} w_{ij} \tag{1.8}$$

1.4.3 Test for Force-Closure Grasps

We came to see that an important condition in grasp planning is that grasps satisfy force-closure. In some cases it is desirable to have a value to show how well the grasp satisfies this condition. The tests used for this goal are classified in two : qualitative and quantitative tests. The qualitative test has a boolean output that tells us if a given grasp is force-closure or not. Some grasp planners can generate several grasps and in this case the quantitative test outputs a value, useful to rank grasps and pick one from the set. This value can be considered a measure of the quality of the grasp.

Qualitative Test

In [Salisbury 85], force-closure condition implies that the positive linear combination of wrenches spans the whole wrench space. In [Mishra 87], it is shown that an equivalent manner to achieve force-closure is that the origin of the wrench space lies inside the interior of the convex hull of the primitive contact wrenches.

More recently, Liu [Liu 99] proposed a linear programming formulation based on the duality between convex hulls and convex polytopes. He demonstrated that a qualitative test can be transformed to a ray shooting problem. Assuming a hard finger model and coulomb friction at contact, the friction cones are approximated by polyhedral convex cones and a convex hull of wrenches is constructed. A point P in the interior of the convex hull is found by choosing a positive convex combination. It is necessary to find the intersection point with the convex hull and the ray from the interior point in direction to the wrench space origin. This is done by transforming the wrench convex hull into a convex polytope, this transformation is dual and has three main properties : 1) A vertex of the convex hull is transformed to a facet of the convex polytope and vice-versa. 2) A facet of the convex hull is transformed to a vertex of the

convex polytope and vice-versa. 3) A point in the interior of the convex hull is transformed to redundant hyperplanes of the convex polytopes and vice-versa.

The problem to detect the convex hull facet intersected by the ray can be reformulated as a problem of maximizing an objective function subject to some constraints. The solution gives the facet of the convex hull intersected by the ray and consequently the intersection point.

With the intersection point, we compute the distance between point P and the origin d_1, and the distance between point P and the intersection point d_2. If $d_1 < d_2$ the wrench space origin lies in the interior of the convex hull and the grasp is force-closure, otherwise the grasp is not force-closure.

Quantitative Test

There are several ways to grasp an object and it is desirable to have a quantitative test for such force-closure grasps to rank them. This offers the possibility to choose one grasp among several for a task. Kirkpatrick and Mishra were the first to propose grasp metrics [Kirkpatrick 91] [Mishra 95] based on convexity theory in wrench space and the Steinitz theorem, which states that given a subset S of Euclidean d-space E^d, any point in the interior of its convex hull is also in the interior of the convex hull of some subset of S with at most 2d points.

A quantitative version of the Steinitz theorem can be applied to the grasping problem in the next manner : for any set of points $S \subseteq E^d$ and its convex hull, there is a constant given by the radius of the largest ball centered at the origin contained in the convex hull. The radius can be considered as a measure for the grasp.

The ratio of largest external force-torque that can be resisted by applying at most unit forces at each contact point expresses the efficiency of the grasp. A ratio of one means the most efficient grasp.

Based on the grasp wrench metrics Ferrari and Canny [Ferrari 91] take into consideration friction and propose two quantitative tests to force-closure grasps.

Two different criteria to evaluate a grasp quality are proposed. The first one, finds grasp configurations that maximize the wrench given independent force limits that minimize the worst case force applied at any contact point. We have already seen the representation of finger forces when we take friction at the contacts and the wrenches. The set of all possible wrenches that can be applied at the contacts can be given by $W_i = \sum_{j=1}^{m} \alpha_{ij} w_{ij}$, and the set of all possible wrenches acting on the object is given by $W_{L_\infty} = W_1 \oplus \ldots \oplus W_n$. The last formula tells us that the total wrench belongs to the set that is the Minkowski sum of the convex sets, exchanging this sum with the convex hull we have

$$W_{L_\infty} = CHULL(\bigoplus_{i=1}^{n} w_{i1}, \ldots, w_{im}) \tag{1.9}$$

where n is the number of contacts and m the number of polyhedral convex cone facets. As the Minkowski sum over a finite number of sets with a finite number of elements is a finite set, then it is enough to compute the convex hull over these sets of elements. The quality measure Q_∞ is the distance of the nearest facet of the convex hull from the origin.

The second criterion minimizes the sum for all the applied forces. If we take the example given in [Ferrari 91] that the magnitude of the force is proportional to the total current in motors, using this criterion will result in the minimization of the power needed to actuate the end-effector. We have the total force $f = \sum_{i=1}^{n} f_i$, and we have seen how to represent the contact forces when considering coulomb friction, then the total wrench on the object is $w_{net} = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_{ij} w_{ij}$, with $\alpha_{ij} \geq 0$ and $\sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_{ij} \leq 1$. The set of all the possible wrenches is

$$W_{L_1} = CHULL(\cup_{i=1}^{n} w_{i1}, \ldots, w_{im}) \tag{1.10}$$

The total wrench on the object belongs to a set of wrenches. Computing the convex hull over the finite set of points in the wrench space, again the quality measure Q_1 is the distance of the nearest facet of this convex hull from the origin.

In [Trinkle 92] a new quantitative test is presented for any number of frictionless contact points that is formulated as a linear program, the value measured implies how far the grasp is to loose the force-closure.

In [Zhu 03] the pseudo metric proposed uses a gauge function of a convex polyhedral set in the wrench space. The measure reflects the maximum magnitude of the external wrench that the hand is capable to apply on the object in the worst direction.

There are different tests that consider the task to perform (the wrenches required for a particular task) in order to obtain a measure that indicates what is the most appropriate grasp. [Li 87] proposes to model the tasks by ellipsoids that need the development of a procedure for modeling such ellipsoids in the wrench space, this procedure is difficult because we need experience to specify the task correctly. A general ellipsoid is $E = \{y \in R^6, < y, Qy > + < a, y > \leq \beta^2\}$, where Q is positive definite symmetric matrix and $a \in R^6$ reflects the asymmetric properties of the task. The task ellipsoid reflects the wrenches that the grasp must apply to the object to actually perform the task. The measure is given by the radius of the largest task ellipsoid that contains the wrench space origin.

In reference [Borst 04], Borst proposes that instead of using only the grasp wrench space, meaning, the set of all wrenches that can be applied to the object through the grasp contact points, an alternative approach is to use the combination of the task wrench space TWS and the incorporation of the object geometry that defines the object wrench space OWS. The TWS indicates the wrenches expected to occur for a given task, when the task is unknown, we can assume that the grasp must hold the object against gravity and accelerating forces, a commonly used approach is to model the unknown task with a unit sphere in the wrench space as we do with the previous measure described here. The OWS was introduced in [Pollard 94] and should contain any wrench that can be created by a distribution of n disturbance forces acting anywhere on the object surface. The number of forces is unlimited, then the object wrench space can be composed by the union of polyhedral wrench cones. Then the wrench space gives us the capability of the grasp and the object wrench space gives the possible disturbances that may occur. The largest factor by which the OWS can be scaled to fit in the grasp wrench space gives a measure of the grasp quality. The OWS is not used directly but approximated by the smallest ellipsoid of quadratic form $x^T Q x \leq 1$ as in [Li 87] to enclose this object wrench space. The factor then is the ellipsoid that reflects the grasp quality.

1.4.4 Contact-Level Force-Closure Grasps Algorithms

Force-closure is a necessary characteristic for any grasp that leads the researchers to develop several algorithms to find grasps satisfying this characteristic. Two general types of algorithms can be found in literature, the first type is focused to find n contact points on the object surface. The second type of algorithms search to find regions where to place the fingers.

Computing Single Contact Grasps

The methods and algorithms that fall into this category compute only a set of contact points on the object surface. Some of the first grasp synthesis algorithms [Mishra 87] using the qualitative force-closure test were proposed during the eighties. In the same investigation [Mishra 87], contact points are placed at the centroids of triangular facets describing the object surface. The algorithm works for frictionless contacts. In [Mirtich 94], using a quantitative measure they proposed to find a three-contact concurrent grasps for 3D Objects, the contact points are spaced 120° from each other and they are on the faces of

equilateral triangular circumscribing prisms. For an object there is a set of three dimensional circumscribing prisms if we let the object rotate.
[Ding 01] [Ding 00a] transforms the problem of finding a set of contact points or grasp on the object surface into a problem of convex quadratic or non linear programming. On both cases an objective function is chosen to minimize a distance. In the first case, the algorithm computes a set of initial contact points by minimizing the distance between the centroid of the contact point wrenches and the wrench space origin. The centroid can be represented by $A + Bx$, where A and B are constant matrices and x denotes a position vector which describes the contact points. The computing for good initial positions is formulated as a quadratic programming problem $min(A+Bx)^T(A+Bx)$, and a qualitative test [Liu 99] is performed for the grasp. If the grasp is force-closure the algorithm ends, if not new contact points are computed by moving the contact wrenches along a local search direction.

In the second case, the distance is computed between the center of the mass of the object and the contact points, the algorithm needs that a minimum of two of m contact points must be specified to find the rest of the contact points. Again a series of constraints are defined in such a way that the contact points rest inside the object facets and the vector forces inside the friction cones, which make some of these constraints non linear.

Computing Independent Regions

One of the pioneer works on the synthesis of force-closure grasps was made by Nguyen [Nguyen 88] a geometrical approach that uses the representation of contacts by wrench convexes. As we have seen, the force-closure is formalized by the use of theory of convexes. Instead of producing only points of contact that satisfy force-closure, independent regions for contact are computed by finding disjoint wrench convexes, such that any n-tuple of wrenches from these convexes is force-closure. Using line geometry of wrenches, these independent regions are constructed based on the shape of the object. These results were applied by Pollard [Pollard 94] to plan grasps using the whole hand.
Ponce and Faverjon [Ponce 95] presented a method for constructing maximal regions in polygonal object facets for three fingers satisfying the force-closure condition. The method characterizes a grasp by a system of linear constraints in the position of the contact points along the polygon facets. The regions that fulfill the constraints are computed by a projection algorithm based on linear parameter reduction using a simplex technique. In [Ponce 93] an extension of the method is made for the synthesis of three and four finger grasps for 3D polyhedral objects. Several sufficient conditions to force-closure grasp are proved. In the case of four finger grasps, using line geometry a characterization is found, the wrench can be seen as the line of action of the force applied at some point, the six coordinate wrench is known as the Plucker coordinates of the line ; intersection of lines can be computed. Equilibrium grasps correspond to the lines that are linearly dependent, then using this result it is shown that when four linearly dependent lines occurs, the lines are coplanar and intersect in a single point (concurrent lines), forming a regulus or two flat pencils.

The method transforms the conditions for force-closure into a set of linear inequalities in the position of the contact points and the position of the intersection point c_{int} of the lines of actions of the forces. For a polyhedron, each contact point can be parameterized by two variables (x_i, y_i) describing its position on the facet plane. As the contact points must lie inside the convex facet, a set of linear constraints are defined similarly as in the ray shooting approach. The set of constraints form a d-polytope in the Euclidean space defined by the intersection of half spaces $HS_i = A_iP \leq b_i \; i = 1, \ldots, n$, where A_i is real matrix, P is the position vector, b_i a real number, and n is the number of half spaces. The objective is to construct the orthogonal polytope projection onto a k-dimensional subspace. Due to the uncertainty, a way to minimize the positioning errors is to find maximal contact regions maximizing a criteria depending on their size

and location. The regions are computed for any set of contact points lying on them satisfying the force-closure condition. These regions can be represented by parallelepipeds [Nguyen 88], then a criterion is to maximize the minimum of the edge lengths of this parallelepiped.

The distance between the center of the mass and the center of the parallelepipeds is taken as the second criterion [Ponce 97]. The two criteria can be formulated as linear constraints and the problem to find the maximal regions is transformed to solve a linear program. The fingertips are placed at the center of the regions. Some of these algorithms or methods are considered here as optimal because of the use of some criteria to find the position of the contact points or regions. A list of the force-closure grasp algorithms can be found in [Pertin-Troccaz 89] [Bicchi 00], for a more general grasp synthesis algorithms, see [Shimoga 96].

1.5 Heuristic Grasp Planners

In all the above synthesis algorithms, a grasp is defined as the contact points on the object surface. As these methods do not consider the structure or geometry of the end-effector to apply the forces on the object, a grasp found has a higher probability that cannot be feasible for the end-effector. The use of heuristics helps to circumvent the complexity of the grasp planning problem and generally is easier to implement. This is advantageous when we want to use the algorithm with an experimental robotic platform.

1.5.1 Generalized Elliptical Cylinders

Based on the observation that humans pre-shape their hands to grasp an object, a taxonomy can be used to generate robotic hand pre-shapes to grasp the object, this grasp pre-shape depends on the object geometry. For an unknown object a model is necessary. In [Bard 95] the object is approximated by generalized elliptical cylinders GEC. The method consists in producing slices of the object similarly as the tomography process. Each slice contains a 2D contour description, which is used to form ellipses that enclose this contour, the ellipses directions could be defined by inertial axes or privileging some directions, e.g. for vertical approaching. If the ellipse does not enclose tightly the contour, it will be overtaken by a polygon and can be divided into smaller parts by finding concavity points [Bard 90]. A tangent line at the concave point is formed and the cut is made perpendicularly to the tangential line. Different line cuts can be computed if neighbor concave points are found. Each part is enclosed again by an ellipse, Fig. 1.4.

The elliptical cylinders are formed by a set of slices where ellipses have similar directions and size. The ellipse axes are used as the four directions to possibly grasp the object. The directions are valid if there is no intersection with other part ellipses on the slice. At the end, each GEC has grasp directions. Taking the size of the ellipse axes, a hand pre-shape is computed.

1.5.2 Natural Grasping Axis

Based on the observation that humans pre-shape their hands to grasp an object, an interesting heuristic is to find a grasp axis from the object geometry called *natural grasping axis* (NGA) [Michel 04] to guide the planning. The NGA axis then is a straight line parallel to the object surface. As a triangular mesh could be generated from a voxel coloring to represent the object geometry [Michel 05], the natural grasping axis is the most parallel line with all pair of facing facets of the object, Fig. 1.5. This line minimizes a criterion that is based on the quadratic sum of the angles between the normal facets that form a pair (opposite facets) and the natural grasping axis. An elimination of facet pairs is made by considering their surface, given that small surfaces do not contribute in a substantial manner for determining the axis.

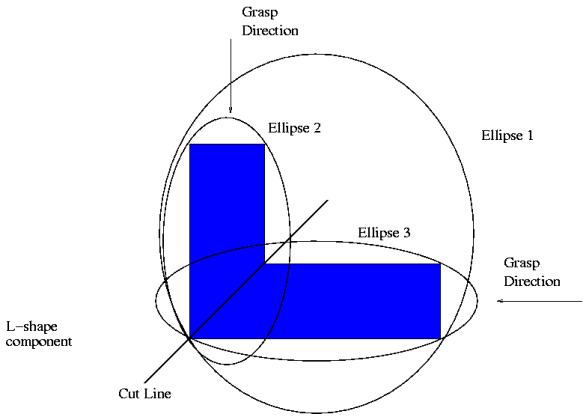


Figure 1.4 – Slice for an L-shape component and the ellipses enclosing the subcomponents after partition with respective grasping directions

Two main configurations can be generated from this grasping axis by taking the line as vector direction or an orthogonal vector.

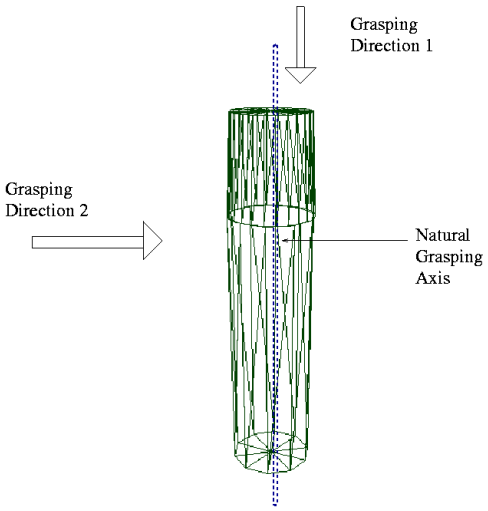


Figure 1.5 – Natural grasping axis and two main grasp directions

1.5.3 Grasping based on Shape Primitives

The object representation is an important aspect in grasp planning, using shape primitives as boxes, cylinders, cones, ... is a good idea to simplify the object geometry. The advantage of using well known geometrical shapes is that they allow us to generate grasps in a simple manner by parameterizing the contact points as well as the approaching directions. Then a complex object can be represented by a few primitive shapes Fig. 1.6, a mug for example can be constructed by a cylinder and a rectangle [Miller 03]. Finally based on some criteria a grasp can be chosen from a set.

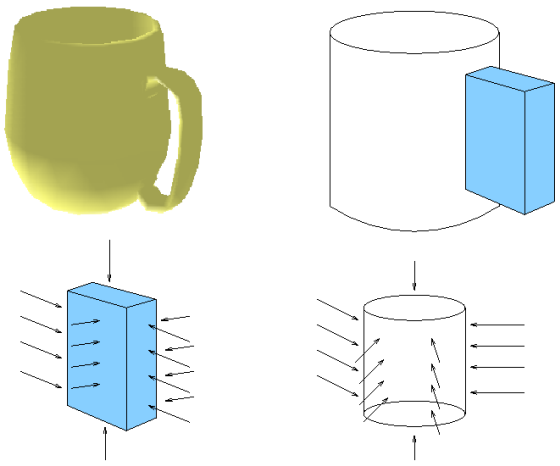


Figure 1.6 – Object approximate by primitive shapes and grasp directions

1.5.4 Random Grasps

The approach assumes that we can generate a sufficient number of candidate grasps and choose the best among them. Reference [Borst 03] shows that it is not necessary to generate optimal grasps. The optimality criterion is here the condition of force-closure with a quality index defined as the length of the smallest wrench in any direction that breaks the grasp. The criterion was applied in human grasps in a set of simple objects and compared with grasps randomly generated.

It is obvious that finding optimal grasps is very complex but finding a fairly good force-closure grasp is not so difficult as there are many good grasps. There is no practical relevance for most objects if no optimal grasps are generated, as these grasps cannot be executed for most robot hands. A statistical study shows that an average quality grasp is an acceptable good grasp making randomized grasp generation suitable for robot grasp planning in manipulation tasks. Fischer [Fischer 97] and Borst [Borst 99] use a random generation strategy. From one arbitrary point and frame inside the object, they launch a first ray in direction of the frame's X-axis. The same for the other rays but rotated about the Z-axis by 120 degrees. Toth [Toth 99], tries to find a Y-shape grasp star. The penetration points on the object surface give the contact points.

1.6 Grasping based on Human Demonstration

An optional approach seeks to reduce the complexity of grasp planning problem by using human motions to guide the robot actions to perform a task. In [Kang 97a] it is shown how a human grasp can be mapped to a manipulator robotic hand (Utah/MIT) for replication. A two level method is used, the first level is a mapping of the human hand position with virtual fingers. This group of virtual fingers act on the object in a similar way that the human hand does. The second is called the physical level where manipulator and object geometry is considered to fit the manipulator grasp. Power and precision grasps can be mapped.

After temporal segmentation of the manipulation task, a data acquisition system (cameras and glove) extracts information from the environment that provides the locations and configurations of the objects and the contacts on the grasped object. A recognition phase is executed to identify the type of grasp and to synthesize the robot grasp. The descriptions of the human grasp are used as cues for the robot to plan

the grasp, a 3D graphical model of the robotic hand is employed to represent the human grasp connecting the effective contact points between the hand and the object. The contacts with the palm or the finger links are modeled as points. A taxonomy is used to recognize the type of grasp on the object, the grasp classification depends on the number of fingers and links in contact with the object and the shape of the object. Once the grasp was recognized, a manipulator grasp is generated. Depending on the type of grasp, the planning varies, having one procedure for power grasps and another for the accurate ones. The procedure of power grasps starts with the alignment of the robot hand relative to the human hand, an approach vector is computed from the position of the human hand, transformations at the robotic hand frame are applied for the two hands to coincide. A hand translation toward the object is performed until the palm makes contact with the object surface preserving the hand orientation all the way, followed by an adjustment of the hand to fit the object to the palm. Finally, the fingers close surrounding the object. For acute grasps, an initial approximate grasp pose G_p is determined by computing a pose frame from contact points, then a kinematic feasible robotic hand pose is searched in a neighborhood of G_p. As in the power procedure, an adjustment of the grasp pose has to be made. This is done by optimizing a compatibility criterion [Chiu 88] $C(P_h(t,r),\theta_{rh})$, where P_h is the hand pose vector and θ_{rh} are the joint angles vector representing the robotic hand posture.

Recently in [Ekvall 04], a similar framework was used taking as a robotic end-effector the Barrett hand. The classification of the type of grasps is again oriented by two general classes : power and acute grasps. A difference is found in the recognition phase where learning techniques as the hidden markov models [Ekvall 05] are used. A glove with position sensors gives the location of the fingers and palm ; the Graspit simulator [Miller 01] serves as graphical interface to adjust the pose. The mapping process uses the sampling of a number of typical human poses and the match with robotic hand poses as inputs to a neural network. The output is the joint values for the robotic hand.

1.7 Integrated Manipulation Systems

Manipulation tasks that for humans or even some animals are simple and present no major difficulties, could not be so easy for a robot. Perception, control and planning are fundamental to try to solve manipulation tasks for autonomous systems. Only a few robotic systems have such functionalities. Grasp planners are essential and critical in such systems, obviously for autonomous robots one of the most difficult tasks is to manipulate arbitrary objects because we cannot simplify the problem by using predetermined grasps after a recognition object phase.

The first attempts to solve manipulation tasks were proposed in the eighties using automatic programming or task-level systems by a decomposition of the task in actions, independently of how these actions were accomplished. Handey [Lozano-Perez 87], SHARP [Laugier 85] used motion planners (grasp, path and trajectory) and vision systems to perform pick and place tasks. Generally the grasp planners made an exhaustive search for grasps. In the nineties an integrated robotic system for dextrous manipulation was proposed in [Bard 95], this integration approach uses vision, sensing, planning and control. The representation of the object and obstacles was made by octrees. One of the original points is the grasp strategy. An object modeling for grasping is implemented in an incremental way by slices constructing elliptical cylinders and picking a shape hand to grasp the object or a part of this. The shape hand is based on human taxonomy. Actually the grasping is made by tactile sensing.

In [Lee 05], an approach to model the 3D workspace with a stereo camera is presented. The approach constructs a set of invariable features from the clouds of 3D points, which allowed the comparison with a database of object model for recognition. If an object is identified, then the cloud of points is substituted by the model. The remaining clouds of points are considered as obstacles and they are represented

by multiresolution octrees, similar as the ones found in [Bard 95]. Once a model of the workspace is constructed, a grasp is found in the selected object. In [Jang 05], we find that a grasp is generated according to the object model in the database. For example, for a box, four approach directions are given by the axes $\pm x$ and $\pm y$ of a frame defined on the object. This only allows for translational motions to reach the object. A line is defined along the frame axes and several contact points are created on the line. In order to know if the grasp can be executed without risk of collision with the obstacles, a test of visibility is performed. This test consists on verifying the visibility of the object, if after the environment and object rendering along the approach direction there are no obstacles pixels with a depth greater than the object pixels, then the object is visible. The second part of the test is to make a sweeping of the end-effector to the object, if there is an overlapping between the end-effector and obstacles, the object cannot be grasped. The sweeping operation is transformed in a rendering operation again, if there are pixels of obstacles between the end-effector and the object, a collision exists.

The strategy to simplify the object shape into a shape was used in [Taylor 02], a vision system approximates the shape of the object by a box and the grasp planner generates grasps for this box. The same approach is used by Petersson [Petersson 02], but he uses a database containing the object label, size, grasping configuration and necessary way points. All these systems have been tested in fetch and carry tasks.

The integrated system Robutler [Hillenbrand 04] uses a vision system for the recognition of objects and for unknown objects uses the generalized cylinder idea [Bard 95]. The difference with other systems is given by the force-closure grasp generation, here a random generation is made. The system was tested for glass bottles only.

1.8 Discussion

We have seen that since the nineties, the methods for the solution of motion planning are sampling-based approaches with probabilistic and diffusion algorithms, offering new techniques that can be used in experimental robotic platforms. The implementation of a general grasp planner for autonomous robots is a very complex problem due to the search for a whole robot grasp configuration in a high dimensionated space and the surface of the object. Others diffculties appear, the consideration of the robot and end-effector structures to plan the motions of the robot to actually grasp the object. Some of the algorithms presented in this chapter find a solution for only a part of the problem, the generation of contact points that sastisfy the force-closure condition. Besides that in general they are complex to implement, making these algorithms unsuitable for our purpose. Nevertheless, we keep the condition of force-closure for the grasps because we need to transport the object from one place to another in a firmly way. From a practical point of view, the heuristics algorithms seem to be a better option because they offer a way to quickly find a grasp on the object.

The use of the natural axis give two good directions to approach the object and try to find a grasp but it is restrictive because we can intuitively see that there are a lot of other grasps in different approach directions and because in the case of more complex objects, the method is incapable to find grasps in the parts of the object. A more interesting solution is to model the object with some primitive shapes, in that way, we can plan grasps for each known shape. The drawback is that for real arbitrary objects there is no suitable strategy because we will need a big set of primitives to model it. The generalized ellipse volume seems to be a more effective method, it considers the partition of the object to grasp at the same time that it constructs the object model. As the method iteratively makes 2D slices of the object to finally obtain a 3D model with the union of all the slices like the tomography process, the partition of the object follows a two dimensional reasoning. The problem of this method is that modeling and grasp generation are intimately linked, many parts of the object cannot be considered for grasp because the high

risk of incompletness of the object model. A more interesting approach to the grasp planning problem is the random generation of grasps [Borst 99] to avoid the complexity of implementation of the first approaches to compute force-closure grasps. We will see in chapter 3 and 4 respectively, that the work presented in this thesis focuses to solve the grasp planning problem in its entire way. In order to obtain the solution, we guide a random grasp generation process using the inertial properties of the object and decomposing the object into subcomponents as in [Bard 90] [Bard 95] but reasoning in three dimensional space. Autonomous robots can be used in diverse environments, notably an increasing research direction is for service robotics to help elderly people or with some kind of handicap. The robot must take decisions in a human environment and will interact with humans. Interactive manipulation capabilities are essential for these purposes. In the next chapter a simple architecture for such interactive manipulation tasks is proposed.

Chapter 2

IMAP : Interactive MAnipulation Planning

Today, research in robots that can act in human environments receives high attention. It is desirable that robots be capable to perform simple daily human activities. The robot has to interact at different levels with other entities present in these environments, particularly with humans. Interacting through gestures and voice commands allows to understand the human intentions and instructions, but there are other kinds of interaction activities between them. We are interested in interactive manipulation tasks and more specifically in hand over tasks.

For a service robot, usually the human will ask for objects to be carried from one place to another or for objects to be delivered to him. The robot has to pick the object and hand it to the human. Cooperative manipulation is another common interactive task [Esteves 06]. The same is true for interaction between robots.

The pick and place operation has been studied intensively in the eighties and nineties, different approaches have appeared. We cite some of the most representative planners : Handey [Lozano-Perez 87], SHARP [Laugier 85] and SPARA [Mazon 90]. All these planners integrate different modules to perform a task.

Handey is composed with the following modules : object modeling, a module for model-based localization, a path planner, and a grasp module. The main steps that the planner makes for a pick and place operation are : an object localization is made and in some cases a modeling step for simple polyhedral objects precedes the localization step. Then, a grasp is found on the object that can be reached in the initial and final position (compatible grasps). A collision free path is computed to move the robot to the grasp position and grasp the object. If a compatible grasp is not found, then a re-grasping process is executed using obstacles free placements until a valid grasp is found for the final destination. A second free collision path is generated to move the robot from its current configuration to another one near the final object destination. Finally, to place the object at destination generating a force-guarded motion or a compliant motion.

SHARP is a system for automatic programming that mainly integrates three planners : a grasp planner, a transfer motion planner and a part-mating planner. These planners are intended to generate robust grasp positions on objects, safe trajectories for robot motion and sensor based motions, respectively. Here, it is implemented a communication mechanism based on constraint propagation, such constraints can be : uncertainty (ex. imprecise object localization) and accessibility.

SPARA architecture is composed of three planners : a grasp planner, a global planner for the transfer motions, and a local planner for motions between obstacles in a plane. A module for uncertainty propagation, a workspace module that computes the space generated by a point attached to the robot end-effector

moving the first three degrees of freedom, a local environment module builds a projection of the object and obstacles in a plane for the approach motions and intermediate linking point generator module, where accessible link points can be reached by the global and local planner and finally a verification module to guarantee that a local motion is valid.

We adopt the approach and some of the ideas from these planners and we propose an architecture to solve the above mentioned interactive manipulation tasks. The IMAP architecture for task planning offers an option to try to solve common essential tasks for interaction. It is not the purpose of this thesis to implement a general automatic task planner. We know that motion planning techniques can be used to solve some basic manipulation tasks, but a task planner is necessary for more complex tasks.

This chapter describes the different modules included in our system and shows how a controller invokes them and how it manages their interaction. Next, we present the control processes which can be implemented to solve *delivery* manipulation tasks. A delivery task can perform two operations or subtasks : hand over or pick and place operations.

The techniques used in our planner for path planning are sampling-based probabilistic methods, which allow us to find free collision paths for high complex mechanisms and make unnecessary the partition of the path planning in a global and a local planners for manipulator arms.

2.1 IMAP Architecture

The approach proposed in this thesis is a simple planner with a modular architecture. Each module performs an operation as shown in Fig. 2.1. Given the task to be performed, a controller follows a sequence of actions to solve the problem. These actions can be given by a finite state machine. Every module in the interactive manipulation planner (IMAP) performs specific operations and they communicate with a central module called the controller.

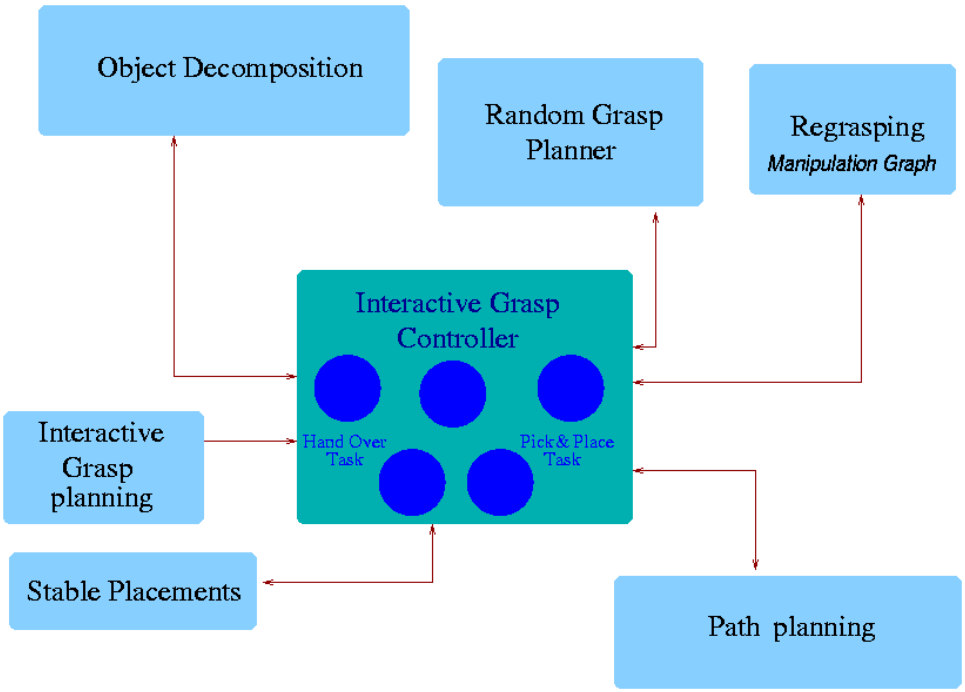


Figure 2.1 – Modular Architecture of Interactive Manipulation Planner

2.1.1 IMAP Controller

This is the part that controls the task. The controller can be composed of a set of machines with a finite set of states, a set of symbols and transitions. Each manipulation task can be described by a finite state machine FSM or automaton. The machine transitions depend on a set of flags that indicate the state of the machine and the next state to go to. The values of these flags are set from the operator output. Each machine state asks for operations and uses the operators to request a module service. The module notifies if the operation has succeeded or not through the flags. The output data from and for each module depends on the operation performed or to be performed respectively. The controller receives and sends the data throughout the operators. The input for the IMAP controller is the initial state (the initial location of the object and robots) and the final state (the final pose of the object) as well as the type of task to be performed.

2.1.2 Planner Operators and State Flags

An operator is an action or operation in a module, if there is only one operator for a module, the operator and the module are the same. The operators take as input the type of action to execute and the corresponding valid data. The output is composed of a state flag that indicates if the action has succeeded or not, and the expected data in case of success.

We define several types of operators. The grasp operator G_{op} indicates that a valid grasp has to be generated, a decomposition operator OD_{op} requests a partition of the object because the geometry of the object does not allows to find a valid grasp on it. The regrasping operator RG_{op} asks for the creation of a manipulation graph to find a manipulation path that consists as we saw in chapter one, in a decomposition of transit and transfer motions.

Regrasping operations are needed when the initial grasp configuration is not compatible with the final interactive grasp.

The motion planner operator PP_{op} calls a probabilistic motion planner to generate a free collision path between two configurations. Finally, the interactive grasp operator IG_{op} is used to plan an interactive grasp (double grasp) on the object to be manipulated. The operators and flags are shown in Table 2.1.

Figure 2.2 – Operator Structure

2.1.3 Random Grasp Planning

The grasp planning module can produce a set of grasps on the object and robot grasp configurations that allows the integration phase with a path planner to be coherent. With the integration of the grasp generator and path planner, a complete solution for the grasp planning problem can be obtained, that is the approach and grasp robot motions for manipulation. The grasps generated can be used for other modules as the regrasp module. The grasp planner has two grasp generators for a three contact gripper

Table 2.1 – Operators and Flags

Operator	Module	Input	Output	Flag
OD_{op}	Object Decomposition	Polyhedron	2 polyhedra	OD_f
IGP_{op}	Interactive Grasp Planner	Polyhedron, arms, end-effectors	Double grasp	IGP_f
GP_{op}	Grasp Planner	Polyhedron, arm, end-effector	Best grasp and grasp robot configuration	GP_f
SG_{op}	Grasp Planner	Polyhedron, arm, end-effector	Set of grasps	SG_f
SP_{op}	Stable Placements	Manipulation site and polyhedron	Set of placements	SP_f
PP_{op}	Path Planner	Initial and final robot configurations	Path	PP_f
CT_{op}	Compatibility	Grasp and polyhedron	Compatible robot configuration	CT_f
RG_{op}	Regrasping	Set of grasps and placements	Compatible grasp	RG_f

and a three finger articulated hand. On the next chapter we will describe a grasp planner to compute random grasps in unknown objects.

2.1.4 Interactive Grasp Planner (IGP)

In some manipulation tasks there is a need to hold an object by two or more end-effectors or for a human and a robot. The function of this module is to plan a double grasp for interaction via manipulation. The structure of two robots is used to produce the grasps, in this way we assure that the grasps are mutually free of collision between them and with the environment. The generation process uses object decomposition to partition it into smaller components and by producing grasps on that components. If the IGP finds the double grasp, the module returns a positive answer setting the corresponding state flag.

2.1.5 Stable Placements

Stable placements can be found by taking the projection of the object facets that may contact with the support plane and compute if the center of gravity lies inside the convex hull of the polygonal contour. At this point, we assume that the object can be placed with the same configuration at the destination site, or that a stable final configuration is given. This module should deal with finding the support places where the object can be positioned without risk of falling, for example : obstacle zones with faces with important slopes or with a very small surface.

2.1.6 Transfer and Transit Path Planning

The transit (robot alone) and transfer (robot with object) planners are basically composed by any probabilistic sampling based method for motion planning. For long transfer or transit path, a basic PRM or a visibility-PRM algorithm can be used, it is convenient because with the same graph we can plan several robot paths. These two types of motion planners were briefly described in the previous chapter.

2.1.7 Regrasping

As we already saw, the fact that we cannot find a compatible grasp for both, the pick and place configuration implies that the object must be reconfigured until a valid grasp is found to move the object to its destination or final configuration. This regrasping process can be seen as a pick and place problem with intermediate goals. This module can use the set of grasps generated by the grasp planner. Sometimes the object can be hidden by an obstacle or surrounded by a kind of cage, a set of transfer and transit motions can solve the problem by using the so called manipulation graph, already reviewed in chapter 1. The execution of this manipulation graph uses stable placements and a set of grasps already generated to move the object to intermediate configurations.

2.1.8 Compatibility

The compatibility module verifies that a grasp that picks the object can be the same for the object place operation. The test implies that a collision free robot configuration exists for the grasp, allowing access to the pick and place locations. The method for the test is simple, we use the grasp as a constraint for the robot and we try to generate a configuration with a random loop generator algorithm described next in the chapter. If such configuration exists and it is collision free, then the grasp is compatible.

2.1.9 Object Decomposition

This module makes a recursive partition of the object into subcomponents. Each subcomponent is used as an input to the grasp planning module to plan grasps on it. The service of this module is used for two purposes : when two grasps are required or to generate a grasp for complex objects (generally a non-convex one). The OD_{op} takes a polyhedron model as input, if the decomposition succeeds to generate the partition, the operator gives as output the model of two polyhedra that corresponds to two subcomponents of the object, otherwise the output is empty and the flag OD_f is set to zero value to indicate failure. Chapter 4 deals with the decomposition approach and the integration with the grasp planner.

2.2 IMAP Delivery Tasks

For demonstration purposes we have chosen representative manipulation tasks to validate our approach. The pick and place task is interesting because even if we cannot consider it as an interactive operation by itself, it can be part of an interactive manipulation task. If we have, for example, a hand over, a task that cannot be performed because of the object size, it will be impossible to generate two grasps on it, then an optional operation is to place the object in the nearest place to the other robot or entity. The IMAP architecture presents the relation between the different planning parts to find a solution. We can notice that this is an automatic way to solve interactive manipulation tasks from a geometric point of view.

2.3 Pick and Place Task

The pick and place task is a common task that any manipulator system has to solve. When a robot or human asks for an object, it happens frequently that planners cannot compute a double grasp because the object is not big enough or its geometry does not allow it, the controller then can generate a plan for a pick and place task. The goal is now to place the object in a place where the robot or human partner can reach it.

We assume that if the object can be grasped in a configuration, then it can be placed in the same position at some designated place. As an object cannot move by itself, it is the action of the robot on the object that makes it move, we can see this differently if an adequate robot description is used. The actions the IMAP controller takes for a pick and place task are :

- Generate a set of grasps and choose one. This operation is executed by the grasp planner, the grasps are ranked using a quality criteria. The output is the complete robot configuration that grasps the object.
- Compatibility test. Verify if the grasp used to pick the object allows it to place it at final destination with final configuration.
- Transit motion. Plan a free collision path from the initial robot configuration to the grasp configuration. In this work we did not consider the constraints of compatibility between the initial and final position of the object to be manipulated. The computation of stable placements that allows the object to rest in equilibrium, some constraints of this kind are treated in [Jones 90].
- Regrasping. Find a valid grasp if there is no compatibility that allows to reach the placement location with the final configuration by reconfiguring the object using stable placements and a set of grasps from the grasp planner and making a set of transfer and transit motions.
- Find a compatible robot configuration for object placement. Using the grasp resulting from the compatibility test and a collision free robot configuration is found near the object destination.
- Transfer motion. Plan a free collision path from the grasp configuration to the final configuration by using a robot description that allows the robot to move the object.

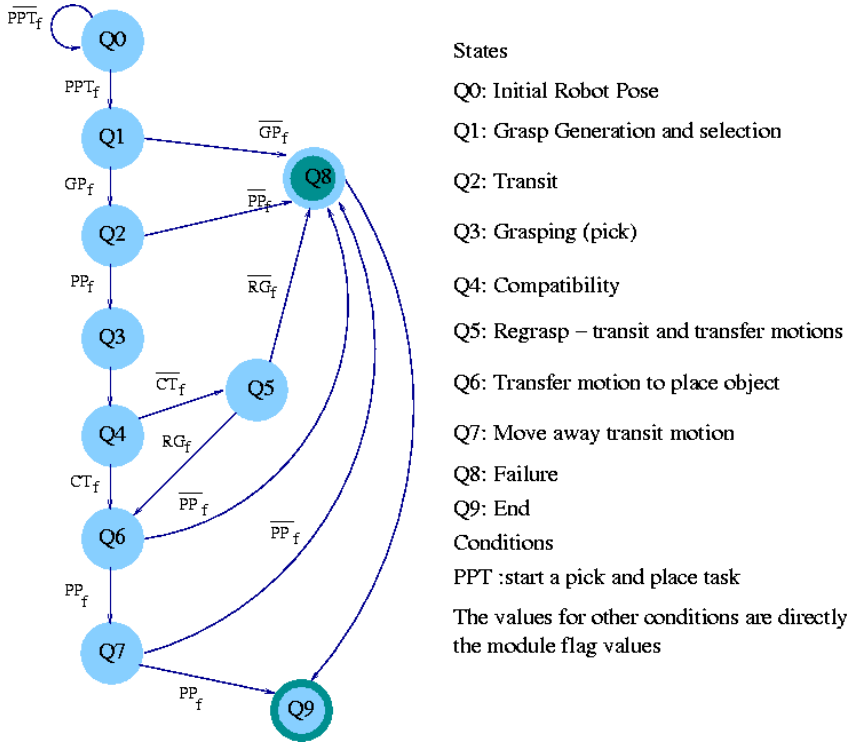


Figure 2.3 – Pick and Place finite state machine

2.3.1 Motion Planning for Robot-Object Chain

We have already seen that manipulator robots are described by a set of links and joints. For planning transfer motions, the robot has to move the object maintaining the grasp found by the grasp planner. To find a solution, we first need to define the static object as a movable object, this can be done by integrating the object model in the robot description as a part of it, and the motions of this new part are the same as a free-flyer joint, the object can be moved and rotated in any direction in the space. The robots we use are mobile manipulators by consequence, they are redundant robots which makes more difficult for the motion planning. New motion planning techniques like sampling-based methods presented in chapter 1 allow us to deal with complex structures.

Probabilistic roadmaps and Rapidly-Random Exploring trees methods sample the configuration space by generating random collision free robot configurations. To plan motions for the robot and the object, we can define the grasp as a kinematic constraint that has to be satisfied in the different phases of the motion planning when the constraint is active. This allows us to plan a path for the object only and to find a configuration for the rest of the robot using inverse kinematic techniques. We can use another method for the robot and object motion planning if we consider that an articulated manipulator forms a chain, if we consider its mobile base and the object, the whole system forms a closed kinematic chain if the object is placed, for example, on the ground. We can plan paths for the robot and the object using motion planning algorithms for closed kinematic chains, even if for transfer motions the object is hold by the robot and not necessarily touches the ground, breaking the closed chain. In algorithms for closed kinematics chains [Lavalle 99] [Han 00], decomposing robots or mechanisms in active and passive chains is a good strategy. In our case the robot arm is seen as the passive chain and the object as an active chain.

The manipulator arm is attached to the object through a frame. After a grasp is planned, this frame is updated allowing the motion of the whole robot to maintain the correct grasp. We will see in the next section that in hand over operations the same kind of algorithms can be used.

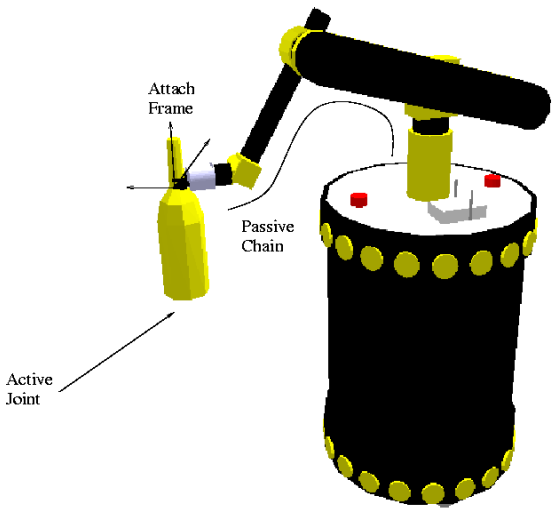


Figure 2.4 – Division of system in active and passive chains for pick and place tasks

2.3.2 Random Loop Generator

For the kinematic robot-object chain motion planning problem we use a method presented in the work of [Cortés 02], the method called random loop generator RLG was proposed to plan motions for closed chain mechanisms. Such system is broken into active and passive chains. Active chains are considered for the sampling with forward kinematics and then inverse kinematics is used for the passive chains to reach the active ones, and in that way follow the motion of the rest of the mechanism. A closed chain configuration is valid if the end of the active chain lies on the passive chain workspace. The probability to generate an active chain configuration that can be reached for the passive chain depends on the workspace intersection of the two chains. The method searches active chains that can always be reached by passive chains in the sampling process. The workspace is delimited by the possible joint range of values. The value of one joint affects the position of the next joints in the chain. The algorithm generates joint by joint the configuration of the active chain. Instead of taking a random value from the joint range of motion given by its limits, a new interval is computed for every configuration to carry the end joint of the active chain to the workspace of the passive joint.

If we have an active chain configuration, the values of the active end joint allow the end joint of the passive chain to reach it and close the loop, forming an interval of closure. To compute these intervals of closure, a simple and effective way is to take two concentric spheres and compute the intersection between them. The radii of the spheres is the distance between the origins and ends of the chains, when the chains are in full and minimum extension respectively. We assume that there is a cone whose vertex intersecting the common center of both spheres. Then, the intersection volume between the cone and the two spheres determines the interval of closure. If the intersection is void, the configuration is rejected.

Algorithm 4: RLG for Single Loop

```
input  : the loop τ
output : the configurations q[n_sol]
begin
    q_a ← SAMPLE_q_a(τ);
    q_p[n_sol] ← COMPUTE_q_p(τ, q_a);
    if n_sol = 0 then
        return Failure ;
    else
        q[n_sol] ← COMPOUNDCONFIGURATION(τ, q_a, q_p[n_sol]) ;
    end
end
```

The algorithm 4 describes the random generation for a single loop. First of all, the parameters of the active chain q_a are computed using the algorithm 5, as input has the loop that defines the kinematic chain involving joints from the base joint J_b to the end joint J_e. The values for each joint are computed sequentially. The algorithm reduces the complexity of the closed chain at each iteration until the passive chain q_p reaches the active chain by its inverse kinematics. The closure interval function returns a set of intervals I_c that approximates the exact intervals of closure. The current variable joint closure interval depends on the previous joint configurations. The closure interval must be recomputed for all the joints in the generation of each new configuration, except for the joint treated. The approach is conservative in the way that all the space must be considered for the sampling process, it is possible that in some iteration the method computes an empty set, in this case the process has to be reinitialized. Generally there are several solutions $q[n_{sol}]$, the configurations are obtained combining q_a with all the different solutions of q_p.

Algorithm 5: Configuration Sampling

```
input  : the loop τ
output : the parameters q^a
begin
    ⋆(J_b, J_e) ← INITIALIZATIONSAMPLE(τ);
    while not ENDACTIVECHAIN(τ, J_b) do
        I_c ← COMPUTECLOSUREINTERVAL(τ, J_b, J_e);
        if I_c = ∅ then
            goto line⋆;
        end
        SETJOINTVALUE(J_b, RANDOM(I_c));
        J_b ← NEXTJOINT(τ, J_b);
        if not ENDACTIVECHAIN(τ, J_e) then
            SWITCH(J_b, J_e);
        end
    end
end
```

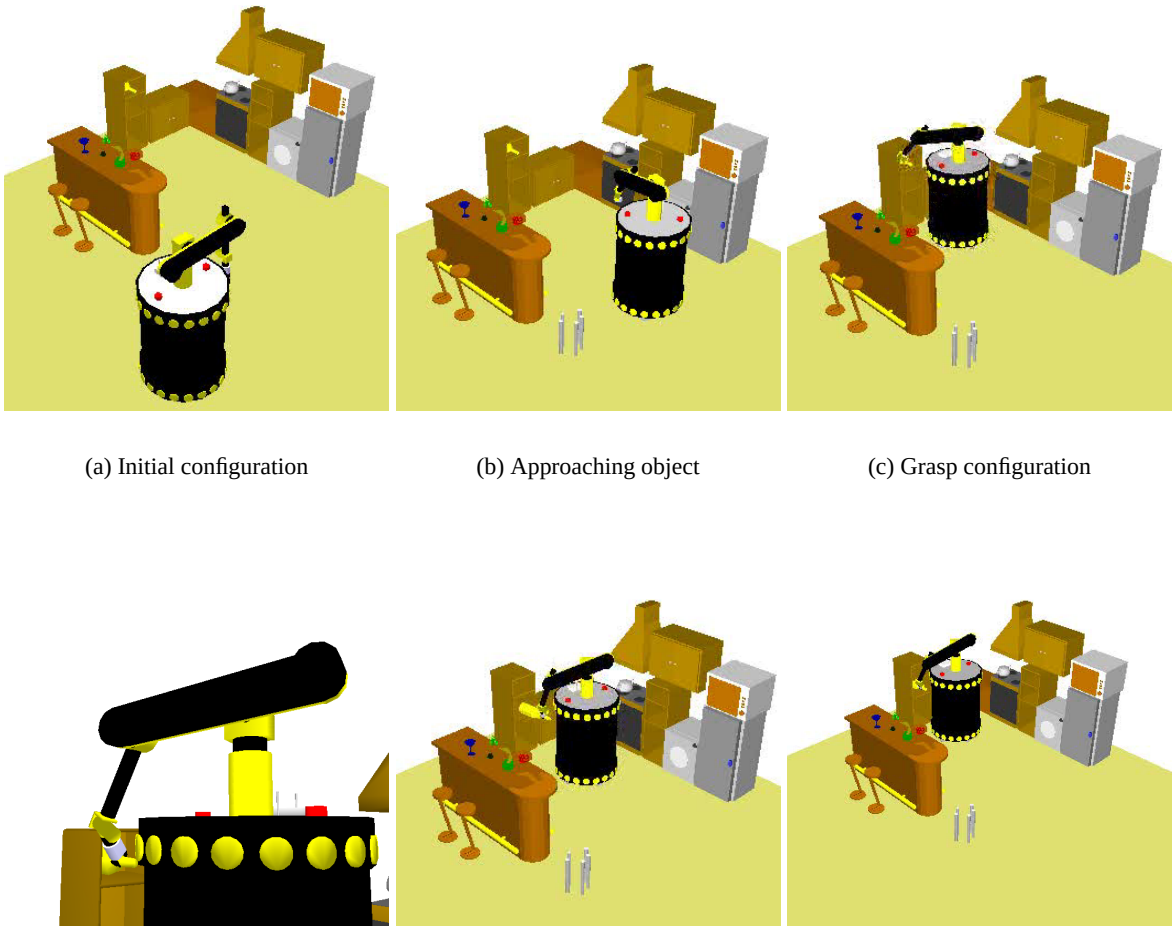
In Fig. 2.5(a-e), we can see a sequence of images for a pick and place task, the goal is to move the bottle from one shelf of the kitchen furniture to a shelf under. The first and final images give the initial and final object-robot configurations, respectively. Once a grasp configuration is computed using an object decomposition, a motion path for the robot is found and executed to pick the object. Fig. 2.5b shows a robot configuration when approaching to the object. When the object is grasped, the object is part of the robot and we indicate this by activating the kinematic constraint that makes the passive chain of the robot to follow the active one (object) as we already explained above, see Fig. 2.5d.

2.4 Hand Over Task

In a hand over task we try to deliver an object from one robot (giver) to another (receiver), this interaction can be solved by using algorithms for parallel mechanisms. At the moment when two mobile manipulators grasp an object, they form a parallel mechanism with mobile bases. Random collision free configurations can be obtained, each configuration can be seen as valid places to carry on the exchange. If the interactive task is between a robot and a human, we make the next assumption, if we can plan a second grasp on the object, a human will find it or even a better grasp.

The actions the IMAP controller takes for a hand over task are :

- Generate two grasps on the object. This operation is done by the interactive grasp planner, two mutually free collision grasps are generated.
- Generate possible exchange configurations. Using a probabilistic roadmap method that allows to find a set of configurations where the object exchange can take place. These obstacle free collision robot configurations are possible because the robots and the object are described as a parallel robot with closed chains.
- Compatibility test. Verify if the *giver* robot grasp is compatible with the initial object configuration.
- Transit motion for giver robot. Plan a free collision path from the initial *giver* robot configuration to the grasp configuration.



(a) Initial configuration (b) Approaching object (c) Grasp configuration

(d) Grasp configuration zoom (e) Moving object (f) Final Configuration

Figure 2.5 – Main steps for a pick and place task, the robot goes to the object and transfers it from the upper shelf to the shelf under.

- Regrasping. Reconfigure the object using stable placements and a set of grasps from the grasp planner making a set of transfer and transit motions until the object can be grasped with the interactive *giver* robot grasp.
- Transfer motion for giver robot. Plan a free collision path from the *giver* robot grasp configuration to the final exchange configuration by using a robot description that allows the robot to move the object.
- Transfer motion for receiver robot. Plan a free collision path from the *receiver* initial robot configuration to the final exchange configuration.

Usually the controller asks the interactive grasp planning module to compute two possible grasps for the operation. One of the grasps is for the giver robot and the other for the receiver or for the human. If the interactive grasp planner succeeds to find the double grasp on the object, then an exchange configuration is computed. Once the two grasps and the exchange configurations have been defined, a path for the giver robot is computed using any sampling probabilistic method. Once the robot is at its grasp configuration,

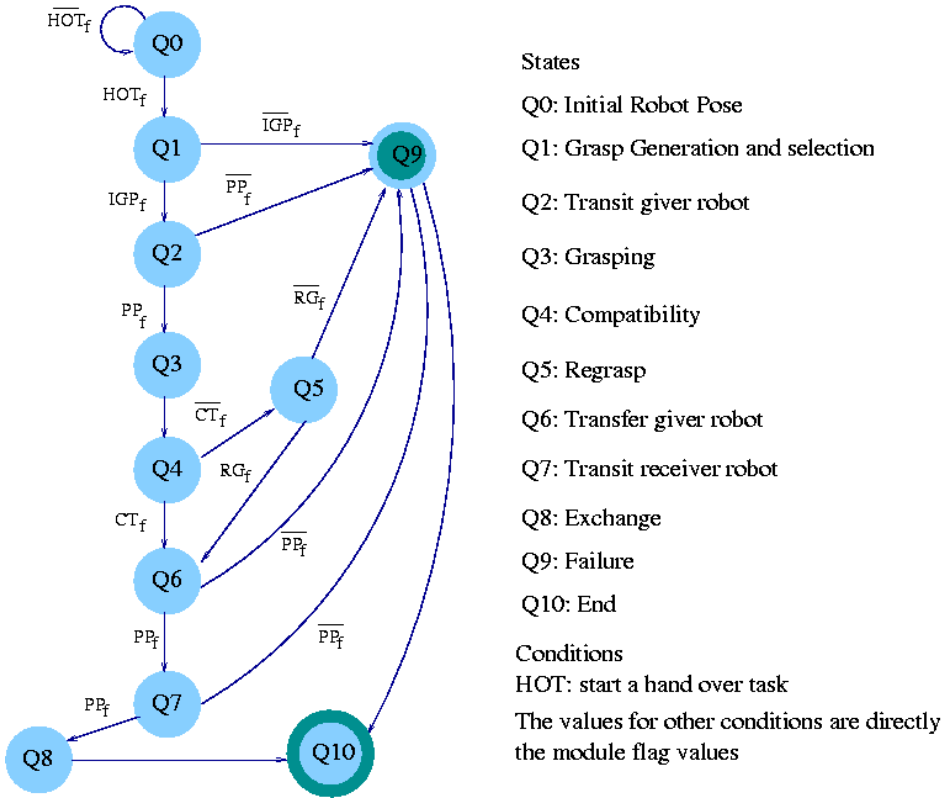


Figure 2.6 – Hand over finite state machine

the object is considered as a part of the robot, a *random loop generator* (RLG) algorithm can now be used with motion planners to find a path for the new formed robot. The robot description as parallel robot and the algorithm to plan motions for it are presented next.

2.4.1 Interactive System as Parallel Robot

The advantage of describing the object and robots as one robot only allows to plan the motions of the whole system by planning for the object alone and using kinematic constraints to make the other parts of the robot to reach it, as shown in the previous section. A parallel system is seen as a system composed by at least two chains that form a loop with the robot base and the end-effector or platform, the algorithm used for the generation of parallel robot configurations [Cortés 03b] is based on the random loop generator [Cortés 02]. In fact, if we have two mobile robot manipulators, the parallel system can be seen as a closed chain because the ground closes the loop constructed by the two robotic systems and the object, that is considered as the end-effector. The workspace for the parallel mechanism is computed from each chain workspace and the position and orientation of the end-effector. Algorithm 6 gives the main pseudocode to generate random configurations of the parallel mechanism. The purpose is that all the m kinematic chains k can reach the platform (object) simultaneously. The configurations of these m chains are computed by the RLG algorithm through the function $Single\_Loop\_RLG$. For redundant kinematic chains, the function returns a finite number of feasible configurations. The configurations correspond to a sampled value of the active q_{k_i} and the n_{sol_i} of the passive q_{k_i}.

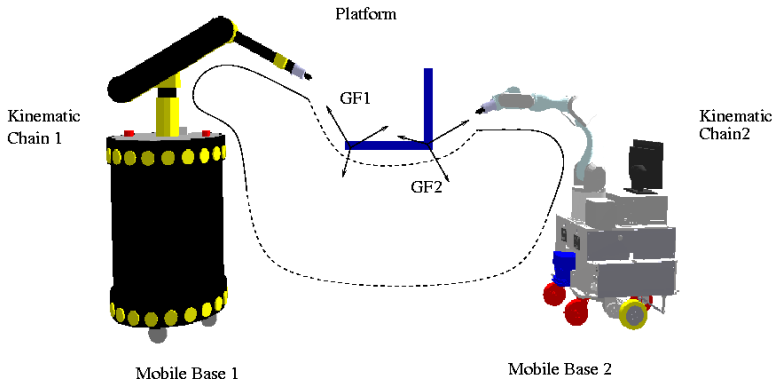


Figure 2.7 – The two robots and the object can construct a parallel mechanism with mobile bases

Algorithm 6: RLG for parallel systems

input : the parallel robot R
output : the configurations $q[n_{sol}]$
begin
  $q_P \leftarrow$ SAMPLE_$q_P(R)$;
  forall $i = 1$ *to* m **do**
    $q_{k_i}[n_{sol_i}] \leftarrow$ SINGLE_LOOP_RLG(k_i);
    if $n_{sol_i} = 0$ **then**
      return Failure ;
    end
  end
  $q[n_{sol}] \leftarrow$ COMPOUNDCONFIGURATION($R, q_P, q_{k_i}[n_{sol_i}]$) ;
end

The joints that define the position and orientation ($pose$) of the robot q_P are selected as active variables, in our case the object is selected as active joint. Then the defined chains (manipulators) are treated as passive single loop closed kinematic chains between the base frame and the object frame associated to the chain, the grasp frame. When chains are not redundant there are no other active variables. For mobile manipulators, the frame associated to the mobile base can freely move on parallel planes and they are considered in the sampling process of q_P by placing them at a position that maximizes the variation of each parameter. Briefly, the mobile manipulators form a redundant chain that can be as well divided in a passive chain (manipulator) and an active chain (mobile base), then random configurations can be computed using the random loop generator. Algorithm 6 starts computing q_P with the sample function that is similar to that of the random loop generator.

The active joint values or parameters are sampled from the computed closure intervals I_c, but considering the several closure constraints imposed at each loop. The configuration of the platform or the object in this case is sampled in two steps : first the position coordinates x_P and y_P are found in a rectangle formed by the intersection of the projection of the approximated workspace of the loops (spheres) and a plane, and then the coordinate z_P is computed considering the intersections of a perpendicular line to the plane at point $(x_P, y_P)$with each workspace volume, defining an interval I_z. Secondly, the orientation of the object is computed by finding the three basic rotations of the plane for the object positions. Each rotation of the position point (x_P, y_P, z_P) describes a circle. The intersection of each circle with

the spheres gives a set from which the sample is taken. Once the configuration of the object is done, the configurations of the single loops are computed using the random loop generator described in the above section.

There are some special cases where the manipulation planner has a high probability to fail, this is the case when the environment is completely divided into two or more independent regions and the robots are placed in different regions. Hand over configurations are difficult to be found at the borders of these regions unless the work spaces of the mobile bases that forms the parallel robot coincides with their respective regions.

In Fig. 2.8, we can see that a set of robot positions can be found by using a PRM method in the learning phase, the nodes of the roadmap represent a collision free robot configurations that can be used as possible exchange placements where the hand over operations can take place.

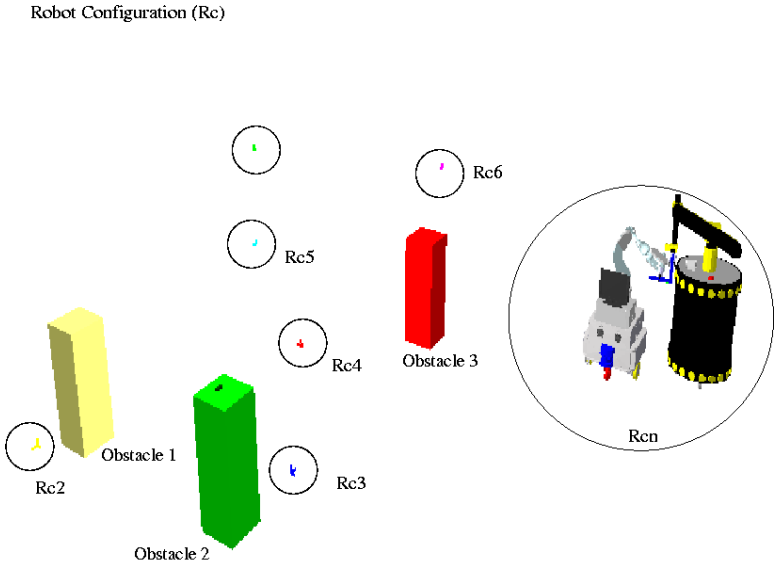


Figure 2.8 – Several configurations can be generated that reflect the places where the hand over operation can be executed. Using a PRM method, graph nodes (robot configurations) are found in the learning phase

2.5 Discussion

As we are interested in the interaction problem between two entities, robot-robot or human-robot pairs, we have presented a modular task planning architecture to solve two essential manipulation tasks for interaction through manipulation where the delivery tasks mainly use pick and place operations.

The architecture modularity allows a more flexible plan of actions, different control processes can be implemented. Specially designed for object manipulation, this can be extended to consider other types of tasks where geometric and kinematic constraints play an important role, for instance object pushing and assembly tasks could be considered. Even if we have not implemented all the modules and we do not consider all the possibilities, we can see that the approach works to automatically plan actions and execute them for manipulation tasks.

It is possible to extend and add new functionalities to the architecture and in that way elaborate more complete and robust plans. For autonomous robots the integration of vision, tactile and force sensors, and motion planning algorithms are necessary to act in human environments.

The approach is not adequate if we search to implement a general task planner for an autonomous robot, different techniques are needed to actually compute a plan of actions given an abstract description (symbolic manner) of the task mission [Fikes 71] and execute these actions considering temporal and resource constraints [Lemai 04b] [Lemai 04a]. However, the selected choice allows us to test our different grasp planners when used to solve delivery manipulation tasks.

Chapter 3

Random Grasp Planner

Grasping an arbitrary object is one of the fundamental tasks that a robot must be able to accomplish. Almost all manipulation tasks begin by a grasp : pick and place, filling a glass, bringing back an object and giving it to a human, using a tool, turning a crank handle... Polyhedral models of objects and environment can be now obtained [Badcock 94] and geometric algorithms for planning path are efficient [Siméon 01]. The important point to solve is the automatic grasp of objects. In this work we discuss a general method to compute a good grasp position given an object and an end-effector, and we present simulation results for some objects computed by our grasp planner.

The quality of a grasp is difficult to define because the forces that the grasp has to resist are mainly unknown. The force-closure concept is a necessary condition for the grasp planning in order to indicate that a grasp can resist up to any perturbation force. Although a grip tool does not define force-closure when the load is known, for example, a crane holding an object through a handle. The comparison of two grasps must consider not only the possibility to resist perturbation with the minimal contact force, but also the easiness to perform the grasp while avoiding collisions and bad positioning of contact points. General criterion influences the ability to accomplish the task like the accessibility to the active zone of a tool or the avoidance of collision for placing. As we have seen in chapter 1, there are several grasp synthesis algorithms to compute force-closure grasps [Shimoga 96] [Ponce 93] [Ding 00b]. Often they have prohibitive computational complexities to be implemented in experimental robots. An alternative is the use of heuristics to improve planning time and to circumvent the complex problem.

3.1 End-Effector Grasp Planning Approach

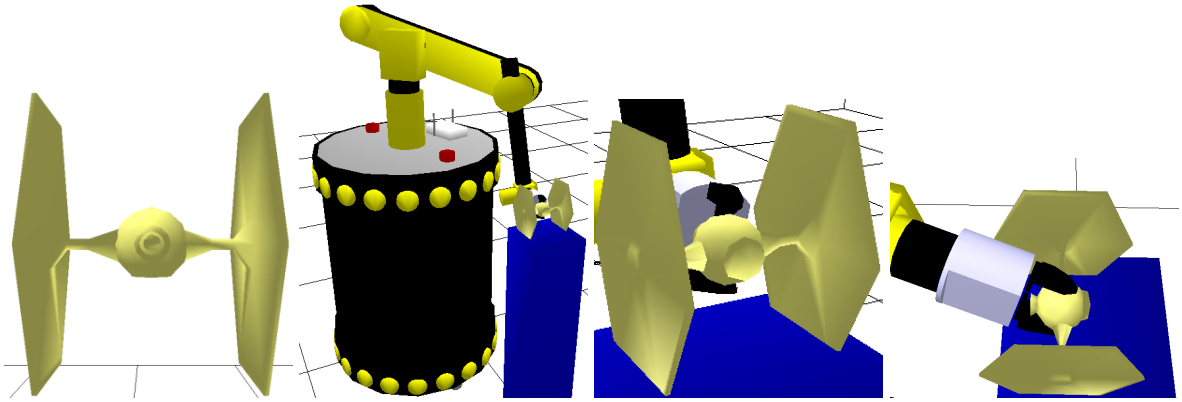
Most of the force-closure grasp planners do not consider the final tool to perform the grasp. From a theoretical point of view, they are capable to find a grasp for an object, even an optimal one ; unfortunately we cannot apply and integrate those planners in a simple way for an autonomous robot. A second approach appears, we called it the end-effector grasp planning. This approach is based on the random grasp generation presented in chapter 1. The grasp is planned to take the geometric and kinematics of robot's end-effector as input. Since the final intention is to integrate the grasp planner as a module of an autonomous robot, we must be capable to plan grasps on-line. Consequently, the random approach gives a good research direction for the grasp planning problem.

3.2 Random Grasps based on Inertial Properties

Instead of generating arbitrary points in any direction and orientation, we propose a different strategy for this purpose. The object is mainly subject to external force of gravity and acceleration that acts on the

mass center. It is a good idea to grasp the object around this point. The forces and torques that we have to apply to the object for keeping it stable will be lower if our grasp is closer to the mass center. This will allow us to obtain a set of grasps with a good quality and it would not be necessary to produce a big number of grasps. Within the same idea, good directions candidates for grasps are given by the principal inertial directions of an object [Lopez-Damian 05a]. In Fig. 3.1 we can see the output of the inertial axes random approach for a non-trivial model. We need to describe the unknown object with a model, then to represent a wide range of objects we use triangular meshes.

In the following section we describe the grasp planner implemented in this work. When a valid grasp is planned, the whole robot can reach and grasp the object. It doesn't matter where the robot is located. This allows the grasp planner to be used in a simple way with a path planner to go from the initial position of the robot to the computed grasp position. The grasp planning problem becomes as defined in [Laugier 85] the process to compute the object grasp configuration and the robot approach motion.



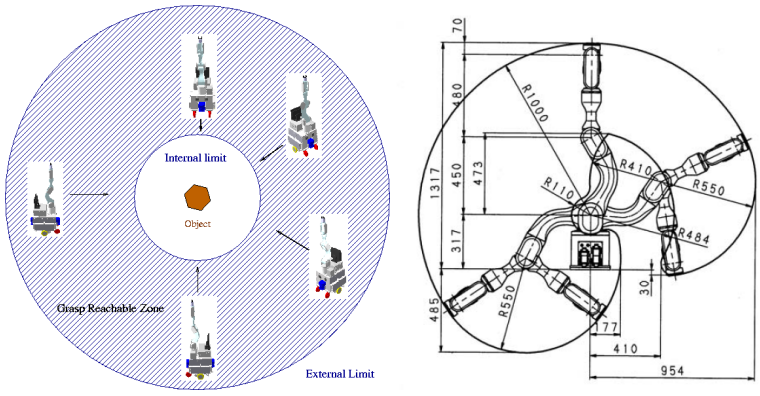
(a) (b) (c) (d)

Figure 3.1 – In a) Scaled spaceship model, b) Grasp found for the planner c) Grasp zoom and d) Grasp zoom seen from above. The grasp is found near the mass center as it is the most logical point to see how forces act on the object

3.2.1 Grasp Planner Algorithm

Here we introduce the basic algorithm for a random generator of grasps for polyhedral objects. We define a grasp as the contact points on the object surface and the frame associated to them, which we have called the grasp frame.

If we have a mobile manipulator and its start configuration is far from the object, the robot cannot be able to grasp it, the mobile platform has to be placed near the object. To place the robot, a random configuration is generated. As we know the location of the object, meaning, its mass center, it is used as the center of a grasp reachable zone (GRZ) given by the workspace of the manipulator. Minimal-maximal limits are used to define this zone considering the joint range of the manipulator which defines the arm workspace (AWS). If the manipulator is placed inside the zone, it will reach the object, see Fig. 3.2. A distance is computed from the arm base to the object. If this distance falls inside the GRZ then the configuration is accepted ; otherwise the process continues until a limit of the number of generated positions is reached.



(a) (b)

Figure 3.2 – a) Mobile platform positioning, b) Joint Range describing AWS

Random generation of platform orientations can produce a great number of configurations that do not allow to reach the object; this is generally due to joint limits for the arm. Then it is a good idea to place the front platform facing the object or the side platform towards the object, or any configuration in between. From this initial configuration the grasp generation begins.

Algorithm 7: Grasp Planner Algorithm Framework

```
input  : Geometric Model : robot R, gripper Grp, the environment E and the object O
output : Grasp G and robot configuration q_G
begin
    MOBILE_PLATFORM_POSITION_GENERATION;
    repeat
        RANDOM_GENERATION;
        if G Satisfy FILTERS then
            ASSIGN_QUALITY_VALUE G ← Q;
            ADDNEWGRASP List_G ← G;
        end
    until Grasps == PREDEFINED_NUMBER;
    G ←CHOOSE_GRASP_BEST_QUALITYVALUE;
    RETURN(G, q_G);
end
```

A) Random Generation Step : Generation of contact points and grasp frame.

B) Filter Step : Since quality determination is computationally expensive, we introduced the filter step to reject as soon as possible unfeasible grasps. Some constraints are imposed by the system itself, the grasp must be kinematically reachable by the whole robot and free of collision with the possible obstacles in the environment. To guarantee that the object is firmly held with no slippage, we use a force-closure test. The collision and force-closure filters are explained in the next sections.

C) Quality Measure Step : Several grasps can be produced after the first two steps are executed. The final step is the assignment of a quality measure to the grasps. Various measures have been

proposed based in wrench space. This shows how efficient the grasp is and allows it to rank the grasp.

In the sequel of the section, we present the generation of grasps for an end-effector composed of two-fingers with three-contacts. We first recall how the axes of inertia and the mass center can be computed.

3.2.2 Axes of Inertia

The calculation of the axes of inertia and mass center is taken from a 3D mesh model of the object that can be obtained from stereo vision or laser range sensors. These parameters depend only on object geometry with a constant density ρ. The object model is composed of facets and vertices. The location of the mass center and the inertia tensor can be computed by the conversion of the integrals of mass into volume integrals. We suppose the polyhedron (P) has a mass m and a uniform density ρ, we can relate the volume as $m = \rho V$, we compute

$$V = \int_P dV \tag{3.1}$$

The volume integrals can be reduced into surface integrals by the divergence theorem.

$$\int_V \nabla \cdot F dV = \int_S F \cdot N dS \tag{3.2}$$

for a vector field F defined on V. Where V is the region bounded by the surface S (union of triangular faces) and N is the vector of the exterior unit normal of V along its boundary. We use the algorithm developed by Mirtich [Mirtich 96]. The complexity of the algorithm is linear depending on the number of object faces. The inertia tensor T is composed by the moments and products of inertia around the mass center (CM).

$$\mathbf{T} = \begin{bmatrix} Ixx & -Ixy & -Ixz \\ -Iyx & Iyy & -Iyz \\ -Izx & -Izy & Izz \end{bmatrix} \qquad \mathbf{CM} = \begin{bmatrix} CM_x \\ CM_y \\ CM_z \end{bmatrix} \tag{3.3}$$

$$CM = \frac{1}{m}\rho[\int_P x dV,\ \int_P y dV,\ \int_P x dV]^T \tag{3.4}$$

$$\begin{array}{ll} I_{xx} = \rho \int_P (y^2 + z^2) dV - m(CM_y^2 + CM_z^2) & I_{xy} = I_{yx}\rho \int_P (xy) dV - m(CM_x CM_y) \\ I_{yy} = \rho \int_P (z^2 + x^2) dV - m(CM_z^2 + CM_x^2) & I_{yz} = I_{zy}\rho \int_P (yz) dV - m(CM_y CM_z) \\ I_{zz} = \rho \int_P (x^2 + y^2) dV - m(CM_x^2 + CM_y^2) & I_{xz} = I_{zx}\rho \int_P (zx) dV - m(CM_z CM_x) \end{array} \tag{3.5}$$

The principal axes of inertia are the principal directions of T.

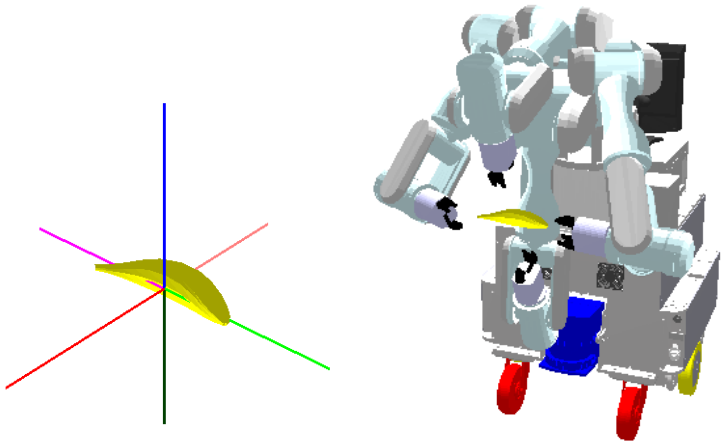
3.2.3 Grasp Generation Algorithm

Following the basic algorithm, the goal of the grasp generation is to produce a set of contact points on the object surface and the grasp frame. We have decomposed the whole algorithm into two algorithms for better understanding. The first one finds the inertial axes and the mass center from the object giving directions to search grasps. The second algorithm computes a set of contact points from the grasp frame.

Algorithm 8: Grasp Generation

```
input  : Geometric Model : gripper Grp, and the object O
output : Grasp G
begin
    COMPUTE_SIX_INERTIAL_AXES;
    COMPUTE_MASS_CENTER;
    COMPUTE_GRASP_FRAME G_F;
    forall Inertial Axes do
        COMPUTE_SEVERAL_ORIENTATIONS(G_F);
        COMPUTE_SEVERAL_DISPLACEMENTS(G_F);
        [CP, G_Forig] ←COMPUTE_CONTACT_POINTS(G_F, Grp);
        G ← [CP, G_F];
    end
end
```

Compute the mass center and the main axes of inertia (*AOI*), Fig. 3.3. The initial grasp frame is given by the axes of inertia and the mass center.
For each axis of inertia compute several orientations of the grasp frame by a small angle θ about the X and Y axes of the frame, this allows us to find grasps in case the gripper is in collision with the object. Compute several grasp frame positions along the axes of inertia when it is not possible to obtain a grasp around the mass center because an obstacle is placed near the object and the gripper or the robot arm is in collision with the obstacle. Compute contact points for each position and orientation of the grasp frame.



(a) Inertial Axes

(b) Grasps in direction of some inertial axes

Figure 3.3 – Six inertial axes and mass center computed for an object and four representative grasps at the inertial axes directions.

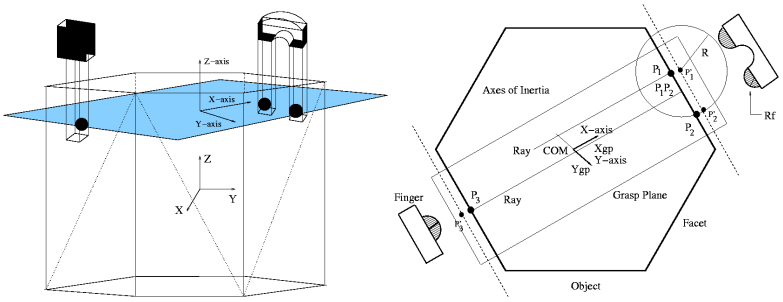
In algorithm 9 we present the generation of contact points. The generator considers the structure of the gripper. The current stage has to be implemented manually for each gripper, here we present the algorithms for two different end-effectors. Algorithm 9 is for a gripper with two finger and three fingertips,

algorithm 10 is for a three fingered hand called Barrett hand. General solutions can be proposed like generate a closing path for the fingers until a collision with the object is detected. To find the contact points, the intersection of polyhedra and the distance of penetration of the fingers must be computed. Unfortunately these kind of solutions are computationally too expensive to be used extensively.

Algorithm 9: Contact Points for two parallel gripper with three contact points

input : Geometric Model : gripper Grp, the object O and G_F
output : Contact Points CP and new G_F origin
begin
G_P =COMPUTE_GRASP_PLANE(G_F);
P_1 =COMPUTE_FIRST_CONTACT_POINT(G_F);
P_2 =COMPUTE_SECOND_CONTACT_POINT(G_F, P_1);
G_{Fr} =RELOCATE_GRASP_FRAME(G_F, P_1, P_2);
P_3 =COMPUTE_THIRD_CONTACT_POINT(G_{Fr}, P_1, P_2);
G_{Forig} =COMPUTE_NEWORIGIN_GRASP_FRAME(G_{Fr}, P_1, P_2, P_3);
$CP \leftarrow [P_1, P_2, P_3]$;
end

A grasp plane *Gp* is formed with the X-axis and Y-axis of the grasp frame, Z-axis is one of the main axis of inertia (see Fig.3.4).
To find the first point (P_1) of contact, we draw up a ray in the direction of the X-axis of the grasp plane. We find the intersection point (P_1) between the ray and the object. As the fingers have spherical fingertips, we define P_1' as the displacement of P_1 in the direction of the object surface normal by the radius (Rf) of the fingertip.



(a) Grasp 3D view

(b) Grasp 2D view with intersection rays

Figure 3.4 – Contact points generation from inertial axis

For the second point of contact (P_2), we find the intersection between the plane Gp and the planes formed by the facets of the object displaced by a distance Rf in the facet normal direction. The line that results of the intersection of planes must intersect a circle with center at P_1' of radius (*R*), where *R* is equal to the distance between the two-contact fingers of the gripper. The intersection point is P_2'. P_2 is

the contact point of the object that satisfies

$$\overrightarrow{P_1P_2} \cdot Y_{gp} > 0 \tag{3.6}$$

Relocation of the grasp frame according to P_1 and P_2

$$Y_{gp} = \frac{P_1' - P_2'}{\|P_1' - P_2'\|} \tag{3.7}$$

$$X_{gp} = Y_{gp} \times Z_{gp} \tag{3.8}$$

We emit a second ray with the same direction of axis X_{gp} of grasp plane from the middle point of $P_1'P_2'$. P_3' is the intersection between the last ray and the object facets displaced by Rf. P_3 is the associated contact point.
The new origin of the grasp frame is the middle point of the line formed between P_3' and the middle point of $P_1'P_2'$. This point will be near of the initial point.

3.3 Multifinger Hand

We have mentioned in the last section that for each end-effector we need a module for the generation of a candidate grasp. In this section we present the generation of grasps for a three-fingered hand.

The manipulation robot hand for our simulations is the Barrett Hand. This hand is composed by three fingers, each one with two joints. One finger is fixed and the other two can spread synchronously from 0 to 180 degrees around the palm. This mechanism is controlled by four motors which implies that each finger has one actuated inner link and a coupled outer link that moves with a ratio τ regarding the inner link. The fourth motor controls the motion of the two fingers from the palm.

As for the two fingers case, the generation step takes as input the geometric model of the object, environment and robot. At the end we produce a set of candidate grasps on the object. Following the algorithm framework 7, the grasp planner produces a set of contact points on the object surface and the grasp frame.

In literature, we found other grasp planners that produce a set of grasps using heuristics and then they try to solve the inverse kinematic problem to place the fingers on the contact points [Borst 02]. In practice this may not be the best option because it is not evident how to place the robot arm and particularly the end-effector to effectively make the contacts, the inverse kinematics of the system could be difficult to solve. In our case we have chosen to consider the geometry and kinematics of the hand in the generation process.

3.3.1 Generator Grasp Algorithm

As two of the fingers of the Barrett hand have a motion concerning the palm of the hand, we generate several start configuration grasps only moving these fingers. Then we have to generate a grasp frame and find the contact points on the object surface.

Grasp Frame (G_F) : The location of the G_F is given by the mass center the first time, then we generate a series of locations along the axes of inertia, this might be useful when we cannot find a grasp about the mass center because of an obstacle. We have six approach directions from the hand to the object being the Z-axis of the G_F an inertial axis. The X-axis and Y-axis of the G_F are given by the other two inertial axes that form a grasp plane. Other grasp frames orientations can be generated if we rotate about the axes of inertia.

Positioning of the hand : We define a hand frame H_F centered on the palm and with a height equal to the half of the fingers dimension in the Z-axis direction of the H_F. The Y-axis is parallel to the fingers and the X-axis is perpendicular to them. Finally the Z-axis is pointing out of the hand. After computing the grasp frame, we place the frame of the hand in the location of the grasp frame in a coincident way using the inverse kinematics of the arm, once the multi-finger hand is in place, we proceed to find the contact points.

Contact Points : Algorithm 10, presents the steps to find the contact points for a multi-fingered hand. We reduce the problem to the intersection of the fingers with the object surface considering each finger like a planar robot manipulator with two linked joints.

Algorithm 10: Multi-Finger Contact Points

input : Geometric Model : gripper Grp, the object O and G_F
output : Contact Points CP
begin
  forall *Fingers* i **do**
    $Cr_F = \text{GENERATE\_CIRCLE}(R_f)$;
    $P_n = \text{COMPUTE\_FACET\_PLANE}(O, Facet_n)$;
    $L = \text{COMPUTE\_INTERSECTION}(Cr_F, P_n)$;
    if $(\exists L)$ **then**
      $Point_1 = \text{COMPUTE\_INTERSECTION\_POINT}(L, Facet_n)$;
      $Cr_f = \text{GENERATE\_CIRCLE}(r_f)$;
      $l = \text{COMPUTE\_INTERSECTION}(Cr_f, P_n)$;
      $Point_2 = \text{COMPUTE\_INTERSECTION\_POINT}(l, Facet_n)$;
    end
    $\theta_1 = \text{SOLVE\_EQUATION\_FINGER}(Grp, Point_1, Point_2)$;
    $C_i = \text{COMPUTE\_CONTACT\_POINT}(\theta_1)$;
  end
  $CP \leftarrow C_i$;
end

We generate two finger trajectories described by circles. The first trajectory with radius R_f corresponds to the case where the finger is in its maximal extension and we compute the intersection between the circle and the plane defined by the object facet. With the resulting line, we compute now the intersection between this and the triangular facet that gives us one intersection point P_1.

Next, we generate a second trajectory with radius r_f, that is the minimal distance when the finger joints take their maximal values, the finger is at its minimal extension. We find a second intersection point P_2. As we can see, both intersection points do not correspond to the real contact point, the actual point is found with the line formed by P_1 and P_2 that is located on the object surface and the geometric relations between the links of the finger, see Fig. 3.5. We assume that both intersection points P_1 and P_2 are lying on the same facet.

Intersection point : the Barrett hand has three independent fingers, each finger has a motored inner link and a coupled outer one $\theta_2 = \tau\theta_1$. The position of the fingertip for each finger is given by

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} cos\theta_1 \\ sin\theta_1 \end{bmatrix} L_1 + \begin{bmatrix} cos((1+\tau)\theta_2 \\ sin((1+\tau)\theta_2 \end{bmatrix} L_2 \tag{3.9}$$

We have to consider the initial position of the finger given by θ_0. Finally taking the equation of the line $P = P_1 + \mu(P_2 - P_1)$ in a matrix form we have :

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} A \\ C \end{bmatrix} + \mu \begin{bmatrix} B \\ D \end{bmatrix} \tag{3.10}$$

We make some operations and we find an equation with one unknown. We solve this equation with a numerical method.

$$\begin{aligned} &D(L_1 \cos\theta_1 + L_2 \cos(\theta_0 + (1+\tau)\theta_1) - A) - \\ &- B(L_1 \sin\theta_1 + L_2 \sin(\theta_0 + (1+\tau)\theta_1) - C) = 0 \end{aligned} \tag{3.11}$$

For the solution of θ_1, we use the equations above and we find the coordinates for the contact point. We do the same for the others fingers. We can see in Fig. 3.6 the result of the grasp planner for a simple object.

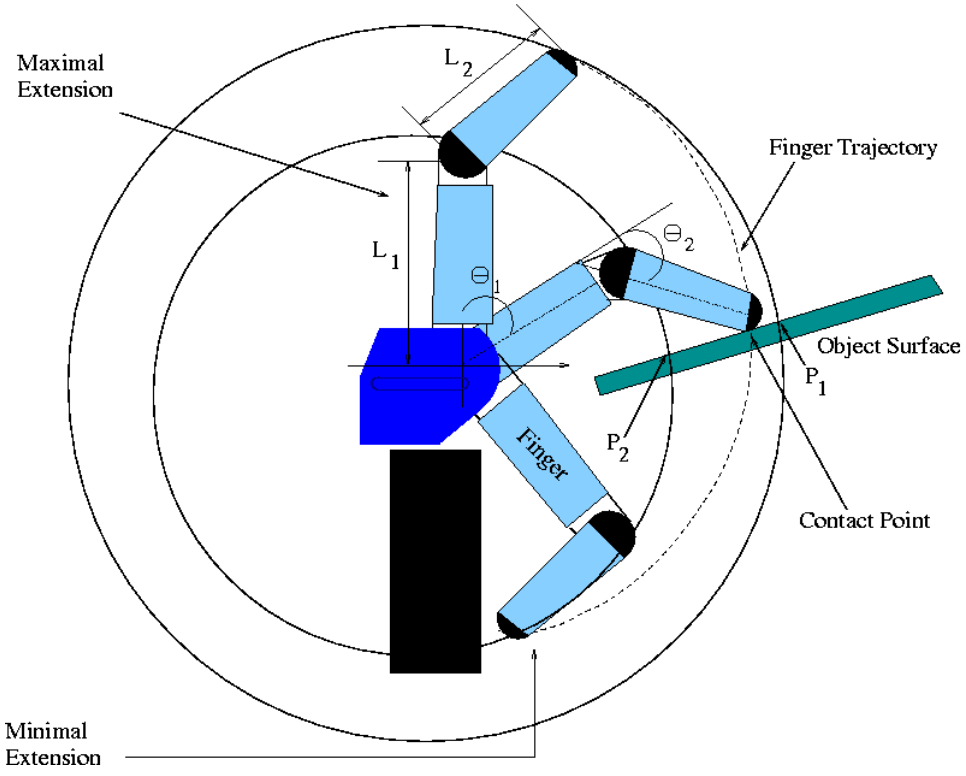
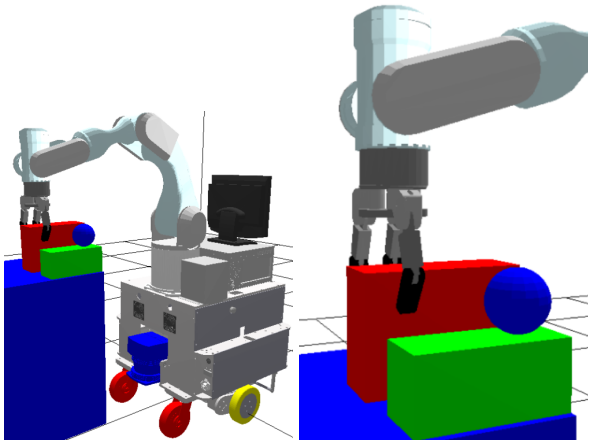


Figure 3.5 – Taking maximal-minimal configuration of finger, two circular trajectories are generated to compute the contact point when real finger trajectory intersects the object surface



(a) Grasp for a cube (b) Zoom

Figure 3.6 – Generation of grasps for an object with obstacles.

3.4 Grasp Filters

It can be seen if we make a simple experiment that humans can grasp an object in different manners for the same task, it is logic then to think that for a robot we are in the same case although maybe with a smaller number of grasps. The planner generator gives us as output a set of grasps on the object, as we are searching for force-closure grasps, we need to test which grasps in the set satisfies this condition in a faster way to eliminate them as soon as possible. The first filter then is the force-closure filter for a three contact point grasp. Also we search that the grasp is collision free with the environment and the object, a collision detector is used for this purpose, we recall here that the grasp is considered with the positioning of the whole robot.

3.4.1 Force-Closure Filter

One of the most important properties for a grasp is the notion of force-closure. We already saw that several works have been made in the analysis and synthesis of force-closure. Nguyen [Nguyen 88] proposes an algorithm for constructing 2D force-closure grasps based on the geometry of the object, Ponce [Ponce 95] computes grasps of polygonal objects using a projection algorithm based on linear programming. Considering only fingers, a basic grasp could be defined geometrically by the position C_i of d hard fingers or contact points on the object surface, with $i = 1, ..., d$. Hard finger contact model and Coulomb friction are assumed between the object and the fingers. Each finger exerts in C_i a force f_i and a moment $C_i \times f_i$ regarding some point on the object, the mass center in our case. Force and moments are combined and form a six-dimensional vector called wrench $w_i = [f_i, C_i \times f_i]^T$. A grasp achieves equilibrium when the sum of the wrenches is zero $\sum_{i=1}^{d} w_i = 0$. A grasp is force-closure if it can balance any external forces and moments (w) exerted on the object. The forces applied by the finger f_i must remain in the friction cone to avoid the slippage, see Fig. 3.7. The grasp is then force-closure if and only if there is a force in each friction cone so that the sum of the corresponding wrenches is zero. For a three-finger grasp, a necessary condition for force-closure is the existence of a point in the intersection of the plane formed by the three contact points with the friction cones at these contact points [Ponce 93] [Li 03], as in Fig. 3.7.

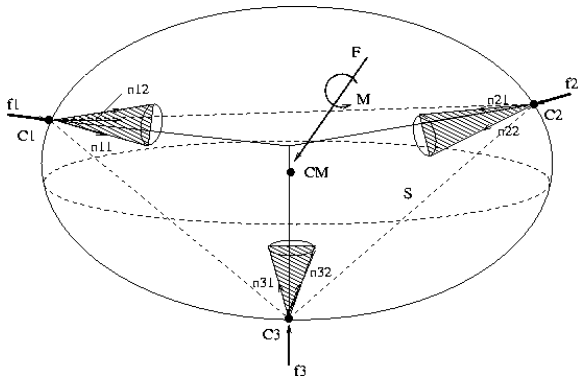


Figure 3.7 – 3D Grasp with three contact points can be seen as a plane grasp considering the sections of friction cones by the plane containing the three points.

Computing 3D force-closure grasps has been treated in [Sudsang 95]. They found that the four fingers grasps fall into three categories and develop new necessary and sufficient conditions for it. However, the algorithm that we implement is based on the work done by Li [Li 03] due to its simplicity and its small computing time. He presents new conditions for computing the 3D force-closure grasps in a geometric way for a robot hand with three hard fingers and contact point with friction. The case of four fingertips is unfortunately more complicated.

3D Grasps Force-Closure Algorithm

The algorithm transforms the 3D force-closure grasp problem to a two-dimensional. A 3-dimensional three-finger grasp is force-closure if two conditions are fulfilled :
1) There is a contact plane *Sp* and contact unit vectors $n_{i,1}$ and $n_{i,2}$, with $1 < i < 3$. Three contact points define a plane *Sp* if they do not lie on the same line. The intersection of the friction cones with the plane *Sp* can be in three ways : at a point, on a line, or on a plane, see Fig. 3.8. In the last case as the apex of the friction cones lies on the contact plane, the intersection is delimited by two lines defined by a pair of contact unit vectors $n_{i,1}$ and $n_{i,2}$.
2) The contact unit vectors form a 2D force-closure grasp in the contact plane.

2D Grasps Force-Closure Algorithm

The algorithm considers a hard finger model, contact points (C_1, C_2 and C_3) and the normal ones at this contact points (pointing inside of the object). The friction cones are bounded in pairs by the unit vectors $n_{i,1}$ and $n_{i,2}$. External forces and torques act in a point of the object, frequently on the mass center.
Proposition : Three non parallel contact forces in the friction cones different to zero achieve equilibrium if they positively span the plane and its lines of action intersect at some point. Li [Li 03] proposes the substitution of the unknown forces in the proposition by the boundary vectors of the friction cones and states a new proposition to compute equilibrium grasps. The algorithm starts with the elimination of the regions in the friction cone that do not contribute to equilibrium, this is called disposition H.

In order to achieve a moment equilibrium for a grasp with three-fingers and forces f_i and f_j applied

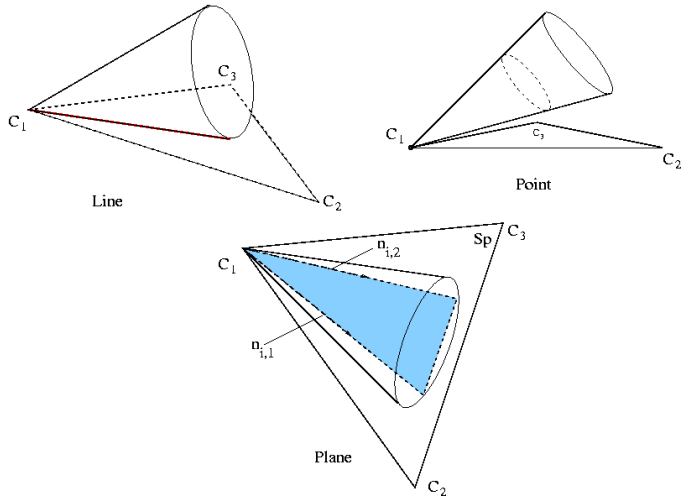


Figure 3.8 – Intersection between friction cones and contact plane

in two contact points, the next condition must be satisfied :

$$\overrightarrow{C_kC_i} \times f_i + \overrightarrow{C_kC_j} \times f_j = 0 \tag{3.12}$$

The friction cone at contact point C_k, can be divided into two regions by a line to the other contact points, for example $\overrightarrow{C_kC_i}$ and $n_{i,1}$ or $n_{i,2}$. There may be one region r_k that does not contribute to satisfy equation 3.12 if the force line lies in this region r_k. We can substitute $n_{i,1}$ or $n_{i,2}$ for $\frac{\overrightarrow{CiCk}}{|C_iC_k|}$ and we do not change the result of equilibrium. This can be computed as shown in the flow diagram. A new proposition states that the three finger grasp is in equilibrium if the intersection of the three friction cones is not empty after the operation of disposition H, see Fig. 3.10.

The next step is the calculation of intersection points by the two boundary lines of each friction cone. The grasp will be force-closure if at least there is one point due to the intersection of two different boundary lines of friction cones that are inside of the third friction cone, such points satisfy the equation

$$(\overrightarrow{C_iB_{jk}} \times n_{i1}) \cdot (\overrightarrow{C_iB_{jk}} \times n_{i2}) < 0 \tag{3.13}$$

B_{jk} is one of the intersection points by the boundary lines of friction cones C_jC_k. The complexity of the algorithm is minimum, only a few operations are needed [Li 03]. When more than three fingers are used there are algorithms in literature, some of them were mentioned in chapter 1, but none of them uses only a few operations, an extension for n-fingers seems a hard task.

3.4.2 Collision Detection Filter

The second type of filter that we use is the collision checker. We verify that there is no intersection between the geometric models of the robot, gripper, object and the obstacles. We use the collision checker [van Geem 01] implemented in the motion planning tool Move3D [Siméon 01]. When a grasp is found, we place the gripper by calculating the inverse kinematics of the arm and we test that the execution is free of collision of any kind. Considering the whole robot and the environment we avoid the risk of collision between arm-object, arm-obstacles and gripper-obstacles. This reduces the need of backtracking at high level task planning.

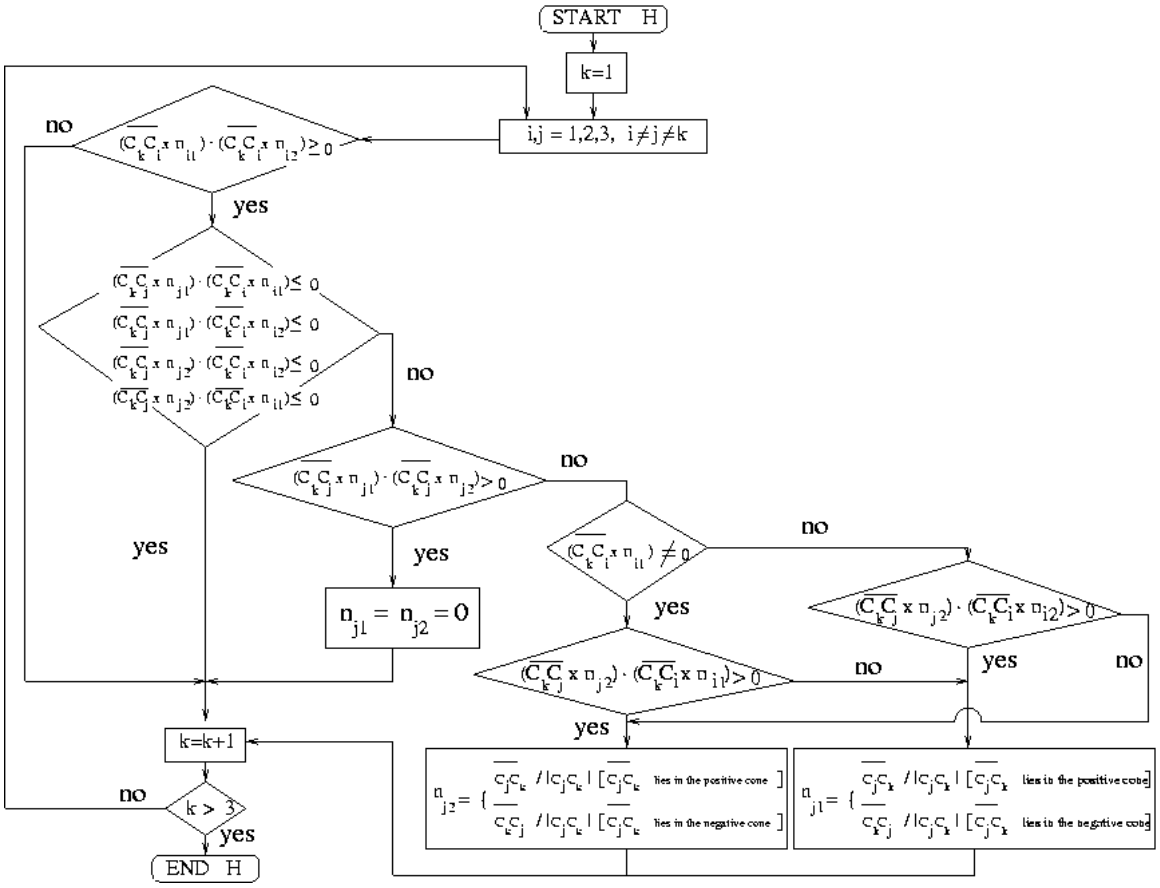


Figure 3.9 – Disposition H of friction cones, from [Li 03]

3.5 Quality Measure

Last section filtered the set of grasps produced by the planner to only have force-closure collision free grasps, but we need to choose one grasp among others, we need to rank them. A quality value reflects the efficiency of the grasp, then an index value is assigned to each grasp of the set. The grasp with the best quality value is chosen for the task. As we have already seen in chapter 1, the wrench space can be used to implement quantitative force-closure algorithms, here we use a simple one to perform the ranking. Quality of a grasp not only means force-closure and some other indices can be defined. In the second part of this section we assume that a grasp has a value assigned, if not, we assign the maximum value to the grasp and we penalized the grasp if it is near of the facet borders.

3.5.1 Quality based on Wrench Space

The planning of a good grasp is important when the robot has to take an object firmly. For this we use one of the quality criteria that has been developed in [Ferrari 91]. The criterion tries to quantify the notion of a good grasp for a force-closure grasp. The quality index indicates the unit strength grasp resistance for the worst case disturbance.

A hard finger contact model and Coulomb friction are assumed between the object and the fingertips. The forces applied by the finger f_i must remain in the friction cone to avoid slippage. We must approximate the friction cone to represent it by a finite set of m vectors. If we assume that each friction cone

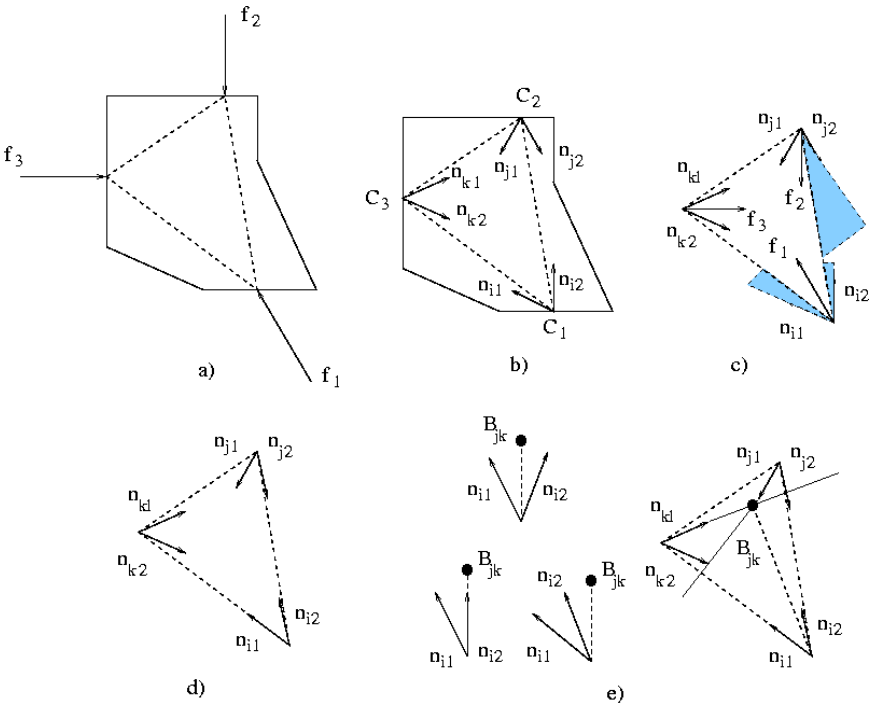


Figure 3.10 – a) Bidimensional object, three finger forces applied on the object and contact plane b) Contact plane and friction cones represented by two vectors c) Friction cones, forces and moment equilibrium. The shaded zones do not contribute to equilibrium. d) Disposition H e) Intersection points cases and one example

has unit height, then the convex hull corresponds to the L_1 metrics in this grasp wrench space described in Ferrari and Canny [Ferrari 91]. This space represents the space of wrenches that can be applied by the grasp considering that the sum of the magnitude of the normal forces applied by the gripper at the *n* contact points is 1, then f_i can be written as :

$$f_i = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_{i,j} f_{i,j} \tag{3.14}$$

with $\alpha_{i,j} \geq 0$. Similarly, the total wrench applied on the object is expressed by :

$$w = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_{i,j} w_{i,j} \tag{3.15}$$

and the set of all wrenches is :

$$W = CHULL(\bigcup_{i=1}^{n} w_{i,1}, ..., w_{i,m}) \tag{3.16}$$

The quality value for the grasp is equal to the distance of the closest facet of the Convex Hull from the origin. The wrench in this direction is the most difficult for the grasp to apply. We have to say that the friction coefficient affects this quality measure, here a constant and predefined value is used. It is important to note that the moment and force have different dimensions, a scale λ could be used obtaining $w = [f \quad \lambda(d \times f)]^T$. The moment multiplier λ is chosen to make $\lambda = \frac{1}{r}$, where r is the

maximum radius from the mass center, in this way the quality of the grasp is independent of the object scale [Pollard 94] [Miller 99].

In Fig. 3.11, we can see the grasp wrench space for two different grasps and their quality represented by the two circles for a 2D case. For the space case, the grasp quality is represented by a sphere.

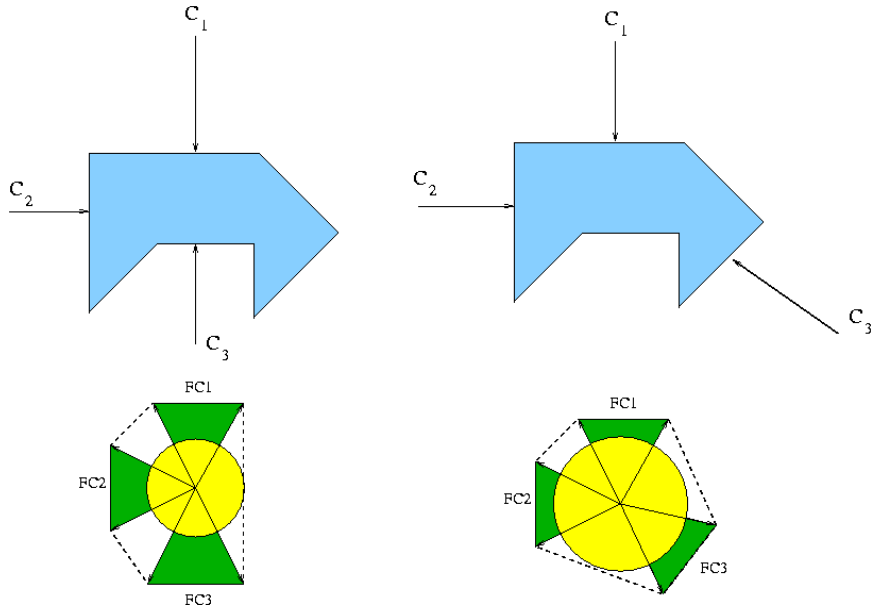


Figure 3.11 – Grasp Wrench Space GWS metric in 2D

3.5.2 Stable Grasps

It may happen that grasps may be generated near the borders of the object. Such grasps are not desirable because they can be unstable at the moment when the gripper grasps the object.

We define a stable grasp as the grasp where the contact points are away from the borders of the object. If the facets that describe the object are big enough, a grasp will be more stable if the contact points are in the middle of the facets.

This is why we have implemented a simple criterion of penalty as follows : we find the circle inscribed for the facet with radius R_c, not necessarily must be a circle but we use it for the simple operations we require to compute it. For a triangular facet, see Fig. 3.12, we know that from a triangle, three other inside triangles can be computed if we take the intersection point I between the lines traced from the vertices and these vertices.

The triangle area is given by $\Delta(ABC) = \Delta(ABI) + \Delta(CIA) + \Delta(BCI)$. We can express the triangle area with the radius of the circle and the semi perimeter of the triangle, after some operations we have that $R_c = \frac{\Delta(ABC)}{s}$, where $s = \frac{a+b+c}{2}$ is the semi perimeter and a, b and c the lengths of the triangle edges. Using the Heron formula finally we have :

$$\begin{aligned} \Delta &= \sqrt{s(s-a)(s-b)(s-c)} \\ R_c &= \sqrt{\frac{s(s-a)(s-b)(s-c)}{s}} \end{aligned} \tag{3.17}$$

From the center point of the inner circle I, we form another circle centered in the facet with radius $\frac{R_c}{2}$. In our case the facets are triangles but the method works for any convex facet. In this way the facet is divided into regions, we assign a penalty weight for each of these regions. The contact points contained in regions 2 and 3 are penalized with a high value resulting in a bad quality.

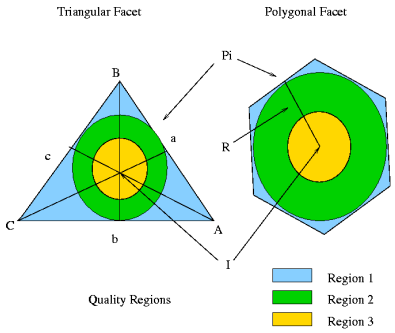


Figure 3.12 – Regions Quality Filter in Facet of contact

3.6 Fingertips Placement

Once a grasp was found and the robot positioned in the grasp configuration applying the inverse kinematics of the manipulator, the fingers have to reach the contact points ; simple closing fingers procedure is implemented for the gripper and the Barrett hand. From the start state position of the end-effector, usually an open state, an increment is added to each degree of freedom of the fingers until a collision with the object is detected. Depending on the increment step, the fingers can penetrate the object unless a very small increment is used, which is more costly that if it closes rapidly and take the fingers back to the object surface. We have at this moment two positions of the fingers, the open position out of collision and the close position in collision, so we define a displacement functions for the fingers.

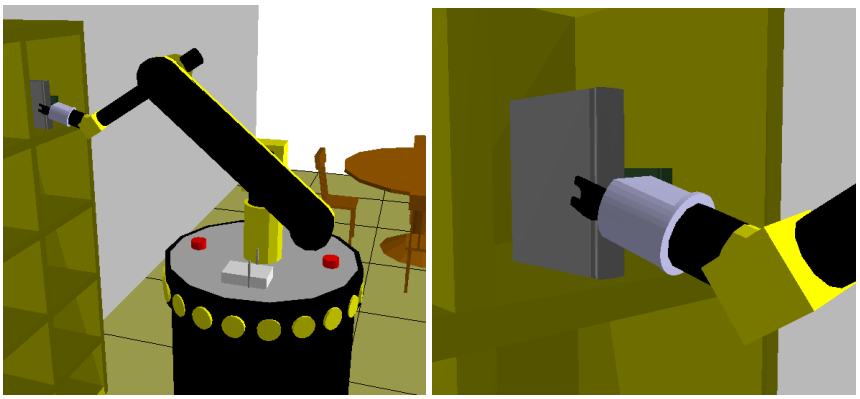
$$\theta_{Gi} = (1 - \mu_i)\theta_{Ci} + \mu_i\theta_{Oi},\ with\ \mu_i \in [0, 1] \tag{3.18}$$

The functions are the linear evolution of fingers positions from an closed state(θ_{Ci}) to a grasp state (θ_{Gi}). The fingers are continuously taken from the collision position $(\mu_i = 0)$ to the collision free open position $(\mu_i = 1)$ until it finds the collision free grasp position, finding the values of μ_i. These values are computed with the displacement functions using the output of the collision detector and varying the parameter μ_i.

3.7 Discussion

In this chapter we have described a grasp planner for 3D objects using a geometric model. We assume that the object has a uniform density but in some cases, this is not true, the grasp chosen may be inadequate. By measuring the position of the mass center, the weight and other physical properties of the object at the moment of the grasp, the grasp can be modified.

The geometrical shape of the object to grasp affects the number of force-closure grasps one can generate for the object and the complexity of the grasp planner depends on it, as well as the number of facets of the object.



(a) Book on a shelf (b) Book zoom

Figure 3.13 – The figure shows mobile robot manipulator in a common human environment, we show how the grasp planner can find a solution in a common task as picking an object, a book on a shelf

We can see that all the grasp planners tackle the problem of generating a set of contact points assuming that the object is small enough for the robot to grasp. We could make a partition of the object into smaller pieces using a convex decomposition process. Convex decomposition of three-dimensional polyhedra is done by iteratively removing the non-convexity of the polyhedron until all components are convex, for example a briefcase object can be decomposed into two subcomponents, the body and the handle, a feasible grasp can be computed easily in the handle part. Decomposition is the subject for the following chapter.

Other extensions that a grasp planner could consider are a series of constraints imposed by the task that the robot has to accomplish. These constraints will be reflected in the grasp selection. In this case the grasp with the best quality measure could not be the adequate one for an specific task. For example Jones and Lozano-Perez [Jones 90] consider constraints imposed by the classical pick and place problem to plan grasps.

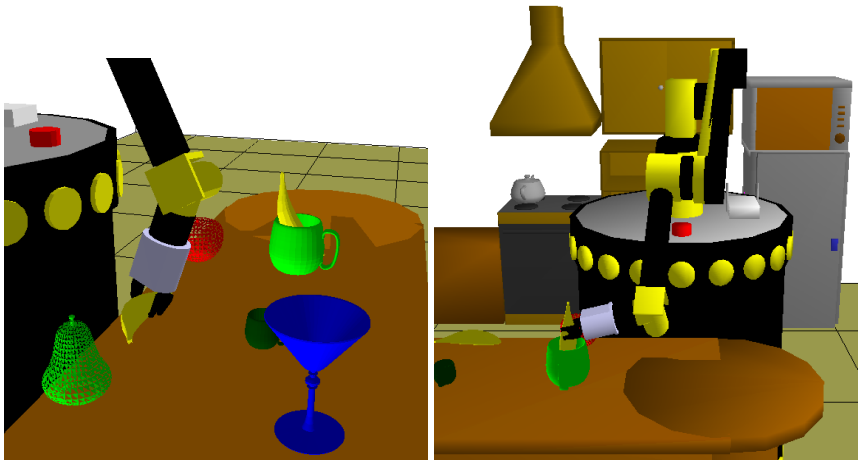
A third quality criterion can be implemented by taking into consideration the distance between the gripper and the environment. We can assign a lower quality value for the grasps that are located near the obstacles around the object.

Vision can be used to locate the object with a better precision and modify the robot trajectory if this location differs from that of the object model that was used for the motion planner.

3.7.1 Human Hand Grasp Generation

We can see that for a human hand or a five-fingered hand, the approach described for the Barrett hand could be used after some modifications. Being the first difference the number of fingers, this increases the computing complexity of the algorithms, this makes that some existing methods cannot be applied. We find some methods for four finger grasps [Sudsang 95] [Ponce 97] and how an n-finger grasp can be transformed to a force-closure grasp knowing at least m finger contact points and finding the position for the other fingers [Liu 00] [Ding 00b].

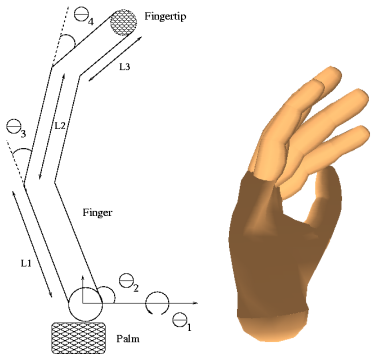
We do not know how to handle easily as the method presented in this chapter, but we can use the thumb, the index and the medium fingers to form a three force-closure grasps and the other two fingers



(a) Banana alone (b) Banana inside mug

Figure 3.14 – The figure shows a service robot in a kitchen and how the grasp planner can find a solution for an object in different situations, for example a banana on the table surrounded by obstacles and a banana inside a mug

will increase the stability of the grasp. The second difference is the number of links for each finger (see Fig. 3.15), considering the human hand we have three links with four joints instead of two as the Barrett hand, there is a dependency between the links caused by a tendon that passes through the finger. This dependency of the last two links can be approximated with a linear relationship given by $\theta_4 = \frac{2}{3}\theta_3$ [Rijpkema 91]. We simplify the human hand model considering a second dependency between θ_2 and θ_3 given by $\theta_3 = \tau\theta_2$.



(a) Finger (b) Hand

Figure 3.15 – Geometric model of a human finger and hand

We generate two trajectories for the finger, these trajectories are represented by circles as we did for the Barrett hand, one corresponding to the case when the finger is at its maximal extension and the radius

of the second circle is the length from the fingertip point to the origin point of the finger, where the finger is at its minimum extension, two intersection points P_1 and P_2 on the object surface will be generated.

A set of finger points are located between the previous two intersection points P_1 and P_2, given that we can generate a set of finger trajectories by setting τ with different values, contrary to the Barrett hand. For the values of θ_1, we apply the same strategy of the Barrett hand and we set from the beginning the value of this joint that has the function to spread the fingers.

At the end we can find an equation with only one unknown as the Barrett hand case.

$$\begin{aligned} D(L_1\cos\theta_2 + L_2\cos(\theta_{inner} + (1+\tau)\theta_2) + \\ + L_3\cos(\theta_{outer} + (1+\frac{5}{3}\tau)\theta_2)) - A) - \\ - B(L_1\sin\theta_2 + L_2\sin(\theta_{inner} + (1+\tau)\theta_2) + \\ + L_3\sin(\theta_{outer} + (1+\frac{5}{3}\tau)\theta_2)) - C) = 0 \end{aligned} \tag{3.19}$$

We consider the current position of the finger given by the inner link joint θ_{inner} and θ_{outer} for the outer joint link.

Chapter 4

Object Decomposition in Grasp Planning

The motivation of this thesis is to develop functions for autonomous robots capable to grasp different kind of objects, almost all practical grasping algorithms assume some knowledge of the object geometry. We are interested in completely unknown objects.

An easy and suitable representation for an arbitrary object is a triangular mesh as shown in Fig. 4.1. Such mesh can be seen as a polyhedral description given that in geometry, a polyhedron is simply a three-dimensional solid which consists of a collection of polygons, usually joined at their edges.

Sometimes the grasp planner does not find any valid grasp. Size and geometry complexity of the object are one of the most common reasons for the planner to fail. One possible solution is to decompose the object into several smaller parts. Robots could use this decomposition strategy in some situations, for example to open a door, the handle could be separated from the rest of the door, allowing the robot to grasp this door handle and push or pull the door to open it.

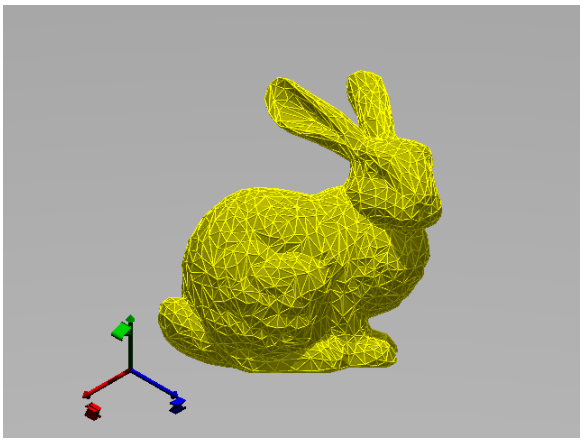
Decomposition of polyhedra is not a new problem; in the computational geometry community a solution is given for convex decomposition [Bajaj 84] [Chazelle 84]. In other communities it has been proved that decomposition is useful to solve difficult problems allowing more efficient algorithms, e.g. computation on volumetric properties, solution of partial differential equations, collision detection.

Similarly to find grasps for two robots in manipulation planning, decomposing the object into two parts and grasp each part with the end-effector can be a good solution.

The first part of the chapter presents convex and approximate convex decomposition approaches for polyhedral objects. The second part shows how the decomposition is used within our grasp planner. An extension of this planner is made to plan two grasps in the same object for interactive manipulation tasks.

4.1 Convex Decomposition of Polyhedra

The general problem of partitioning complex structures into simpler components was treated in computational geometry. One reason is that most geometric algorithms cannot be applied to non-convex structures in a simple manner. A common strategy to overcome this difficulty is to decompose the structures into convex components. We can say that the general problem is the partitioning of 3D polyhedra into a minimum number of convex pieces. The problem is known to be NP-hard. A first solution was proposed by Chazelle [Chazelle 84] finding a lower bound and worst case optimal algorithm.



(a) Rabbit polyhedron

Figure 4.1 – Polyhedron description with triangular surfaces

4.1.1 Notions

A series of notions had been developed in [Chazelle 84] [Bajaj 84], we use these concepts. The representation of polyhedra is made by the boundaries which consists of zero-dimensional faces called vertices, one-dimensional faces called edges, and two-dimensional faces called facets. We say that two facets are adjacent if and only if they share an edge.

A notch is a reflex edge of a polyhedron. This edge has an inner dihedral angle subtended by two incident facets that is greater than $180°$. When the polyhedron surface is a manifold, notches are reflex edges only. The polyhedron surface is a 2-manifold if each point on the surface has a neighborhood that is homeomorphic to an open 2D ball or half ball [Munkres 00]. Then we can say that polyhedra with 2-manifold surfaces are called manifold polyhedra, this notion was first defined in [Bajaj 84]. If a polyhedron has a number of holes n, it is called of $genus\ n$. A $shell$ is an internal void in the polyhedron.

4.1.2 Basic Decomposition

The basic method consists to resolve the several concave features in manifold polyhedra characterized by notches. The decomposition is then a set of cuts performed through the polyhedron to resolve the notches by a cutting plane C_p. It is not difficult to notice that an infinite number of C_p can be chosen. The notch is resolved if the cutting plane splits the reflex angle, and if both resulting angles as measured from the inner side of the facets are not reflex. Solving certain notches can generate subnotches. A subnotch is then a subsegment of a notch, Fig. 4.2. This can create an exponential number of parts, to avoid this, a plane is associated to each notch C_{p_i} that resolves the reflex edge, the subnotches generated will be resolved by taking the constraint that the cuts should be coplanar with the plan C_{p_i}.

The partition process considering only polyhedra of genus zero is as follows : the intersection of the cutting plane and the polyhedron produces a series of edges that forms a polygon or cross section on the cut plane surface S. The cut is then represented by the polygon P_c. The result of the process is the creation of two polyhedra. These polyhedra are generated from the data structure of the original polyhedron P.

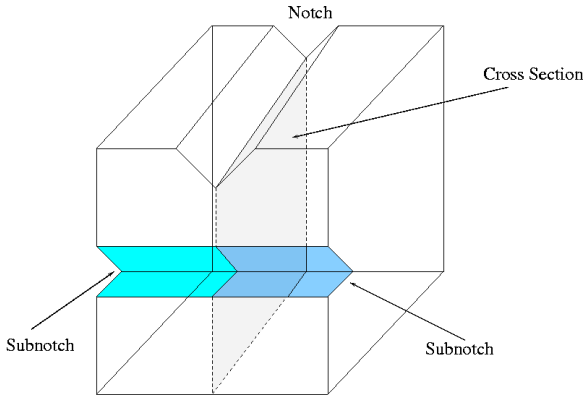


Figure 4.2 – Subnotches caused by two orthogonal notches

When the polyhedron presents holes, the intersection of C_p and P may generate a series of polygons. The consequence is that the algorithm described above does not work properly because resolving the notch, the polyhedron may not be split by simply cutting a handle of P and preserving its connectivity. The algorithm is then modified to support this situation.

To generalize the algorithm, firstly it is computed the intersection of each facet of P with C_p, producing a set of series of edges S. Then it is computed the exterior boundary (EB) of S, identifying the series of edges which contains the reflex edge. Taking EB, all the interior boundaries are computed (IB) forming $P_c = EB \cup IB$, now the cut includes a set of polygons. The cutting of each edge of P which intersects P_c is made by updating the adjacency lists. A depth-first search is executed in the lists to verify the connectivity of P. If the polyhedron is no longer connected, P is decomposed into two different pieces P_1 and P_2. If P is still connected, an update of the data structure lists is made and a recursive call of the algorithm is performed.

The algorithm produces a worst case optimal number $O(r^2)$ convex polyhedra, with a $O(nr^2(r + \log n))$ time and in $O(nr^2)$ space, where n is the number of polyhedron edges and r is the number of reflex edges, this algorithm computing time analysis is from [Chazelle 84].

4.1.3 Non-manifold Polyhedra Decomposition

This algorithm was proposed by Bajaj [Bajaj 84] and it is based on the recursive cutting method of Chazelle with an extension to handle non-manifold polyhedra when special notches are presented on the structure, see Fig. 4.3. The strategy followed is firstly to resolve these special notches to build only manifold polyhedra, subsequently a cutting and splitting method is applied.

As the precedent section a data structure DS is used to represent the polyhedra. In this case, the DS is composed of a list of vertices (coordinates and adjacencies), a list of edges (pointers to the corresponding vertices and pointers to orientation depending on the facet that belongs to), list of facets (facet equation and cycles of edges defining the boundary of the facet), all stored similarly to a star-edge representation. The process proceeds as in algorithm 11.

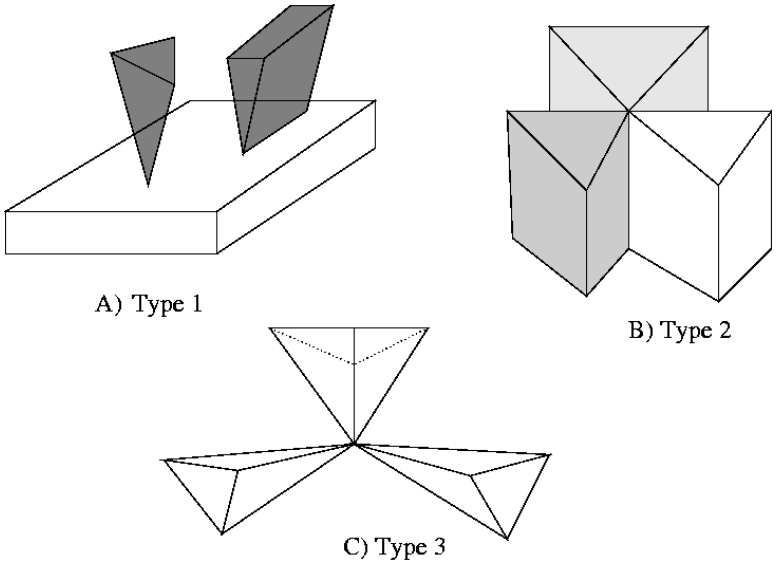


Figure 4.3 – Types of notches in non-manifold polyhedra

Algorithm 11: Non-manifold Decomposition Algorithm Framework

```
input  : Polyhedron P
output : polyhedra P_1,P_2,..,P_n
begin
    C_i ← REMOVE_SPECIAL_NOTCHES;
    forall Manifold Components C_i do
        while not StopCondition (notches N_i = ∅) do
            C_pi ←ASSIGN_CUT_PLANE for all N_i;
            subn_i ← FIND_ALLSUBNOTCHES of N_i;
            REMOVE_SUBNOTCHES(C_pi, N_i);
            c_i ← REMOVE_SPECIAL_NOTCHES;
        end
    end
    ASSIGN_COMPONENT_TO_POLYHEDRA_SET P_i ← c_i;
end
```

Given a polyhedron P, it is first partitioned by eliminating all special notches producing manifold components. For each component computed the notches or reflex edges. To each of these notches a cutting plane is assigned. Solving the notches of manifold components may produce subnotches or subpolyhedra with special notches. Proceeding as before, each subpolyhedra is split to generate manifold ones. A more detailed description of each step of the algorithm is given next. Based on Fig. 4.3, the first kind of special notch is treated by the algorithm as follows. The single vertex or single edge on the facet causing the non-convexity is detached from the rest of the polyhedron making a search of adjacency. This may take at most $O(n)$ time. For the second type of notch, where more than two facets share the same edge, the algorithm finds a plane normal to the mutual edge e_m and a circular sort of edges of the resulting intersection between the plane and the facets is done, then a facet pairing $(f_1, f_2), (f_3, f_4), .., (f_{k-1}, f_k)$ is carried on between the adjacent facets, an edge is created between each pair of facets and we eliminate e_m. Finally, grouping the facets that enclosed a volume makes the partition of the polyhedron resolving

the notch. The third type of special notch is resolved by grouping the edges and facets adjacent starting from the common vertex v, that cause the non-convexity, this set can be crossed without passing trough v, creating for each set the vertex v which separates the sets from each other. A combination of this three types of notches can be found, in this case all the notches of type one are resolved, then the notches of type three and finally those of type two.
Once there are no special notches, we continue to resolve reflex edges of manifold polyhedra by cutting planes. The intersection between a manifold polyhedron and a cutting plane is determined by taking the plane equation $Cp \rightarrow Ax+By+Cz+D = 0$ that defines two half-planes $HS_r = Ax+By+Cz+D \leq 0$ and $HS_l = Ax + By + Cz + D \geq 0$. To cut the polyhedron with the plane C_p, it is necessary to compute the intersection points, sorting these points and creating lines between adjacent points we can form polygons on both half-planes. One of these polygons contain the notch becoming the polygon of cut P_c at each half-plane. If the polyhedron is not separated into two pieces by the P_c, then the others polygons on the half-planes are taken to partition the polyhedron.

The algorithm runs in approximately $O(nr^2 + nr \log n + r^4)$ time and $O(nr + r^{\frac{5}{2}})$ space, computing times from [Bajaj 84].

4.2 Mesh Decomposition Approach

Based on that an arbitrary object can be represented by a mesh, the approach treats the problem of object decomposition in finding contours for cutting the mesh. Same as above approaches, the concave features are used to guide the search of these cutting contours CC or $cuts$ that follow minimum negative curvatures. An operator is defined called scissoring operator to split the mesh into two disjoints meshes along a closed contour crossing the mesh. Once a contour is found, a snake is created, this can adapt itself to the mesh shape in order to improve the CC. The scissoring operator performs three basic operations to finally split the mesh [Lee 04]. These three operations are the feature contour extraction, the loop completeness and the snake motion that are described next.

4.2.1 Feature Contour Extraction

The first step is to compute a contour feature not necessarily closed from where the algorithm begins to create the cut. In a mesh the concave features can be seen as negative curvatures. From this assumption, a curvature function is used to find the features. A minimum curvature value is assigned to each mesh vertex, this value is $C_f(v) = \frac{T(v)-\mu}{\sigma}$, where μ is the mean and σ the standard deviation, and $T(v)$ is the tensor field computation. A hysteresis threshold on the curvature values allows it to define triangulated areas with these vertices. From these areas, the skeletons can be computed unifying those whose directions are similar and endpoints are close, the contour is selected from the series of skeletons considering the length and its centricity.

4.2.2 Loop Completeness

Let CC be the contour chosen, almost always the contour is incomplete, it is not a closed boundary on the mesh surface. A crossing mesh operation to complete the contour is performed. The difficulty of completing an open contour to a closed one is to find an encircled path on the mesh that passes through mesh features as much as possible. A combination of three functions to search this path is defined. If a mesh vertex is taken, $C_d(v) = \sum_{v_i \in CC} \frac{1}{g(v,v_i)}$, where C_d is the sum of distances from the vertex to all vertices in CC, the value of this function will be higher in the vicinity of the contour CC.

The second function is given by C_n and its values depend on the facet normal, then for example it will be lower for the facets with normal with the opposite direction of CC. If n_{CC} is the center vector of

the normal cone of all vertex normal of $v_i \in CC$ and α is the angle of this cone taking n_v as the normal at vector v, the function becomes

$$C_n = \begin{cases} 1 & \text{if } n_c \cdot n_v \leq \cos(\alpha) \\ \frac{nc \cdot nv + 1}{\cos(\alpha)+1} & \text{otherwise} \end{cases}$$

The weighted elements of the third function are applied to the mesh edges and are composed by the two above functions, C_f is a function that represents the curvature of the mesh that was used for feature extraction and $l(e)$ is the length of the edge. These components are used as the average of the values at the two extreme vertices that form the edge. Then the edge cost is $f(e) = l(e) \cdot C_d(e)^{w1} \cdot C_n(e)^{w2} \cdot C_f(e)^{w3}$, evaluating the function tells the interest of taking the edge as a part of the contour.

4.2.3 Snake Motion

The snake is fit to the mesh and it defines the cutting contour or simply the cut. The snake is moved by minimizing an energy function E_S. This function is formed as $E_S = \int E_I(M) + E_O(M)dt$. The internal energy E_I is used to smooth its shape and shorten its length, the snake external energy E_O is used to capture nearby features. To constraint the snake on the mesh in the minimization process, a local parameterization that embeds faces around the snake into a 2D plane is used. The snake position is found by solving a set of linear equations. The algorithm seems to give good results when the contours are manually chosen, contrary to the automatic way that in some cases finds a non convenient cutting contour [Lee 04].

4.3 Decomposition based on Inertial Axes

A practical and fast solution to decompose an object to grasp is to partition it into some few components C_i, constructing a series of cutting planes passing through the axes of inertia C_{pi}. A sketch of the inertial axes decomposition algorithm IAD is presented next. As we have seen, a polyhedron P can be described by a set of vertices, edges and facets. It is important to say here that the facets we considered are triangles, affecting the split process because a triangulation has to be made.

Algorithm 12: IA Decomposition Algorithm Framework

```
input  : Polyhedron P
output : polyhedra P_1,P_2,..,P_n
begin
    IA_i ←COMPUTE_INERTIAL_AXES;
    forall Each Inertial Axis IA_i do
        C_pi ←COMPUTE_CUT_PLANE(IA_i) ;
        C_i ← SPLIT(P, IA_i);
    end
    ASSIGN_COMPONENT_TO_POLYHEDRA_SET P_i ← c_i;
end
```

The decomposition of the object is made by generating cut planes and by separating the object. Inertial axes are used to form cut planes. Each axis gives us the normal of one plane and the other axes generate the plane itself. The process of splitting is a basic one, finding all the facets and edges that intersect the cutting plane C_p. At each iteration a partition gives only two components that belong to the set C_i and then is used to form two new subpolyhedra P_1 and P_2. As a result, we assign each vertex

depending on which side of the cutting plane it lies. For the vertices that lie on C_p, they form a polygon, this one is triangulated and included in both polyhedra structures.

The algorithm can be used recursively, taking each component of the polyhedra set as an input ; a series of subpolyhedra will be generated. This approach does not guarantee convex components.

In Fig. 4.4(a) we can see the model of one object and its correspondent axes of inertia. A candidate plane is defined by the mass center and one axis of inertia. A cut plane example can be seen in Fig. 4.4(b).

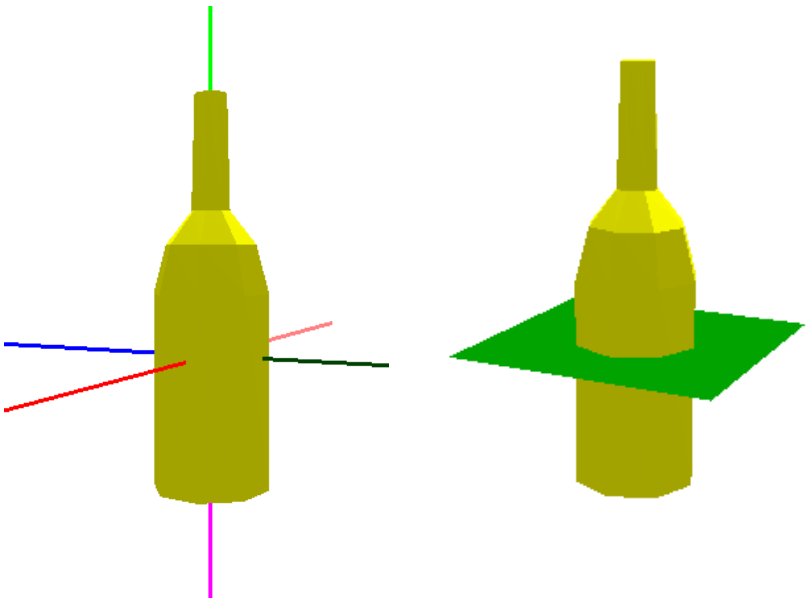


Figure 4.4 – a) Inertial Axes b) Cut plane taking one inertial axis for object decomposition

The decomposition of the object can be defined as :

$$DP(P_H) = \{C_i \subset P_H \; with \; \cup_{i=1}^{n} C_i = P_H\} \\ and \; C_i \cap C_j = \emptyset, \forall_{i,j} \; i \neq j \tag{4.1}$$

Each cut plane C_p is used to separate the object model into two components. Facets are cut by this C_p and a set of points on the plane are produced after the cut. These points are common for both new components. A generation of triangular facets is executed and added in the corresponding data structure of each polyhedron. An example of the cut process is shown in Fig.4.5.

One of the main uses of this kind of decomposition could be in planning two grasps in the same object for manipulation tasks, where two robots take part or in tasks where robot and human have to manipulate an object together. This method offers a simple and fast way to decompose the object. We can see in the last part of the chapter how this method can be part of a grasp planner.

4.4 Approximate Convex Decomposition

The decomposition approach by inertial axes is fine when a speed partition of the object is sufficient to find a grasp, when the object presents a more complex geometry, an approximate convex decomposition (ACD) seems to be a better option. Convex objects are generally easier to grasp than non-convex

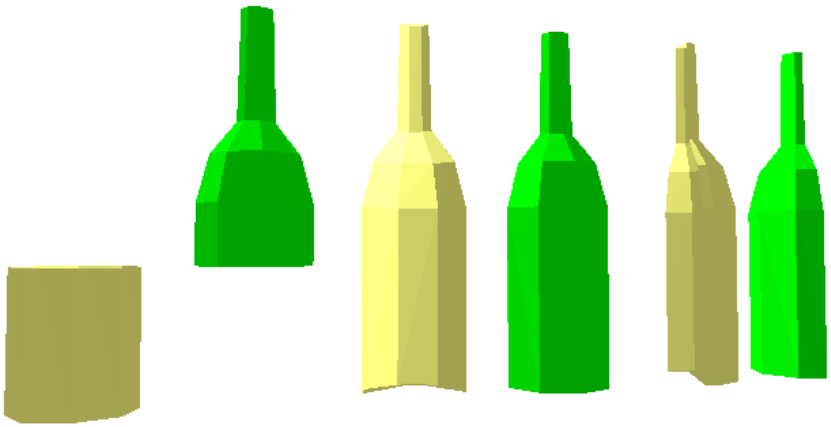


Figure 4.5 – Result of decomposition of a bottle by the three inertial axes planes, each plane produces two parts.

ones. For the object grasp planning problem, a decomposition strategy is a good one as we have seen at the beginning of the chapter. The analysis of previous algorithms shows that they can decompose the polyhedra in convex pieces but these can be small. Obviously, small pieces make object manipulation almost impossible because a grasp planner has no chance to find a grasp on these components. That is why, a better approach for this problem is an approximate decomposition. Of course we will not have convex pieces but parts that satisfy a certain convexity criterion. The resulting components after partition are less complex and it could be easier to grasp them.

The algorithm that we describe in this section is based on the work presented in the technical report [Lien 05] [Lien 03], here the notion of convexity measure is introduced and a framework for decomposing polyhedra in λ-approximate convex components, being λ the non-concave tolerance of the component.

As before, some preliminaries of the ACD are introduced for better understanding. A manifold polyhedron is represented by its boundaries δP_H. Such polyhedron consists of a set of vertices, a set of edges and a set of facets, $\delta P_H = \delta P_0, ..., \delta P_i$, where δP_0 is the outer boundary and the others are the inner ones due to holes. The polyhedron can be decomposed in a series of subcomponents in such a way that we have $P_H = \cup_i^n P_i$, as disjoint components.

The convex hull C_H is the minimum convex volume that encloses the polyhedron composed by a set of vertices, edges and facets. The polyhedron is convex if $P_H = C_H$.

To cut a polyhedron, an intuitive manner is to start with the most visible feature. The visibility is then related to the concavity of the notch $Co(N)$, such property is used as a measure, the concavity of P_H is the maximum concavity of its notches.

P_H is λ-approximate convex if concavity of $P_H \leq \lambda$, then the decomposition of P_H is the partition that contains only λ-components. The definition of an λ-approximate convex component is a polyhedron whose concavity is at most λ.

Algorithm 13: Approximate Convex Decomposition Algorithm Framework

```
input  : polyhedron P_H, tolerance λ
output : max(concave(P_i)) ≤ λ
begin
    C_H ←COMPUTE_CONVEXHULL(P_H);
    N ←COMPUTE_NOTCHES(P_H);
    B_N ←FIND_NOTCH_BIGGEST_CONCAVITY;
    if B_N ≥ λ then
        C_p ←COMPUTE_CUTTINGPLANE(B_N);
        P_i ←COMPUTE_PARTITION(P_H);
        RETURN(P_i);
    end
    RETURN(P_H);
end
```

The framework algorithm 13 shows the main steps for one iteration to decompose polyhedra in approximate convex components. The ACD takes as an input the geometric model of a component. This can be the object or a part of the object represented by a polyhedron P_H and a tunable convex tolerance λ.

The approximate convex decomposition removes at each iteration the most significant non-convex feature or notch that has the biggest concavity value. A cutting plane is generated, that splits the object into exactly two components.

The resulting components are the input for the next iteration of the algorithm. The process is repeated until all components C_i satisfy the convex tolerance λ. The resulting convex decomposition after iteration is :

$$\begin{aligned} CD(P_H) = \{P_i \subset P_H | concavity(P_i) \leq \lambda\}\ with \\ \cup_{i=1}^{n} P_i = P_H and\ P_i \cap P_j = \emptyset, \forall_{i,j}\ i \neq j \end{aligned} \tag{4.2}$$

A description of the principal operations of the decomposition algorithm is given in the next subsections. We have already seen that a concavity measure is needed to cut the polyhedron, but how this is measured ? How to compute a cutting plane ?

4.4.1 Concavity Measure

To identify the most significant notch we need a measure of its concavity. In contrast of some measures as radius, area and volume, concavity is not a well defined measure. To measure concavity, Lien and Amato propose the notions of bridges and pockets for polyhedron, see Fig. 4.6.

$$\begin{aligned} Bridge(P_H) = \{f_{C_H} | f_{C_H} \in C_H \wedge f_{C_H} \backslash S_{P_H} \neq 0 \\ Pocket(P_H) = \{f_{P_H} | f_{P_H} \in S_{P_H} \wedge f_{C_H} \backslash C_H \neq 0 \end{aligned} \tag{4.3}$$

where, C_H is the convex hull of the polyhedron, f_{C_H} represents a facet of convex hull and f_{P_H} is a facet of the polyhedron and S_{P_H} is the surface of the polyhedron P_H.

The bridges as defined in equation 4.3, are facets of the C_H that are not part of the polyhedron. We identify the match between facets of C_H and P_H and discard them. Pockets are those facets of polyhedron

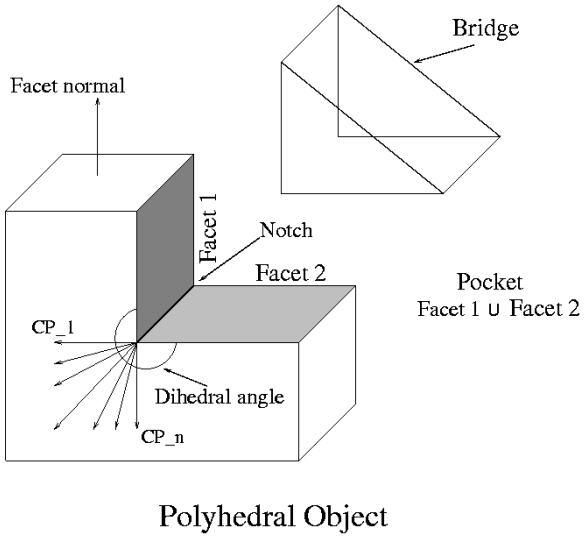


Figure 4.6 – Example of Polyhedral object, bridges and pockets are indicated. The notch is the reflex edge formed by the two facets with an internal angle greater than $180°$. A series of cutting planes CP_1 ... CP_n are shown.

P_H that are not part of the convex hull. As the facets of C_H are not generated from the facets of P_H, meaning, if P_H would be convex its correspondent C_H will be the same P_H, a test is required to find the facets that correspond to the pockets. Such test is done by a user tolerance in the distance between the P_H facet and the C_H facet and a tolerance in the orientation between facets approximate planes. After the test we obtain the pockets. It is interesting to observe that a real concavity measure would be the trajectory described by a point of the notch to the bridge, such trajectory would be given by $\int_0^1 \tau(t)dt$ as shown in Fig. 4.7a, but we don't know how to define this $\tau(t)$ function. A simple option is to take the distance between the middle point of each notch to the bridge. We measure the concavity as the length of the straight line from the concave notch to the convex hull of the object, see Fig. 4.7b. The notches with a value bigger than λ are taken to decompose the object. We begin to decompose by taking the notch with the biggest value.

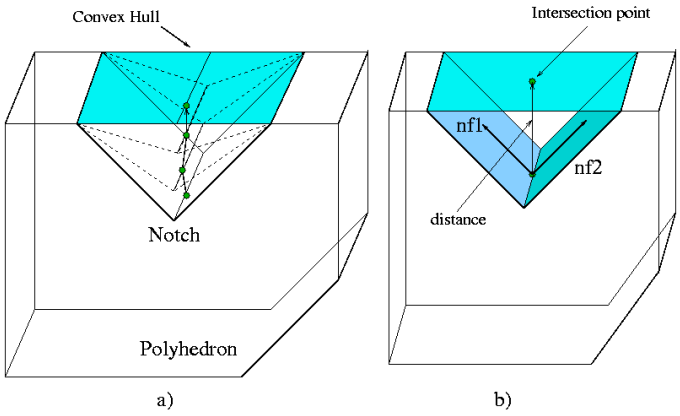


Figure 4.7 – Concavity property

4.4.2 Cutting Plane Selection

When bridges and pockets are computed and a concavity value is assigned to each notch. The notch with the highest concavity is selected and a series of cutting-planes (C_p) are formed to remove the notch. We select one C_p taking as criterion that the cut has the smallest area surface. We can see in Fig. 4.8 how different cutting planes generate different cross surfaces.

The cross surface area S due to the intersection with the polyhedron can be computed by picking the edge e_i generated by every intersected facet and constructing triangles from an end point of the notch to the edge e_i. The surface area of each triangle is given by a δs. The sum of all δs gives the total surface area $S = \sum_{i=1}^{n} \delta s_i$.

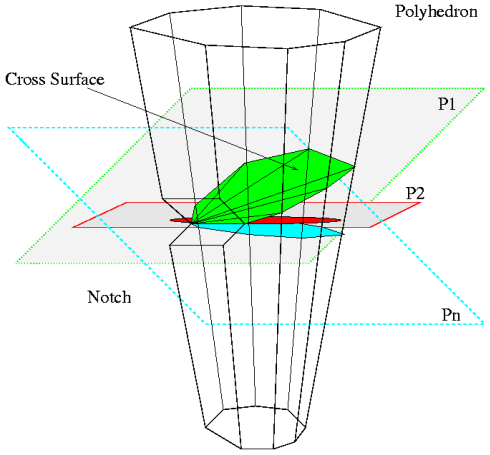


Figure 4.8 – Cross surface for cutting plane selection

4.4.3 Computing the Partition

Once the notch with the biggest concavity measure was found and a cutting plane was chosen, a cutting process is activated, which actually produces only two disjoint components. Three steps are required to separate the polyhedron.

- ⋆ Propagation and label
- ⋆ Split edges and triangles
- ⋆ Build new facets

The propagation takes the vertices that define the reflex and the C_p. All the vertices of the polyhedra are labeled depending on their location in relation with the cutting plane. The vertices that are located to the left of C_p are considered to belong to P_1 and if they are on the right to P_2. The vertices v_i located on the C_p surface are labeled as not determined because these belong to both components. Once each vertex has been labeled, we detect which edges and triangles are to be cut by the plane C_p. From all these points we can build boundaries on the cutting plane that we call cross surface S.

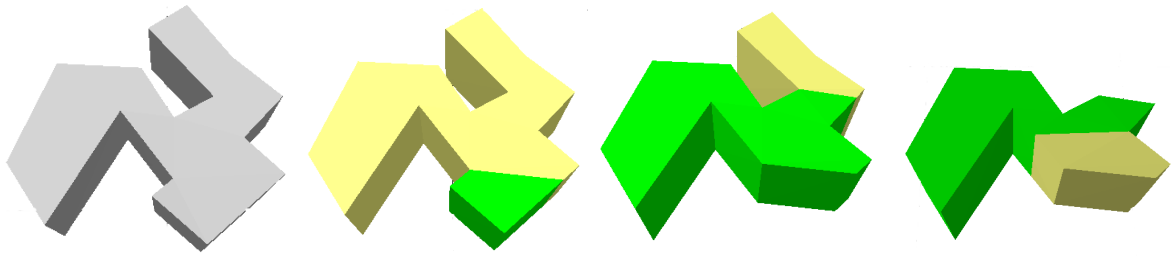
Algorithm 14: Split Algorithm

input : the polyhedron P_H, the cutting plane C_p, the notch N
output : the polyhedra P_1 and P_2
begin
  COMPUTE_CROSS_SECTION from (P_H, C_p);
  SPLIT_FACETS;
  while not *StopCondition (P_H separated into two subcomponents)* **do**
    $C \leftarrow$ SPLIT_UNHANDLE_POLYGONS;
  end
  RETURN (P_1, P_2);
end

As we are interested in that at each iteration only two pieces are generated the split operation begins the partition of the polygon P_c on the cross section that contains the reflex edge or notch N as in the non-manifold algorithm. After the split, if the polyhedron is actually separated into two pieces P_1 and P_2, we can continue to search for other significant notches ; otherwise we continue to cut, taking the other polygon boundaries on the cross surface, a pseudo code is presented in algorithm 14.

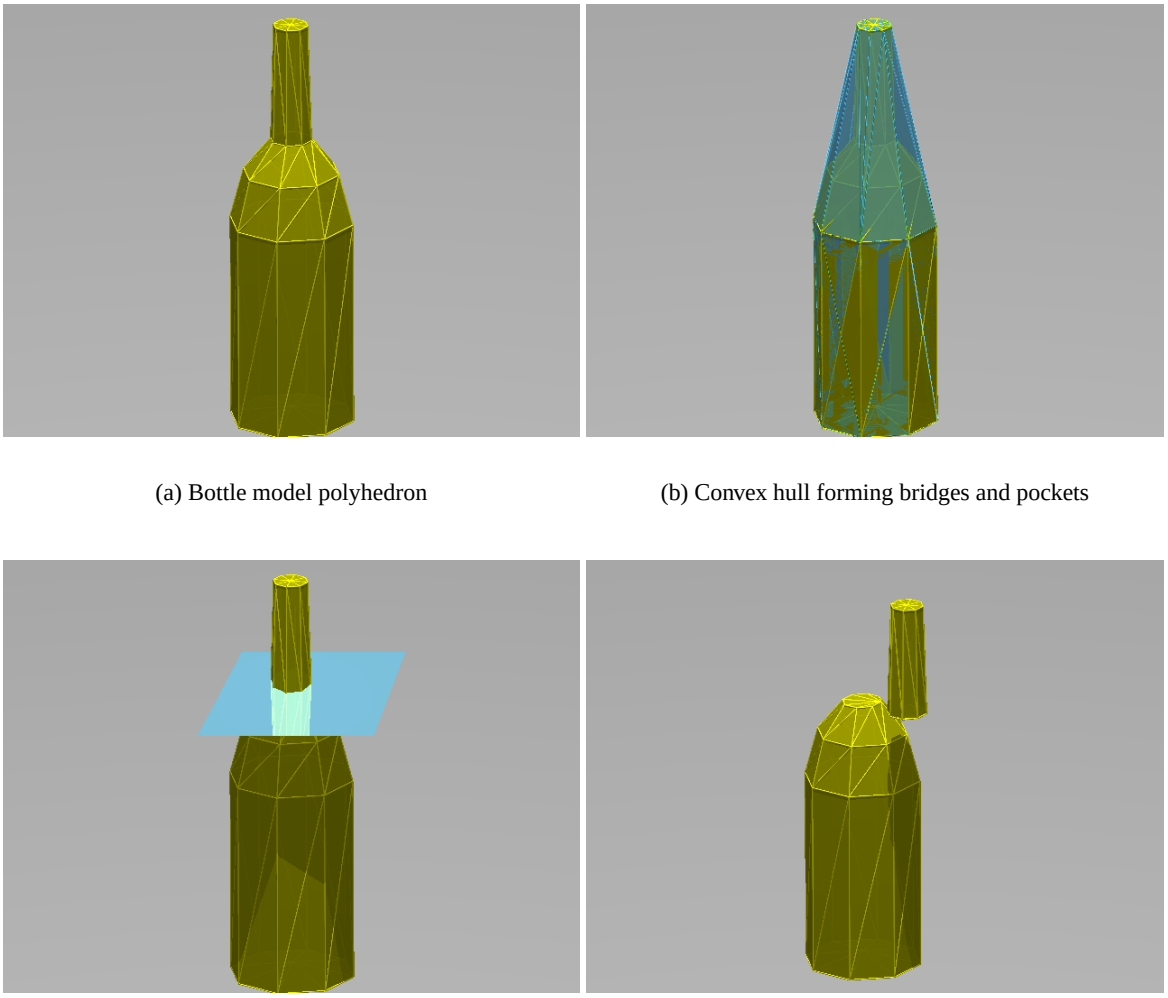
The step to create new facets corresponds to the fact that we use triangular facets as a representation for polyhedra, this is because we can easily represent arbitrary shape objects. New facets correspond to triangulate the polygon created by splitting the polyhedron using well known algorithms, creating monotone polygons [de Berg 00].

Resulting subcomponents are shown in Fig. 4.9 after four decomposition iterations by computing the cutting plane without considering the surface area. The cutting plane is always the middle plane between the facets that form the notch. A sequence of the decomposition process is shown in Fig. 4.10 with the computing of surface area.



(a) Polyhedral Object (b) First iteration (c) Second iteration (d) Third iteration

Figure 4.9 – The figure shows the output of the ACD algorithm after three iterations. When the first iteration is done, two components are produced. These two components are the input for the second and third iterations. The second iteration produces two new subcomponents. In the third iteration, the component is convex and it is not decomposed.



(a) Bottle model polyhedron

(b) Convex hull forming bridges and pockets

(c) Cutting plane

(d) Decomposition components

Figure 4.10 – ACD process steps

4.5 Integrating Object Decomposition in Grasp Planning

We mentioned in the previous chapter that a grasp is generally the beginning of most manipulation tasks and robots must be capable to handle most common objects in the surroundings. The shape of these objects varies and many are non-convex difficulting the grasp planning. The object model can be obtained by a stereo vision or 3D laser sensors. These models are generally complex and composed by many details. Grasping an unknown object is one of the tasks that an autonomous robot must be able to accomplish. A clue point to solve this is the automatic grasp of the object. The grasp planning problem becomes the generation of feasible grasps on the object that satisfies some fundamental criteria : the grasp must be stable, the grasp must be reachable, and the grasp must be free of collision. Grasp planning is computationally intensive due to both : the large search space and the incorporation of geometric and task-oriented constraints. The search space can be reduced by using certain heuristics. Shape of the object, relative size of the object and gripper are the principal geometric characteristics to consider in the planning. Shapes are generally non-convex and constitute a more difficult problem to solve for a grasp planner. In the following sections we describe a planner to deal with such kind of objects.

4.5.1 A Non-Convex Grasp Planner Algorithm

The approach that we propose decomposes the object in smaller components and computes a set of grasps for each generated component. Every grasp is ranked depending on a quality criterion. The best quality grasp is chosen as the output of the planner. We present here the basic algorithm framework of the random generator of grasps for non-convex objects [Lopez-Damian 05b].

Algorithm 15: Non-Convex Grasp Planner Algorithm Framework

```
input  : the robot R, the environment E, the object O
output : the grasp G
begin
    while not StopCondition (G Founded or ACD(O) = ∅) do
        C ← OBJECT_DECOMPOSITION;
        forall Components of C_i do
            COMPUTE_INERTIAL_AXES(O);
            G ←RANDOM_GENERATION_GRASPS;
            if G Satisfy FILTERS then
                ASSIGN_QUALITY_VALUE G ← Q;
                ADDNEWGRASP List_G ← G;
            end
        end
    end
    CHOOSE_GRASP_BEST_QUALITY_VALUE;
    RETURN G;
end
```

The grasp planner takes as an input the geometric model of the environment, including the object and the geometric and kinematic models of the robot and gripper. The object decomposition process is the first step of the algorithm. Here two strategies for the planner can be followed. The first one is to call for a complete decomposition process that gives as a result the series of components of the object and generates a set of grasps for each component in the list. The second one, as we presented it in the framework, at each iteration of the decomposition process two components of the input are produced.

We choose one of the two components and we try to generate a feasible grasp on it. If a grasp is found, the planning process is ended and the grasp is given as output. If not, we take the second component and we call again the grasp planner. In case that a feasible grasp cannot be generated for both components, a new iteration is performed.

This process is repeated until one of two conditions arrive : a grasp is found in one component, or decomposition is terminated. A feasible grasp can be found before completing decomposition. The last components found are generally small and less interesting.

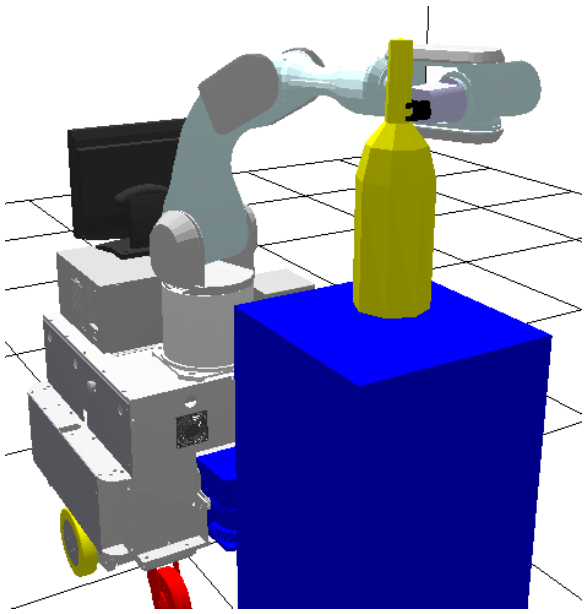


Figure 4.11 – Grasp founded on the upper part of the bottle because bottom part is very big, the grasp computed is in the bottom of the bottle neck.

4.6 Planning for Interactive Grasps

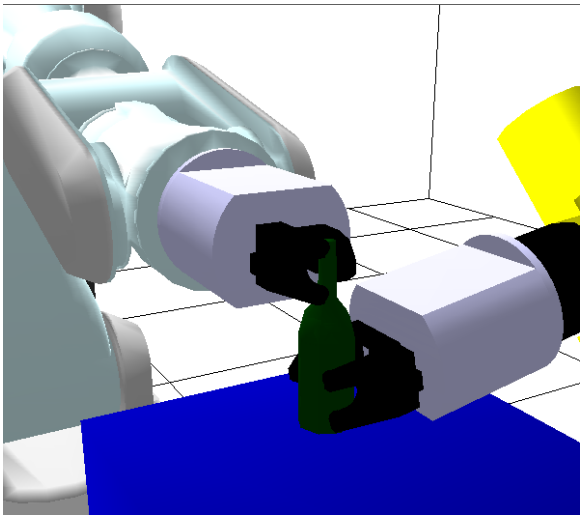
For interactive grasps, we mean that a double grasp can be planned on the same object to perform manipulation tasks that require an exchange of this object between entities in the environment. The combination of two entities can be found to perform this kind of task, robot-robot or human-robot pairs.

Starting from the hypothesis that if the planner can find a grasp for a second manipulator, we can assure that the human will find it or even a better one. The algorithm 16 presents the main steps to plan these interactive grasps.

Using the same strategy as before, a decomposition of the object at each iteration gives two subcomponents ; we apply the grasp planner to the components within the decomposition process. If a grasp is found at each component the algorithm stops, otherwise it continues the decompose-planning loop. The other two stop conditions for the algorithm shows a limit of tries or that we cannot decompose the object any longer. A limit has to be established for the decomposition methods that do not depend on the concavities as the IAD. A result of the use of this algorithm is shown in Fig. 4.12.

Algorithm 16: Interactive Grasps Algorithm

input : the robots $R1$ and $R2$, the environment E, the object O
output : the grasps G_1 and G_2
begin
  while not *StopCondition* ($Ntry == LIMIT$ *or* G_1 *and* G_2 *or* $CD = \emptyset$ **do**
    $[C_1, C_2] \leftarrow$OBJECT_DECOMPOSITION;
    $[G_1, G_2] \leftarrow$RANDOM_GRASP_PLANNER (C_1, C_2);
    if $G_1 \neq True$ *or* $G_2 \neq True$ **then**
      INCREMENT_NTRY;
    end
    UPDATE_ROBOT_CONFIGURATIONS;
  end
end



(a)

Figure 4.12 – Interactive grasps for bottle object

4.7 Discussion

The grasp planning for non-convex 3D objects is a challenge for service robotics. In this chapter we have proposed a grasp planner for these kinds of objects. The idea to decompose the object in smaller parts seems to be a good solution. The strategy to generate grasps on components at each iteration of the decomposition process for the grasp planning allows to save computing time in the execution of the algorithm. The decomposition by removing concavities gives interesting results, but a main point that would help to improve the algorithm is an optimal selection of the cutting plane. Until now, the criterion used does not guarantee the best way to partition an object for grasping, a more intentional manner has to be found. Since our approach is a geometric one, it is not obvious how we should cut the object in a

pertinent manner that helps to grasp the object to perform a desirable task. If we consider another kind of information a better directed decomposition can be obtained.

We can say anyway that for delivery tasks the approach gives satisfactory results and seems to be suitable for a grasp planner. Some improvements can be made to the algorithms expressing the nature of the task and the object in the decomposition process.

If we compare this approach with the method in [Bard 95], we can notice that the main difference is that the approach followed in our work is in a 3D space while the other make the partition in 2D. The unions of the similar slices that forms 3D components limit the way the object can be partitioned. Both approaches can decompose the object in subcomponents but only our method gives us a criterion to select the subcomponent to grasp the object. It seems easy to adapt other criteria in order to consider possible constraints at the moment of object decomposition process.

In the decomposition process, the generation of cuts can be improved if a set of neighbors concavity points are linked forming a continuous simple closed curve called knot, the cuts are made with these knots [Lien 05].

One of the challenges that still has to be solved in an optimal way is the concavity metrics, how to measure concavity that reflects really how concave a feature is, in such manner that solving these features we decompose the object advantageously. One option could be the use of a deformable surface encapsulating the object, as the surface change its shape according to the object, the main deformations will be at the concavities, the value representing how deformed is the surface indicates the concavity value.

The holes in polyhedra are another difficult problem to deal with, finding meaningful cuts is not trivial, reducing the genus of the polyhedron to zero is the strategy followed, but how the reduction is made does not guarantee that the final cut is the best from the grasp planning point of view. For example, in the case of a mug object, the handles can be solved by cuts around the handles which give as a result a series of small components. The best option is to cut the whole handle from the object.

Chapter 5

Experimental Results

The objective in this chapter is to present the results obtained for the grasp planner. These are divided into two parts : tests made in simulation and tests with an experimental mobile manipulator. The first part of the chapter presents the software and hardware platforms that were used for experimentation. A general architecture for autonomous robot and a generation of modules for distributed architectures developed at LAAS are briefly described, the latter one helps for the integration of the grasp planning algorithms into the physical robot with other functionalities to plan and execute tasks with such architecture. We continue with the description of a grasp based architecture with modules that will allow to solve manipulation tasks for unknown objects for service robots. The second part of the chapter presents simulations results for the grasp planner, some of these results use object decomposition algorithms as those presented in chapter 4. The grasp planner is tested in various situations with different objects for different robots to illustrate the planner behavior and performance. Double grasps are computed next for delivery tasks, we show an example for a pick and place task. The last part of the chapter presents some possible scenarios where the grasp planner can be used and some experimental results with the physical mobile manipulator Jido for grasping operations.

5.1 Experimental Platforms

This section presents the software and physical platforms that were used to obtain the results. We test our algorithms in simulation before integration into a physical robot. First we describe the motion planning tool Move3D, then we continue with the end-effectors, and finally we introduce the physical robot *jido*.

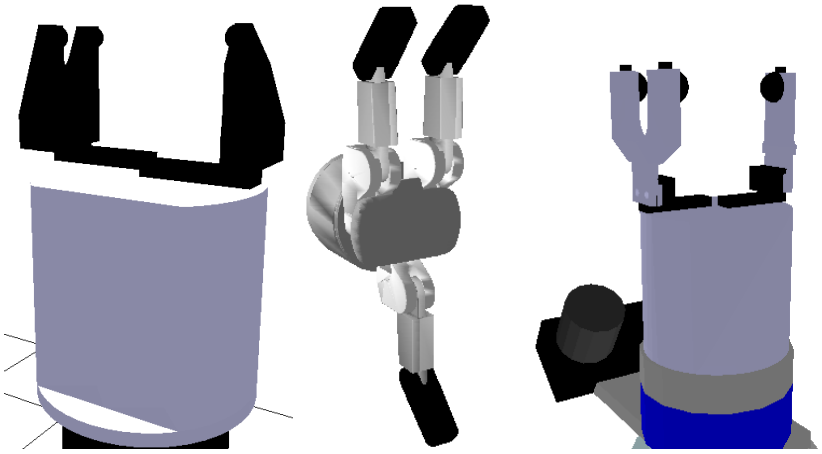
5.1.1 Move3D : Motion Planning Platform

Move3D [Siméon 01] is a generic motion planning tool to compute collision free paths for a broad set of applications, from industrial factories to mobile robots. The platform is based on probabilistic roadmaps methods as described in chapter 1. We recall briefly what was presented in chapter 1, once a roadmap is computed, the planner tries to connect the initial to the final configuration to the roadmap and searching a path in the roadmap using the specified steering method. Move3D consists of several modules : the *modeling* module allows the user to describe mechanical systems and environments to describe the motion problem and motion constraints. The *geometric tool* module is used to filtering a geometric database and initialize interference and collision checkers. The *steering methods* module computes local paths that satisfy kinematic constraints of the mechanical systems for admissible motions. We can find a linear method that computes a straight line between two configurations for holonomic

systems, a nonholonomic method for car-like mechanisms, and a Manhattan method that considers the constraint to move one degree of freedom each time. The *planning* module that contains the roadmap search and builder. Finally, a *display* method to visualize the results.

5.1.2 Robots and End-Effector Models

Using the modeling module of Move3D we have described robot models of mobile manipulators, the mobile platform can be holonomic represented by a Nomadic4000 robot or a nonholonomic Neobotix platform. The end-effector can be modeled by grippers, we use a gripper with two parallel fingers with three semi-spherical contact points or articulated multi-fingered hands, particularly the Barrett hand that was described in chapter 3; in Fig. 5.1 we show three end-effectors used for the tests. Throughout the document we can find the robot models that combine three main elements. A first model is composed by an holonomic mobile platform, a six D.O.F Gt6a arm with a gripper. The second model is described by a holonomic platform, a six D.O.F PA10 arm and a Barrett hand. The next two models are similar with a nonholonomic Neobotix mobile platform, a six D.O.F PA10 arm and differ in the end-effector, one has the Barrett hand, while the other is equipped with a gripper. The last model is that of our experimental platform with the addition of two pairs of stereo vision cameras as shown in Fig. 5.2.



(a) (b) (c)

Figure 5.1 – a) Gripper with three semi-spherical fingertips, b) Three multi-fingered Barrett hand and c) Jido gripper

5.1.3 Physical Robot Jido

The robotics group at LAAS has recently acquired a new experimental mobile robot to carry on a set of tasks, object manipulation falls in this set. The robot is composed of a Neobotix MP-L655 mobile platform equipped with a differential drive locomotion system, ultrasound and laser sensors SICK, and a manipulation arm Mitsubishi PA10-6C with six degrees of freedom. The end-effector is a gripper with two fingers and three fingertips. A force sensor is mounted at the wrist and resistor tactile sensors at the fingers. For vision perception two cameras are used, one at the wrist and the other at the center of the platform in a support with a pan-tilt system as shown in the Fig. 5.2.

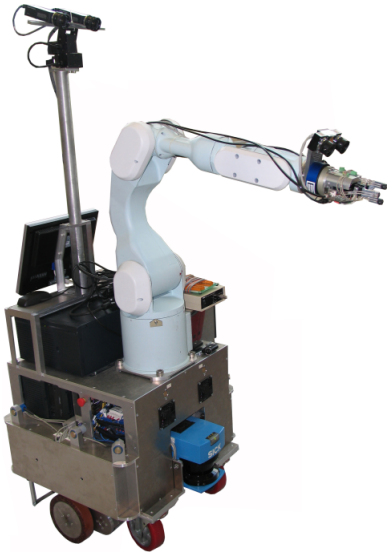


Figure 5.2 – Experimental platform

5.1.4 Robot Scenarios

For some tasks we cannot hope that the robot has at the beginning a database with all the object models in the environment, the capacity to manipulate unknown objects becomes primordial. Another scenario we are interested in is a “curious robot”, once the robot perceives an object that attracts its attention, it approaches the place where the object lies and starts to inspect it to construct a three-dimensional model. Once a model representation was obtained, the robot can grasp the object to ask for more information to a human for example.

The other scenario is the hand over task, we are interested in making the robot grasp an object located at some known place, transport it to a place where the human stands and present the object before him.

The work in this document can be used to solve the grasp planning part of the tasks, and for pick and place operations in the inspection process in the first scenario. These two scenarios are used as a guide to equip the autonomous robot with capacities that will allow it to move in a human environment and interact with humans. This is one of the goals in the European project *COGNIRON*, where LAAS is a member.

5.2 Robot Control Architecture

In order for an autonomous robot to accomplish a mission or task given by a user, it is convenient that this task can be transformed in a set of subtasks or simple actions that the robot will execute with the resources that it possesses in a finite time. For that a general architecture for autonomous robots becomes important. It is briefly described below.

5.2.1 LAAS Architecture

First we present the general architecture developed at LAAS-CNRS [Alami 98] for autonomous robots. It is divided in three levels :

Functional : this level integrates the functions to control the sensors and actuators as well as the algorithms that perform a specific operation or action. This level contains a module library. Each

module is an autonomous entity performing a set of specific functionalities and can communicate between them.

Execution control : this level allows to control and coordinate the execution of functions from the precedent level following a set of constraints imposed by the higher level, verifying that the actions do not take the robot to an incoherent state.

Decisional : this level is composed of a mission planner and a supervisor. The planner receives a description of the state of the environment and a mission from a user, a sequence of actions and the execution modalities are given as output. The modalities describe the constraints to be respected in the execution of the plan. A supervisor interacts with the lower level and looks for the good execution of the actions and reacts to the events within a finite time.

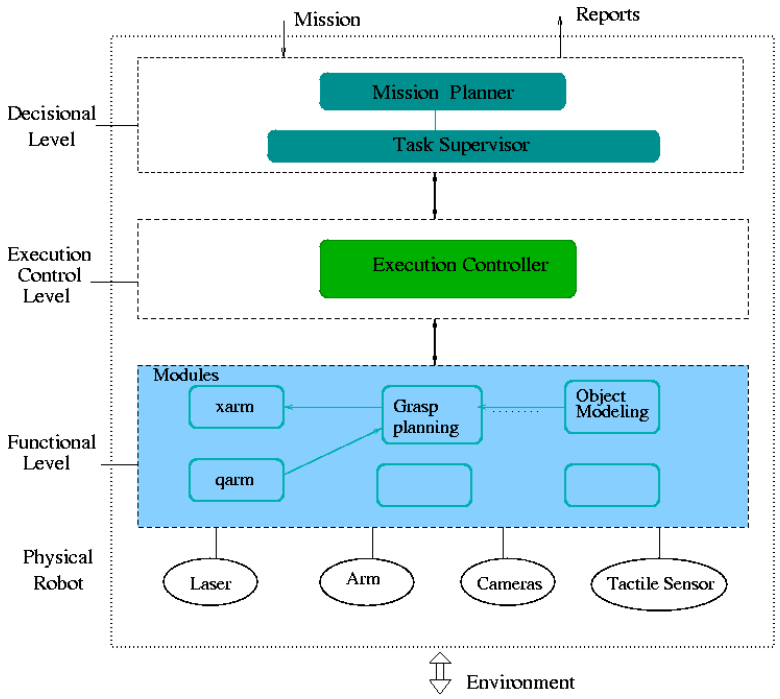


Figure 5.3 – Architecture for autonomous robots

5.2.2 Generator of Modules GenoM

All the algorithms in this thesis are in the functional level and can be integrated to the robot architecture as a module using the Genom tool [Fleury 96] [Fleury 97]. Genom is a module generator where each module encapsulates a set of functions, the activities and data produced by the functions, and the mechanisms to control their activity. The module considers the functions as services and allows the access to the service and, produces data through standard interfaces. The functions are controlled through requests, this can be used indifferently for other modules, operators, or a supervisor. When the service has ended, a reply is returned to the module that made the request, an execution report tells if the service succeeded or if there were errors. There are two types of requests : the execution request that actually starts a service, and a control request that controls the service execution (parameter modifications, interruption). An activity is a function in execution, the activity can control the physical robot (sensors and actuators) and uses services from other modules or produces data. The data are updated in data structures

called posters that can be accessible during execution or at the end of the service. All transmitted data from and to the modules are memorized in a data base $SDIf$.

The module counts with a control data structure aside from the $SDIf$ for management and with tasks that execute the functions. These tasks are divided in control and execution tasks. The control task manages the module taking the requests for the module, verifying the parameters validity and registering these parameters in the $SDIf$, demanding the start of the execution task, and returning the results to the client when the activity is over. The execution tasks are in charge of the function algorithms. If this algorithm is to be periodical (visual servoing, monitoring, filters, ...), you will have to use a periodical execution task and specify its period. It is also possible to use aperiodical tasks and a sequential scheduling. Tasks are given priorities depending on their constrains in terms of resources and CPU requirements.

Modules provide two standard interface libraries : a) A service library, which handles requests emission and reception, and b) A poster library, which contains the necessary functions to read the module posters.

The integration of the algorithms (the code) in the module consists in associating the code to the requests. We have to tell GenoM which are the functions that must be executed to handle requests. Your algorithms must be split into several parts (startup, main function, end, interruption, ...) Each of these parts is called a $codel$ (elementary code), at this time, codels are C functions. A module is the result of the linked edition between the code generated by GenoM and the codel libraries.

5.2.3 Object Grasping Architecture

The architecture presents the main functions of the system to carry out automatic grasping tasks for unknown objects. As we mention in the beginning of the chapter, this is a collaborating work in the robotics group at LAAS-CNRS. The algorithms developed in this work are situated in the grasp planning module. Almost all modules are implemented using GenoM tool. The context of unknown objects implies that a model has to be constructed from a perception system. For modeling an object is necessary to acquire data from several views.

How to position the perception sensors to build a complete model is a Next Best View NBV problem and must be included in the architecture. The NBV module is simplified by a discrete semi-sphere centered at the object, each point or cell on the semi-sphere gives a position to place the stereo vision system mounted in the arm wrist to take images. The module $ObjMod$ processes each pair of images from the stereo vision cameras to find 3D points and to compute a model of the object by creating a cloud of 3D points with all images and computing a triangular mesh with them, as well as the location of the object given by a reference frame. The object model and location are the input for the grasp planning module gP, this finds a configuration grasp for the robot and the motion path to grasp it, the path is saved in a poster.

Module $qarm$ is the low level interface with the PA10 arm that moves the arm from one configuration to another. The module $Xarm$ is a trajectory time planner and execution module for the manipulator arm. As input it takes a list of robot joint configurations using the forward kinematic model, it computes for each configuration a vector with the position and orientation of the robot end-effector. With the list of these cartesian vectors and assuming that from one point to the next the motion is linear, a trajectory is computed, the output is the joint configurations where the arm has to move [Herrera-Aguilar 05]. The module $fingm$ closes the fingers incrementally until contact is detected.

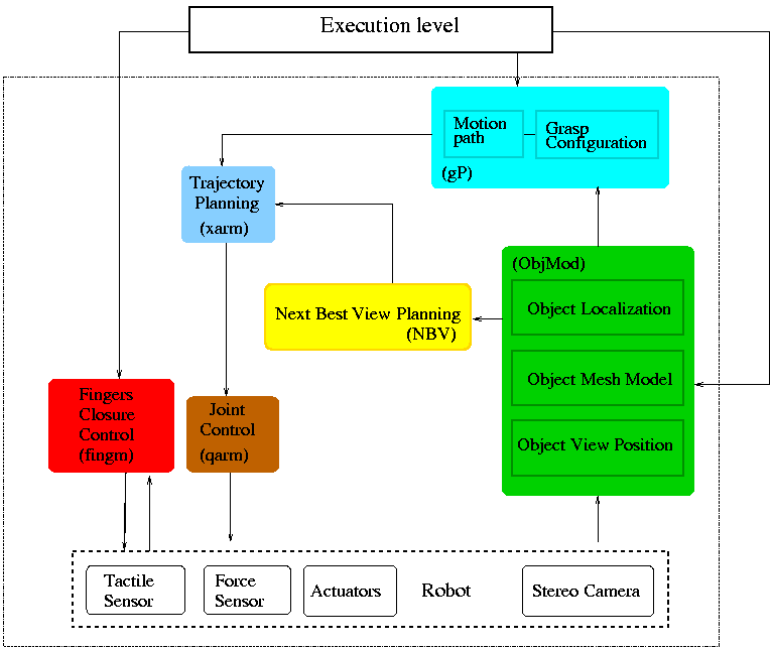
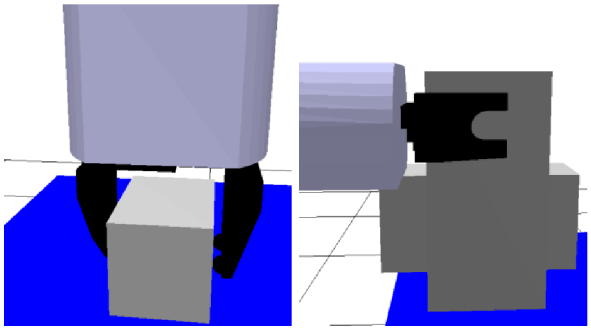


Figure 5.4 – Software architecture for grasping

5.3 Grasp Planner

We have tested the algorithm framework 7 in chapter 3 and the algorithms 8 and 9 with the gripper for several object geometric models. For the force-closure test and the quality value we predefined a friction coefficient corresponding to surfaces with rubber and steel materials.

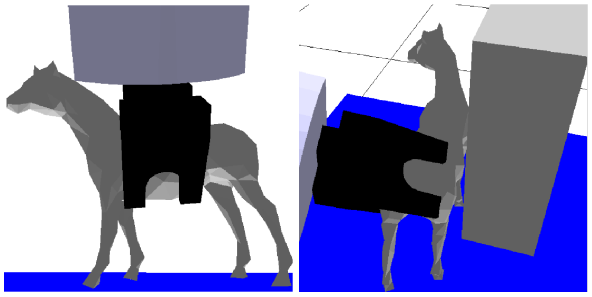
To show some planner properties we use a very simple object : a box composed with only a few facets located on a table. After the grasp planning execution, a grasp is found near the center of the mass and the manipulator approaches the object in direction of one inertial axis as shown in Fig. 5.5a. For the same object in a different configuration and with an obstacle in one of its sides, a different grasp is found Fig. 5.5b, we observe that a different inertial axis guides the approach and the grasp is located at the top of the object above the mass center avoiding the obstacle.



(a) (b)

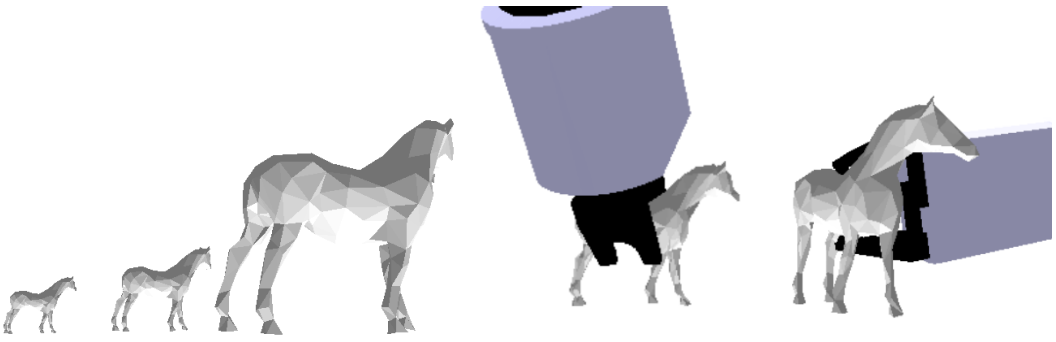
Figure 5.5 – Cube Model. For the same model, two different situations give very different grasps.

In Fig. 5.6, we show the solution for a more complex model like a horse formed with a bigger number of facets. We can see the grasps generated by the grasp algorithm. In the cases when there are no obstacles in the manipulation space, the grasps are near the mass center of the object as the previous example. When obstacles are placed near the objects, we observe how the planner is capable to find another grasp avoiding collision with the obstacle. Here we maintain the same object configuration and location. In Fig. 5.7a, we can see three different sizes of the object. This difference of size allows some other grasps on the object, in Fig. 5.7b a grasp is found by contacting the surface between the legs in the second model that was not possible in the first model, the second model being almost twice as big. The biggest model is very big for the planner to find a grasp.



(a) (b)

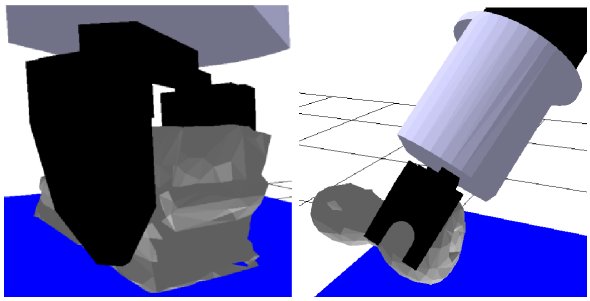
Figure 5.6 – Horse Model with and without obstacles. In the first figure without obstacles, the planner finds a grasp near the mass center. The obstacle in b makes the planner find another different grasp avoiding collision. The horse model is due to the Large Geometric Models Archive of Georgia Institute of Technology.



(a) (b) (c)

Figure 5.7 – Different object size

The last models in Fig. 5.8 are models of real objects acquired by a range scanner. A laser range scanner acquires the three-dimensional geometry of the object producing a cloud of three dimensional points representing the object scanned, and then a triangulated mesh is produced by using classical triangulation methods [de Berg 00] like Delauney or with other approaches : ball pivoting [Bernardini 99] [Restrepo 05] developed to deal with reconstruction and fusion of images acquired from different points of view. The process begins with a seed triangle where its three vertices are contained in a sphere with a tunable radius. The sphere pivots around the triangle edges to find other points to form triangles, the process ends when all points have been used. Marching cubes method [Lorensen 87] computes a mesh from volume data, where the base element is the voxel, the unit of division of space. In deformable surfaces [Kobbelt 99] [Lagarde 04] method, the surfaces are deformed with the application of forces on an initial mesh usually created with a discrete sphere.



(a) (b)

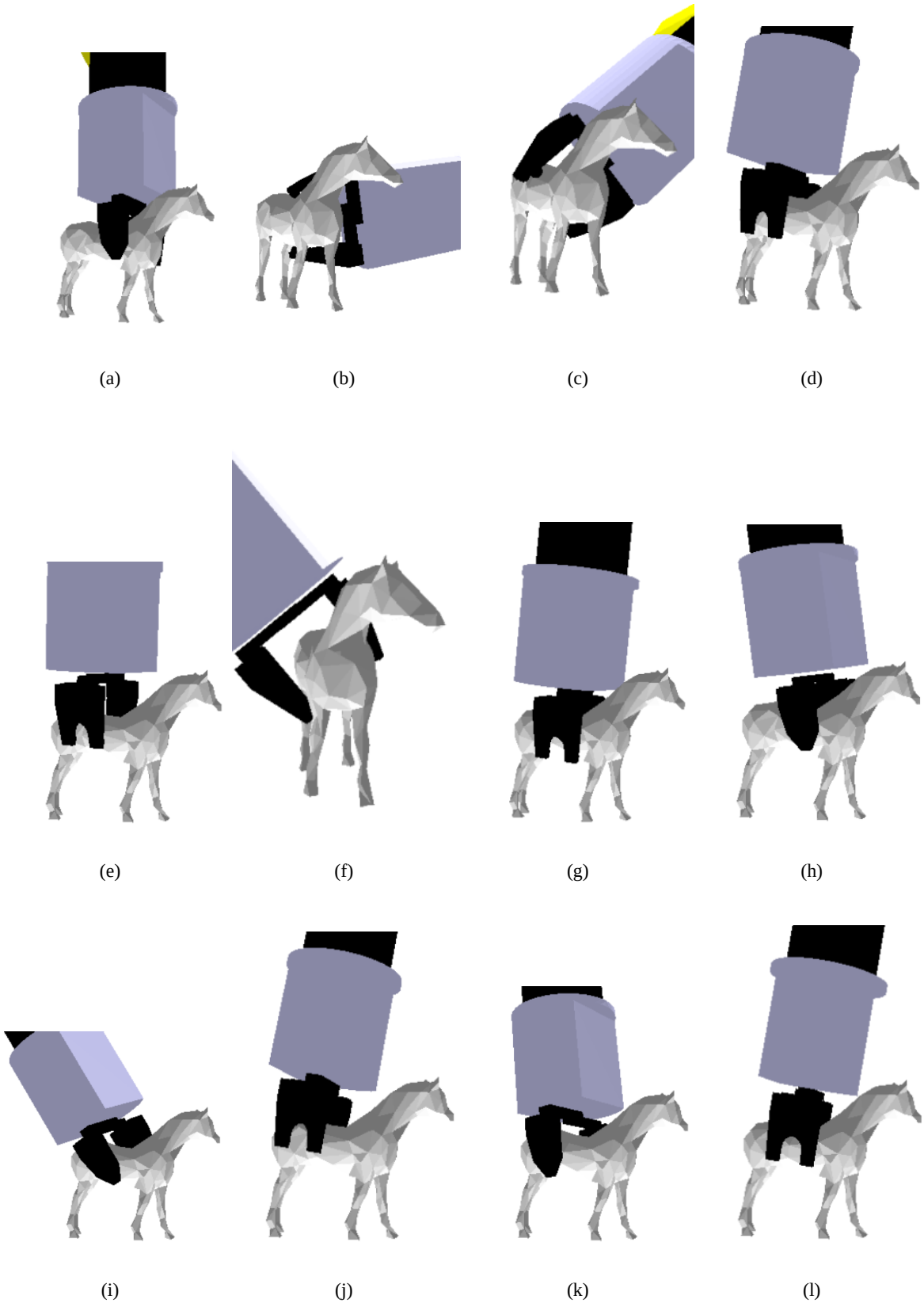
Figure 5.8 – Grasp planner tested in Real Couch and Doll Models. The models come from the image gallery of Ohio State University.

The experiments for the grasp planner were performed using a 500 MHz Solaris SunBlade. The time that the algorithm 7 requires to generate the final grasp is determined by the complexity of the object model. In Table 5.1 we can see the number of force-closure grasps generated by the algorithm for each object, the quality of the best grasp and the total processing time of the whole process. Since the grasp planner reasons on the whole robot grasp configuration, the computing time increases in comparison if only the object and a free-flyer end-effector were considered, so even for relatively simple objects like the cube, the planner takes a while. One can see in Fig. 5.9 some of the grasps found for the grasp planner.

Table 5.1 – Results for Several Objects

Object	Total Time (seconds)	Grasps	Quality	Facets
Horse	7.41	17	0.66	600
Couch	10.77	26	0.512	1000
Spaceship	8.67	21	0.87	617
Doll	5.45	20	0.44	500
Cube	2.45	30	0.13	12

The number of grasps is directly related to the six inertial axes, the object length is divided to generate grasping points along the inertial axes and the rotations around the inertial axes at each grasping point. After passing the filters the number of grasps decreases substantially.



(a) (b) (c) (d)

(e) (f) (g) (h)

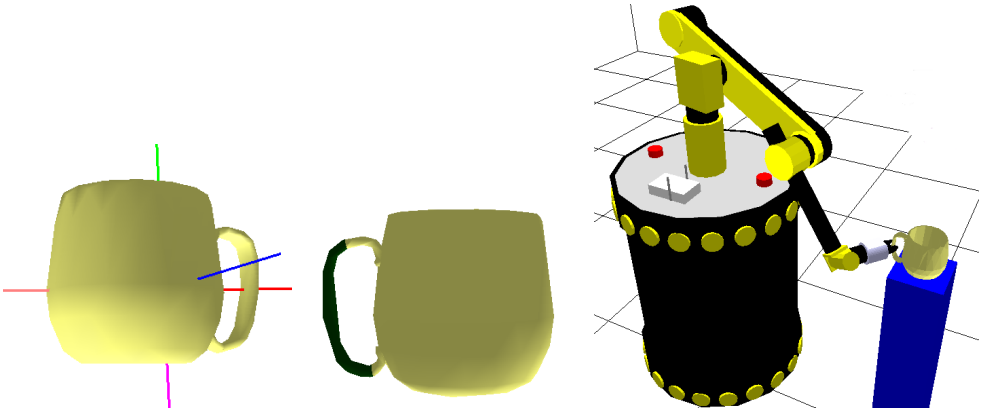
(i) (j) (k) (l)

Figure 5.9 – Set of grasps

5.3.1 Object Decomposition

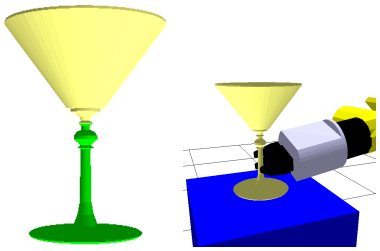
The results presented here are obtained using the algorithm 13 of approximate convex decomposition of objects integrated to the grasp planner algorithm 15 as described in chapter 4. In Fig. 5.10 we can observe the inertial axes of a mug that were used to generate the grasps on the object. After the first call to the object decomposition process, the grasp planner found one feasible grasp on one of the generated subcomponents.

Three object models were used with the approximate convex decomposition : a mug, a glass and a bottle. In the first two cases, the mug and the glass are decomposed into two subcomponents and grasps were found in the handle of the mug, see Fig. 5.11 and the supporting base of the glass, see Fig. 5.10.



(a) (b) (c)

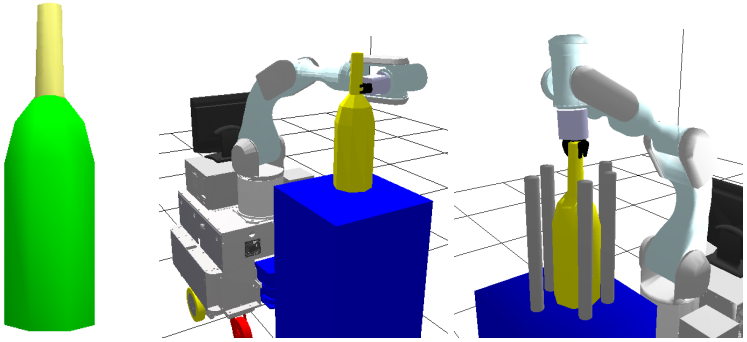
Figure 5.10 – a) Inertial axes used in grasp planner to generate a set of grasps on the object b) Decomposition of object after one iteration of the process c) The final grasp founded by the grasp planner after one iteration in the decomposition process



(a) (b)

Figure 5.11 – a) Decomposition of glass after one iteration of the process b) Feasible grasp generated after object decomposition

The last object is a bottle and we can see that the bottle size is big compared to the gripper, the only possibility is to grasp the bottle neck, see Fig. 5.12b. Finally in Fig. 5.12c, we can see how the planner is capable to compute a different grasp when the object is surrounded by obstacles. The experiments were performed using the same machine for the basic grasp planner, a 500 MHz Solaris SunBlade. The processing time that the algorithm requires to decompose an object is determined by the complexity of the object model. In Table 5.2 we can see the time after one iteration of the object decomposition algorithm 15 in chapter 4 for different object models.



(a) (b) (c)

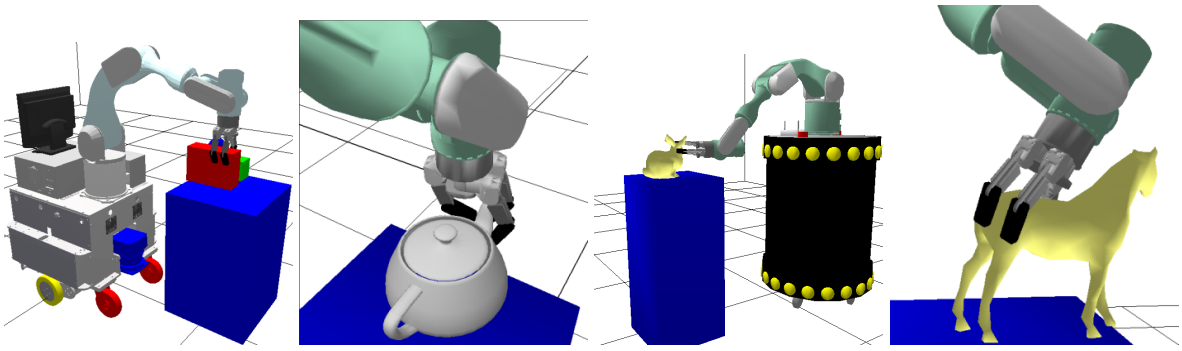
Figure 5.12 – a) Decomposition of a bottle b) Grasp founded on the upper part of the bottle because bottom part is very big. The grasp computed is in the bottom of the bottle neck. c) The bottle is surrounded by cylindrical obstacles, an alternative grasp is found

Table 5.2 – Grasp planner computing and Approximate convex decomposition time (seconds)

Object	Cut Time (s)	Grasp Time (s)	Total Time (s)	Facets
Bottle	0.12	36.7	36.82	80
Mug	0.71	34	34.71	499
Glass	13.58	135.56	149.14	2750

5.3.2 Multifingered Hand

The Fig. 5.13 presents examples of tests for various objects with the three-fingered Barrett hand. We can see in Fig. 5.13a, the solution found by the grasp planner, the fingers are located in two parallel faces of the object and avoiding the collision with other obstacles. On the next examples for more complex objects, the planner finds grasps that are not close to the center of the mass, even if no obstacles are present. The step to generate grasp points along the inertial axes causes that not all the facets are visited to find the grasps. Despite of the good grasps directions that inertial axes can generate, there are several other possibilities that the planner does not take into consideration.



(a) Grasp for a box model

(b) Grasp for a teapot model

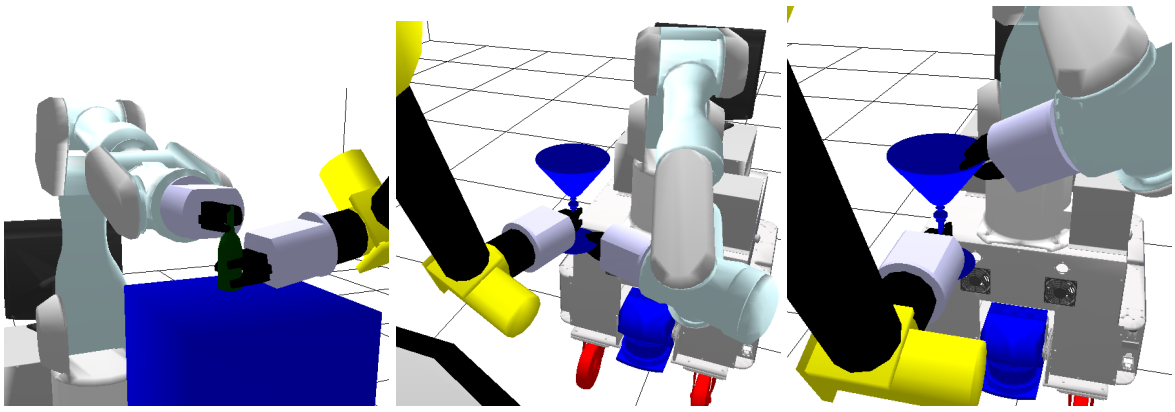
(c) Grasp for a rabbit model

(d) Grasp for a horse model

Figure 5.13 – Generation of grasps for a multi-finger hand Barrett on different models.

5.3.3 Double Grasps

For interactive manipulation tasks we need at least two grasps, in chapter 4 we have proposed the algorithm 12 and algorithm 15 to find such double grasps. We show other results for other different objects. These grasps are intended to be used for hand over tasks, at one moment of the task the two robots have to hold the object at the same time and then exchange it.



(a) (b) (c)

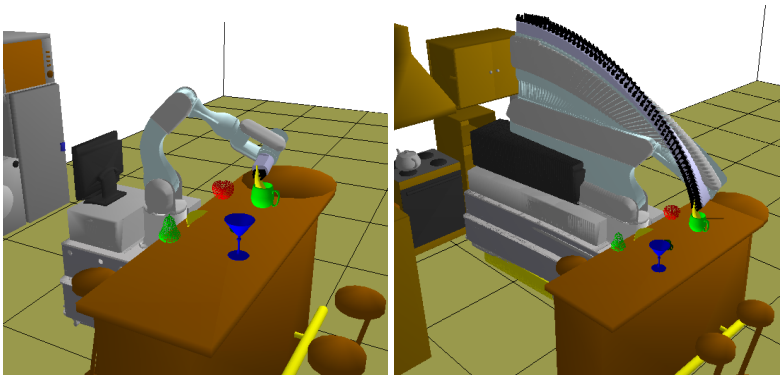
Figure 5.14 – a) Decomposition of bottle object, two grasps, each one in different components b) Grasps for a glass in the same component, c) Grasps for the same glass in different components

Table 5.3 – Grasp planner computing and Inertial axes decomposition time (seconds)

Object	Cut Time (s)	Grasp Time (s)	Total Time (s)	Facets
Bottle	6.11	42.86	48.97	80
Glass	21.54	301.2	322.74	2750

5.3.4 Complete Grasp Planning Motion

For a complete solution of grasp planning, we have to consider the approaching motion to grasp the object. In our case as we compute the grasp and the whole robot configuration for grasping, the approach is simplified by using a path planner from the initial robot configuration to the grasp configuration. As we can see in Fig. 5.15a, the example shows a banana object inside a mug, the grasp and robot configuration are found in the top part of the object to avoid the collision with the mug. In Fig. 5.15b, a visibility-PRM motion planner was used to find the path.



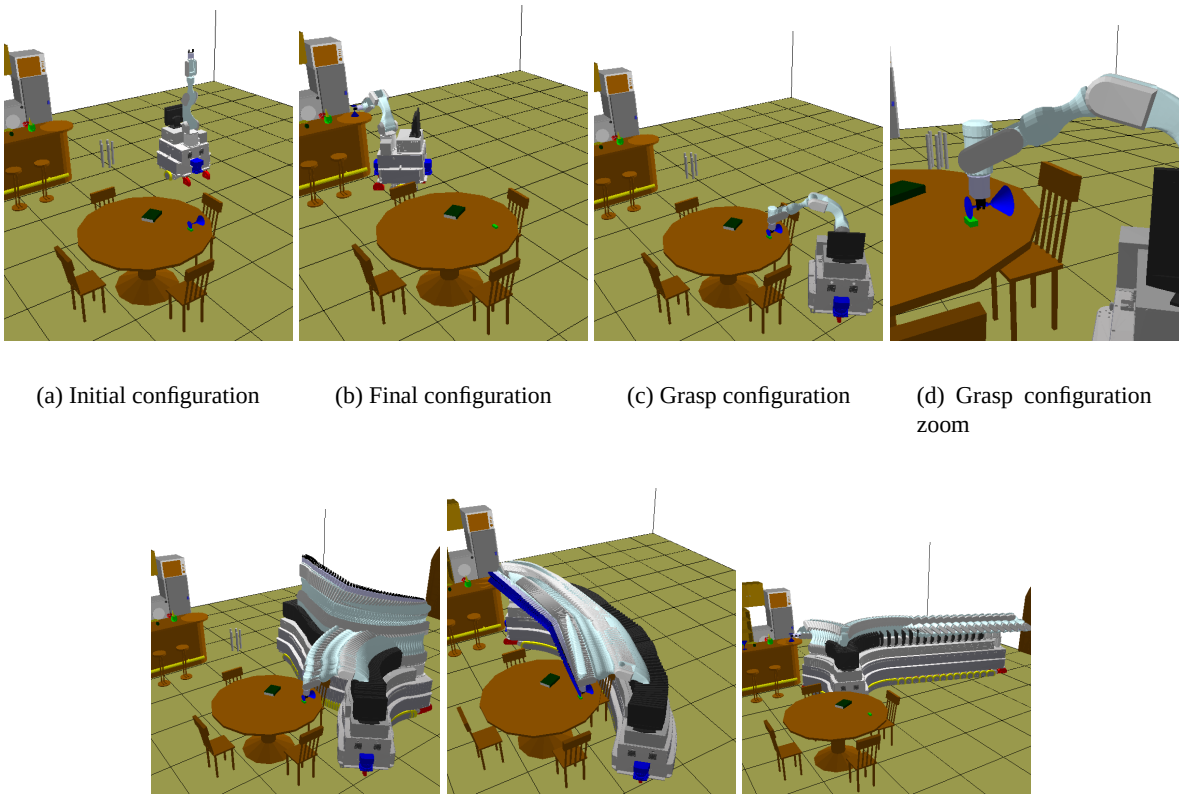
(a) Banana inside mug (b) Path for robot

Figure 5.15 – a) A robot grasp configuration is shown here for a banana object inside a mug, b) The approaching motion path to grasp the object without risk of collision with the mug.

5.4 Pick and Place in Delivery Task

The pick and place operation is a common manipulation task that an autonomous robot must be capable to solve. It cannot be considered properly as an interactive task, but it can be used as a complement for others tasks, for example when given the size of the object it is not possible to deliver the object directly to the human or robot, then a possibility is to leave the object at a near place. The pick and place operation as we saw in chapter 2 is considered as a part of delivery tasks, and even hand over operations can be seen as a kind of pick and place task. However, the problem is hard to solve when we are in real situations. The object model is not known and its exact location neither. Besides, the object can be surrounded by obstacles and sometimes these ones can be over the object, which may need a regrasping strategy to finally pick the object. Regrasping can be needed to reconfigure the object that allows to place the object to the final destination. In Fig. 5.16 we present a common task in human environments,

the robot is instructed to take the glass object from the living-room table Fig. 5.16a to the kitchen table Fig. 5.16b. The IMAP planner computes the grasp configuration and the motion from the initial position of the robot to the grasp configuration, then a second motion path is computed taking the grasp constraint to transport the object to the destination. Finally, the last motion is to leave the place to another one. The paths are computed with the visibility-PRM method and with a steering method for the nonholonomic mobile base. Regrasping is not considered here and we assume that grasp to pick the object is compatible with final object pose.



(a) Initial configuration (b) Final configuration (c) Grasp configuration (d) Grasp configuration zoom

(e) Path for pick motion (f) Path for transfer motion (g) Leaving path

Figure 5.16 – Actions for a pick and place task. From an initial and a final position of the object

5.5 Experimental Results

We started to perform tests with the experimental mobile manipulator *Jido*. A grasp planner module called gP has been developed that uses the functions described previously, it is able to run in simulation and in connexion with the real robot Jido. The grasp planner module interacts with the $qarm$ module that can control the robot joints directly and deliver information of these joint values. The $xarm$ module receives as input the path generated by the grasp planner and produces a trajectory that the manipulator executes to place the manipulator at the grasp configuration with the fingers opened. To grasp the object a $fingm$ module closes the fingers until contact using tactile sensors at the fingertips. The intensity of

the closure is tuned. The gP module is the only one that was developed in this work, the other modules were developed by other PhD students and researchers [Restrepo 05] [Herrera-Aguilar 05].

The test that we performed was to plan and execute the motions to grasp an object lying alone on a table. We give as input to the grasp planner module the model and its location. The gP module reads the $qarm$ poster to know the current robot position and updates its robot model configuration, this configuration becomes then the initial robot configuration for the planner. A grasp and the robot configuration are planned, the path from the initial robot position to the grasp configuration is computed using a bidirectional RRT motion planner. The mobile platform is already near the table and this kind of planner seems a better option in this case. The gP poster data is updated with the motion path generated. We can see the three first images in Fig. 5.17 that correspond to the initial robot configuration after the update step, the grasp configuration generated by the grasp planner, and the path to approach and grasp the object. The next images show the trajectory executed by the experimental manipulator, we can notice that the $xarm$ trajectory follows the path successfully. Once the manipulator is at the grasp configuration, a closing request is send to the $fingm$ module that closes the fingers. All the modules are executed online by the computer embedded in the robot, a processor with Fedora linux.

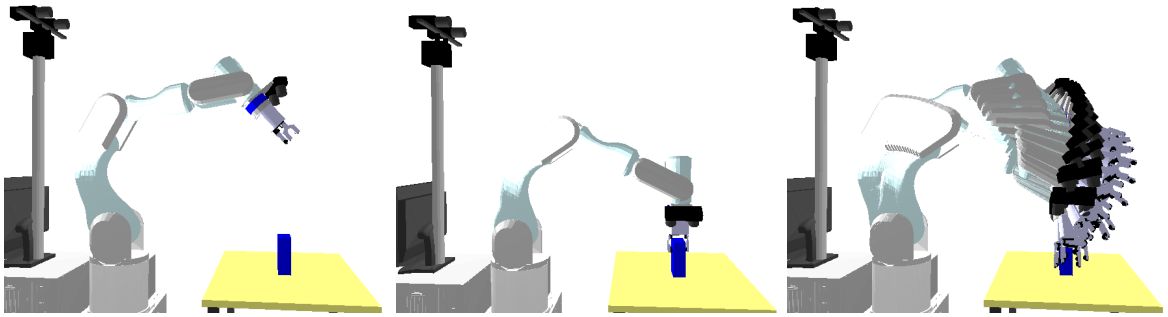
For the next test we introduce a big obstacle near the object that hinders the robot to use the same configuration to grasp the object. After execution of the grasp planner, a secondary grasp is generated that allows the robot to grasp the object from one of its sides. The motion path and trajectory execution from an initial robot position to the final grasp position are shown in Fig. 5.18. Here, we do not show a complete sequence of pick and place task because our interest is the grasp planner and because the place sequence only implies the call of a path planner and execution of the trajectory.

5.6 Discussion

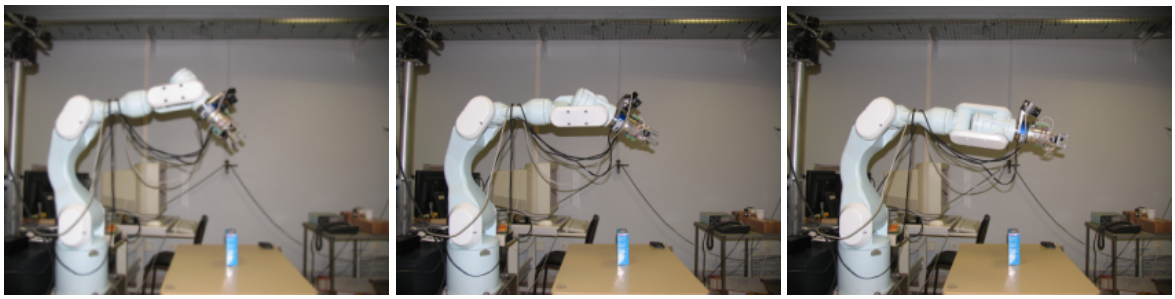
This chapter has presented the results obtained in this work for the manipulation planner, and mainly for the grasp planner in its different instances. Object decomposition helps in the generation of grasps for complex objects making the planner more robust. However, one of the limitations of the grasp planner is that grasps are only found on the object facets. While from experience we observe that in many cases a better grasp can be found at vertices and edges, for example for a tetrahedron model. Another important limitation is that we cannot deal with deformable objects [Hirai 01].

Some improvements have to be made for the grasp planners. In the planning for multifingered hands we need to change the contact model, a cylinder model for the fingertips is needed and we have to deal with the case when a fingertip contacts different facets on the object at the same time. As we want a robot that can interact in a natural manner with humans, when the robot hands over an object to the human, it must take care of the way to present the object. We need to express and consider the grasp planning constraints, for example, when the object has sharp surfaces like a knife, and if the object contains liquids, ... This is difficult because we do not know the material of objects neither their use, a pure geometric object representation is not enough for the planner, one simple solution could be the use of etiquettes [Kim 04].

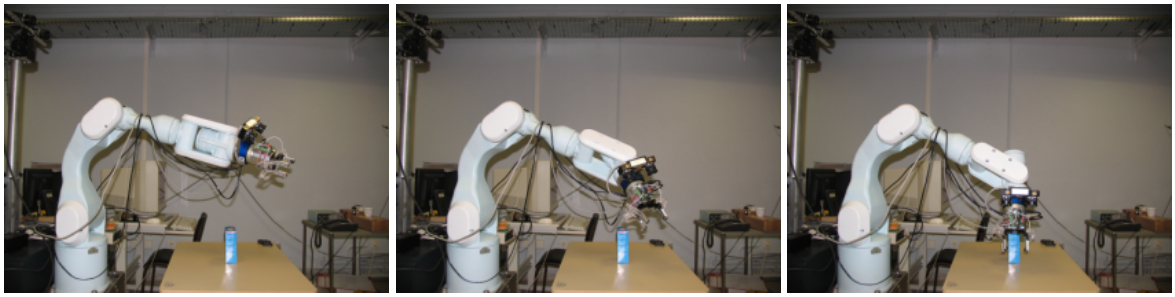
We have shown the implementation of the algorithms in the motion planning platform Move3D to validate them. The architecture of the planner can be extended in this scope, in particular the use of parallel robot model to express kinematic constraints on the object to manipulate. We have presented the integration of our grasp planner in an experimental mobile manipulator developing a functional module to make possible their execution. In spite of the limited number of the experimental tests, the results show that the random grasp planning approach seems to be suitable for manipulation of unknown objects.



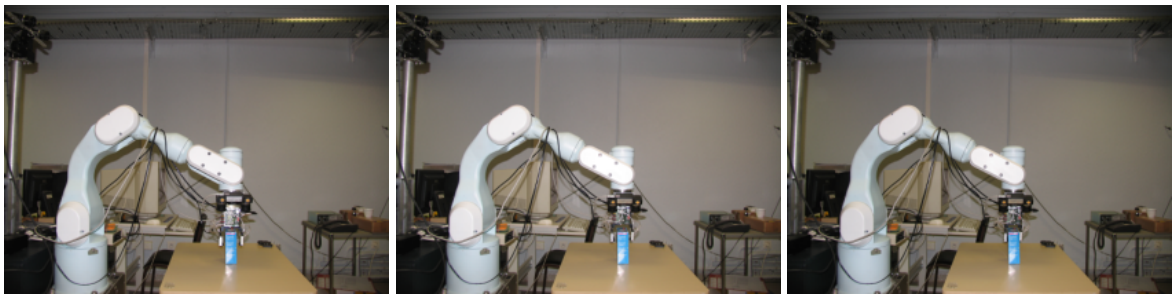
(a) Initial configuration (b) Final configuration (c) Path



(d) (e) (f)

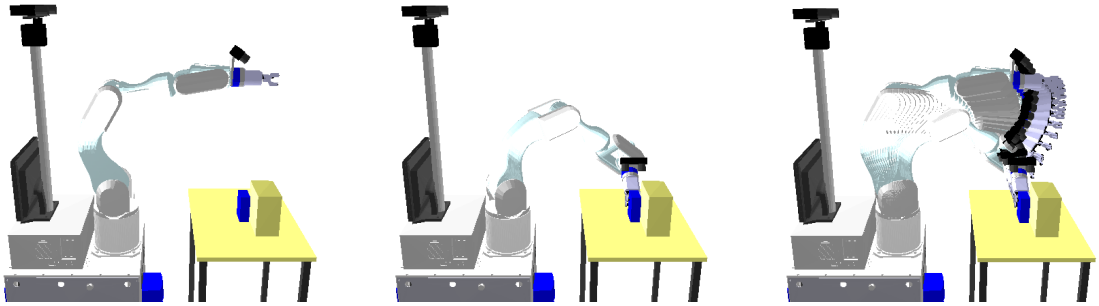


(g) (h) (i)



(j) (k) (l)

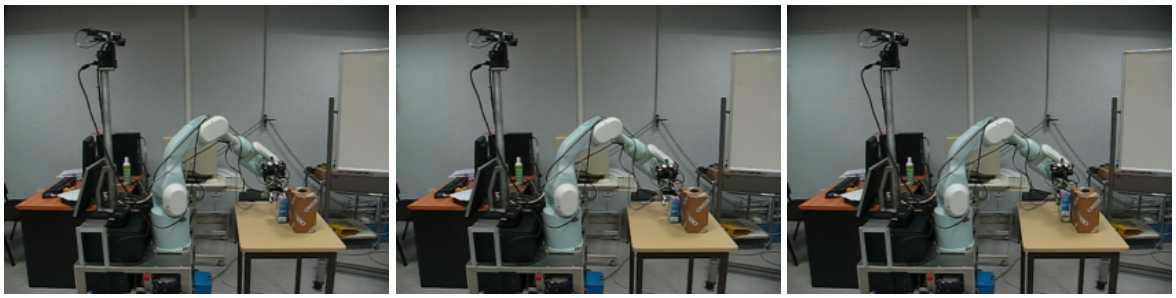
Figure 5.17 – Motion sequence of the robot to grasp the object



(a) Initial Configuration (b) Grasp Configuration (c) Path Generated

(d) (e) (f)

(g) (h) (i)



(j) (k) (l)

Figure 5.18 – Motion path to grasp the object when obstacle is presented and the trajectory followed by the robot

Conclusions

In this thesis we have presented a simple task planner for interactive manipulations based on a modular architecture, which allows us to test our grasp planner in the context of object manipulation. All the algorithms are integrated in a tool for motion planning called Move3D. Given that objects cannot move by themselves, we have attacked the problem by introducing the object as part of robot description and by dividing the robot in kinematics chains. This allows to define kinematic constraints between these chains and in that way the object moves and we can find a configuration for the rest of the robot. This kind of description helps us to define two robots and the object as a parallel robot that forms a closed kinematic chain making possible the use of motion planning techniques for these kind of structures. This was thought to plan interactive manipulations as hand over. Even if we have not solved the complete task, we have shown that the approach is feasible and promises good results.

When the robot has to manipulate objects using its end-effector to grasp them, we can find several algorithms that try to plan optimal grasps depending on certain criteria, where in many cases even the humans grasps are not optimal. The hypothesis that it is not necessary optimal grasps because there are several good ones helps to investigate alternative approaches. The use of heuristics to generate a set of grasps on the object is more suitable for the grasp planning problem from an algorithm complexity point of view that is reflected in the implementation and integration of such algorithms with an experimental robot. Among the set of heuristics, using inertial properties to guide the grasp generation gives good results and it is of great utility where the object to grasp is unknown.

The strategy to grasp 3D objects by decomposing them into approximate convex parts makes the planner more robust. This allows to obtain possible good size parts for grasping them, contrary to decompose the object obtaining strictly convex subcomponents. However, given that the algorithms to decompose the object are basically geometric, it cannot be guaranteed that the object will be cut in an adequate way for the task. Again, in a context where robots have to manipulate unknown objects to inspect them or to transport them, a set of grasps can be used for the task and not necessarily we have to compute an optimal one.

It was shown that the planner can be integrated relatively easily as a module for an experimental mobile manipulator. The generated motion paths to grasp the object are well executed online. Many more experiments have to be made to test the robustness of the planner, and some modifications can be made to improve it.

Perspectives

The thesis does not deal with one important aspect in the integration of the planner with experimental robots, the data obtained by the sensors that are intrinsically uncertain. The grasp planner could be modified to generate grasps with an associated radius of placement, if we move the contact point on the object surface within this radius, the grasp continues to satisfy the filters of the grasp planner, this allows it to have a margin error when the gripper establishes contact with the object.

The grasp planner for hands can be extended to generate enveloping grasps, for that we need to compute the intersections between the surface object and each part of the hand, these intersections are not points of contact but surfaces, this increases the computing complexity for the geometric algorithms. According to Nguyen [Nguyen 88], if we have a line in contact, this can be represented by two wrenches applied at the end points of this line, similarly for a plane, four wrenches in each corner of the plane in contact. An optional solution is to construct an end-effector equipped with a set of spherical contacts along the fingers and palm to make contact with the object.

Another interesting extension of this thesis is to plan how to grasp the object using the whole body of the robot, one can think for example that if we ask the robot to take a long cylinder shape tube bigger than its end-effector, the machine cannot be available to take it, as it is convex, no decomposition is possible to create smaller parts, there is no solution for the task with the current planner. But if the planner considers the whole robot as humans do, we can imagine that the robot can use its arms to surround the object for grasping. In the same way, for humanoids the grasp planner can be extended for simplified human hand models as described in chapter 3 and planning with the two arms, hands and body trunk.

Résumé

Depuis ses origines l'homme cherche à se faciliter la vie, d'abord en fabriquant des outils, ensuite en construisant des machines, puis en automatisant ces machines et maintenant en essayant de concevoir des robots autonomes. La robotique débute dans la seconde moitié du vingtième siècle avec le développement de bras manipulateurs articulés, commandés à l'aide d'un programme qui contient toutes les actions que le robot doit exécuter. La réussite de ce type de machine débute avec leur introduction sur les chaînes industrielles de production qui représentent encore leur plus important débouché économique.
Peu de temps après, les chercheurs se sont intéressés à rendre ces machines plus performantes en les munissant de capteurs extéroceptifs pour percevoir l'environnement de façon à ce qu'elles puissent réagir en modifiant leurs actions. Les robots mobiles Shakey à Stanford et Hilare au LAAS sont des exemples de telles machines capables d'utiliser la perception pour se déplacer dans le but de réaliser des tâches simples. Soutenus par l'avancée rapide des technologies de l'information et de la communication, les roboticiens cherchent à construire une machine autonome capable de prendre ses propres décision en fonction de ses capacités et de son environnement évolutif pour effectuer la tâche assignée. Un autre objectif est leur introduction dans un environnement humain pour en faire des robots de service ou personnels. L'interaction avec les humains et les autres robots se retrouve alors au coeur de la problématique et nécessite de nouvelles fonctionnalités.
Ce travail de thèse s'inscrit dans ce contexte de robots autonomes, et plus spécifiquement dans celui de la planification de la saisie pour la manipulation d'objets. Il y a plusieurs manières d'interagir avec une entité, la manipulation en constitue une de privilégiée. Nous nous intéresserons surtout aux deux tâches de manipulation représentatives de ces interactions : la tâche de prise et dépose d'un objet et la tâche d'échange d'un objet, voir Fig. 19. Ces deux activités nécessitent la conception et le développement d'un planificateur de saisie. Une dimension supplémentaire du problème est donnée par l'environnement très changeant de l'homme pouvant contenir des objets inconnus du système. Le planificateur doit alors prévoir une phase d'acquisition de données sur l'objet avant de calculer une saisie.

L'objectif de ce résumé en français est de donner un aperçu général de ce travail de thèse sans rentrer dans tous les détails, mais en développant les parties les plus importantes pour les francophones qui ne souhaitent pas lire tout le manuscrit en anglais ou ne maîtrisent pas bien cette langue. Nous suivons le même ordre que le document précédent mais présentons les résultats en même temps que la description des méthodes. Nous commençons par un état de l'art et quelques concept théoriques pour la planification de saisie et la manipulation, puis nous proposons une architecture pour résoudre les tâches mentionnées. Nous décrivons ensuite le planificateur de saisie qui est présenté en deux parties, la première décrit l'idée de base pour la recherche d'un ensemble de prises sur l'objet et une série de filtres permettant de choisir une bonne prise parmi cet ensemble. La deuxième propose une extension utilisant une stratégie de division de l'objet en parties convexes approchées pour traiter le cas des objets non convexes.

Figure 19 – Echange d'un objet, le planificateur génère deux saisies, une est exécuté par le robot et l'autre nous dit qu'au mois l'objet est saisissable, même si nous ne pouvons pas garantir que l'humain trouvera la même prise.

Manipulation : fondements et algorithmes pour la planification

Les robots manipulateurs mobiles sont le plus souvent constitués d'une plate-forme et d'un ou plusieurs bras manipulateurs munis d'un outil terminal. Ces manipulateurs sont des mécanismes articulés constitués de corps rigides décrits par la géométrie euclidienne. Le mouvement de ces corps articulés est représenté par les relations liant les repères attachés à ces corps au niveau des liaisons. La position relative de deux repères consécutifs est représentée par le modèle de Denavit-Hartenberg qui utilise quatre paramètres. Pour connaître la position dans l'espace cartésien d'une partie du robot par rapport à un repère en fonction des coordonnées articulaires du robot, une série de transformations est effectuée. Ces transformations constituent le modèle géométrique direct. Si au contraire, à partir d'une position cartésienne du robot nous voulons connaître les coordonnées articulaires correspondantes, le processus est connu comme le problème géométrique inverse, mais il constitue généralement un problème plus difficile. Pour une structure de robot bien définie, le problème peut se résoudre d'une manière analytique, sinon il faut faire appel à des méthodes numériques. Ces méthodes sont utilisés dans les algorithmes de planification.

Planification de mouvements

La planification pour la manipulation et la saisie d'objets constitue un sous-problème du problème général de la planification de mouvements. Dans les années quatre-vingt, la notion d'espace des configurations est introduite par Lozano-perez [Lozano-Perez 83b] dans la communauté des roboticiens. Il transforme la recherche d'un chemin sans collision pour un robot se déplaçant parmi des obstacles en une recherche de trajectoire pour un point se déplaçant dans l'espace des configurations CS_{free}. Nous pouvons classifier les approches pour la planification de mouvements en trois groupes : les décompositions cellulaires, les champs de potentiel et les roadmap ou carte de chemin [Latombe 91].

La première famille de méthodes essaie de décomposer le zone libre de l'espace des configurations en un certain nombre de régions appelées cellules. Un graphe de connectivité est construit pour représenter l'adjacence de ces cellules, la sortie des algorithmes est constituée d'une séquence de cellules, un chemin continu peut être obtenu à partir de cette séquence. La seconde famille de méthodes considère

le robot comme une particule évoluant dans un champ de force artificiel, cette particule est attirée par le champ de potentiel positif du but et repoussée par les champs négatifs des obstacles. La troisième et dernière famille construit un graphe qui capture la connectivité de la zone libre de l'espace des configurations. Les noeuds correspondent à des configurations sans collision et les arêtes définissent chacune un chemin de l'espace libre reliant deux noeuds. Un chemin reliant les noeuds associés aux configurations initiale et finale constitue alors une solution.

Au milieu des années quatre-vingt-quinze, une nouvelle approche est proposée. Au lieu de représenter explicitement les obstacles dans l'espace des configurations, de nouveaux algorithmes capturent la connectivité de CS_{free} par échantillonnage. Un graphe dont les noeuds représentent des configurations sans collision du robot et les arêtes les liaisons entre ces noeuds est alors construit en appliquant une méthode locale. Cette dernière dépend de la structure cinématique du robot et des contraintes du mouvement. Ces algorithmes de planification de mouvements par échantillonnage peuvent être divisés en deux grandes classes : les méthodes probabilistes aléatoires et les algorithmes d'accroissement.

Les premières se décomposent en trois phases, la phase d'apprentissage où échantillonnage construit la carte des chemins ou roadmap (voir la Fig. 20), la phase de recherche d'un chemin sur ce graphe et la phase de lissage. La phase de lissage essaye de simplifier le chemin pour que le robot effectue un mouvement plus direct vers le but. Il est évident que ce type d'algorithme a besoin d'un détecteur de collision très performant.

Les algorithmes d'accroissement encore dits de diffusion reposent sur l'idée de créer une carte de chemin sous forme d'arbre en capturant la connectivité de CS_{free}. La racine de cet arbre correspond à la configuration initiale, l'algorithme échantillonne l'espace pour trouver de nouvelles configurations correspondant à de nouveaux noeuds de l'arbre, puis de nouveaux arcs reliant ces noeuds. Après un certain temps, l'algorithme vérifie si la configuration finale peut être atteinte à partir d'un noeud. Une variante intéressante consiste à construire deux arbres en même temps en partant simultanément des configurations initiale et finale. L'algorithme s'arrête lorsque les deux configurations sont reliées ou qu'un nombre d'essais maximum a été atteint. Une description plus détaillée est présentée dans [Choset 05].

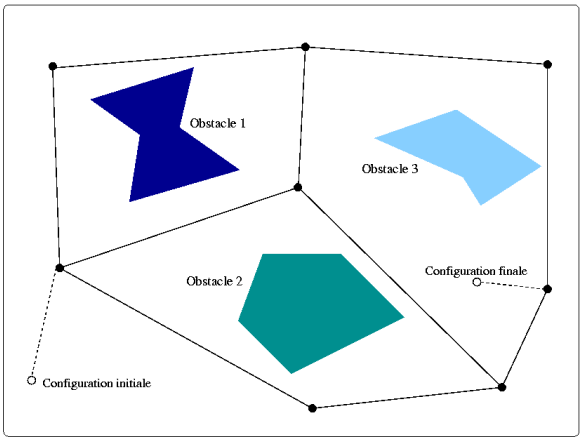


Figure 20 – Roadmap ou carte de chemin pour la planification de mouvements

Planification pour la manipulation

La planification pour la manipulation fait partie du problème de la planification de mouvements, elle introduit des objets mobiles au problème du paragraphe précédent où les objets sont statiques. Un plan de manipulation est une séquence alternant des mouvements dits de transit et de transfert. Sur un chemin de transit, le robot se déplace seul tandis que sur un chemin de transfert le robot déplace aussi un objet. Ces deux types de chemins appartiennent à des espaces différents, le problème consiste a trouver un chemin sans collision dans un espace de configurations combinées. Cet espace est défini comme le produit cartésien $CCS = CR \times CO_1 \times \cdots \times CO_m$ entre l'espace de configuration du robot et les espaces de configuration d'objets.

Deux contraintes supplémentaires sont nécessaires, elles concernent la stabilité de la saisie des objets et la stabilité des positions de pose des objets. Pour cela, deux sous espaces appelés $GRASP$ et $PLACEMENT$ sont construits.
Leur intersection défini un troisième sous-espace $GRASP \cap PLACEMENT$. La planification peut alors être définie comme la recherche d'un chemin sans collision dans les composantes connexes de ces trois sous-espaces.Un graphe de manipulation dont les noeuds sont les composantes connexes de $GRASP \cap PLACEMENT$ et les arcs correspondent à des chemin de transit ou de transfert, peut alors être construit. Le problème de la planification des manipulations d'objet a été formulé ainsi pour la première fois dans [Alami 89].

La manipulation commence généralement par la saisie de l'objet à manipuler, elle nécessite la planification de la prise et du mouvement d'approche. La prise s'effectue par l'application de forces aux points de contact en utilisant ou non le frottement. Le modèle de contact entre les doigts du préhenseur et l'objet peut être sans friction, l'effort est alors appliqué dans la direction de la normale à la surface de contact, ou avec friction. La friction introduit un effort tangent et éventuellement un moment de pivotement autour de la normale dans le cas de doigts "mous". Pour ce travail, nous avons choisi le modèle avec friction et utilisons le modèle de Coulomb. Ce modèle suppose que la force de frottement reste dans un cône de friction défini par le coefficient de friction qui dépend des matériaux en contact. Les contraintes non linéaires introduites par ce modèle peuvent être approchées par un cône polyédrique convexe, voir Fig. 21.

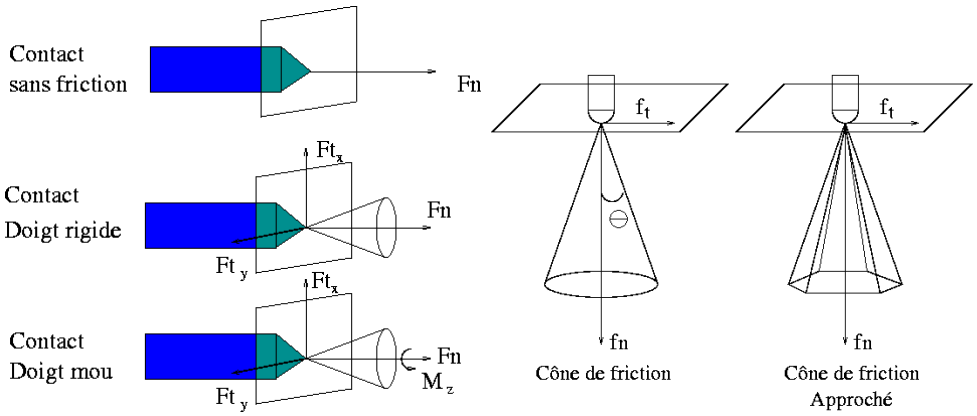


Figure 21 – Modèles de contact et les forces et moments associés pouvant être appliqués, ainsi que le cône de friction dû au modèle de Coulomb.

Planification des saisies

La version la plus simple de la planification de saisie correspond à la recherche des points de contact sur la surface de l'objet. Nous étendrons cette formulation pour définir une saisie par les points de contact, un repère de prise et le mouvement pour aller prendre l'objet. Nous prenons aussi en compte la structure cinématique du préhenseur. Il existe des approches qui considèrent aussi le mouvement de dégagement [Laugier 85]. Il semble être accepté que quatre points de contact avec friction sont suffisants pour maintenir en équilibre stable n'importe quel objet, et même que trois points de contact sont suffisants si on considère des objets ayant une surface dérivable. Pour nos expérimentations, nous utilisons une pince à deux mors parallèles avec trois doigts terminés par une demi sphère.

Pour la planification des prises, une propriété importante est la fermeture de force qui définit des prises pouvant résister à n'importe quel effort extérieur sous l'hypothèse d'efforts de contact infinis. Les forces extérieures sont constitués d'une force et d'un moment regroupés dans un torseur d'effort. Des algorithmes existent pour tester la condition de fermeture de force de façon qualitative et quantitative. Ils utilisent l'espace des torseurs de forces et la géométrie convexe. La fermeture de force implique que les combinaisons linéaire positive des torseurs de contact parcourent tout l'espace des forces [Salisbury 85].

Il a été montré dans [Mishra 87] de manière équivalente que la démonstration de la fermeture de force peut se ramener à montrer que l'origine de l'espace des forces est à l'intérieur de l'enveloppe convexe d'un ensemble de torseurs. La plupart des algorithmes transforme le problème de vérifier si une prise définit une fermeture de force en un problème de programmation linéaire [Liu 99]. Les algorithmes quantitatifs sont basés sur le théorème de Steinitz qui a été appliqué à la robotique par [Kirkpatrick 91], ce théorème nous permettre de prendre le rayon d'une sphère contenue dans l'enveloppe convexe construit à partir des torseurs, comme la valeur de la efficacité de la prise.

Le module du plus grand torseur extérieur auquel puisse résister une prise qui appliquerait une force unitaire à chaque contact donne une image de l'efficience de la prise. Dans [Ferrari 91], deux mesures basées sur ce théorème sont présentées pour les cas où la prise utilise la friction. Le premier correspond au torseur maximum pouvant être obtenu par des forces de contact bornées, la mesure correspond à la direction où le torseur est minimum. Le deuxième cherche le minimum de la somme des efforts de contact. Ce deuxième critère peut être interprété comme la minimisation du couple nécessaire au niveau des moteurs des actionneurs des doigts de la pince.

Parallèlement à ces algorithmes de test et de comparaison, des algorithmes de synthèse de prises on été développés à partir des mesures précédentes ou de la formulation initiale du problème, ils permettent de calculer ou placer des points de contacts sur la surface de l'objet. Une approche par la résolution d'un problème linéaire ou non linéaire est proposé dans [Ding 00a] en considérant des restrictions. Une approche géométrique consiste à placer les points de contact au centre des faces de l'objet [Mishra 87]. Il est aussi possible de ne pas rechercher des points de contact, mais des régions indépendantes qui satisfassent la condition de fermeture de force. Le premier a avoir travaillé dans cette direction est Nguyen [Nguyen 88], suivi par les travaux de Ponce [Ponce 97] qui construit des régions maximales. La méthode caractérise les prises par un système linéaire de contraintes sur la position des points de contact dans des facettes polygonales de l'objet. Les régions satisfaisant ces contraintes sont calculées par une projection de polytopes basée sur une réduction de paramètres linéaires, en maximisant la longueur des arêtes des régions et en minimisant la distance entre le centre de la région et le centre de masse de l'objet.

Aucun de ces algorithmes ne prend en compte les mécanismes qui appliquent les forces et d'un point

de vue pratique ils ne sont ni complets ni utilisables, à cause de leur complexité algorithmique, pour une intégration sur un robot expérimental.

Approches heuristiques

Les approches heuristiques semblent constituer une option séduisante et permettent parfois de trouver un compromis entre complexité et robustesse. Nous en mentionnons trois qui nous ont paru intéressantes. La méthode des cylindres elliptiques généralisés crée des coupes de l'objet de manière similaire au examen du cerveau par tomographie. Chaque tranche contient un contour 2D qui est encapsulé par une ellipse. Les directions des ellipses sont définies par des directions privilégiés comme les axes d'inertie ou des directions particulières telle que la verticale. Si l'ellipse n'englobe pas étroitement le contour, celui-ci peut être représenté par un polygone et divisé en sous-composantes en coupant le polygone par les points de concavité. Des cylindres sont générés à partir d'ensembles de tranches où les ellipses ont des directions et des tailles similaires. Les directions des cylindres constituent les directions de saisie. Un préhenseur multi-doigts peut être positionné à partir de cette taxonomie [Bard 90].

Une autre méthode vise à trouver un axe, appelé axe naturel de prise, à partir de la géométrie de l'objet. Cet axe correspond à la direction parallèle aux paires de facettes du maillage de l'objet et qui sont face à face. Cet axe minimise un critère quadratique basé sur la somme des angles entre les normales des paires de facettes et l'axe. Comme pour la méthode précédente, le préhenseur est pré-configuré et l'axe donne deux directions de saisie. Une suit la direction de l'axe et l'autre est définie par la direction perpendiculaire [Michel 05].

La dernière méthode que nous décrivons consiste à construire une approximation de l'objet à partir d'un ensemble de primitives comme des cubes, des cylindres et des sphères, pour n'en citer que quelques unes. Pour chaque primitive, une direction d'approche et des points de contacts sont calculés pour définir un ensemble de prises parmi lesquelles une prise sera sélectionnée [Miller 03].

Partant de démonstrations ou l'homme montre comment saisir un ensemble restreint des objets, il existe des approches qui cherchent à reproduire ces mouvements en les adaptant aux manipulateurs et préhenseurs [Kang 97b]. Ces techniques d'apprentissage peuvent être utilisées pour essayer de prendre toute une série d'objets ayant des formes similaires en reproduisant le même type de prise [Ekvall 04].

Planificateur pour des tâches de manipulation interactive

La littérature présente quelques systèmes intégrés pour des tâches de manipulation. Ces intégrations correspondent soit à des modules limités à la planification, soit à des modules pour l'exécution consistant à ajouter des éléments de nature diverses comme, par exemple, un module pour la vision. Un des premiers planificateurs est Handey [Lozano-Perez 87], développé pour effectuer des opérations de prise et dépose. Il est notamment composé d'un module de vision pour la localisation des objets et d'un module de planification de prise qui prend en compte la position de dépose. Si la configuration finale de l'objet n'est pas accessible en utilisant la même prise que celle choisie pour la saisie initiale, un module de reprise est activé. Ce module reconfigure la prise de l'objet en réalisant une dépose dans une position stable qui permette une saisie compatible avec la position de dépose finale.

SHARP [Laugier 85] intègre trois modules de planification, un pour la génération de prises robustes, un pour le calcul des trajectoires de mouvements de transfert et finalement un pour les mouvements basés sur des capteurs. SPARA [Mazon 90] est aussi composé de trois modules de planification. Le premier

concerne la prise, le deuxième est un planificateur de mouvement global pour transférer l'objet et le troisième définit le mouvement local pour prendre et déposer l'objet parmi les obstacles éventuels.

Nous reprenons quelques idées de ces planificateurs et nous décrivons un planificateur orienté vers la résolution de tâches de manipulation interactives. Nous nous intéressons plus particulièrement à l'échange d'objets qui nécessite des opérations de prise et de dépose d'objets. Pour cela, nous proposons une architecture composée de modules qui interagissent avec un contrôleur, voir Fig. 22.

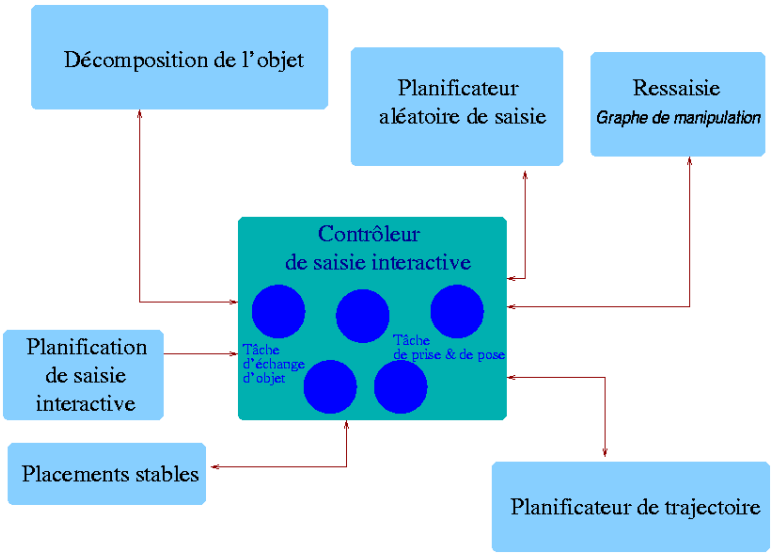


Figure 22 – Architecture modulaire pour un planificateur de manipulation interactif

Nôtre planificateur de tâches de manipulation est constitué de plusieurs modules de planification basés sur des algorithmes d'échantillonnage probabilistes. Ceci nous permet de travailler avec des mécanismes plus complexes que ceux présentés pour les planificateurs précédents. L'architecture est composé des modules suivants :

- Contrôleur de tâches : c'est la partie qui régule tout le fonctionnement, le contrôleur peut être vu comme un ensemble de machines à états où un automate gère chaque tâche de planification. Les transitions entre états dépendent de drapeaux qui indiquent si l'opération a réussi à trouver une solution. Chaque état de la machine correspond à une action effectuée par des opérateurs. Les actions sont réalisés par les modules. Les données de sortie de chaque module dépendent de l'opération effectuée. Les modules échangent les données dont ils ont besoin au moyen d'opérateurs.
- Opérateurs et drapeaux : Pour effectuer une action, le contrôleur utilise les opérateurs. Ceux-ci prennent en entrée le type d'action à effectuer et les données nécessaires pour la réaliser. A la sortie les opérateurs délivrent un drapeau indiquant si l'action a ou non réussi et, selon le cas, les données correspondantes.
- Planification de saisie aléatoire : ce module fournit en sortie à la fois la prise ou un ensemble de prises, la configuration de prise pour le robot et le chemin pour aller prendre l'objet.
- Génération de deux prises : pour les tâches de manipulation interactives, il est nécessaire que deux robots saisissent simultanément le même objet. Ce module génère ces deux prises et les deux saisies correspondantes. Une partition de l'objet peut être demandé par ce module.

- Poses stables : les poses stables sont calculées en prenant la projection des facettes en contact dans le plan support et en calculant si la projection du centre de gravité se trouve dans l'enveloppe convexe du contour polygonal, appelé polygone de sustentation.
- Planification de mouvements : les mouvements de transit et de transfert du robot avec l'objet sont planifiés en utilisant les techniques de cartes de chemins aléatoires probabilistes par échantillonnage ou par arbres d'exploration rapides.
- Test de compatibilité : ce module vérifie qu'une même prise peut être utilisée par le planificateur pour saisir l'objet dans sa configuration initiale et finale. En définissant la prise comme une contrainte à respecter par le robot, si la génération aléatoire, pour cette prise, d'une configuration du robot sans collision est possible pour la configuration finale de l'objet, alors la prise est compatible.
- Planification de ressaisie : si le système ne trouve pas de prise compatible pour les configurations initiale et finale, l'objet peut être reconfiguré un certain nombre de fois pour trouver une série de prises compatibles. Ce module utilise les données du module de poses stables.
- Décomposition d'objets : ce module calcule une partition en éléments volumiques de l'objet. Ces éléments sont utilisées comme entrées pour le planificateur de saisie.

Tâche de pose et dépose d'objets

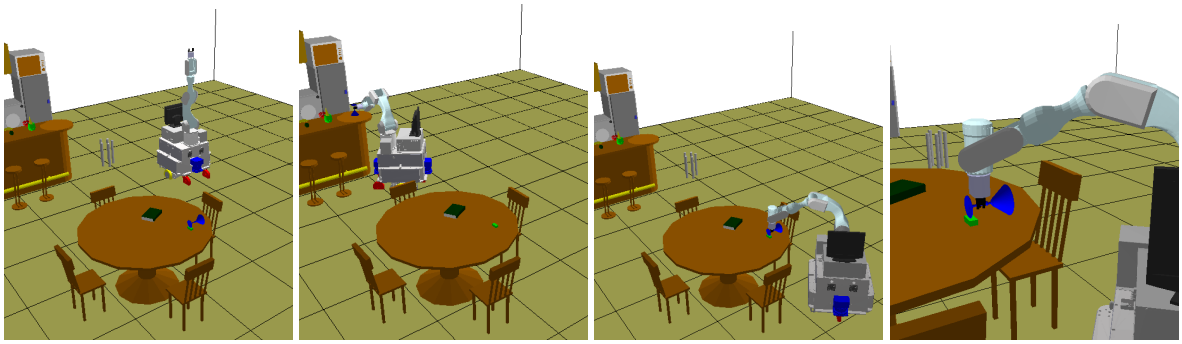
Pour la tâche de manipulation consistant à prendre l'objet puis à le déposer (Fig. 23), au départ, la machine à états est dans l'état initial qui correspond à la configuration initiale du robot et de l'objet. L'état suivant qui représente le début du calcul est la génération de la saisie. Si une prise et la configuration du robot associée sont trouvés, le système passe dans l'état de transit et demande la planification d'un chemin pour amener le préhenseur de l'état initial à la configuration de prise. En cas d'échec, si le planificateur n'a pas trouvé de configuration de prise, la machine passe dans l'état échec et arrête le processus.

Une fois que le robot s'est déplacé et a saisi l'objet, le système est dans l'état saisie et teste la compatibilité de la prise. Si elle est compatible, cela signifie que le planificateur a trouvé une configuration de dépose de l'objet pour cette prise et le planificateur peut évoluer vers l'état de compatibilité. Si le test échoue, l'état de ressaisie est activé. Il donnera en sortie une configuration de prise supplémentaire et la configuration de pose compatible ou se placera en échec. Une demande de planification du mouvement de transfert pour amener l'objet de sa position de saisie à la position de dépose est effectuée. Si un chemin est trouvé, la machine passe dans l'état d'objet posé. L'avant dernier état est alors atteint et correspond au mouvement de transit pour amener le robot dans sa configuration finale permettant d'atteindre le dernier état de fin de tâche. Dans le cas où les réponses sont négatives la machine évolue vers l'état d'échec.

Approche par chaînes cinématiques

Comme les objets ne peuvent pas se déplacer seuls, le robot doit leur appliquer des actions. Le problème de la planification des mouvements du robot pour déplacer un objet a été résolu dans la communauté de robotique en planifiant le chemin pour l'objet puis en cherchant une configuration du robot qui satisfasse la contrainte de prise de l'objet. Pour cela, à la description du robot nous devons ajouter celle de l'objet. Le robot est vu comme une chaîne cinématique qui peut être divisé en chaînes actives et passives (Fig. 24). Une chaîne active fait évoluer le système, une chaîne passive utilise son modèle géométrique inverse pour rejoindre la chaîne active. Cette stratégie est utilisé pour résoudre les problèmes de planification pour les mécanismes qui forment une chaîne cinématique fermée [Lavalle 99] [Han 00].

Dans notre cas, lorsque le robot saisit l'objet posé, il forme un chaîne fermée. Au moment de soulever l'objet, cette chaîne est rompue, nous pouvons alors utiliser ce type d'algorithme en déclarant l'objet comme partie active et le manipulateur comme la chaîne passive pour calculer les configurations du robot entier par une technique aléatoire. Nous avons choisi d'utiliser l'algorithme de génération aléatoire pour chaînes fermées développé dans [Cortés 02] [Cortés 03a] pour le module de planification des mouvements de transfert.

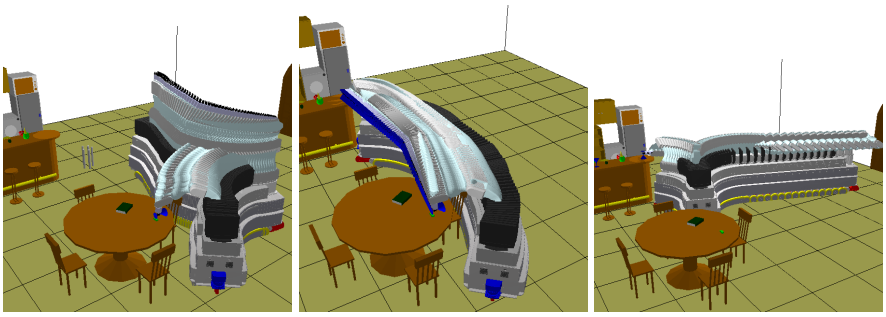


(a) Configuration initiale

(b) Configuration finale

(c) Configuration de saisie

(d) Configuration de saisie agrandie



(e) Chemin du mouvement de prise

(f) Chemin du mouvement de transfert

(g) Chemin vers la position finale

Figure 23 – Actions pour une tâche de prise et dépose d'un objet.

Pour cette méthode, la chaîne active est considérée dans le phase d'échantillonnage du planificateur, elle utilise la cinématique directe et la cinématique inverse des chaînes passives comme nous l'avons déjà mentionné. En prenant en compte l'espace de travail des deux chaînes, l'algorithme augmente la probabilité de générer des configurations de la chaîne active qui peuvent être atteintes par les chaînes passives. L'espace de travail est décrit par les intervalles admissibles par les coordonnées articulaires positionnant les corps de la chaîne cinématique. La valeur d'une articulation conditionne les valeurs possibles pour les articulations suivantes de la chaîne, ces valeurs constituent des intervalles de fermeture. Les intervalles sont calculés en considérant deux sphères concentriques et en calculant leur intersection. Les rayons des sphères correspondent à la distance entre les extrémités de la chaîne lorsque celle-ci est

en extension maximale d'une part et en extension minimale d'autre part.
Nous supposons qu'il existe un cône pour lequel le sommet coïncide avec le centre commun des deux sphères. Le volume d'intersection entre le cône et les deux sphères détermine alors l'intervalle. Les valeurs pour chaque articulation de la chaîne active sont calculées de façon séquentielle. En général, il existe plusieurs solutions, ainsi les configurations sont obtenues par la combinaison des configurations de la chaîne active avec les différentes solutions de la chaîne passive. La planification pour les mouvements de transit et de transfert est effectuée avec la même méthode mais en activant ou désactivant la contrainte de prise.

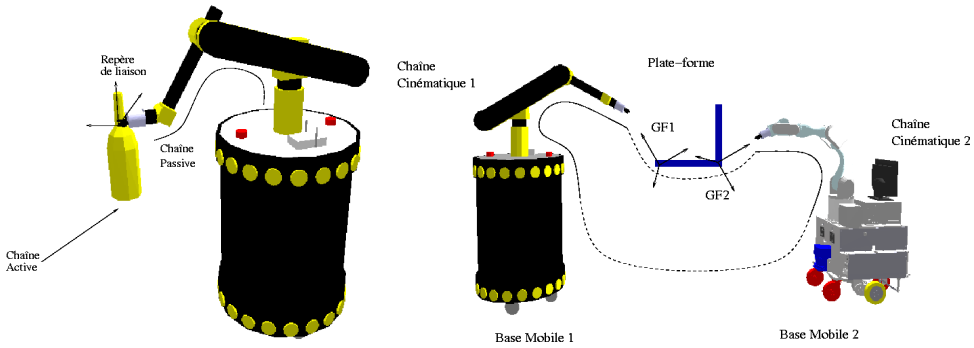


Figure 24 – Division des systèmes en chaînes actives et passives pour les tâches de manipulation

Tâche d'échange d'objets

Pour la tâche d'échange d'un objet (Fig. 25), nous souhaitons qu'un robot appelé donneur saisisse un objet puis le donne à un autre robot appelé receveur. Pendant que les deux robots tiennent tous les deux l'objet, un mécanisme parallèle est formé. La chaîne cinématique fermée est constituée de l'objet, des deux préhenseurs, de la base mobile de chacun des robots et de leur bras. Comme les bases mobiles sont en contact avec le sol, le système parallèle constitue une chaîne cinématique fermée. Nous pouvons donc diviser le robot en chaînes actives et passives puis encore rediviser ces chaînes en sous-chaînes actives et passives. Par exemple, dans nôtre cas, la chaîne active est l'objet et les chaînes passives correspondent aux robots, mais chaque chaîne passive est divisée en une chaîne active correspondant à la base mobile et une chaîne passive correspondant au bras manipulateur, voir Fig. 24.

Les configurations du robot parallèle sont basées sur la méthode précédente. Dans ce cas, l'espace de travail est calculé à partir de chaque espace de travail des chaînes cinématiques et de la position et de l'orientation de l'objet. Ce type d'approche a été utilisé pour la planification de mouvements de co-opération entre mannequins virtuels [Esteves 06]. L'algorithme est utilisé avec un planificateur de mouvements probabiliste d'échantillonnage. Dans la phase d'apprentissage de ce planificateur chaque noeud représente une configuration du robot parallèle sans collision. Ces configurations sont prises comme lieux d'échange. Les actions à réaliser pour ce type de tâche de manipulation interactive sont similaires à celles des tâches de prise et dépose. Par conséquent, la machine à états est aussi similaire avec quelques différences au niveau des opérations demandées pour quelques états.

L'état de génération de saisie demande deux opérations : la génération d'un ensemble de prises sur l'objet et la génération de la prise et de la configuration de saisie pour le robot donneur. Deux états

supplémentaires sont introduits, l'un pour la génération des configurations d'échange et l'autre pour l'état de transit du robot récepteur afin qu'il rejoigne la configuration d'échange à partir de sa configuration initiale.

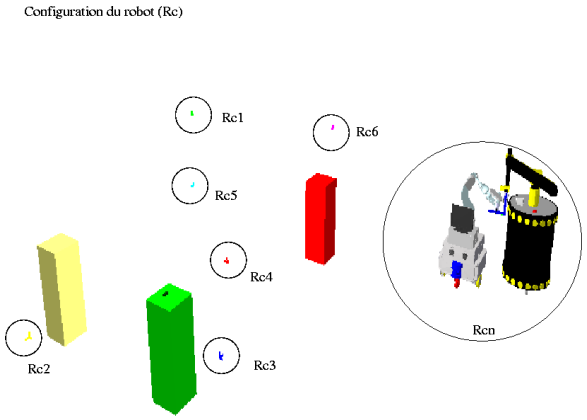


Figure 25 – Plusieurs configurations Rc_i peuvent être générées qui donnent les lieux où une opération d'échange peut être exécutée. En utilisant une méthode probabiliste et aléatoire, les noeuds du graphe (configuration du robot) sont trouvés dans la phase d'apprentissage.

Un planificateur de saisie aléatoire

Comme nous l'avons vu dans les derniers paragraphes, un planificateur de saisie est nécessaire pour une grande partie des manipulations d'objets qui nécessitent une prise. Rappelons que la manipulation d'un objet peut s'effectuer de différentes manières, le robot peut aussi pousser l'objet ou le soutenir sans le serrer entre des doigts.

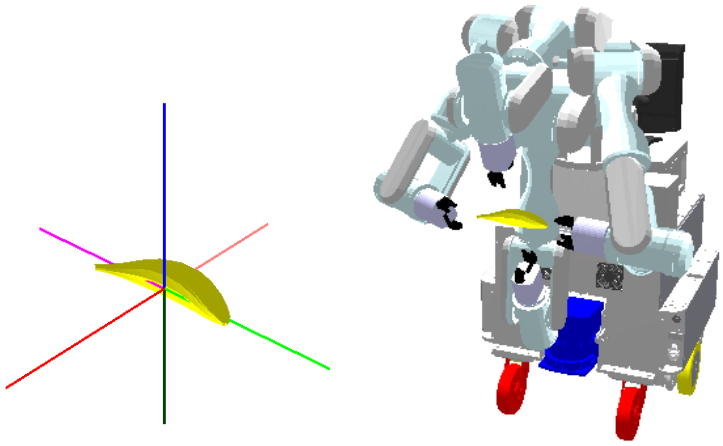
Nous nous focalisons sur la manipulation d'objets par prise. Ces prises sont calculées pour un préhenseur muni de trois doigts qui satisfont la condition de fermeture de force. La stratégie suivie est la planification de prises aléatoires guidée par des propriétés inertielles de l'objet [Lopez-Damian 05a]. Nous supposons l'existence de plusieurs prises [Fischer 97] et nous ne chercherons pas à obtenir la prise optimale pour accomplir la tâche [Borst 03]. Nous proposons un planificateur de prises pour objets inconnus. Les objets étant modélisés par un maillage triangulaire. Tous les algorithmes proposés prennent en compte ces hypothèses.

Le planification aléatoire suppose que le système peut générer un nombre suffisant de prises et choisir la meilleure parmi celles-ci au sens d'un critère de qualité. En raison de cette hypothèse, pour effectuer une tâche de manipulation, le système utilise des prises qui ne sont pas forcément optimales.

En pratique, une prise optimale de l'objet peut être sans intérêt car non accessible par le manipulateur. Une étude statistique [Borst 03] a montré qu'une prise de qualité moyenne est une prise acceptable. De ceci il découle que la génération de prises aléatoires est appropriée pour la planification de saisie dans le contexte de la manipulation. Une façon rapide de trouver une prise pour un objet est l'utilisation d'heuristiques.

Au lieu de générer simplement des points de contact aléatoires, nous proposons une stratégie un peu différente. L'objet étant principalement sujet à la pesanteur et aux forces d'accélération qui agissent toutes deux sur le centre de masse, il est préférable de saisir l'objet autour de ce point. Les forces et couples appliqués à l'objet pour le maintenir en position seront plus faibles. En suivant la même idée, de bons candidats pour la direction d'approche peuvent être donnés par les directions des axes principaux d'inertie de l'objet, voir Fig. 26.

Avec notre planificateur, quand une saisie est trouvée, cela signifie que le robot complet peut approcher l'objet et le prendre, peu importe où le robot était localisé avant. Ceci permet au planificateur de saisie d'être utilisé directement avec un planificateur de chemins pour aller d'une configuration courante vers une configuration de saisie. Le problème de la planification de saisie devient un processus qui non seulement calcule des prises mais aussi les mouvements d'approche et de saisie de l'objet, voir Fig. 30.



(a) Axes d'inertie

(b) Prises dans les directions des axes inertiels

Figure 26 – Six axes inertiels et le centre de masse calculés à partir de l'objet et quatre directions de prise représentatives suivant quatre axes inertiels.

Algorithme de planification de saisie

Nous définissons une prise par les points de contact sur la surface de l'objet et un repère que nous appelons le repère de saisie. L'algorithme 17 commence par vérifier si la distance entre la base du bras manipulateur et l'objet permet au préhenseur d'atteindre l'objet ou s'il est nécessaire que la base effectue un mouvement d'approche. Le mouvement d'approche est obtenu en générant une configuration aléatoire de la plate-forme, mais en interdisant les configurations incompatibles avec la saisie de l'objet. Cette configuration doit se trouver à l'intérieur d'une aire comprise entre deux cercles situés autour de l'objet et représentant l'espace de travail du bras manipulateur. A partir de cette configuration initiale du robot localisée près de l'objet à manipuler, le système commence la génération de la saisie. Celle-ci correspond aux trois étapes : la génération des prises, le filtrage et l'assignation d'une valeur de qualité pour les prises.

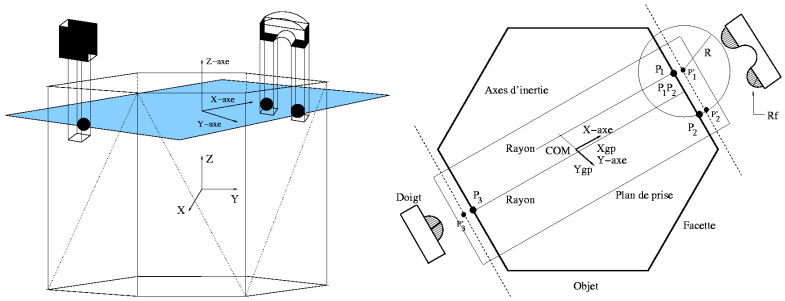
Génération de prises

La génération est divisée en deux phases : la première correspond au calcul des repères de prises. Tout d'abord les axes d'inertie et le centre de masse de l'objet sont calculés. Ils fournissent les directions de recherche pour les prises. Douze repères sont générés à partir du centre de masse et des les directions principales d'inertie. A partir de ces premiers repères, d'autres repères de saisie sont aussi générés soit par une translation, soit par une rotation suivant les axes d'inertie ou par une combinaison des deux.

Algorithm 17: Principe de l'algorithme pour le planificateur de saisie

Entrée : Modèle géométrique : robot R, Outil terminal Ot, environnement E et l'objet O
Sortie : Saisie S et configuration du robot q_S
begin
  GÉNÉRATION_POSITION_PLATEFORM_MOBILE;
  repeat
    GÉNÉRATION_ALÉATOIRE;
    if S *Satisfait* FILTRES **then**
      ASSIGNATION_VALEUR_QUALITÉ $S \leftarrow Q$;
      AJOUTE_NOUVELLE_SAISIE $Liste_S \leftarrow S$;
    end
  until $Saisies ==$ NOMBRE_PREDÉFINI;
  $S \leftarrow$CHOISIR_MEILLEURE_SAISIE;
  RETOUR(S, q_S);
end

Le deuxième phase correspond au calcul des points de contact associé à un repère de saisie. Ce calcul prend en compte la structure cinématique du préhenseur et nécessite donc une adaptation pour chaque préhenseur. Dans ce travail nous présentons deux exemples, une pince à deux mors parallèles et trois doigts à bouts sphériques et la pince de la société Barrett qui comporte trois doigts articulés. Pour la première pince, les points de contact sont calculés en construisant des droites à partir du repère de saisie et en calculant les intersections entre ces droites et la surface de l'objet. Le repère de saisie est réorienté pour s'adapter à la surface de l'objet et les dimensions de la pince jouent un grand rôle, voir Fig. 27.



(a) Vue 3D d'une saisie

(b) Vue 2D : saisie avec points d'intersection

Figure 27 – Génération des points de contact à partir d'un repère de saisie

Pour la pince Barrett, chaque doigt a deux articulations. Un des trois doigts est fixe et les deux autres peuvent tourner symétriquement de 180 degrés autour de la paume. Les positions limites correspondent à la situation où le doigt fixe a la même orientation que les deux doigts mobiles, et à celle où il leur est opposé. Le mécanisme est actionné par quatre moteurs, ce qui implique que pour chaque doigt les deux articulations sont couplées par un mécanisme interne. Cette pince peut être décomposé en un mécanisme de fermeture des doigts et un mécanisme d'orientation des doigts autour de la paume. L'algorithme génére plusieurs angles d'orientation pour chaque repère de saisie.

Les points de contact sont ensuite calculés pour chacune de ces orientations. Chaque doigt est considéré comme un manipulateur planaire a deux articulations couplées. Nous cherchons l'intersection entre la trajectoire décrite par le doigt et la ligne définissant la surface de l'objet dans le plan du doigt. Nous calculons d'abord l'intersection entre cette ligne et les deux cercles extrèmes correspondant au doigt en extension et au doigt replié. Nous recherchons ensuite le point d'intersection entre ces deux points, voir Fig. 28.

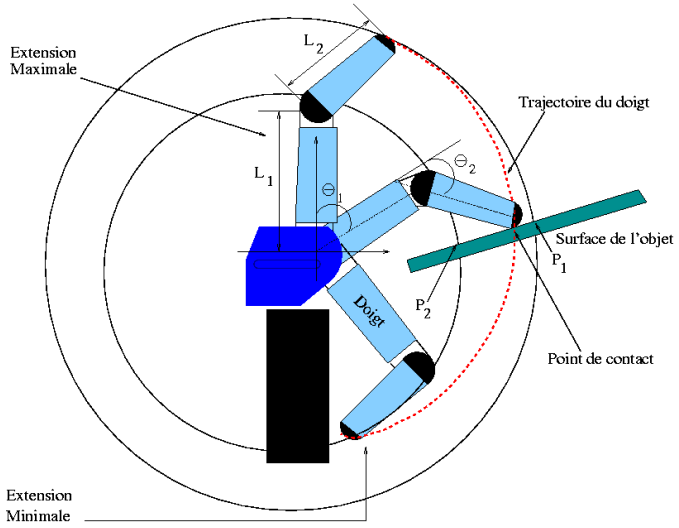


Figure 28 – A partir des configurations repliée et en extension, deux trajectoires circulaires sont générées pour calculer le point de contact qui est l'intersection de la trajectoire réelle et de la surface de l'objet

En égalent l'équation paramétrique de la surface et l'équation de la courbe de la trajectoire planaire du doigt, nous obtenons l'équation à une inconnue (θ_1).

$$D(L_1 \cos\theta_1 + L_2 \cos(\theta_0 + (1+\tau)\theta_1) - A) - B(L_1 \sin\theta_1 + L_2 \sin(\theta_0 + (1+\tau)\theta_1) - C) = 0 \quad (1)$$

Les termes A,B,C et D correspondent aux paramètres de l'équation d'une droite. Comme les deux parties de chaque doigt de la pince sont couplées, τ est la proportion de mouvement de la partie supérieur du doigt par rapport au mouvement de la partie inférieur. Finalement, θ_0 est la valeur initiale du position du doigt.

Cette équation est résolue en utilisant une méthode numérique, la solution nous permet de trouver les coordonnées du point d'intersection du doigt. La répétition de ce calcul au deux autres doigts permet de calculer les trois points de contact.

Move3D considère que deux objets en contact sont en collision, il est donc impossible de planifier un chemin directement. Nous pouvons nous approcher à une faible distance de la position de contact, cette configuration approchée constitue une bonne approximation de la position de prise.

Filtrage

Comme le calcul du critère de qualité est coûteux en temps de calcul, nous introduisons cette étape pour éliminer le plus tôt possible les prises qui ne sont pas réalisables. Quelques contraintes sont imposées par le système : la prise doit être géométriquement accessible par le robot, il doit exister une configuration du robot sans collision avec les obstacles pour atteindre la prise. Pour garantir que l'objet puisse être tenu fermement sans glisser, un test de fermeture de force est effectué.
L'algorithme de fermeture de force utilisé considère trois points de contact avec friction. Nous utilisons le modèle de friction de Coulomb.

Une prise satisfait la condition de fermeture si ces trois points : a) forment un plan et le cône de friction coupe le plan en définissant deux vecteurs unitaires, et b) les vecteurs unitaires forment une fermeture de force dans le plan.

En partant de la proposition qui établit que trois forces de contact qui ne sont pas parallèles entre elles et sont situées dans des cônes de friction atteignent l'équilibre si elles couvrent le plan et si les lignes d'action s'intersectent en un point. Li [Li 03] propose de substituer les forces inconnues par les vecteurs de frontière des cônes de friction et établit une nouvelle proposition. L'algorithme commence par l'élimination de régions dans les cônes de friction qui ne contribuent pas à l'équilibre. Ce processus est appelé *disposition H*.
La proposition dit alors que les forces sont en équilibre si l'intersection des trois cônes de friction n'est pas vide après l'opération de la disposition H.

Ramené au niveau des points d'intersection entre les droites limites des cônes de friction, une nouvelle proposition dit que les trois forces forment une fermeture de force si au moins un des points d'intersection de deux droites frontières des cônes de friction se trouve à l'intérieur du troisième cône [Li 03].

La vérification de la fermeture à partir de ce théorème est très rapide.

Mesure de qualité

Plusieurs prises peuvent être produites après les deux premières étapes ; l'étape finale est l'assignation d'une valeur de qualité aux prises. La mesure de qualité reflète l'efficacité de la prise et permet de classer les prises pour en choisir une. Nous utilisons le critère de qualité développé par Ferrari et Canny [Ferrari 91] ; le critère essaye de mesurer la notion d'une prise efficiente. L'index de qualité indique la résistance d'une prise de force unitaire pour le pire cas de perturbation. Comme nous l'avons déjà dit, nous considérons des points de contact avec friction suivant le modèle de doigt dur.

Le cône de friction est approximé par un cône polyédrique convexe représenté par un ensemble fini de vecteurs. Si nous supposons que chaque cône de friction a une hauteur unitaire, alors nous pouvons construire l'ensemble des torseurs de forces pouvant être générés par les efforts unitaires des arêtes de ces cônes polyédriques. L'enveloppe convexe de ces torseurs dans l'espace des torseurs de forces peut ensuite être construite. La valeur de qualité de la prise est alors égale à la distance à l'origine de la facette de l'enveloppe convexe la plus proche de l'origine, voir Fig. 29. Le torseur associé à cette direction est le plus difficile à appliquer par la prise. Il est important d'observer que la force et le moment ont des dimensions différentes, un paramètre multiplicatif est utilisé pour obtenir une mesure de qualité indépendante de la taille de l'objet.

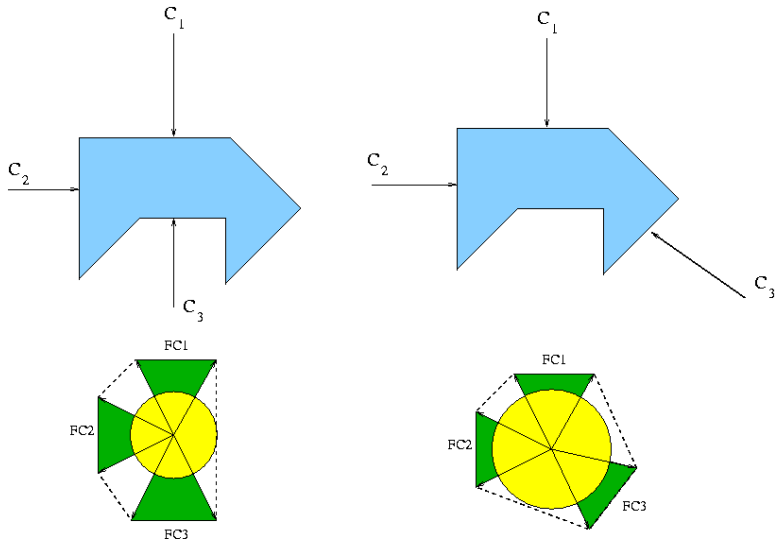
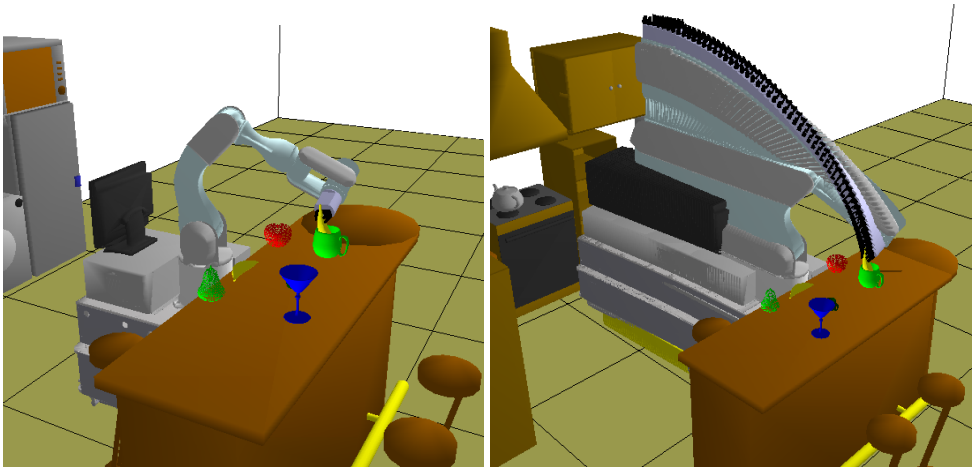


Figure 29 – Espace de force de saisie, métrique en 2D

L'utilisation de l'espace des torseurs n'est pas la seule méthode pour définir une mesure de qualité des prises. Nous pouvons par exemple souhaiter pénaliser les prises en fonction de l'éloignement des points de contacts avec la frontière des facettes de l'objet. Tant que les points de contacts sont proches des dites frontières, les prises reçoivent une mauvaise valeur de qualité. Pour cela, nous définissons des régions autour du centre de la facette, et leur associons une valeur ou poids que nous utilisons en combinaison avec d'autres mesures de qualité.



(a) Banane à l'intérieur d'une tasse (b) Chemin pour le robot

Figure 30 – a) La configuration de saisie du robot est montrée ici pour une banane à l'intérieur d'une tasse, b) Le mouvement d'approche sans collision pour la saisie de l'objet.

Décomposition d'objets pour la planification de prises

Parfois l'objet à saisir possède une forme complexe ou une taille importante mettant en échec le planificateur de saisie précédent. Pour contourner ce problème, nous proposons d'introduire une stratégie de décomposition de l'objet pour obtenir des sous-composants susceptibles d'être saisis. Le problème de la décomposition n'est pas nouveau en géométrie algorithmique qui propose la décomposition de l'objet en éléments convexes. Cette approche donne, dans les cas pratiques, une multitude de petites composantes convexes que le préhenseur ne peut pas utiliser pour saisir l'objet. Nous utilisons l'approche de la décomposition convexe approchée basée sur les travaux d'Amato [Lien 03] [Lien 05]. Ceci nous permet d'obtenir des composants de l'objet suffisamment grands pour être saisis. En intégrant cette étape de décomposition dans le planificateur de saisie nous le rendons plus robuste. La même stratégie peut être appliquée pour calculer les deux prises nécessaires à deux robots devant échanger un objet.

Méthodes de décomposition

La première méthode de décomposition [Chazelle 84] ou méthode basique consiste à résoudre toutes les concavités associées à une entaille (arête concave du polyèdre caractérisée par un angle interne entre ses deux facettes adjacentes supérieur à 180 degrés) trouvées sur l'objet de façon récursive jusqu'à ce que toutes les sous-composantes soient convexes. La décomposition est obtenue par un ensemble de coupes planes qui passent chacune par une entaille. Il n'est pas difficile de voir qu'il y a un nombre infini de plans de coupes possibles. L'entaille est résolue si le plan de coupe sépare l'entaille de telle sorte que les deux arêtes créées sur l'entaille ne sont pas des entailles.

Quand la surface du polyèdre est une variété, toutes les entailles sont des arêtes concaves. La surface du polyèdre est une variété en 2-D, si chaque point de la surface a une voisinage qui est homéomorphe à une boule 2-D ouverte ou à une demi-boule. Un polyèdre dont la surface est une variété en 2-D est un polyèdre variété [Bajaj 84].

La deuxième méthode [Bajaj 84] est basée sur la méthode récursive précédente avec une extension pour résoudre de entailles spéciales dans la structure du polyèdre. La présence de ce type d'entaille fait que le polyèdre est un polyèdre non-variété. L'algorithme pour la décomposition résout dans un premier temps toutes les entailles spéciales et ensuite les arêtes en entailles.
Les entailles spéciales sont de trois types, voir Fig. 31 : a) une arête ou un sommet d'une partie du polyèdre est sur la surface d'une autre partie du polyèdre, b) différentes parties du polyèdre partagent une arête et c) différentes parties du polyèdre partagent une sommet.

Dans le premier cas, l'algorithme détache le sommet ou l'arête en faisant une recherche de contiguïté. Pour le deuxième cas, on définit un plan qui intersecte les facettes en formant deux paires de facettes adjacentes, une arête est crée entre chaque paire de facettes en remplacement de l'arête partagée qui est éliminée, ce qui résout ce type d'entaille.

Le troisième cas est résolu en groupant les arêtes et facettes adjacents en commençant pour le sommet commun. Chaque ensemble doit être parcouru sans passer par le sommet commun. On crée ainsi un sommet pour chaque ensemble, ce qui sépare les ensembles entre eux. Une combinaison de ces trois types d'entailles peut se rencontrer. Dans un tel cas, nous commençons par résoudre les entailles de type 1, puis ensuite les entailles de type 3 et enfin les entailles de type 2.

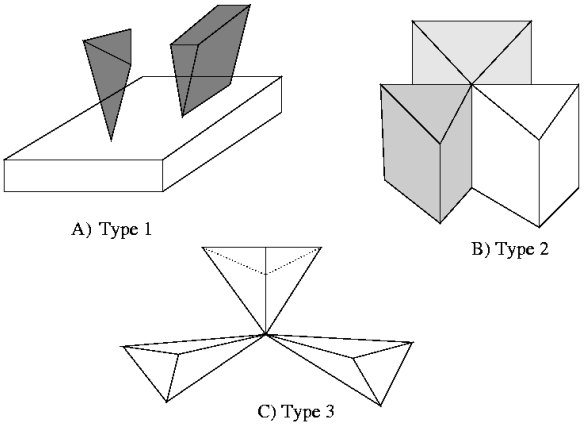
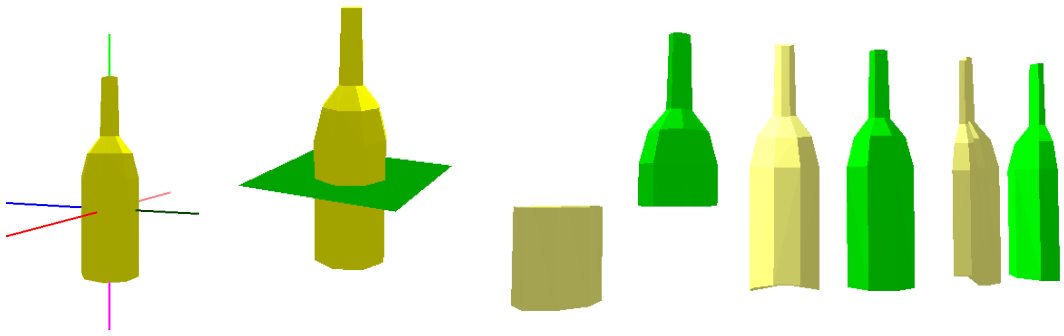


Figure 31 – Trois types d'entailles spéciales

Autres approches

Il y a d'autres méthodes pour décomposer un polyèdre en sous-composantes. Comme nous utilisons une représentation par maillage triangulaire des polyèdres, nous pouvons utiliser des outils plus spécifiques pour réaliser ces opérations sur le maillage. Pour décomposer ce maillage en deux parties, une alternative est de trouver une découpe dans celle-ci. Comme précédemment les concavités guident la recherche des découpes qui suivent les courbures négatives minimales. La décomposition est effectuée en trois étapes : 1) l'extraction de la découpe qui n'est pas nécessairement fermée, 2) la complétude de la découpe, la fermeture de la courbe est réalisé en parcourant le maillage jusqu'à trouver un chemin en essayant de passer par le plus possible d'entailles, et 3) le mouvement de la découpe pour l'adapter au maillage tout en recherchant à la lisser et a en diminuer sa longueur.



(a) Axes inertielles (b) Plan de coupe (c) Sous-composantes

Figure 32 – Résultat de la décomposition d'une bouteille par trois plans qui correspondent au trois axes d'inertie ; chaque plan produisant deux parties.

Une deuxième alternative de décomposition consiste à couper l'objet par les plans passant par le centre de masse et parallèles aux directions principales d'inertie. A chaque itération deux sous-composantes

sont générées qui forment à leur tour deux polyèdres, voir Fig. 32. Chacun de ces nouveaux polyèdres peut être décomposé récursivement en générant un ensemble de sous-polyèdres. Cette méthode peut être utilisée pour calculer les deux prises simultanées nécessaires pour les manipulations à deux préhenseurs. Aucune des deux méthodes précédentes ne garantit d'obtenir des composantes convexes.

Décomposition Convexe Approchée DCA

Pour les objets complexes une décomposition simple n'est pas suffisante. De plus, les algorithmes de décomposition convexe peuvent donner en sortie une multitude de petites composantes convexes que l'on ne peut pas exploiter pour trouver des prises. Une solution est alors de décomposer le polyèdre en sous-composantes convexes approchées. Bien sûr, nous n'obtenons pas des composantes convexes mais des composantes $\lambda-$approchées, où λ est la tolérance réglable de non-convexité de la décomposition. Quelques notions de base sont nécessaires pour mieux comprendre la méthode. La description d'un polyèdre est effectuée par un ensemble de sommets, un ensemble d'arêtes et un ensemble de facettes. Le polyèdre peut être décomposé en sous-polyèdres disjoints. L'enveloppe convexe est le volume convexe minimal qui contient le polyèdre ; il est aussi composé d'ensembles de sommets, d'arêtes et de facettes. Pour couper le polyèdre, une façon intuitive consiste à commencer par les entailles les plus profondes. Cette profondeur, difficile à définir proprement, peut être utilisée comme une mesure pour classer les entailles. On coupe alors le polyèdre par un plan contenant l'entaille la plus profonde.

Un polyèdre est $\lambda-$approché si son entaille la plus profonde a une profondeur inférieure ou égale à λ. Une décomposition $\lambda-$approchée ne contient que des composantes $\lambda-$approchées.

Cette méthode est récursive comme les précédentes et résout l'entaille la plus profonde à chaque itération en découpant le polyèdre par un plan de coupe et en produisant deux sous-composantes qui sont à leur tour décomposées. L'algorithme 18 peut être appelé récursivement jusqu'à ce que toutes les sous-composantes soient $\lambda-$approchées.

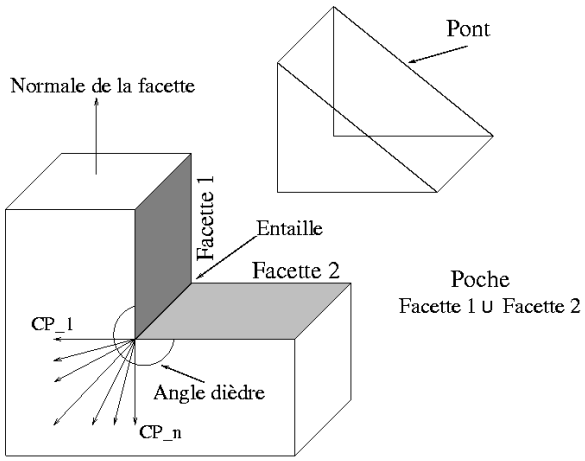


Figure 33 – Exemple d'un objet polyédrique, les ponts et les poches sont indiqués. L'entaille est l'arête concave formée par deux facettes ayant un angle intérieur supérieur à $180°$. Une série de plan de coupes possibles CP_1 ... CP_n sont montrés.

La notion de concavité n'étant pas aussi bien définie que celle de volume ou d'aire, il est nécessaire de définir une mesure. Pour cela nous introduisons les notions de *pont* et de *poche* proposés dans les références [Lien 03] [Lien 05]. Un pont est constitué de toutes les facettes adjacentes qui appartiennent à l'enveloppe convexe et qui n'appartiennent pas au polyèdre. Inversement une poche est constituée par les facettes adjacentes qui appartiennent au polyèdre et qui n'appartiennent pas à l'enveloppe convexe, voir Fig. 33.

La mesure de la concavité pourrait idéalement être donnée par la longueur moyenne de la trajectoire décrite par un point de l'entaille pour atteindre la surface du pont. Cette trajectoire apparaît difficile à définir, d'autres options doivent être proposées. Une option simple est de prendre comme mesure de la profondeur de l'entaille la distance entre le point milieu de l'entaille et l'enveloppe convexe.

Pour couper le polyèdre nous devons choisir un plan de coupe, comme il y en a un nombre infini, nous en calculons un certain nombre et conservons celui qui résout l'entaille avec la surface coupée la plus petite. Un exemple pour une itération de l'algorithme est présenté sur la Fig. 34.

Algorithm 18: Cadre de l'algorithme pour la décomposition convexe approchée

Entrée : polyèdre P_H, tolérance λ
Sortie : max(concave(P_i)) $\leq \lambda$
begin
  $C_H \leftarrow$CALCULER_ENVELOPPECONVEXE(P_H);
  $N \leftarrow$CALCULER_ENTAILLES(P_H);
  $B_N \leftarrow$TROUVER_ENTAILLE_PLUSGRANDE_CONCAVITÉ;
  if $B_N \geq \lambda$ **then**
    $C_p \leftarrow$CALCULER_PLANCOUPURE(B_N);
    $P_i \leftarrow$CALCULER_PARTITION(P_H);
    RETOUR(P_i);
  end
  RETOUR(P_H);
end

(a) Modèle polyédrique d'une bouteille

(b) Enveloppe convexe avec ponts et poches

(c) Plan de coupe

(d) Composantes de la décomposition

Figure 34 – Etapes de la décomposition convexe approchée

Intégration de la décomposition d'objets dans la planification de saisie

Les silhouettes et la taille des objets varient en influençant la difficulté de la planification des prises pour un objet. D'ailleurs la plupart des objets de notre environnement sont non-convexes. Saisir un objet arbitraire est un tâche que n'importe quel robot autonome doit être capable de réaliser. L'approche que nous proposons pour la planification de la prise d'objets complexes consiste à étendre le planificateur de saisie par un module de décomposition des objets polyédriques puis à chercher des prises sur les sous-composantes. Deux stratégies sont envisageables pour cette intégration, une décomposition complète suivie d'un calcul de prises sur chaque composantes ou un calcul des prises à chaque étape de la décomposition en deux sous-composantes. Nous avons utilisé la deuxième stratégie en arrêtant l'algorithme 19 lorsque une prise est trouvée.

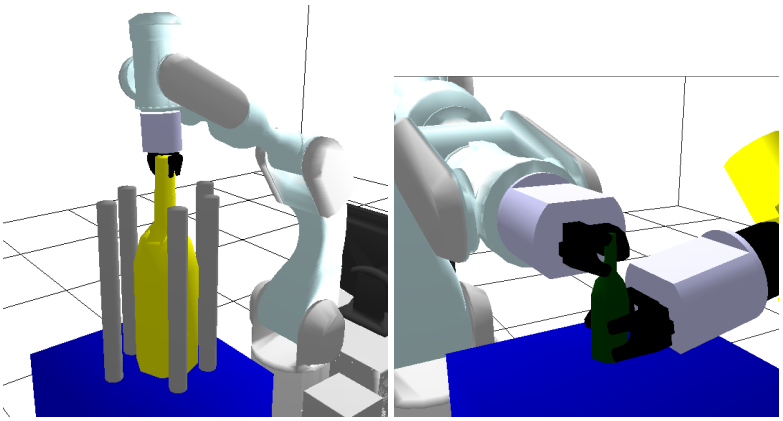
Un exemple est présenté sur la Fig. 35a. La même méthode peut être utilisée pour trouver une solution au problème de la planification d'une double prise pour le même objet, voir Fig. 35b. A chaque itération dans le processus de décomposition de l'objet, deux composantes sont produites et nous essayons de trouver une prise sur chacune de ces composantes pour l'un des préhenseurs.

Algorithm 19: Cadre de l'algorithme pour un planificateur de saisie non-convexe

```
Entrée  : robot R, environnement E, objet O
Sortie  : saisie S
begin
    while not ConditionArrête (S Trouvée ou DCA(O) = ∅) do
        C ← DÉCOMPOSITION_OBJET;
        forall Composants de C_i do
            CALCUL_AXES_INERTIE(O);
            S ←GÉNÉRATION_ALÉATOIRE_SAISIE;
            if S Satisfait FILTRES then
                ASSIGNATION_VALEUR_QUALITÉ S ← Q;
                AJOUTE_NOUVELLE_SAISIE List_S ← S;
            end
        end
    end
    CHOISIR_MEILLEURE_SAISIE;
    RETOUR S;
end
```

Intégration dans un robot expérimental

La dernière partie du travail a consisté à tester les algorithmes sur une plate-forme robotique expérimentale. Tous les algorithmes de planification de saisie ont été implémentés dans la plate-forme logicielle de planification Move3D et l'intégration des algorithmes sur le robot a été réalisé au travers de l'écriture d'un module avec l'outil Genom du LAAS-CNRS.



(a) (b)

Figure 35 – a) Configuration de saisie trouvée par le planificateur non convexe b) Configuration de saisie interactive pour une bouteille

Plate-forme logicielle et robotique

Move3D est un outil générique de planification de mouvements pour calculer des chemins sans collision pour un grand ensemble d'applications qui peuvent aller de l'industrie aux robots mobiles. La plate-forme est basée sur les méthodes de carte de chemin probabilistes. Le logiciel dispose de plusieurs modules : le module de modélisation permet à l'utilisateur de décrire les systèmes mécaniques, l'environnement, les contraintes de mouvements et de poser le problème de mouvement. Le module géométrique est utilisé pour initialiser les détecteurs de collision. Le module de méthodes locales calcule des chemins locaux satisfaisant les contraintes cinématiques des systèmes mécaniques pour délivrer des mouvements réalisables. Le module de planification contient le constructeur de cartes de chemins et un moteur de recherche du chemin dans la carte de chemin. Finalement, le module d'affichage permet de visualiser les résultats.

La plate-forme robotique expérimentale *Jido* est construite autour d'une plate-forme mobile Neobotix et d'un bras manipulateur Mitsubishi PA-10. Elle est munie de nombreux capteurs : télémètres laser SICK, paire de caméras stéréoscopiques, ultrasons, accéléromètre, etc. Le bras manipulateur dispose de six degrés de liberté et d'une pince à deux mors parallèles et trois doigts à bouts sphériques. Au niveau du poignet se trouve une deuxième paire de caméras stéréoscopiques et un capteur de force à six dimensions.

Architecture de contrôle

Pour qu'un robot autonome puisse accomplir une mission ou une tâche donnée par un utilisateur, il est intéressant de transformer la tâche en un ensemble de sous-tâches ou d'actions simples que le robot exécutera avec ses propres ressources dans un temps borné. Pour cela, une architecture générale devient importante. L'architecture du LAAS-CNRS est divisée en trois niveaux : le premier est le niveau fonctionnel qui intègre les fonctions pour commander les capteurs et actionneurs ainsi que les algorithmes qui réalisent une opération ou action. A ce niveau il existe une librairie de modules, chaque module est une entité indépendante qui exécute un ensemble de fonctions et peut communiquer avec d'autres modules.

Le deuxième est le niveau d'exécution qui permet de contrôler et coordonner l'exécution des fonctions du niveau précédent en suivant une série de contraintes imposées par le niveau supérieur, tout en vérifient que les actions ne mettent pas le robot dans un état incohérent. Le troisième est le niveau décisionnel qui est composé d'un planificateur de mission et d'un superviseur. Le planificateur reçoit une description de l'état de l'environnent et une mission de l'utilisateur, une séquence d'actions et les modalités d'exécution sont donnés en sortie. Les modalités décrivent les contraintes à respecter par le plan d'exécution. Le superviseur interagit avec le niveau inférieur, surveille la bonne exécution des actions et réagit aux événements dans un temps borné.

Générateur de modules

Tous les algorithmes de la thèse sont au niveau fonctionnel et peuvent être intégrés à l'architecture du robot sous la forme d'un module en utilisant l'outil Genom [Fleury 96]. Cet outil est un générateur de modules où chaque module encapsule un ensemble de fonctions, les activités et données produites par les fonctions et les mécanismes pour contrôler ses activités. Le module considère les fonctions comme des services accessibles et produit des données à travers des interfaces standard.
Les fonctions sont contrôlées par des requêtes envoyés par les opérateurs ou le superviseur. Quand le service a terminé une action, une réponse est retournée au module qui en a fait la requête, un rapport d'exécution dit si le service a réussi ou s'il y a eu une erreur. Il y a deux types de requêtes : les requêtes d'exécution qui commencent une activité, et les requêtes de contrôle qui contrôlent l'exécution d'un service (modification de paramètres, interruptions). Une activité correspond à une fonction en cours d'exécution, l'activité peut commander le robot physique (capteurs et actionneurs), elle peut utiliser les services d'autres modules et produire de données. Les données sont mises à jour dans des structures de données appelées *posters*, ils sont accessibles pendant l'exécution ou à la fin d'un service.

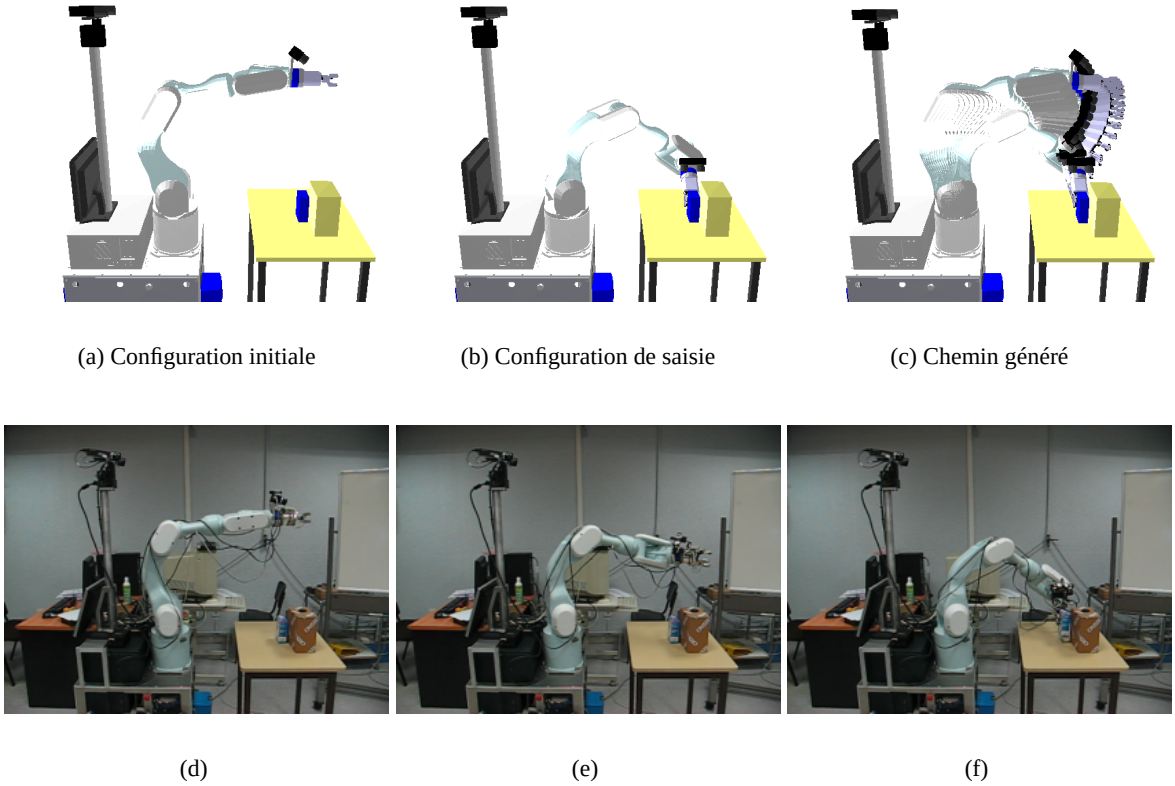
Architecture pour la saisie

Cette architecture présente les fonctions principales du système pour réaliser des tâches automatiques de saisie pour des objets arbitraires, mais les travaux de cette thèse se focalisent dans la partie relative à la planification de mouvements. Dans le contexte des objets inconnus, il faut un modèle de l'objet qui représente sa géométrie, ce modèle doit être acquis par des capteurs externes depuis plusieurs points de vues. Le module chargé de faire cette modélisation est *ObjMod*. Le modèle de l'objet et sa localisation sont les entrées pour le module de planification de saisie appelé *gP**. Celui-ci calcule, comme nous l'avons déjà vu, la configuration de saisie du robot ainsi que les mouvements pour la saisie. Le chemin calculé est sauvegardé dans un poster. Ce chemin résultat constitue l'entrée du module *xarm* qui est un module de planification et d'exécution de trajectoires dans le temps pour un bras manipulateur. Ce module est en communication constante avec le module *qarm* qui est l'interface de plus bas niveau du bras. Le dernier module est *fingm* qui commande l'ouverture et la fermeture des doigts ainsi que les efforts de contact.

Test sur Jido

Le module de planification de saisie a été testé en connexion avec la plate-forme Jido. La tâche à accomplir était la saisie d'un objet dans deux situations : dans la première l'objet est seul sur une table, la deuxième comporte un obstacle tout près de l'objet. Le test montre le fonctionnement du module de planification et l'interaction avec les autres modules à l'exception du module de modélisation simulé ici. L'environnent est donné comme entrée au planificateur ainsi que la position et le modèle de l'objet. La tâche se déroule en séquence, le module gP lit le poster du module qarm pour connaître la configuration

courante du robot et actualise la configuration du modèle du robot en simulation. Cette configuration est la configuration initiale pour le planificateur. La prise et la configuration de saisie sont calculés, le chemin de la configuration initiale à la configuration de saisie est calculé en utilisant un arbre d'exploration rapide ou RRT pour son sigle en anglais. Le poster du module gP est mis à jour avec le chemin généré. Ce poster est utilisé par le module xarm qui exécute la trajectoire. Une fois le bras positionné dans la configuration de saisie, une requête est envoyée au module fingm pour déclencher la fermeture des doigts. Sur la Fig. 36 une séquence de vue de la tâche est présentée. Elle montre la configuration de saisie trouvée par le planificateur.



(a) Configuration initiale (b) Configuration de saisie (c) Chemin généré

(d) (e) (f)

Figure 36 – Chemin pour la saisie d'un objet en présence d'un obstacle et trajectoire suivie par le robot.

Conclusions et perspectives

Dans ce mémoire, nous avons présenté un planificateur simple de tâches de manipulations interactives basé sur une architecture modulaire. Nous avons testé ce planificateur dans le contexte de la manipulation d'objets. Tous les algorithmes ont été intégrés avec l'outil de planification Move3D. Etant donné que les objets ne peuvent pas se déplacer par eux mêmes, nous avons construit le problème en introduisant l'objet comme une partie de la description du robot, et en divisant le robot en chaînes cinématiques. Ceci a permis de définir des contraintes cinématiques entre les chaînes pour permettre au robot de déplacer l'objet. Ce type de description facilite la définition des deux robots et de l'objet comme un système parallèle qui forme une chaîne cinématique fermée et permet d'utiliser les techniques de la planification

de mouvements pour ce type de structures. Ceci a été fait pour planifier des manipulations interactives comme l'échange d'un objet. Même si nous n'avons pas réalisé de tâche complète, nous avons montré que l'approche est faisable et promet de bons résultats.

Il existe de nombreux algorithmes pour planifier des prises pour la manipulation d'un objet par un robot. Certains essaient de trouver une prise optimale suivant un certain critère. Ces méthodes sont très coûteuse en temps de calcul. D'autre part, très souvent les prises humaines ne sont pas optimales. L'hypothèse qu'il existe un grand nombre de bonnes prises pour exécuter une tâche encourage les approches par génération heuristique d'un ensemble de prises qui ont une complexité algorithmique plus faible. Parmi les heuristiques, l'utilisation des propriétés inertielles pour guider la génération de prises donne de bons résultats et elle est de grande utilité lorsque l'objet a une forme arbitraire et inconnue.
L'utilisation de la décomposition en parties convexes approchées de l'objet pour le calcul des prises rend le planificateur plus robuste. Cette décomposition permet d'obtenir des parties de l'objet de taille adéquate pour les saisir, contrairement a l'utilisation d'une décomposition strictement convexe. Cependant, par construction et du fait qu'ils font des choix arbitraires ces algorithmes ne peuvent pas garantir que l'objet soit coupé de façon adéquate pour la tâche. Nous avons montré que le planificateur est facilement intégrable dans un module pour un manipulateur mobile expérimental. Les chemins de déplacement générés pour saisir l'objet sont bien exécutés en ligne. Bien sûr, de nombreux tests du planificateur devront encore être réalisés et de nombreuses améliorations devront y être apportées pour le rendre plus robuste.

Plusieurs extensions sont envisageables, nous en mentionnons seulement quelques unes. La génération de prises englobantes serait un complément aux prises dites de précision. Une autre extension intéressante est de planifier une saisie en utilisant toute la structure du robot, nous pouvons penser par exemple le cas où l'utilisateur a demandé au robot de prendre et transporter un grand cylindre, plus grand que sa pince. La machine ne peut pas le prendre, et comme l'objet est convexe, la décomposition n'est plus possible pour créer de plus petites parties. Il n'y a pas de solution avec le planificateur actuel, mais si nous considérons tout le robot comme l'humain le fait, nous pouvons imaginer que le robot utilisera son bras pour le mettre autour de l'objet. Dans le même esprit, le planificateur de saisie peut être étendu pour un modèle simplifié de la main humaine et planifier, en plus, pour les bras et le tronc.

Bibliography

[Alami 89] R. Alami, T. Siméon & J.P. Laumond. *A geometrical approach to planning manipulation tasks. The case of discrete placements and grasps*. In 5th International Symposium on Robotics Research, Tokyo, Japan, 1989.

[Alami 94] R. Alami, J.P. Laumond & T. Simeon. *Two Manipulation Planning Algorithms*. In Workshop on the Algorithmic Foundations of Robotics, Boston, 1994.

[Alami 98] R. Alami, R. Chatila, S. Fleury, M. Ghallab & F. Ingrand. *An Architecture for Autonomy*. International Journal Of Robotics Research, vol. 17, no. 4, pages 315–337, 1998.

[Amato 98] N. Amato, O. Bayazit, L. Dale, C. Jones & D. Vallejo. *OBPRM : An obstacle-based PRM for 3D workspaces*. In Workshop on the Algorithmic Foundations of Robotics, 1998.

[Arechavaleta 04] G. Arechavaleta, C. Esteves & J.P. Laumond. *Planning cooperative motions for animated characters*. In International Symposium on Robotics and Automation, Mexico, 2004.

[Badcock 94] J.M. Badcock & R.A. Jarvis. *Wire-frame modelling of polyhedral objects from rangefinder data*. Robotica, vol. 12, pages 65–75, 1994.

[Bajaj 84] C.L. Bajaj & T.K. Dey. *Convex Decomposition of Polyhedra and Robustness*. SIAM Journal of Computing, vol. 13, no. 3, pages 488–507, 1984.

[Bard 90] C. Bard & J. Trocazz. *Automatic preshaping for a dextrous hand from a simple description of objects*. In Proc. IEEE International Workshop on Intelligent Robots and Systems, pages 865–872, 1990.

[Bard 95] C. Bard, C. Laugier, C. Milési-Bellier, J. Trocazz, B. Triggs & G. Vercelli. *Achieving Dextrous Grasping by Integrating Planning and Vision-Based Sensing*. International Journal of Robotics Research, vol. 14, no. 5, pages 445–464, 1995.

[Barraquand 91] J. Barraquand & J.C. Latombe. *Robot Motion Planning : A Distributed Representation Approach*. International Journal of Robotics Research, vol. 10, no. 6, December 1991.

[Bernardini 99] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva & G. Taubin. *The Ball-Pivoting Algorithm for Surface Reconstruction*. IEEE Transactions on Visualization and Computer Graphics, vol. 5, no. 4, pages 349–359, 1999.

[Bicchi 95] A. Bicchi. *On the Closure Properties of Robotic Grasping*. International Journal of Robotics Research, vol. 14, no. 4, pages 319–334, 1995.

[Bicchi 00] A. Bicchi. *Hands for dexterous manipulation and robust grasping : a difficult road toward simplicity*. IEEE Transactions on Robotics and Automation, vol. 16, no. 6, pages 652–662, December 2000.

[Borst 99] Ch. Borst, M. Fischer & G. Hirzinger. *A Fast and Robust Grasp Planner for Arbitrary 3D Objects*. In Proc. IEEE International Conference on Robotics and Automation, pages 1890–1896, Detroit, Michigan, May 1999.

[Borst 02] Ch. Borst, M. Fischer & G. Hirzinger. *Calculating hand configurations for precision and pinch grasps*. In Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1553–1559, Lausanne, Switzerland, October 2002.

[Borst 03] Ch. Borst, M. Fischer & G. Hirzinger. *Grasping the Dice by Dicing the grasp*. In Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3692–3697, Las Vegas, Nevada, October 2003.

[Borst 04] Ch. Borst, M. Fischer & G. Hirzinger. *Grasp Planning : How to Choose a Suitable Task Wrench Space*. In Proc. IEEE International Conference on Robotics and Automation, pages 319–325, New Orleans, 2004.

[Cambon 05] S. Cambon. *Planifier avec les contraintes géométriques du movement et de la manipulation*. Thèse de doctorat, Université Paul Sabatier, Juin 2005.

[Canny 88] J.F. Canny. The complexity of robot motion planning. MIT Press, 1988.

[Chazelle 84] B. Chazelle. *Convex Partitions of Polyhedra : A Lower Bound and Worst-Case Optimal Algorithm*. SIAM Journal of Computing, vol. 13, no. 3, pages 488–507, 1984.

[Chiu 88] S.L. Chiu. *Task compatibility of manipulator postures*. International Journal of Robotics Research, vol. 7, no. 5, pages 13–21, 1988.

[Choset 05] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki & S. Thrun. Principles of robot motion : Theory, algorithms and implementations. MIT Press, 2005.

[Cortés 02] J. Cortés, T. Siméon & J.P. Laumond. *A Random Loop Generator for Planning the Motions of Closed Kinematic Chains using PRM Methods*. In Proc. IEEE International Conference on Robotics and Automation, 2002.

[Cortés 03a] J. Cortés. *Motion Planning Algorithms for General Closed-Chain Mechanism*. Thèse de doctorat, Institut National Polytechnique de Toulouse, December 2003.

[Cortés 03b] J. Cortés & T. Siméon. *Probabilistic Motion Planning for Parallel Mechanism*. In Proc. IEEE International Conference on Robotics and Automation, 2003.

[de Berg 00] M. de Berg, M. van Kreveld, M. Overmars & O. Schwarzkopf. Computational geometry. Springer, 2000.

[Ding 00a] D. Ding, Y.H. Liu & S. Wang. *Computing 3-D Optimal Form-Closure Grasps*. In Proc. IEEE International Conference on Robotics and Automation, pages 3573–3578, 2000.

[Ding 00b] D. Ding, Y.H.Liu & S. Wang. *Computing 3-D Optimal Form-Closure Grasps*. In Proc. IEEE International Conference on Robotics and Automation, pages 3573–3578, San Francisco, April 2000.

[Ding 01] D. Ding, Y.H. Liu, J. Zhang & A. Knoll. *Computation of Fingertip Positions for a Form-Closure Grasp*. In Proc. IEEE International Conference on Robotics and Automation, pages 2217–2222, 2001.

[Ekvall 04] S. Ekvall & D. Kragic. *Interactive Grasp Learning Based on Human Demostration*. In Proc. IEEE International Conference on Robotics and Automation, 2004.

[Ekvall 05] S. Ekvall & D. Kragic. *Grasp Recognition for Programming by Demonstration*. In Proc. IEEE International Conference on Robotics and Automation, 2005.

[Esteves 06] C. Esteves, G. Arechavaleta, J. Pettre & J.P. Laumond. *Animation planning for virtual mannequins cooperation*. ACM Transactions on Graphics, 2006.

[Ferrari 91] C. Ferrari & J. Canny. *Planning Optimal Grasps*. In Proc. IEEE International Conference on Robotics and Automation, pages 2290–2295, Nice, France, May 1991.

[Fikes 71] R.E. Fikes & N.J. Nilsson. *STRIPS : A New Approach to the Application of Theorem Proving to Problem Solving*. Artificial Intelligence, vol. 2, 1971.

[Fischer 97] M. Fischer & G. Hirzinger. *Fast Planning of Precision Grasps for 3D Objects*. In Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 120–126, Grenoble, 1997.

[Fleury 96] S. Fleury. *Architecture de contrôle distribuée pour robots mobiles autonomes : principes, conception et applications*. Thèse de doctorat, Université Paul Sabatier, Février 1996.

[Fleury 97] S. Fleury, M. Herrb & R. Chatila. *GenoM : a Tool for the Specification and the Implementation of Operating Modules in a Distributed Robot Architecture*. In Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 842–848, Grenoble, France, 1997.

[Gravot 04] F. Gravot. *ASyMov : Fondation d'un Planificateur Robotique Intégrant le Symbolique et le Géométrique*. Thèse de doctorat, Université Paul Sabatier, Mai 2004.

[Han 00] L. Han & N. Amato. *A kinematics-based probabilistic roadmap method for closed chain systems*. In Workshop on the Algorithmic Foundations of Robotics, 2000.

[Herrera-Aguilar 05] I. Herrera-Aguilar & D. Sidobre. *On-line trajectory planning of robot manipulator's end effector in cartesian space using quaternions*. In 15th International Symposium on Measurement and Control in Robotics, Belgium, November 2005.

[Hillenbrand 04] U. Hillenbrand, B. Brunner, Ch. Borst & G. Hirzinger. *The Robutler : a Vision-Controlle Hand-Arm system for Manipulating Bottles and Glasses*. In Proc.35th International Symposium on Robotics, Washington D.C., 2004.

[Hirai 01] S. Hirai, T. Tsuboi & T. Wada. *Robust Grasping Manipulation of Deformable Objects*. In 4th IEEE International Symposium on Assembly and Task Planning, pages 411–416, May 2001.

[Hsu 98] D. Hsu, L. Kavraki, J. Latombe, R. Motwani & S. Sorkin. *On finding narrow passages with probabilistic roadmap planners*. In Workshop on the Algorithmic Foundations of Robotics, 1998.

[Jang 05] H. Jang, H. Moradi, S. Lee & J. Han. *A visibility-based accessibility analysis of the grasp points for real-time manipulation*. In Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3586–3591, 2005.

[Jones 90] J. Jones & T. Lozano-Perez. *Planning Two-Fingered Grasps for Pick-and-Place Operations on Polyhedra*. In Proc. IEEE International Conference on Robotics and Automation, pages 683–688, Cincinnati, Ohio, May 1990.

[Kallman 03] M. Kallman, A. Aubel, T. Abaci & D. Thalmann. *Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping*. EUROGRAPHICS, vol. 22, no. 3, 2003.

[Kang 97a] S.B. Kang & K. Ikeuchi. *Toward Automatic Robot Instruction from Perception-Mapping Human Grasps to Manipulator Grasps*. IEEE Transactions on Robotics and Automation, vol. 13, no. 1, pages 81–95, February 1997.

[Kang 97b] S.B. Kang & K. Ikeuchi. *Toward automatic robot instruction from perception-mapping human grasps to manipulator grasps*. IEEE Transactions on Robotics and Automation, vol. 13, no. 1, pages 81–95, 1997.

[Kavraki 96] L. Kavraki, P. Svestka, J.C. Latombe & M.H. Overmars. *Probabilistic Roadmaps for Path Planning in High Dimensional Configuration Spaces*. IEEE Transactions on Robotics and Automation, vol. 12, no. 4, 1996.

[Khatib 85] O. Khatib. *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*. In Proc. IEEE International Conference on Robotics and Automation, St. Louis, Missouri, March 1985.

[Kim 04] J. Kim, J. Park, Y. Hwang & M. Lee. *Advanced Grasp Planning for Handover Operation between Human and Robot : Three Handover Methods in Esteem Etiquettes using Dual Arms and Hands of Home-Service Robot*. In International Conference on Autonomous Robots and Agents, pages 34–39, December 2004.

[Kirkpatrick 91] D. Kirkpatrick, B. Mishra & C.H. Yap. *Quantitative Steinitz's Theorems with Applications to Multifingered Grasping*. Rapport technique, Courant Institute of Mathematical Science, N.Y. University, 1991.

[Kobbelt 99] L. Kobbelt, J. Vorsatz, U. Labsik & H.P. Seidel. *A shrink wrapping approach to remeshing polygonal surfaces*. EUROGRAPHICS, vol. 18, no. 3, 1999.

[Koga 94a] Y. Koga, K. Kondo, J.J. Kuffner & J.C. Latombe. *Planning Motions with Intentions*. In Proc. of SIGGRAPH, pages 395–408, 1994.

[Koga 94b] Y. Koga & J.C. Latombe. *On Multi-Arm Manipulation Planning*. In Proc. IEEE International Conference on Robotics and Automation, San Diego, CA, 1994.

[Kuffner 00a] J.J. Kuffner & J.C. Latombe. *Interactive manipulation planning for animated characters*. In Proc. of Pacific Graphics, Hong Kong, October 2000.

[Kuffner 00b] J.J. Kuffner & S.M. LaValle. *RRT-Connect : An Efficient Approach to Single-Query Path Planning*. In Proc. IEEE International Conference on Robotics and Automation, 2000.

[Lagarde 04] J.M. Lagarde, M. Devy, L. Moreno, P. Lacroix & P. Lecoutaller. *Projet BODY SCAN : modélisation corporelle par profilométrie*. In Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle, January 2004.

[Lakshminarayana 78] K. Lakshminarayana. *Mechanics of form closure*. Rapport technique 78-DET-32, ASME, 1978.

[Latombe 91] J. C. Latombe. Robot motion planning. Kluwer Academic Publishers, 1991.

[Laugier 85] C. Laugier & J. Pertin-Trocazz. *SHARP : A System for Automatic Programming of Manipulation Robots*. International Symposium of Robotics Research, pages 125–132, 1985.

[LaValle 98] S.M. LaValle. *Rapidly-Exploring Random Trees : A New Tool for Path Planning*. Rapport technique, Computer Science Department, Iowa State University, October 1998.

[Lavalle 99] S. Lavalle, J.H. Yakey & L. Kavraki. *A probabilistic roadmap approach for systems with closed kinematic chains*. IEEE Transactions on Robotics and Automation, 1999.

[LaValle 05] S.M. LaValle. Planning algorithms. University of Illinois, 2005.

[Lee 04] Y. Lee, S. Lee & A. Shamir. *Intelligent Mesh Scissoring using 3D Snakes*. In Proc. 12th Pacific Conference on Computer Graphics and Applications, October 2004.

[Lee 05] S. Lee, D. Jang, E. Kim, S. Hong & J. Han. *A real-time 3D workspace modeling with stereo camera*. In Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 965–972, 2005.

[Lemai 04a] S. Lemai. *Ixtet-Exec : Planning, Plan Repair and Execution Control with Time and Resource Managment*. Thèse de doctorat, Institut National Polytechnique de Toulouse, June 2004.

[Lemai 04b] S. Lemai & F. Ingrand. *Interleaving Temporal Planning and Execution : IxTeTeXeC*. In 14ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle, January 2004.

[Li 87] Z. Li & S. Sastry. *Task Oriented Optimal Grasping by Multifingered Robot Hands*. In Proc. International Conference on Robotics and Automation, pages 389–394, 1987.

[Li 03] J.W. Li, H. Liu & H.G. Cai. *On Computing Three-Finger Force-Closure Grasps of 2-D and 3-D Objects*. IEEE Transactions on Robotics and Automation, vol. 19, no. 1, pages 155–161, February 2003.

[Lien 03] JM. Lien & N. Amato. *Approximate convex Decomposition*. Rapport technique TR03-001, PARASOL LAB, Texas A&M University, January 2003.

[Lien 05] J-M. Lien & N. Amato. *Approximate Convex Decomposition of Polyhedra*. Rapport technique TR05-001, Texas AM University, Texas, janvier 2005.

[Lin 97] M.C. Lin, D. Manocha, J. Cohen & S. Gottschalk. Collision detection : Algorithms and applications. In J.P. Laumond and M. Overmars, editors, Algorithms for robotic motion and manipulation, AK Peters, 1997.

[Liu 99] Y.H. Liu. *Qualitative Test and Force Optimization of 3-D Frictional Form-Closure Grasps Using Linear Programming*. IEEE Transactions on Robotics and Automation, vol. 15, no. 1, pages 163–173, February 1999.

[Liu 00] Y.H. Liu. *On computing n-finger force-closure grasps on polygonal objects*. International Journal of Robotics Research, vol. 19, no. 2, pages 149–158, 2000.

[Lopez-Damian 05a] E. Lopez-Damian, D. Sidobre & R. Alami. *A grasp planner based on inertial properties*. In Proc. IEEE International Conference on Robotics and Automation, pages 766–771, Barcelona, April 2005.

[Lopez-Damian 05b] E. Lopez-Damian, D. Sidobre & R. Alami. *Grasp Planning for Non-Convex Objects*. In Proc. 36th International Symposium on Robotics, Tokyo, Japan, November 2005.

[Lorensen 87] W.E. Lorensen & H.E. Cline. *Marching Cubes : A high resolution 3D surface construction algorithm*. Computer Graphics, pages 163–169, 1987.

[Lozano-Perez 83a] T. Lozano-Perez. *Spatial planning : A Configuration Space Approach*. IEEE Transactions on computer, vol. 32, no. 2, pages 108–120, 1983.

[Lozano-Perez 83b] T. Lozano-Perez. *Spatial Planning : A Configuration Space Approach*. IEEE Transactions on Computers, vol. 32, no. 2, pages 108–120, 1983.

[Lozano-Perez 87] T. Lozano-Perez, J. Jones, E. Mazer, P. O'Donnell, P. Tournassoud & A. Lanusse. *Handey : A Robot System that Recognizes, Plans and Manipulates*. In Proc. IEEE International Conference on Robotics and Automation, Raleigh, 1987.

[Markenscoff 90] X. Markenscoff, L. Ni & C.H. Papadimitriou. *The Geometry of grasping*. The International Journal of Robotics Research, vol. 9, no. 1, pages 61–74, February 1990.

[Mazon 90] I. Mazon, R. Alami & P. Violero. *Automatic Planning of Pick and Place Operations in presence of uncertainties*. In Proc. IEEE International Workshop on Intelligent Robots and Systems, pages 33–40, 1990.

[Michel 04] C. Michel, V. Perdereau & M. Drouin. *An approach to grasp planning in constrained environments*. In Proc. IEEE International Conference on Mechatronics and Robotics, Aachen, Germany, September 2004.

[Michel 05] C. Michel, V. Perdereau & M. Drouin. *Extraction of the Natural Grasping Axis from Arbitrary 3D envelopes provided by Voxel Coloring*. In Proc. 36th International Symposium on Robotics, Tokyo, Japan, November 2005.

[Miller 99] A. Miller & P.K. Allen. *Examples of 3D Grasp Quality Computations*. In Proc. IEEE International Conferece on Robotics and Automation, pages 1240–1246, Detroit, May 1999.

[Miller 01] A. Miller. *GraspIt : A versatile Simulator for Robotic Grasping*. Thèse de doctorat, Columbia University, 2001.

[Miller 03] A. Miller. *Automatic Grasp Planning Using Shape Primitives*. In Proc. IEEE International Conferece on Robotics and Automation, pages 1824–1829, September 2003.

[Mirtich 94] B. Mirtich & J. Canny. *Easily Computable Optimum Grasps in 2-D and 3-D*. In Proc. IEEE International Conference on Robotics and Automation, pages 739–747, 1994.

[Mirtich 96] B. Mirtich. *Fast and Accurate Computation of Polyhedral Mass Properties*. Journal of Graphics Tools, vol. 1, no. 2, pages 31–50, 1996.

[Mishra 87] B. Mishra, J.T. Schwartz & M. Sharir. *On the existence and synthesis of muilti-finger positive grips*. Algorithmica, vol. 2, no. 4, pages 541–558, 1987.

[Mishra 95] B. Mishra. *Grasp Metrics : Optimality and Complexity*. In Algorithmic Foundations of Robotics, pages 137–165, 1995.

[Munkres 00] M. Munkres. Topology. Prentice Hall, 2000.

[Nguyen 88] V. Nguyen. *Constructing Force-Closure Grasps*. International Journal Of Robotics Research, vol. 7, no. 3, pages 3–16, 1988.

[Nissoux 99] C. Nissoux, T. Simeon & J. Laumond. *Visibility Based Probabilistic Roadmaps*. In IEEE International Conference on Intelligent Robots and Systems, pages 316–1321, 1999.

[Pertin-Troccaz 89] J. Pertin-Troccaz. *Grasping :a state of the art*. The Robotics Review, vol. 1, Spring 1989.

[Petersson 02] L. Petersson, P. Jensfelt, D. Tell, M. Strandberg, D. Kragic & H.I. Christensen. *Systems Integration for Real-World Manipulation Tasks*. In Proc. IEEE International Conference on Robotics and Automation, pages 2500–2505, Washington D.C., 2002.

[Pollard 94] N. Pollard. *Parallel Methods for Synhesizing Whole-Hand Grasps from Generalized Prototypes*. Thèse de doctorat, Massachusetts Institute of Technology, 1994.

[Ponce 93] J. Ponce, S. Sullivan, J.D. Boissonnat & J.P. Merlet. *On Characterizing and Computing Three- and Four-Finger Force-Closure Grasps of Polyhedral Objects*. In Proc. IEEE International Conference on Robotics and Automation, pages 821–827, 1993.

[Ponce 95] J. Ponce & B. Faverjon. *On Computing Three-Finger Force-Closure Grasps of Polygonal Objects*. IEEE Transactions on Robotics and Automation, vol. 11, no. 6, pages 868–881, December 1995.

[Ponce 97] J. Ponce & B. Faverjon. *On Computing Four-Finger Equilibrium and Force-Closure Grasps of Polyhedral Objects*. International Journal of Robotics Research, vol. 16, no. 1, pages 11–35, February 1997.

[Restrepo 05] J.A. Restrepo. *Modélisation d'objets 3D par construction incrémentale d'un maillage triangulaire dans un contexte robotique*. Thèse de doctorat, Université Paul Sabatier, Janvier 2005.

[Rijpkema 91] H. Rijpkema & M. Girard. *Computer animation of knowledge-based human grasping*. Computer Graphics, vol. 25, no. 4, pages 339–348, July 1991.

[Salisbury 85] J.K.. Salisbury & M. Mason. Robot hands and the mechanics of manipulation. MIT Press, 1985.

[Shimoga 96] K.B. Shimoga. *Robot Grasp Synthesis Algorithms : A Survey*. The International Journal of Robotics Research, vol. 15, no. 3, pages 230–266, June 1996.

[Siméon 01] T. Siméon, J-P. Laumond & F. Lamiraux. *Move3D : A Generic Platform for Motion Planning*. In Proc. 4th International Symposium on Assembly and Task Planning (ISATP'2001), 2001.

[Siméon 04] T. Siméon, J-P. Laumond, J. Cortés & A. Sahbani. *Manipulation Planning with Probabilistic Roadmaps*. International Journal of Robotics Research, vol. 23, pages 729–746, juillet 2004.

[Sudsang 95] A. Sudsang & Jean Ponce. *New Techniques for Computing Four-Finger Force-Closure Grasps of Polyhedral Objects*. In Proc. IEEE International Conference on Robotics and Automation, volume 1, pages 1355–1360, Nagoya, Japan, 1995.

[Taylor 02] G. Taylor & L. Kleenman. *Grasping Unknown Objects with a Humanoid Robot*. In Proc. Australian Conference on Robotics and Automation, pages 191–196, Aucland, November 2002.

[Toth 99] E. Toth. *Stable Object Grasping with Dextrous Hand in Three-Dimension*. Periodica Polythechnica SER. EL. ENG., vol. 43, no. 3, pages 207–214, 1999.

[Trinkle 92] J. Trinkle. *A Quantitative Test for Form Closure Grasps*. In Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1670–1677, July 1992.

[van den Bergen 99] G. van den Bergen. *A fast and robust GJK implementation for collision detection of convex objects*. Journal of Graphics Tools, vol. 4, no. 2, pages 7–25, 1999.

[van Geem 01] C. van Geem & T. Siméon. *KCD : a collision detector for path planning in factory models*. In Submitted to International Conference on Robotics and Automation, 2001.

[Zhu 03] X. Zhu, H. Ding & J. Wang. *Grasp Analysis and Synthesis Based on a New Quantitative Measure*. IEEE Transactions on Robotics and Automation, vol. 19, no. 6, pages 942–953, December 2003.

Planification de saisie pour la manipulation d'objets par un robot autonome

L'évolution autonome d'un robot dans un environnement évolutif nécessite qu'il soit doté de capacités de perception, d'action et de décision suffisantes pour réaliser la tâche assignée. Une tâche essentielle en robotique est la manipulation d'objets et d'outils. Elle intervient non seulement pour un robot seul mais également dans des situations d'interaction avec un humain ou un autre robot quand il s'agit d'échanger des objets ou de les manipuler conjointement. Cette thèse porte sur la planification de tâches de manipulation d'objets pour un robot autonome dans un environnement humain. Une architecture logicielle susceptible de résoudre ce type de problèmes au niveau géométrique est proposée. Généralement, une tâche de manipulation commence par une opération de saisie dont la qualité conditionne fortement la réussite de la tâche et pour laquelle nous proposons un planificateur basé sur les propriétés inertielles de l'objet et une décomposition en éléments quasi-convexes tout en prenant en compte les contraintes imposées par le système mobile complet dans un environnement donné. Les résultats sont validés en simulation et sur le robot sur la base d'une extension des outils de planification développés au LAAS-CNRS. Le modèle géométrique 3D de l'objet peut être connu a priori ou bien acquis en ligne. Des expérimentations menées sur un robot manipulateur mobile équipé d'une pince à trois points de contacts, de capteurs de force et d'une paire de caméras stéréoscopiques ont montré la validité de l'approche.

Mots-clés: Manipulation d'objets, planification de saisie, manipulateur mobile, robotique

Grasp planning for object manipulation by an autonomous robot

The autonomous robot performance in a dynamic environment requires advanced perception, action and decision capabilities. Interaction with the environment plays a key role for a robot and it is well illustrated in object and/or tool manipulation. Interaction with humans or others robots can consist in object exchanges. This thesis deals with object manipulation planning by an autonomous robot in human environments. A software architecture is proposed that is capable to solve such problems at the geometrical level. In general, a manipulation task starts by a grasp operation which quality influences strongly the success of the overall task. We propose a planner based on object inertial properties and an approximate convex decomposition. The whole mobile system taken into account in the planning process. The planner has been completely implemented as an extension of the planning tools developed at LAAS-CNRS. Its results have been tested in simulation and on a robotic platform. Object models may be known a priori or acquired on-line. Experiments have been carried out with a mobile manipulator equipped with a three fingers gripper, a wrist force sensor and a stereo camera system in order to validate the approach.

Keywords: Object manipulation, grasp planning, mobile manipulator, robotics