Roadmap construction for regrasping problems

Ward Kockelkorn

August 28, 2009

Master's thesis
Cognitive Artificial Intelligence
Utrecht University
60 ECTS

Supervisors:
Dr. A.F van der Stappen
Dr. R.J. Geraerts
Dr. V. van Oostrom

Abstract

We consider the problem of finding a path among obstacles for an object that can only be moved by means of a so-called form closure grasp by four frictionless fingers. The goal is to determine if there is a preferable method to create a roadmap needed for solving the path planning problem. Along the path both the object and the fingers must avoid collisions with the obstacles. The presence of the obstacles may necessitate occasional regrasps, but we require that a form closure grasp of the object is maintained at all times.

The problem will be split into two separate parts. First a grasp graph is constructed that connects grasps that can be reached from each other while maintaining form closure. We consider two different methods of regrasping. One method moves all fingers at the same time, the other method moves the fingers one at a time.

The second part of the problem is the creation of a roadmap that can be used for path planning. A grasp of the graph created in the first step is assigned to each node in the roadmap. Then the nodes are connected to each other, while taking the constraints given above into account. The two regrasping methods are combined with four different methods to regrasp between two roadmap nodes.

The results will show that the regrasping method where the fingers are moved simultaniously generates denser grasp graphs. The roadmap connection methods do not differ much from each other, but a preferred method can still be designated.

On basis of the insights obtained we will close with some hints on how to increase the regrasping and roadmap creation.

Preface

When I started this project my knowledge on the subject was modest. I had followed some courses in which grasping and path planning were discussed, but I knew nothing about regrasping. So I started with searching for literature and then studied it. When I was familiar with the subjects, the basis for the methods that would be used were devised by me in consultation with my first supervisor, Frank van der Stappen.

Besides that I had never programmed in C++. Learning this programming language, which is quite similar to Java which I had used for several earlier projects, has taken serious amounts of time in the first few months of the project. An additional difficulty was that I wanted to implement everything needed to run the needed test myself. But when that proved to be a too big exercise to start programming in C++, my second supervisor, Roland Geraerts, advised me to use the Callisto package by Dennis Nieuwenhuisen. This meant that the handling of environments and collision checking were dealt with. Unfortunately this also meant that I was bound to the Microsoft Windows platform. At the beginning I wanted to create software that was platform independent, which is also not the easiest thing to do in a new programming language.

It was the intention to create a system that could any arbitrary two dimensional polygon, thus including non-convex polygons. The implementation of the calculation of the offset of non-convex polygons proved to be a project worthy of its own thesis. The CGAL-framework has got a module for calculating the offset for arbitrary polygons, but unfortunately I did not succeed in getting this module to work. This was mostly caused by my inadequate knowledge of C++ and the subject of offsetting polygons. It was therefore decided to only study convex polygons.

The grasping and regrasping part had to be done by myself, because there is no package, that I am aware of, that provides the needed functionality. This was reasonably straight forward once I understood the algorithms needed. The path planning algorithms were also implemented by myself, with help from some code for the local planner, provided by Roland.

One of the biggest difficulties not related to the content of this thesis was to manage such a large project for the first time. Until the last few weeks of the project I have had difficulties with judging how much time the work that had to be done would take. Also combining a full-time job and the finishing of a master's thesis is not recommendable, as I have found out the hard way.

Therefore I would like to thank my family for supporting me throughout the whole project. And I would also like to thank my supervisors for their commitment to making sure I could finish this project.

Ward

Contents

Chapter 1

Introduction

For most humans it is not very difficult to grasp an object in a way such that the object is immobilized between the contact points, which most often are the fingers. Many times it is even possible to reposition one or more fingers on the object without letting go of the object, which is facilitated by the friction between the fingers and the object. This repositioning of the fingers on objects they are holding is done most of the time to relax specific muscles or to get a more comfortable grip. But there are also situations in which the object has to be placed behind obstacles. In these cases it may also be necessary to change the grasp on the object. An example of such a situation is the positioning of a screw or bolt in a tight space like a car-engine or a computer case.

Machines that are used to grasp and move objects, which are most often robotic arms, are programmed in a way such that they will not be obstructed while moving in the environment in which they have to operate. This is possible because all variables are known beforehand. There is no need to assume the handled objects or the environment of the machine will change in a way that the machine will not be able to execute its task. If there are one or more changes however, the machine will have to be reprogrammed to cope with these alterations, which means the machine cannot be used until the program that runs it is modified.

In many industrial situations this is not a problem, because most robotic arms are used for a specific task that must be precisely repeated numerous times. And when there are changes in the surroundings or handled objects, this usually means there is something wrong. But there are situations where we want a robotic arm to operate autonomously and find its own path to a goal.

We will look at the combination of two separate problems here. First there is the problem of grasping an arbitrary object. How can we hold an object in a way that it cannot move? If we securely hold that object, is it possible to modify our grasp without losing the grip on the object? But when we can regrasp an object, can we move it at the same time? This brings us to the second problem, how can we determine whether it is possible to move an object from its current to its desired position without collisions? If we know how to move the object

through the environment, what is the best way to hold the object? And finally when we are moving the object, but we discover that we cannot pass a narrow passage when using the current grasp, is it an option to regrasp the object on the spot so the movement can continue afterwards?

1.1 Related work

1.1.1 Grasping

The fixation of objects in a mechanical way is first described by Reulaux [12] in 1876. He describes a method to determine if a two dimensional object is immobilized by the contacts it has with its surroundings. This method is based on the theory that all displacements in $\mathbb{R}^2$ can be seen as a rotation around some point[1]. Therefore an object can be immobilized by making sure no instantaneous centers of velocity exist. An instantaneous center of velocity is any point on a rigid body and its extension that has zero velocity. The extension of a rigid body is the theoretical extension of the body to fill all space. The extension has no influence on the movement of the body. Form closure is the situation where no infinitesimal motions of an object are possible. Thus, if and only if there are no velocity centers for both clockwise and anti-clockwise rotation possible, the object is in form closure. See Chapter 2 for a more detailed description of Reuleaux's algorithm.

We use wrench space analysis [5] to be able to determine whether or not an object is placed in form closure by its contact points. The position and the direction of the force exerted by each finger are combined to form vectors, which are called wrenches. Form closure is achieved if and only if the wrenches form a positive span in wrench-space. In this way it is possible to determine algebraically whether or not a grasp places an object in form closure. Another advantage of wrench-analysis over Reuleaux's method is that wrench-analysis also applies to three dimensional objects.

With both methods the only forces used to hold the object in place are applied by the fingers. These forces are applied at the contact points in inward direction, perpendicular to the surface of the object. We assume there is no notion of friction between the fingers and the object. In [10] Markenscoff and Papadimitriou show that in order to create a form closure grasp, four fingers are sufficient for all two dimensional polygons. The omission of friction makes it more difficult to place the object in form-closure, but if form-closure is accomplished, the addition of friction will make the grasps only more secure. Therefore we will use four frictionless fingers to generate grasps.

[1] Infinitesimal translations can be seen as rotations around a center point that is placed at infinity.

1.1.2 Regrasping

We allow regrasping of the object if form closure is maintained. The way an object can be regrasped depends on the type of fingers that is used.

Sudsang and Phoka [15] have researched the possibility to regrasp a two dimensional object with four frictional fingers while maintaining force closure, which is a constraint similar to form closure, but takes friction into account. The advantage of the use of frictional fingers is that less fingers are needed to create a valid grasp. Regrasping in this case can for example be possible if three of the four fingers form a force closure grasp, so the fourth finger can be moved without being in contact with the object.

When using frictionless fingers it is not possible to move a finger away from the boundary of the object, because this will in general destroy form closure. Therefore the fourth finger must remain in contact with the object at all times when regrasping. An advantage of this method is that to perform a regrasp near an obstacle the clearance around the object has to be greater than or equal to the diameter of the fingers to avoid collisions. When moving a finger away from the object, the clearance has to be greater than the diameter of the finger to avoid collisions. So regrasping while maintaining contact between all fingers and the object can be done in more confined spaces.

1.1.3 Path planning

The field of path planning research is broad. The goal of path planning is to find a path between the start and goal configurations. The problems that are solved with planning are not only in artificial intelligence, but also in robotics. Artificial intelligence researchers use planning algorithms to solve puzzles like Rubik's cube. In robotics motion planning is used to move a robot in an environment.

The freedom of movement of the robot is restricted by several constraints. The first restrictions come from the environment. All motions should be made without collisions with the obstacles that are present in the environment. Other constraints are implied by the robot. The physical capabilities of the robot my prevent it to reach certain configurations. A car for example cannot turn on the spot, it has to move forwards or backwards in order to turn around.

The space that represents all possible placements of a robot, by means of a number of parameters, is called the configuration space, denoted as C_{conf}. The constraints divide the configuration space into a free space C_{free} where the robot can move freely and a forbidden space. Motion planning problems are translated from the high level assignment, like move from place a to place b without colliding with anything, to a lower level problem. In this case that would be, move from a to b which both lie in C_{free} while staying completely inside C_{free}.

The more constraints there are, the more difficult the problem will be to solve. There are several classes of algorithms to cope with motion planning problems. The first class consists of combinatorial algorithms. These methods are complete in the sense that they will find a solution if there is one and else

return a failure. This makes these algorithms the preferred ones to use. But they have got a drawback that they are very hard to define and implement, especially when robots with more that two or three degrees of freedom are concerned. In addition there are approximate methods, that approximate the free space C_{free}. They are easier to implement, but may fail to find a path even if one exists.

Another way of solving motion planning problem is using sampling based methods. These methods sample the configuration space and check for each sample if the placement lies in C_{free}. Neighboring samples are connected by means of a local planner. If the object can follow the path without collisions, the nodes are connected. When a sufficiently dense graph of connected nodes is created the first phase is done.

The second phase is the query phase in which a start and goal configuration are given. These configurations are then added to the graph generated in the first phase. Then a path is found by applying a graph search algorithm on the graph with given start and goal configurations.

Since sampling is used to explore the configuration space these methods cannot guarantee to return a valid path, or to even return at all if no path exists. A weaker notion of completeness is therefore used with these methods. The algorithms are probabilistically complete, which means that for any given input for which a solution exists, that solution will be found if sampling is continued sufficiently long. If no solution exists, there is no guarantee the algorithm will terminate in finite time and return failure to find a solution.

Despite this weaker notion of completeness, excellent results have been found using sampling based algorithms. The algorithm we will use is sampling based and is called the probabilistic roadmap method [8]. This method follows the steps described and creates a roadmap from the connected nodes. There are different ways in which the roadmap can be built. The way the configuration of a node is chosen, the nodes that are regarded as neighbors and the way the local planner works can be chosen depending on the problem at hand.

In this case the configurations of the nodes are created randomly. The local planner interpolates linearly between the configurations. The number of nearest neighbors of a node is set to be higher than the number of nodes in the roadmap. Therefore all combinations of two different nodes are checked for connection. This is done in order to generate roadmaps that will show the differences between the regrasping algorithms used.

1.1.4 Current state of research

The most research done on the combination is centered around multiple robot arms with manipulators fitted. Cheng [1] has researched the regrasping between two separate robot arms. Here the manipulator on one arm that holds the object has to move in the work space of the manipulator on another arm in order to let the second manipulator regrasp the object from the first manipulator. A comparable method can be found in [7]. Here situations with two or three arms are considered. Again the regrasping is done between multiple manipulators.

They also give a method for letting two arms manipulate the same object at the same time.

Siméon et. al. [14] have looked at the possibilities to use a single robotic arm to move an object amongst static obstacles. They allow the object to be put down in order to regrasp it. In order to do so, they divide the configuration space into subspaces where the object can be grasped by the robot while it is placed in a stable position. In each subspace a separate part of the planning problem is solved. In this way planning problems can be solved more efficiently. A probabilistic roadmap method is used for the planning problem. Their method has shown to be feasible and efficient.

Research on the planning of regrasps is also done. In [2] Cho et. al. proposed a method for grasp planning in which the feasible task space of a robot is captured in a look-up table to be able to quickly determine which regrasps are possible for the robot. They do not use a planner for the path that the robot will follow. Instead the path is given.

1.2 Problem description

In many real life situations when machines have to grasp an object that has to be moved in cluttered environments, that environment is known beforehand and is also static. Even then it is difficult to find a path between the start and goal configuration, especially in our case where we demand form closure while moving and regrasping a passive object. We will try to find a way to effectively combine regrasping and path planning of passive objects. In order to do so we will split the problem into two parts.

To start, we are interested in the different ways the fingers can move during regrasping. We will therefore look at two different methods. The first moves all fingers at the same time while regrasping and the second will move the fingers one at a time. In order to compare these two methods, a graph will be built to collect the data on the possibilities to regrasp between different grasps using both methods. The number of connected edges and connected components of the graphs created will be used to see if there is a method that is more favorable to use when regrasping. In order for a regrasp to be valid, form closure must be maintained at all times.

In the second part we will compare four methods that determine when to regrasp while moving the object. The graph built in the first part will be used as input for the probabilistic roadmap method. The four options are (1) regrasping before starting the movement, (2) regrasping after finishing the movement, (3) regrasping at the point closest before a collision would occur and (4) continuously during the movement, so the regrasping and the movement will start and end at the same time. What we want to see is if there is a method that will generate a denser roadmap. A denser roadmap is an indication that the responsible method has a better chance of finding a regrasp option if one exist.

This leads to the following research questions.

Research question 1 *How do the two different methods to regrasp perform on convex polygons?*

Research question 2 *How do the four methods of applying regrasping while moving an object perform when building a roadmap?*

Research question 3 *What can be done to possibly improve the regrasping and roadmap creation?*

1.3 Outline

We will continue as follows, first the relation to the field of artificial intelligence is given. After that we will look at the grasping and regrasping aspects we want to research, the associated results will also be given as the two main parts are separated. Then the path planning part will be treated, again with the results. Finally we will draw our conclusions and we will give some possibilities to use the results in continued research.

1.4 Regrasping in a broader perspective

One of the main ideas behind artificial intelligence is to replicate behavior of human beings with machines. The regrasping of objects is such a form of behavior that lends itself for replication. While regrasping is done unconsciously by human beings most of the time, it is a non-trivial problem for machines. In this research sampling based methods are used to determine the possible options and the notion of friction between the fingers and the object that is held is not taken into account, the results will give an indication of the directions that can be chosen for further research. Maybe information on the way humans determine how they regrasp an object gathered through psychonomic research can be used. And certainly methods used in the field of machine learning will be useful to transform the preprocessing approach into an online one.

Chapter 2

Grasping of 2D polygons

2.1 Offsetting a polygon

We assume that the fingers are discs with a fixed radius. In order to be able to represent a finger as a point instead of a circle, we offset the polygon with the disc. This offset is created by means of the Minkowski-sum operation on the original polygon and a disc-shaped finger. This can be seen as tracing the polygon with a pencil in the center of a finger sized circle. In Figure 2.1 a polygon is shown with its offset drawn around it. Because the calculation of the offset of the polygon is not the primary goal of this research, only convex polygons are considered, for the sake of simplicity.

The first step is the creation of the vertices of the offset of the polygon. The offset will consist of straight edges and arcs. The straight edges are generated by moving the edges of the input polygon perpendicularly by the radius of the fingers. The arcs are created by drawing a circular arc centered at the vertex with the same radius as the fingers on each vertex of the input polygon. The outline of the straight edges and the arcs combined form the offset. Note that

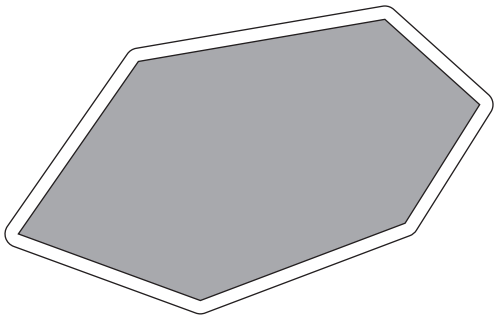


Figure 2.1: A polygon and its offset.

each vertex of the polygon has got two corresponding vertices in the offset.

Now that the offset of the polygon is known, grasps can be created. First the perimeter of the offset is calculated. This perimeter is equal to the perimeter of the input polygon together with the circumference of a finger. Note that this only holds for convex polygons as used here. For non-convex polygons the calculation of the perimeter is more intricate [16].

First the perimeter p of the offset is uniformly parametrized from a fixed starting point on the offset by a value in the interval $[0, 1]$. A random position on the boundary is then obtained by taking a rondom value $s \in [0, 1]$. The position of a finger is can then be calculated by first multiplying s by p. This is the length of the perimeter that has to be followed from the vertex specified as start, to reach the finger. The edge on which the finger lies is then found by traversing the offset for the distance calculated. When we know the relative position of a finger on an edge, the coordinates of the finger are found by interpolating the coordinates of the vertices of the edge.

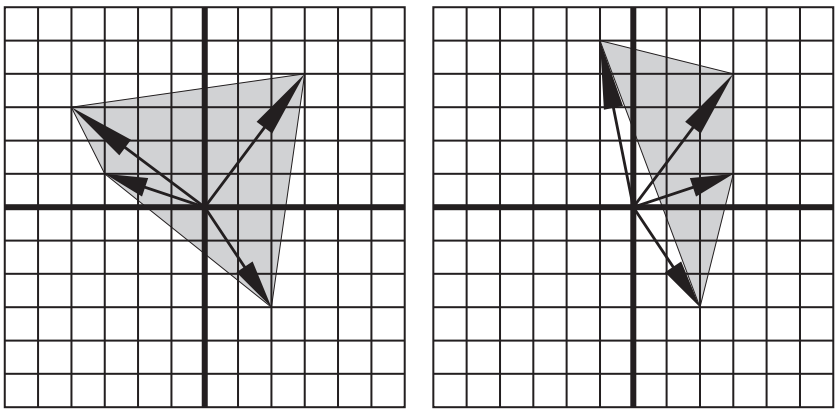
2.2 Form closure

As said before, form closure is the situation where no infinitesimal motions of an object are possible. Reuleaux [12] was the first to come up with an method to test whether a two dimensional object was placed in form closure by its constraints. This method, often called half-plane analysis makes use of the fact that any displacement in $\mathbb{R}^2$ can be seen as a rotation about some point. Both the direction and the velocity of the motion can now be completely determined by the rotation center and a sign (+ for counter-clockwise, − for clockwise). Reuleaux's half-plane analysis method uses the elimination of possible centers of both counter-clockwise and clockwise rotation. The half-planes on the left side of each normal vector of the contact points going in inward direction of the polygon eliminate the centers of clockwise rotation and the half-planes on the right side eliminate the centers of counter-clockwise rotation. If and only if the left-side half-planes cover the whole space and the right-side half-planes also cover the whole space, there are no center of rotation for the given grasp. An object is said to be in form closure if no possible centers of clockwise rotation and no centers of counter-clockwise rotation exist.

Half-plane analysis has one disadvantage. It is only suited for two-dimensional objects, so for determining if a grasp creates form closure on a three-dimensional object, another method must be used. Moreover, we prefer the wrench analysis as described below for ease of implementation.

2.2.1 Wrench-analysis

Wrench analysis makes use of a property of vectors, namely when several vectors together form a positive basis, the combination of these vectors can resist all external vectors. Let us take the vectors $\bar{v}_1$, $\bar{v}_2$, $\bar{v}_3$, $\bar{v}_4$ as an example. See Figures 2.2(a) and 2.2(b) for a graphical representation of four vectors forming a positive



(a) Positive span with 4 vectors in 2D. (b) Failure to create a positive span with 4 vectors in 2D.

Figure 2.2: Positive span in 2D.

span and the failure to do so respectively. If these four vectors form a positive basis, they can resist all external vectors. So any vector $\bar{v}$ can be written as $\bar{v} = \alpha_1\bar{v}_1+\alpha_2\bar{v}_2+\alpha_3\bar{v}_3+\alpha_4\bar{v}_4$, for some $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \geq 0$. It has been shown that checking this condition is equal to verifying that $\alpha_1\bar{v}_1 + \alpha_2\bar{v}_2 + \alpha_3\bar{v}_3 + \alpha_4\bar{v}_4 = 0$, for some $\alpha_1, \alpha_2, \alpha_3, \alpha_4 > 0$. The latter is the case, if and only if the origin will be inside the convex hull of $\bar{v}_1, \bar{v}_2, \bar{v}_3, \bar{v}_4$. To use this theory in grasping problems wrenches have to be made. A wrench is the normal force that a finger can exert on a object combined with the moment of that force. A wrench w is defined as follows:

$$w = \begin{pmatrix} n_x \\ n_y \\ p \times n \end{pmatrix}$$

where p is the location of the finger, n is the inward normal at p and $\times$ stands for the cross product. Because both the position and the direction of the force are given in two dimensions, the wrench space is three dimensional.

In wrench analysis form closure is defined as follows: wrenches $w_1, \ldots, w_k$ induced by fingers should be able to resist any external wrench, so $w_1, \ldots, w_k$ form a positive basis in wrench space, thereby containing the origin O inside the convex hull of $w_1, \ldots, w_k$.

Cramer's rule

In order to determine whether or not O is inside the convex hull of the wrenches a system of linear equations must be solved. The system that has to be solved is derived from Equation 2.1, and is written out in Equation 2.3.

$$\begin{aligned} \alpha_1 w_1 + \alpha_2 w_2 + \alpha_3 w_3 + \alpha_4 w_4 &= 0 \\ \alpha_4 w_4 &= -\alpha_1 w_1 - \alpha_2 w_2 - \alpha_3 w_3 \end{aligned} \tag{2.1}$$

Here $w_1, \ldots, w_4$ are the four wrenches and $\alpha_1, \ldots, \alpha_4$ are factors that all have to be positive in order to create a convex hull for the four wrenches that contain O. Then Equation 2.1 is divided by α_4 so we get Equation 2.2, where all α_i' have to be positive to form a positive span.

$$w_4 = -\alpha_1' w_1 - \alpha_2' w_2 - \alpha_3' w_3 \tag{2.2}$$

For clarity we will rename α_i' to α_i and multiply them with -1, so we will get $w_4 = \alpha_1 w_1 + \alpha_2 w_2 + \alpha_3 w_3$, with all $\alpha_i < 0$. We can write this as a system of linear equations, see Equation 2.3.

$$\begin{aligned} w_{4,x} &= \alpha_1 w_{1,x} + \alpha_2 w_{2,x} + \alpha_3 w_{3,x} \\ w_{4,y} &= \alpha_1 w_{1,y} + \alpha_2 w_{2,y} + \alpha_3 w_{3,y} \\ w_{4,z} &= \alpha_1 w_{1,z} + \alpha_2 w_{2,z} + \alpha_3 w_{3,z} \end{aligned} \tag{2.3}$$

To find the factors $\alpha_1, \alpha_2, \alpha_3$ Cramer's rule [3] is used. This rule states that the system of equations can be represented in matrix multiplication form as:

$$Ax = c \tag{2.4}$$

where A is an invertible matrix, x is the column vector of the variables and c is the column vector with constants. For this system the multiplication is as follows:

$$\begin{bmatrix} w_{1,x} & w_{2,x} & w_{3,x} \\ w_{1,y} & w_{2,y} & w_{3,y} \\ w_{1,z} & w_{2,z} & w_{3,z} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} w_{4,x} \\ w_{4,y} \\ w_{4,z} \end{bmatrix} \tag{2.5}$$

From Equation 2.5 can be derived that $\alpha_1, \alpha_2, \alpha_3$ are equal to the Equations 2.6, 2.7 and 2.8.

$$\alpha_1 = \frac{\begin{vmatrix} w_{4,x} & w_{2,x} & w_{3,x} \\ w_{4,y} & w_{2,y} & w_{3,y} \\ w_{4,z} & w_{2,z} & w_{3,z} \end{vmatrix}}{\begin{vmatrix} w_{1,x} & w_{2,x} & w_{3,x} \\ w_{1,y} & w_{2,y} & w_{3,y} \\ w_{1,z} & w_{2,z} & w_{3,z} \end{vmatrix}} \tag{2.6}$$

$$\alpha_2 = \frac{\begin{vmatrix} w_{1,x} & w_{4,x} & w_{3,x} \\ w_{1,y} & w_{4,y} & w_{3,y} \\ w_{1,z} & w_{4,z} & w_{3,z} \end{vmatrix}}{\begin{vmatrix} w_{1,x} & w_{2,x} & w_{3,x} \\ w_{1,y} & w_{2,y} & w_{3,y} \\ w_{1,z} & w_{2,z} & w_{3,z} \end{vmatrix}} \tag{2.7}$$

$$\alpha_3 = \frac{\begin{vmatrix} w_{1,x} & w_{2,x} & w_{4,x} \\ w_{1,y} & w_{2,y} & w_{4,y} \\ w_{1,z} & w_{2,z} & w_{4,z} \end{vmatrix}}{\begin{vmatrix} w_{1,x} & w_{2,x} & w_{3,x} \\ w_{1,y} & w_{2,y} & w_{3,y} \\ w_{1,z} & w_{2,z} & w_{3,z} \end{vmatrix}} \tag{2.8}$$

Where $|A|$ denotes the determinant of matrix A. The polygon is in form closure by the grasp if and only if $\alpha_1 < 0, \alpha_2 < 0$ and $\alpha_3 < 0$.

2.2.2 Feasible grasps

Not only is it necessary for a grasp to place the object in form closure to be a valid grasp. The grasp has also to be feasible, which means the fingers must be placed at a large enough distance from each other, to prevent them from intersecting. In a real-life situation this is inherently the case, because two physical fingers cannot intersect. But in a virtual environment this has to be verified. This is done by calculating the distance between the center points of each possible pair of fingers and checking whether this distance is larger than twice the finger radius. If any two fingers of a grasp intersect, the whole grasp is discarded.

Chapter 3

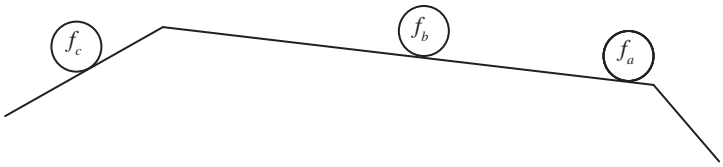
Regrasping

3.1 Constraints

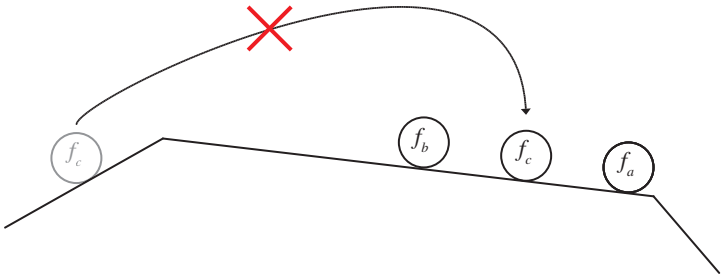
In order to change the grasp on an object, or to regrasp the object, form closure has to be maintained during the whole process of moving the fingers. Because frictionless fingers are used, all fingers must remain in contact with the polygon while regrasping. This puts some constraints on the way in which regrasps can be accomplished.

The first and most obvious constraint is that fingers cannot switch places. Consider the three fingers f_a, f_b and f_c, which are placed on the edge of a polygon in counter-clockwise order, as shown in Figure 3.1(a). It is then not possible to move the fingers so they would end up in another order, for example f_b, f_a, f_c, as shown in Figure 3.1(b).

The impossibility of switching finger positions implies that some regrasps will not be possible, independent of the question whether form closure will be maintained. An example of a situation meant here can be seen in Figure 3.2. Finger f_a starts at position s and has to be moved to position e. But this will not be possible as long as finger f_b is positioned between s and e. To prevent this from happening the same check as described in Section 2.2.2 is used.



(a) Starting order



(b) Impossible re-grasp result

Figure 3.1: Fixed finger order while regrasping.

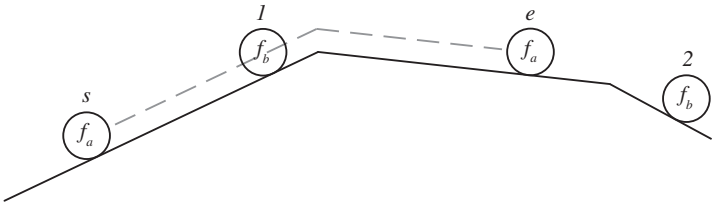


Figure 3.2: Finger f_a can only be moved from s to e after finger f_b has been moved from 1 to 2.

3.2 Regrasping methods

We use two different methods of regrasping. The first is a parallel method, which means that all fingers are moved at the same time. The way it is implemented is that all fingers move from their start position to their end position in the same time. Therefore each finger will have a different speed while moving. This may not seem to be the most desirable option when implementing such method for a real robotic hand, but this problem can be overcome by determining the total time needed for this re-grasping method by scaling the speeds of the fingers proportionally, so the finger that has to move at the highest speed does not exceed the maximum speed imposed by the used hardware. We will only look at the situation where there is no maximum speed defined for the fingers, as we are only interested in the possibilities of regrasping while moving all fingers at the same time.

The other method is a sequential method, which means that the fingers are moved one after each other. This method will be easier to use in real-life situations, as the finger speed is of no influence on the possibility to transform one grasp into another. Unlike the parallel method where only one option to move the fingers is available, the sequential method yields a maximum of potentially $4! = 24$ possible orders in which we can move the fingers. If at least one of these sequences makes it possible to regrasp between two grasps, these two grasps will be connected in the grasp graph (see Section 3.4).

3.3 Regrasping distance

In order to determine how a regrasp can be done with the least effort, the distances the fingers have to move are used. In order to be able to determine the distance each finger has to travel a mapping of the fingers between two grasps has to be made. Because we are using four fingers there are $4! = 24$ possible mappings between the fingers of two grasps, each resulting in a different distance. There are several ways to determine the finger mapping between grasps. The mapping that results in the smallest indicative distance is chosen. Which distance measure is most suitable, depends on the way in which the fingers are to be moved.

3.3.1 Maximum distance between fingers

For this method for each finger the distance that finger has to travel is determined. Out of these four distances the largest distance is used to compare the mapping of the fingers between two grasps. The finger mapping with the smallest maximum distance will be the mapping of our choice. This method of measuring the similarity between grasps is used when the fingers are moved simultaneously during regrasping, also called the parallel method.

3.3.2 Cumulative distance

With this method the distances that all fingers have to travel are added. The mapping with the smallest cumulative distance is chosen as the used mapping. This way to determine the finger correspondence is used when moving the fingers sequentially, which we will therefore call the sequential method. By minimizing the total distance the time it takes to regrasp will be minimized also, assuming the speed of movement of all fingers is equal and constant.

3.4 Graph representation

In order to make it easy to find the grasps that are reachable from a specific grasp, a graph data structure is used. In this graph $G_g = (V_g, E_g)$, with $E_g \subseteq V_g^2$, a vertex represents a grasp and an edges represents the possibility to regrasp while maintaining form closure between the two connected grasps at all times. The graph used is undirected, which means that if we can go from vertex v to vertex v', we can also go from v' to v.

One important point that must not be overlooked when getting a regrasp sequence for sequentially regrasping is that when grasp v is connected with grasp v' by moving the fingers in the order $1, 3, 4, 2$ for example, v' will be connected to v by using the reversed finger order $(2, 4, 3, 1)$.

When using parallel regrasping the edges of the grasp graph are undirected because all fingers start and end at the same time. If this was not the case, the graph would be directed.

3.4.1 Vertices

The vertices of the graph are grasps that place the polygon in form closure. This data makes it easy to search the graph for connected grasps and is used when determining the number of connected components in the graph.

3.4.2 Edges

Edges are placed between two vertices if the two vertices are adjacent. Two vertices are reachable if and only if form closure is maintained during the transition between them. Each edge is labelled with all possible ways to reach the final grasp from the initial grasp. The number of possibilities is 1 for when all fingers are moved at the same time and 24 when the fingers are moved one at a time. This results in a total of 25 possibilities per edge. Especially with the latter method, it is necessary to keep track of the valid ways to move the fingers, so when the graph is used to plan a regrasp, a valid finger sequence is used. If there is at least one possibility to reach the final grasp, both grasps are reachable, but this does not mean that all possibilities to move the fingers are valid.

3.4.3 Graph construction

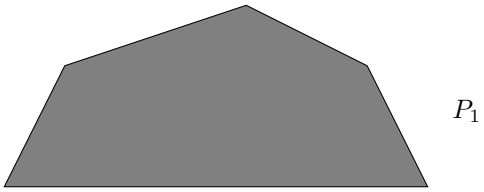
The grasp graph G_g is constructed in two steps. In the first step a given number of vertices is generated. Therefor random grasps are created. If a grasp places the polygon in form closure and the fingers do not overlap, that grasp is added as a vertex to the graph. This continues until the desired number of vertices is reached.

The following step is to try to connect the vertices. All combination of two vertices are checked for all 25 possible ways to regrasp between the corresponding grasps. When all combinations are checked, the graph is completed. In order to be able to compare graphs, the number of connected edges is used together with the connected components of the graph. See Section A.1 for more information about graphs and connected components in particular.

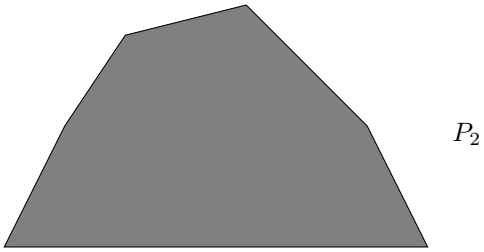
3.5 Input for the experiments

Four different polygons are used as input for testing, the algorithm is designed to accept all two dimensional convex polygons. The four polygons are shown in Figure 3.3. The polygons are created in such a way that there are two defining properties that can be used to describe the polygons. The first property is the edge length ratio. Polygons I (Figure 3.3(a)) and II (Figure 3.3(b)) each have one edge that is significantly longer than the other edges. Polygons III (Figure 3.3(c)) and IV (Figure 3.3(d)) are built from edges that all are roughly of the same length.

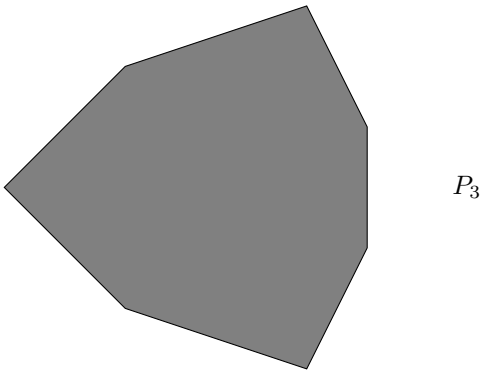
The second defining property of the polygons is the elongation of the polygon, which can for example be defined as the aspect ratio of the narrowest bounding box. Polygons I and IV are thin. The length is approximately twice the width of the polygon. The other two polygons II and III have a roughly square bounding box.



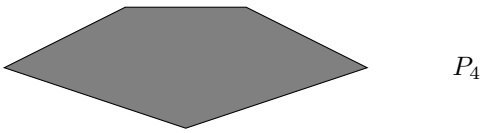
(a) Thin polygon with long edge



(b) Fat polygon with long edge



(c) Fat polygon with equal edges



(d) Thin polygon with equal edges

Figure 3.3: The used polygons: (a) I, (b) II, (c) III and (d) IV

3.6 Regrasp testing

3.6.1 Sampling

We will use sampling to determine if form closure is maintained during regrasping. This is done for simplicity of implementation. When checking the validity of a regrasp, the intermediate grasps are determined by linear interpolation of the finger positions along the perimeter of the polygon between the two grasps. These interpolated positions are then uniformly sampled and each sample is checked for form closure.

First we have to determine which sample rate leads to reliable regrasping results. It is not possible to use the highest possible sample rate, because the time it takes to calculate a regrasp sequence will increase as the sample rate increases. A trade-off between accuracy and running time must then be made, as long as sampling is needed to determine the ability to regrasp.

One of the biggest problems of sampling is the loss of continuity; it is always possible to insert a sample between two samples. Thus it is also possible that there is an obstacle between two samples that prevents the movement of the polygon or fingers. In Figure 3.4 the problem of under-sampling can be seen. If only the grey circles are sampled it will seem as if the fingers can pass between the obstacles. But when the sampling rate is doubled and the transparent circle in the middle is also sampled, it becomes clear that there will be a collision between the circle and the obstacle. Similarly, even though two consecutive samples may correspond to form closure, an intermediate sample may actually indicate that form closure is temporarily not achieved.

Because the methods used do not define a default size of the input polygons, another way to determine the sample rate must be used. Instead of giving an absolute number of samples between the start and end position of a finger, the overlap between two samples is given as measure to determine the sample rate.

For the fingers the overlap o between two consecutive placements is given as a ratio of the diameter of the fingers. This overlap means that in two consecutive

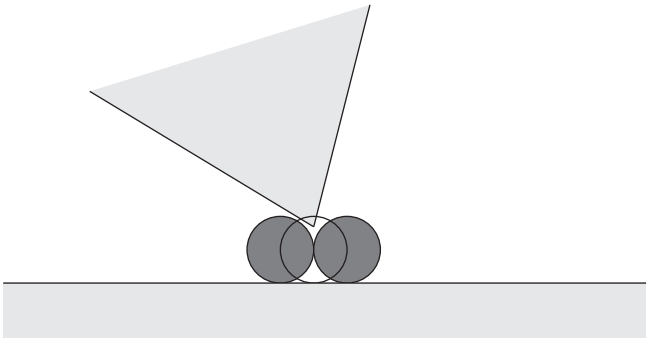


Figure 3.4: The problem of under-sampling.

samples the placements of the fingers are at most $(1 - o)$ times the diameter of the fingers apart from each other. Because the fingers are circular in shape, the orientation and direction of movement of the fingers is of no influence on the way two samples overlap.

3.6.2 Input variables

The variables that are investigated during testing are:

- the overlap between samples,
- the finger radius,
- the regrasp method.

The values for the overlap that we consiser are 0.80, 0.90, 0.95 or 0.97. The finger radius is either $1/25^{\text{th}}$ or $1/75^{\text{th}}$ of the diameter of the polygon. The diameter of a polygon d_P is the largest distance between any two of the vertices of that polygon. The regrasp method is either parallelly $\mathcal{M}_p$, where all fingers are moved at the same time, or sequentially $\mathcal{M}_s$, where the fingers are moved one after the other.

For each combination of the above variables an experimental run consists of the creation of a grasp graph. With each combination 25 runs are done. Over these runs the following statistics are collected:

- the number of trials needed to generate the grasps,
- the number of edges when using $\mathcal{M}_p$,
- the number of edges when using $\mathcal{M}_s$,
- the number of possible finger sequences for edges created with $\mathcal{M}_s$,
- the number of connected components in the graph,
- the sizes of the connected components.

The results are averaged over all 25 runs. The trails needed to generate the grasps are used to see what the influence of the finger radius is on the ease of generating random valid (eq. form closure and feasible) grasps. The number of connected edges will be used to determine the differences between the different settings for each of the input variables.

3.7 Regrasping results

3.7.1 Creating random grasps

First we consider the results for the creation of the random grasps. For each polygon the average number of tries needed to generate a valid (i.e., form closure and feasible) grasp is listed in Table 3.1. Here the columns labeled 'Att.'

contains the average number of attempts needed to generate a single valid grasp. The columns 'F.C.', 'Dist' and 'Both' contain the average number of times a random grasp was not valid due to the lack of form closure, sufficient distance between the fingers or the lack of both form closure and sufficient distance respectively. The number of attempts per grasp is equal to the total number of invalid grasps plus one for the valid grasp.

Table 3.1 shows that for polygon P_1 less attempts are needed than for the other polygons. In case of a finger radius of $d_P/25$, 23.28 attempts are needed to create a valid grasp for P_1 against at least 28.92 attempts for the other polygons. When using a finger radius of $d_P/75$ these values are 30.51 and 37.23 respectively.

	Tries per grasp							
	finger radius = $d_P/25$				finger radius = $d_P/75$			
Polygon	**Att.**	**F.C.**	**Dist.**	**Both**	**Att.**	**F.C.**	**Dist.**	**Both**
P_1	23.28	3.77	1.56	16.94	30.51	3.83	2.30	23.38
P_2	28.92	4.93	2.19	20.80	39.75	5.34	3.31	30.10
P_3	31.25	3.38	2.57	24.30	37.61	2.86	3.25	30.51
P_4	30.53	3.23	2.49	23.81	37.23	2.89	3.39	29.95
Average	*28.49*	*3.83*	*2.20*	*21.46*	*36.27*	*3.73*	*3.06*	*28.49*

Table 3.1: The number of tries needed to generate a randomly created valid grasp.

3.7.2 Creating the grasp graph

Edges

Once the desired number of grasps is created we try to connect all grasps to form the grasp graph G_g. First we will look at the influence of the overlap (o) setting on the number of connected edges. The results for this analysis are shown in Table 3.2. The different polygons with different finger radii connected by one of the regrasping methods are studied for the different sample overlap settings. The values in the table represent the average number of edges in the grasp graph over 25 runs. As there are 50 vertices in a grasp graph, there are $50 \cdot 49 = 1225$ edges at maximum in a graph.

What is most striking when comparing the different settings for the sample overlap is that there is no clear difference between the settings. One would expect the number of connected edges to decrease as the sample overlap and thus the sample rate increases. A connection that would be impossible to make given the restrictions of form closure and inter-finger distance can be made if the sample rate is low enough to only sample outside the areas where the restrictions are not met. In this case we do not see this behavior and this is probably caused by the fact that for each run all grasps are created randomly. Apparently the number of runs is not high enough to level out the influences of this random

creation of grasps. Therefore a slightly arbitrary decision is made to only look at a sample overlap of 0.95 for the rest of the experiments. This value is chosen because it is a good trade-off between accuracy and run time.

One thing that does become clear when looking at Table 3.2 is the fact that by using $\mathcal{M}_p$ roughly twice the number of edges are made compared to using $\mathcal{M}_s$. This observation can be seen with all overlap settings, for all four polygons and for both finger radii.

When looking at the influence of the finger radius on the number of edges in G_g we see that the smaller fingers (r = $1/75 d_P$) result in a denser grasp graph than the larger fingers. This holds also for all polygons, finger radii and sample overlap values considered.

The last thing we can see in Table 3.2 is that the shape of polygon P_1 is suited better for regrasping than the those of the other polygons. Polygon P_2 also shows higher numbers of connected edges than polygons P_3 and P_4, which have similar results. Thus in this case the polygons with one significant longer edge perform better than the polygons build from equally sized edges. There is no difference noticeable from these results between fat and thin polygons.

			Number of Edges			
Polygon	**Radius**	**Method**	o**=0.80**	o**=0.90**	o**=0.95**	o**=0.97**
P_1	$d_P/25$	$\mathcal{M}_p$	683.28	681.04	685.80	676.36
		$\mathcal{M}_s$	326.20	327.24	352.76	314.52
	$d_P/75$	$\mathcal{M}_p$	804.76	830.36	779.48	817.72
		$\mathcal{M}_s$	441.52	442.72	419.44	425.88
P_2	$d_P/25$	$\mathcal{M}_p$	444.76	423.08	438.60	435.32
		$\mathcal{M}_s$	205.56	195.44	197.84	201.76
	$d_P/75$	$\mathcal{M}_p$	520.24	538.56	515.72	529.68
		$\mathcal{M}_s$	265.60	292.76	278.16	281.92
P_3	$d_P/25$	$\mathcal{M}_p$	335.72	314.80	309.72	305.20
		$\mathcal{M}_s$	151.16	133.60	133.72	135.12
	$d_P/75$	$\mathcal{M}_p$	328.24	330.24	345.48	338.44
		$\mathcal{M}_s$	158.68	158.44	167.08	169.12
P_4	$d_P/25$	$\mathcal{M}_p$	311.84	318.80	309.36	302.60
		$\mathcal{M}_s$	144.72	143.92	138.12	126.16
	$d_P/75$	$\mathcal{M}_p$	339.16	308.96	323.04	325.04
		$\mathcal{M}_s$	162.76	140.28	155.92	158.84

Table 3.2: The number of edges in the grasp graph, out of 1225 edges maximum, averaged over 25 runs.

Connected Components

When comparing the grasp graphs, we can also look at the connected components of the graphs. We will look at the size of the largest connected component to see what the differences between the two regrasping methods are. The largest

connected components are chosen because their size will tell us something about the ease of regrasping. The size of the largest connected component shows how many grasps are connected in the graph. The higher this number, the more options there are when searching for regrasp options in the graph. The results are shown in Table 3.3. Here the average, smallest and largest number of grasps in the largest connected components of 25 graphs are given. Again we compare the regrasping methods, the finger radii and the polygons. In order to see if the combination of both regrasping methods is beneficial to the size of the largest connected component, the results of creating the connected components when both methods are allowed at the same time are also given. This is indicated by $\mathcal{M}_p + \mathcal{M}_s$ in the 'Method' column in Table 3.3.

When looking at the differences between the methods we see that the largest connected components are larger when moving the fingers simultaneously ($\mathcal{M}_p$), than when moving the fingers sequentially ($\mathcal{M}_s$). This is as expected from the results of the number of edges in the grasp graph. The connected components created with the use of method $\mathcal{M}_p$ are almost as large as the complete graph. The connected components created with method $\mathcal{M}_s$ are significantly smaller. This can also be seen when looking at the smallest and largest connected components of the 25 runs. Both values are always equal or less with $\mathcal{M}_s$ than with $\mathcal{M}_p$. The differences between the minimum values are sometimes very large, but we must remember that we saw that the influence of the random generation of grasps can be significant. Therefore these values can only be used as an indication.

If both regrasping methods are combined, we see a slight increase in the average size of the largest connected component. This holds for all polygons and both finger radii. Because the largest connected components when using $\mathcal{M}_p$ are almost as large as the full graph, the addition of edges only available when using $\mathcal{M}_s$ is not worth the effort when only creating the grasp graph. The other way around it is certainly worth the effort to include edges that can only be connected with $\mathcal{M}_p$ to a graph consisting of edges made with $\mathcal{M}_s$, but then the question rises why the graph was initially build with $\mathcal{M}_s$.

The finger radius has less influence on the size of the largest connected components than on the number of edges in the graph. This would mean the connected components are densely connected, so many edges could be removed from the graph, without breaking up the largest connected component. The differences are generally so small that we can say that the finger radius is of no influence on the size of the largest connected component.

		Largest connected components					
		r = $d_P/25$			r = $d_P/75$		
Polygon	**Method**	**Avg.**	**Min.**	**Max.**	**Avg.**	**Min.**	**Max.**
P_1	$\mathcal{M}_p$	49.12	47	50	48.64	45	50
	$\mathcal{M}_s$	44.84	37	50	45.60	40	50
	$\mathcal{M}_p + \mathcal{M}_s$	49.16	47	50	48.72	45	50
P_2	$\mathcal{M}_p$	45.64	40	49	45.28	40	48
	$\mathcal{M}_s$	39.44	33	44	41.24	38	45
	$\mathcal{M}_p + \mathcal{M}_s$	45.72	40	49	45.44	40	48
P_3	$\mathcal{M}_p$	47.24	44	50	45.04	22	50
	$\mathcal{M}_s$	39.44	20	47	31.12	18	46
	$\mathcal{M}_p + \mathcal{M}_s$	47.48	45	50	45.76	23	50
P_4	$\mathcal{M}_p$	45.52	24	49	44.52	22	49
	$\mathcal{M}_s$	35.72	18	48	33.20	21	48
	$\mathcal{M}_p + \mathcal{M}_s$	46.12	24	49	46.08	28	49

Table 3.3: The sizes of the largest connected component of G_g.

3.8 Conclusions

We wanted to see if there is a difference between regrasping while moving all the fingers at the same time and regrasping while moving the fingers one at a time. From the results presented in this chapter we see that we will get a denser grasp graph when using $\mathcal{M}_p$ than when using $\mathcal{M}_s$. The combination of both methods increases the size of the largest connected component of the graph only slightly. This increase is so small that it does not justify the extra time needed to generate a graph with both methods instead of only using $\mathcal{M}_p$. Therefore only when the sequential movement of the fingers is required, it is advisable to use method $\mathcal{M}_s$, in all other cases $\mathcal{M}_p$ is the better option.

The finger radius is only of slight influence on the connectedness of the grasp graph. This is good to know as it is not often possible to change the size of the fingers in real life situations.

Chapter 4

Path Planning

We use the probabilistic roadmap method to solve the planning problem for a passive convex polygon moving amidst polygonal obstacles. The probabilistic roadmap method was developed independently by Overmars and Švestka at Utrecht University and Kavraki and Latombe at Stanford University. They published this method together in [8].

With the probabilistic roadmap method a topological graph $G_r = (V_r, E_r)$, called a roadmap, is created. It consists of a set of nodes V_r that represent configurations in C_{free} and a set of edges $E_r \subseteq V_r^2$ that represent the possibility to move within C_{free} between the nodes they connect. The basic algorithm for building a roadmap is shown in Algorithm 1, taken from [9]. Here N stands for the number of nodes in the roadmap, $\alpha(i)$ is a sampled configuration in C_{conf} that we would like to add to the roadmap as the i^{th} node. The creation of a sample of C_{conf} can be done in different ways. The easiest way is to pick a configuration at random, but more intricate methods that are optimized for a specific problem are also possible. When $\alpha(i)$ is in C_{free}, it is connected to the already existing nodes of the roadmap. In order to limit the number of nodes that are checked for connection, only the nodes v that lie within the neighborhood of $\alpha(i)$ in G_r are checked. The neighborhood is the set of n nodes that are closest to $\alpha(i)$ in G_r. How closest and n are defined can be different for each specific planning problem. For efficiency, an edge between v and α is only made when α and v are not already part of the same connected component of the roadmap and if the path between these nodes lies completely within C_{free}. A local planner is used to check the latter. The local planner can also be chosen to be optimal for the specific problem at hand. The adding of nodes is done till there are N nodes in G_r.

The creation of the roadmap is the first step in solving planning problems with the probabilistic roadmap method. This step can be preprocessed, so it can be used for multiple queries.

The second step is the query phase in which a pair of configurations v_{init} and v_{goal} that we wish to connect is given. These configurations are most often not part of G_r, therefore for both v_{init} and v_{goal} the respective nearest nodes

$\in G_r$ that can be connected to, must be found. The check whether connection is possible is done using the same local planner as used in the preprocessing step. When both v_{init} and v_{goal} are connected to G_r, a graph search algorithm can be used to find a path between these nodes. We will not do this step as we will focus on the creation of the roadmap.

Algorithm 1 The basic probabilistic roadmap algorithm, from [9]

```
G_r.init();
i ← 0;
while i < N do
  if α(i) ∈ C_free then
    G_r.add_vertex(α(i)); i ← i + 1;
    for all v ∈ NEIGHBORHOOD(α(i), G_r) do
      if (not G_r.same_component(α(i), v) and CONNECT(α(i), v)) then
        G_r.add_edge(α(i), v)
      end if
    end for
  end if
end while
```

4.1 Building the roadmap

The nodes $v_r \in V_r$ of the roadmaps we will create consist of a placement $p = (x, y, \theta)$ of the polygon, combined with a grasp $g = (f_1, f_2, f_3, f_4) \in [0, 1]^4$, so $v_r = (p, g)$. The placement of the passive polygon consists of a position $(x, y) \in \mathbb{R}^2$ and a rotation $\theta \in \mathbb{R}^2 \times [0, 2\pi)$ and is generated randomly to be in C_{conf}. A grasp consists of four finger placements. These placements are the parametrized distances on the offset from the designated starting point of the offset.

The placement–grasp pair is generated by choosing a placement in C_{free} and combining that with a grasp from G_g, which is generated beforehand. We will randomly choose a placement in C_{free} and combine that with a randomly picked grasp from G_g.

As stated before, the nodes of the roadmap are connected by edges $e_r \in E_r$. An edge connects placement–grasp pairs (p_1, g_1) and (p_2, g_2) that are reachable from each other by a collision-free motion where:

- first the fingers are moved from g_1 to g_2 without collisions after which the polygon and the fingers are jointly moved from (p_1, g_2) to (p_2, g_2) without collisions. We will refer to this method as 'S', for 'Start'.
- the combination of polygon and fingers is moved from (p_1, g_1) without collision to (p_2, g_1) after which the fingers are moved without collisions from g_1 to g_2. We will refer to this method as 'E', for 'End'.

- the combination of polygon and fingers is moved from (p_1, g_1) to (p_2, g_1) until a collision occurs at (p', g_1). If that happens the combination is moved back in the direction of (p_1, g_1) until the fingers can be moved collision free from g_1 to g_2 at a stationary position p''. If this can be done, the combination is moved from (p'', g_2) to (p_2, g_2), again without collisions. This method is referred to as 'M', for 'Midway'.
- the combination of polygon and fingers is moved from (p_1, g_1) to (p_2, g_2), thus both the placement and the grasp are changed simultaneously. Denoted as 'C', for 'Continuous'.

When regrasping the polygon at a stationary position, thus going from (p, g) to (p, g'), the finger positions are linearly interpolated between g and g'. When we move the object while keeping the same grasp, thus transitioning from (p, g) to (p', g), the position is interpolated linearly between p and p'. When using method C, both the finger positions on the object and the position of the object in the environment are linearly interpolated. For each roadmap the choice is made which method of regrasping is used, $\mathcal{M}_p$ or $\mathcal{M}_s$.

In order to check for collisions when moving the fingers and the polygon, the transitions are sampled uniformly. The sample rate depends on the distance between p_1 and p_2. This distance d is defined as the Euclidean distance between (x_1, y_1) and (x_2, y_2) together with half the absolute difference between θ_1 and θ_2. The inclusion of the rotation in this distance is used to get sufficient samples when the distance between (x_1, y_1) and (x_2, y_2) is small. In that case the rotational factor will increase the distance used to determine the number of samples. The factor of 0.5 is chosen rather arbitrarily and one could easily argue that another value should be used. The sample rate is calculated by dividing the distance by the maximum distance between two samples, which is given as an input variable before the creation of the roadmap starts.

For simplicity the edge between (p_1, g_1) and (p_2, g_2) will not be made if g_1 and g_2 are not adjacent in G_g, that is if $(g_1, g_2) \notin E_g$. A possible enhancement would be to only reject a roadmap edge if g_1 and g_2 are not connected in G_g. Edges are thus only created between two nodes if the grasps of the nodes are directly regraspable and there are no collisions while moving and regrasping.

We assume that the neighborhood of a roadmap node consists of all other nodes in the roadmap, so every new pair (p, g) is checked for connection with all other pairs $(p', g') \in G_r$. We will skip the check if the nodes we want to connect are already in the same connected component. In this way we will be able to see what the differences in terms of connectedness of the roadmap are between the eight ways to connect the nodes. We will keep on adding new nodes to the roadmap until the roadmap consists of either 50 or 100 nodes. This means it is possible that the roadmap consists of multiple connected components, which is not desirable when querying the roadmap for possible paths, but as we are mostly interested in the number of connected edges in the roadmap, this is of lesser concern. In order to connect the separate connected components, extra nodes that lie between the components, could be added to the roadmap in order to try and connect these components.

4.2 Path planning experiments

Our aim is to see what the differences between the four methods of connecting roadmap nodes are. Because we keep the regrasp methods separated as stated earlier, the actual number of connection methods is eight. For each of these methods we will look how the number of edges in the roadmap is influenced by:

- the maximum distance between two samples,
- the radius of the fingers,
- the number of nodes in a roadmap.

For the maximum distance between two samples, the values 0.1, 0.5 and 0.9 are evaluated. The radius of the fingers will either be $1/25^{\text{th}}$ or $1/75^{\text{th}}$ of the diameter d_P of the polygon and we will create roadmaps of 50 or 100 nodes.

With each combination of these variables 25 roadmaps are created. Over these runs the averages of the following statistics are collected:

- the number of edges,
- the number of connected components in the graph,
- the sizes of the connected components in the graph,
- the number of edges that are not added due to a collision,
- the number of edges that are not connected, because the grasps are not adjacent in G_g.

For each statistic the average, the extremes (minimum and maximum) and the standard deviation are given. The grasp graphs used are created at random for each run with the following settings: 50 grasps, an overlap of 0.95 and a finger radius of also either $1/25^{\text{th}}$ or $1/75^{\text{th}}$ of the diameter of the polygon.

4.3 Motion planning results

4.3.1 Sample rate

First we will look at the influence of the sample distance *sd* on the number of edges in the roadmap. Table 4.1 shows the results for polygon P_3, with a finger radius of $d_P/75$ and 50 nodes in the roadmap. The values in this table denote the number of edges (Edges) in the roadmap and the number of collisions (Coll) that occurred while trying to connect the nodes. The values are the averages over 25 runs. The methods used to connect the nodes of the roadmap are separated in the parallel method ($\mathcal{M}_p$) for regrasping and are indicated by the subscript 'p'. When the regrasp method where the fingers are moved sequentially ($\mathcal{M}_s$) is used it is denoted with the subscript 's'.

	Edges and collision					
	$sd = 0.1$		$sd = 0.5$		$sd = 0.9$	
$\mathcal{M}$	**Edges**	**Coll**	**Edges**	**Coll**	**Edges**	**Coll**
Sp	135.88	210.76	124.88	210.72	136.56	211.40
Ep	135.84	210.80	124.24	211.36	135.88	212.08
Mp	135.84	210.80	124.32	211.28	136.00	211.96
Sp	135.96	210.68	124.80	210.80	136.44	211.52
Ss	71.80	111.68	66.12	109.68	69.28	108.32
Es	71.76	111.72	65.76	110.04	68.92	108.68
Ms	71.76	111.72	65.80	110.00	68.96	108.64
Cs	72.00	111.48	66.32	109.48	69.40	108.20

Table 4.1: The number of edges in the roadmap and the number of collisions while building the roadmap for different sample distances.

We see that the influence of the sample distance is not obvious. The middle setting for the sample distance (0.5) has less connected edges than both the lower and higher sample distances. It may seem that the differences in the number of edges differs significantly between the different sample distances, but the largest distance between two sample distances is in this case still less than one percent of the possible edges. In this case the roadmap can have a maximum of $(50 \cdot (50 - 1))/2 = 1225$ edges and the greatest difference is 136.56 - 124.88 = 11.68. The other configurations show equivalent results. Therefore further comparison between the different sample distances will not add more insights to the differences between the methods used to connect the nodes of the roadmap. We use a sample distance of 0.1 for the rest of the results, as that is the most accurate setting.

4.3.2 Finger radius

Next we will look at the differences between the different finger sizes used. Table 4.2 shows the results of the roadmaps built for polygon P_2 while the roadmap consists of 100 nodes. As stated before the sample distance is 0.1. The number of edges in the roadmap are in the columns labeled 'Edges' and the columns labeled 'No regrasp' contains the number of connections that are not made because the corresponding grasps are not adjacent in the grasp graph.

We see that the smaller fingers result in more edges in the grasp graph, but we also see that the number of connections that cannot be made because it is not possible to regrasp between nodes, is lower when using the smaller fingers. The latter is caused by the fact that the grasp graph is denser when using the small fingers, whereby the chance of two grasps being adjacent increases. The larger number of edges in the roadmap when using a finger radius of $d_P/75$ can be seen with all configurations.

	Edges and imossible regrasps			
	finger radius = $d_P/25$		finger radius = $d_P/75$	
$\mathcal{M}$	**Edges**	**No regrasp**	**Edges**	**No regrasp**
Sp	505.16	3124.36	590.04	2734.92
Ep	503.76	3124.36	588.60	2734.92
Mp	539.32	3124.36	632.36	2734.92
Cp	505.08	3124.36	590.04	2734.92
Ss	227.20	4123.56	321.72	3735.76
Es	227.12	4123.56	321.00	3735.76
Ms	241.76	4123.56	343.20	3735.76
Cs	228.32	4123.56	322.08	3735.76

Table 4.2: The influence of the finger radius on the number of edges in the roadmap and the number of failed connections due to impossible grasps.

4.3.3 Number of nodes

The influence of the number of nodes on the number of edges in the roadmap can be seen in Table 4.3. Here we see the results for polygon P_4 with a finger radius of $1/25^{\text{th}}$. In the table the absolute number of edges (Edges) and the number of edges as a percentage (%) of the maximum possible number of edges in the roadmap are given. The maximum possible numbers of edges are 1225 and 4950 for roadmaps consisting of 50 and 100 nodes respectively.

For the parallel regrasped roadmaps the percentage of possible edges that are created only slightly decreases when the number of nodes increases. For the serial regrasped roadmaps the opposite holds. The differences we see between the roadmaps consisting of 50 and 100 nodes are negligible. When we look at the eight methods to connect the roadmap nodes individually, we see that there is no identifiable difference in the connectedness of the roadmap between the roadmaps consisting of 50 or 100 nodes.

Table 4.4 shows what happens to the number of collisions when the number of nodes increases. Again the absolute number (Collisions) of collisions and the number of collisions as a percentage (%) of the maximum possible number of connections are given. A connection can only be denied due to a collision if a regrasp is possible between the nodes. If it is not possible to regrasp between two nodes, the attempt to connect these node is canceled and no collision checks are done. As can be seen in Table 4.2, the number of connections that is denied due to the failure to regrasp is lower when the regrasping is done with method $\mathcal{M}_p$ then when using method $\mathcal{M}_s$. This information combined with the data from Table 4.3 explains the differences between the methods that use parallel regrasping and those that use serial regrasping. More connections between roadmap nodes are checked when using parallel regraping than when serial regrasping is used, therefore there is a greater chance of collision when using parallel regrasping. When looking at the four connection methods separately without the regrasping method taken into account, the number of collisions as a

	Number of edges			
	50 nodes		100 nodes	
$\mathcal{M}$	**Edges**	**%**	**Edges**	**%**
Sp	128.84	10.52%	511.96	10.34%
Ep	128.72	10.51%	511.52	10.33%
Mp	128.80	10.51%	511.88	10.34%
Cp	128.76	10.51%	512.08	10.35%
Ss	58.68	4.79%	254.00	5.13%
Es	58.60	4.78%	253.88	5.13%
Ms	58.64	4.79%	254.04	5.13%
Cs	58.92	4.81%	254.68	5.15%

Table 4.3: The number of edges and the connection percentages of the roadmap for different number of nodes.

	Number of collisions			
	50 nodes		100 nodes	
$\mathcal{M}$	**Collisions**	**%**	**Collisions**	**%**
Sp	205.00	16.73%	785.72	15.87%
Ep	205.12	16.74%	786.16	15.88%
Mp	205.04	16.74%	785.80	15.87%
Cp	205.08	16.74%	785.60	15.87%
Ss	98.20	8.02%	388.00	7.84%
Es	98.28	8.02%	388.12	7.84%
Ms	98.24	8.02%	387.96	7.84%
Cs	97.96	8.00%	387.32	7.82%

Table 4.4: The number and percentage of collisions while building the roadmap for different number of nodes.

percentage of of all possible connections slightly decreases when there are more nodes in the roadmap. Again there are no identifiable differences between the four connection methods individually.

4.3.4 Connection methods

Finally we will compare the different methods used to connect the roadmap nodes. The results for creating a roadmap consisting of 50 nodes, while using a finger radius of $1/75^{\text{th}}$ the diameter of the polygon, can be seen in Table 4.5. Here the eight different methods are studied for the four different polygons. The number of edges are again averaged over 25 runs. The higher the number of edges, the higher the performance of the method.

The first thing we can see in Table 4.5 is that the number of edges roughly doubles if we use $\mathcal{M}_p$ instead of $\mathcal{M}_s$, independent of the method used to combine the movement of the polygon with the regrasping action, while constructing the

	Edges			
$\mathcal{M}$	P_1	P_2	P_3	P_4
Sp	200.56	149.88	135.88	150.72
Ep	200.20	149.68	135.84	150.20
Mp	215.20	159.96	135.84	150.36
Cp	200.64	150.04	135.96	150.72
Ss	114.32	81.24	71.80	80.28
Es	114.16	81.20	71.76	80.20
Ms	122.00	86.40	71.76	80.24
Cs	114.60	81.44	72.00	80.28

Table 4.5: The different methods for connecting roadmap nodes.

roadmap. This was to be expected given the similar results when building the grasp graph.

When only looking at either one of the regrasping methods there is one method that sometimes performs better than the other methods. The method that performs slightly better is method 'M' where first an attempt is made to move from a given placement-grasp-pair (p, g) to another placement-grasp-pair (p', g) without regrasping. Only when a collision occurs at placement p'' the object and fingers are jointly moved back in the direction of (p, g) until it is possible to regrasp the polygon. When the regrasp succeeds, the movement is continued in a second attempt to reach (p', g'). Because of this second attempt we expected this method to perform better than the other methods that only make one attempt to go from (p, g) to (p', g') in their respective ways. As we can see the other methods perform just as good as method 'M'. Only for polygons P_1 and P_2 a slightly higher number of edges is seen. We can therefore conclude that most of the collisions are between the polygon and the obstacles or that it is not beneficial to regrasp in the used environment. This shows that method 'M' has the possibility to outperform the other methods, but the layout of the environment is of great influence on the performance gain for method 'M'.

4.4 Conclusions

When looking at the differences between the methods provided to connect the nodes of the roadmap, we see that the roadmap will be denser when the regrasping is done with simmultanious finger movement. Besides that we see that the method which backtracks the movement of the polygon when a collision occurs performs slightly better than the other methods.

Therefore we can conclude that the best way to create a roadmap for a regrasping and moving problem for a passive convex polygon is to parallel regrasp and to backtrack on a collision till the regrasp can be made.

Chapter 5

Improvements on the used methods

5.1 Improving the grasp graph

A possible way to raise the average number of vertices in the connected components of the grasp graph, is to generate one grasp randomly and then create grasps that have three fingers in common with the grasps they are connected with. This causes all adjacent grasps to differ only by one finger. This may seem contradictory to the results that show that the simultaneous movement of fingers will generate a denser grasp graph. But if we know that only one finger is different between two adjacent grasps in the grasp graph, we can plan the regrasps better when moving the polygon. When a specific finger would collide with an obstacle, but the other three fingers would not, it is most natural to only move the finger that would cause the collision. This is be more efficient than moving all the fingers and prevents the stationary fingers to move in a position where they could cause a collision. In a grasp graph where we know we only move one finger between adjacent grasps, it will be easy to find all grasps for which the desired finger is moved while the other fingers stay at their current position. This will result in more efficient movement of the fingers, fingers that do not have to move will not move, except when there is no other way to regrasp the object in a satisfiable way. The notion of the regrasping methods we use here will disappear with this alteration of the grasp graph.

5.2 Improving the roadmap

The easiest way to improve the roadmap is to use general optimizations that can be used for all probabilistic roadmap methods and problems. One of these optimizations is to use the connected component check when connecting nodes. This will not raise the number of nodes in the connected components, but it

will lower the time needed to connect new nodes to the roadmap, thus enabling more nodes to be added to the roadmap in the same amount of time. Another optimization would be to lower the number of neighbors, so only nearby nodes are connected. The fully random sampling of configurations for the nodes could also be altered to a method that makes use of properties of the environment. In [6] Geraerts and Overmars give a good overview of the different methods to improve the probabilistic roadmap method.

The first improvement that could be made to the roadmap (besides the general roadmap optimizations mentioned above and which we have not used) is to use the information available in the grasp graph when assigning grasps to roadmap nodes. We could for example choose the grasp for the node we are adding to the roadmap after the first connection with that node is made. We then can make sure the grasp for the node we are adding is adjacent or at least connected to the grasp of the connected node.

This gives another way to improve the number of edges in the roadmap. Instead of only allowing adjacent grasps in the grasp graph to be connected, we could allow more than one regrasp between two roadmap nodes. This will probably lower the number of connections that cannot be made in the roadmap due to the impossibility to regrasp significantly. How the regrasps are performed during the transition between two roadmap nodes will be another problem that has to be solved, but this may well be worth the effort due to the improved connectedness of the roadmap.

Chapter 6

Conclusions

6.1 Results

We compared two methods for the regrasping of convex polygons with four disc-shaped fingers in two dimensions while constantly maintaining form closure. To do this a so called grasp graph was made. In this graph a number of randomly created grasps are connected to each other if and only if it is possible to regrasps between these grasps without losing form closure.

The first method that is considered is one where all fingers are moved simultaneously and the other method is one where the fingers are moved sequentially. The performance of the two methods is measured by the number of edges and the size of the largest connected component of the grasp graph. The more edges there are and the larger the largest connected component, the better the grasp graph and thus also the method for regrasping.

The results show that the simultaneous method consistently performs better than the sequential method, both on the number of connected edges as on the size of the largest connected component in the grasp graph. The combination of both methods performs only slightly better than the simultaneous method and is therefore not recommended if the simultaneous method is preferred. If the sequential method is preferred for regrasping, it may be worth the extra time to create a grasp graph that combines both methods, because the extra time needed to generate the information for the simultaneous method is roughly $1/24^{\text{th}}$ that of the time needed to generate a grasp graph for the sequential method.

Once the grasp graph can be created, we want to see how we can solve planning problems involving the regrasping and transportation of an object amongst obstacles in two dimensions. The planning problem will be solved with a probabilistic roadmap method. The nodes of the roadmap consist of a placement-grasp-pair that combines a random position and rotation of the object with a random grasp from the priorly compiled grasp graph. Four different methods to connect the roadmap nodes are studied. Again we want to see if there are one or more methods that perform better when generating a roadmap. The perfor-

mance is measured by the number of edges in the roadmap. The four methods are combined with either one of the two regrasping methods, so we can also see their influence on the connectedness of the roadmap.

The influence of the chosen regrasping method is greater than the influence of the chosen connection method. When the four methods for connecting roadmap nodes are compared when all using the same regrapsing method, the differences are negligible. But when comparing the same connection method with different regrasping methods, the results show again that the simultaneous regrasping method is the preferred one.

Three of the four methods regrasp the object while it remains stationary. The fourth one regrasps the object while simultaneously moving it. There are no results that show any advantage of the latter method when looking at the connectedness of the roadmap. An advantage this method has, is that it will take less time to perform both the motion and the regrasping of the object, but that will also increase the forces on the fingers, which could be a disadvantage when used in physical systems.

The four methods can also be divided into two groups when looking at the way in which collisions are handled. Three methods will cancel the attempt to connect two roadmap nodes when a collision occurs. The fourth method will move the polygon back until a regrasp can be done before starting a second attempt to reach the goal node. It is expected that this method will perform better than the other three. The results show that in some cases this happens, but most of the time the extra attempt to reach the goal node does not cause the number of edges in the roadmap to increase.

To conclude we can say that it is always recommended to choose the regrasping method where the fingers are moved simultaneously. For the roadmap node connection method the preferred one is the one where the regrasp is done after a collision would occur and continue the movement of the object with the new grasp.

6.2 Discussion

The fact that differences between the regrasping methods occur, but not between the roadmap connection methods raises some questions.

- Why are the differences we see between the regrasping methods as they are? It seems unnatural to move all fingers at the same time while regrasping. Are these differences a result of using frictionless fingers or is something else causing this? Answering this question will give more insights on improvements that could be made to the regrasping methods.

- Why do the four methods to connect roadmap nodes give similar results? And when we see differences when comparing the results, why are they so small and not consistent?

- What is the influence of the placement of the obstacles in the environment? It would be interesting to see how the methods perform in different environments. Will the presence of more obstacles reveal that there are differences between the methods used to connect the roadmap nodes, or will these methods still preform equally well?

- Are there better ways to generate the grasps and the configurations for the roadmap than randomly picking them? The results are unclear when comparing the different settings that determine the sample rate. Will these results be more clear when other methods are used to generate the nodes for the grasp graph and the roadmap?

6.3 Future work

Several approaches for future work on regrasping passive polygons amongst obstacles while regrasping the polygon while maintaining form closure are given. Besides the possible improvements given in Chapter 5, the questions still unanswered are maybe the best place to start. We can split the future work into different parts, starting with regrasping and path planning. The future work could also head in another direction where completely different methods for regrasping are used, while still solving the planning problem with a probabilistic roadmap method.

6.3.1 Passive objects and environments

When we look at the objects that we can handle, it would be a good addition to also be able to handle non-convex polygons. To accomplish this several of the algorithms used must be altered or extended. To start, the offset creation must be able to cope with concave corners of a polygon. A decision must also be made what happens when a finger is placed in a concave section of the polygon and touches the polygon at two or even more places at the same time. The most natural thing to do would be to determine the force each finger applies. In that way a situation as can be seen in Figure 6.1 will not be ambiguous. When the force of the finger is unknown, it is not clear whether the finger only applies force to the left side or the right side or to both sides in case the finger presses downwards.

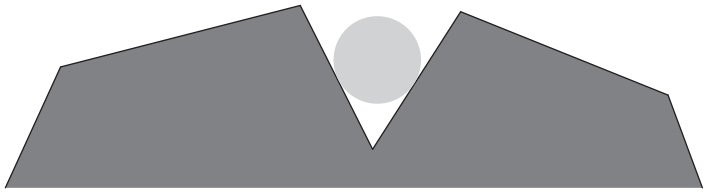


Figure 6.1: Ambiguous finger force in concave parts of a polygon.

6.3.2 Regrasping

The causes of the differences in performance between the two regrasping methods will be a good starting point to continue this research. If it is known what causes the differences, it will probably be possible to use this knowledge to improve the methods or to devise other methods.

The use of friction will change the regrasping part also, it will then be easier to regrasp between grasp, because less fingers are normally needed to immobilize an object when friction is taken into account.

Another idea for the future work is to consider different types of grasps. The use of enveloping grasps [17], which encapsulate the object, could be an interesting possibility. With this method an object is grasped by using the length of the fingers to hold the contact in place. If the fingers are articulated, it is possible for one finger to touch the object multiple times. Therefore it is easy to place an object in form closure with this type of grasp. But by enveloping the object, the room to regrasp will be smaller then when using only the finger tips to hold the object. Despite this last disadvantage, enveloping grasps could be interesting because they can place an object in form closure with less than four fingers, but still have four contact points with the object are needed. This can then be solved by using articulated fingers or by making line contacts between a finger and the object.

6.3.3 Path planning

Also for the path planning part it is interesting to find out why the differences between the used methods are so small. More information on the performance of these methods in different environments is certainly useful to make a decisive distinction between them. And also for these methods holds that there may be better methods to connect the nodes in the roadmap.

6.3.4 Learning algorithms

Instead of generating a grasp graph, it would be interesting to see if it is possible to use machine learning algorithms to let the system learn which grasps are valid and how to regrasp. In [4] a method is proposed to use a Hopfield neural network to determine a collision–free path a finger must take while regrasping. This approach could maybe be extended to also use a neural network to determine which finger positions are optimal. The use of machine learning methods could yield systems that can better handle objects of different shapes and different environments.

The use of a similar method as described in [4] for the finger movement while regrasping could maybe also be used in a slightly altered form for planning the motion of the entire object together with the fingers.

Bibliography

[1] X. Cheng. On-line Path Planning for Assembly Operations by a Two-Arm Robot. *iastedra93*, 1993.

[2] K. Cho, M. Kim, and J.-B. Song. Complete and rapid regrasp planning with look-up table. *J. Intell. Robotics Syst.*, 36(4):371–387, 2003.

[3] G. Cramer. *Introduction à l'analyse des lignes courbes algébriques*. Geneva, 1750.

[4] G. A. de Paula Caurin and L. C. Felicio. Learning based regrasping applied to an antropomorphic robot hand. In *ABCM Symposium Series in Mechatronics*, volume 2, pages 555–562, 2006.

[5] C. Ferrari and J. Canny. Planning optimal grasps. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 2290–2295 vol.3, 1992.

[6] R. Geraerts and M. Overmars. A comparative study of probabilistic roadmap planners. In *Workshop on the Algorithmic Foundations of Robotics[20] Y. Yu and K. Gupta. On sensor-based roadmap: Workshop on the Algorithmic Foundations of Robotics[20] Y. Yu and K. Gupta. On sensor-based roadmap: Workshop on the Algorithmic Foundations of Robotics*, 2002.

[7] Y. Koga and J. C. Latombe. On multi-arm manipulation planning. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 945–952, 1994.

[8] J. C. Latombe, P. Svestka, M. Overmars, and L. Kavraki. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *In IEEE International Conference on Robotics and Automation*, page 171, 1994.

[9] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[10] X. Markenscoff and C. H. Papadimitriou. Optimum grip on a polygon. Technical report, Department of computer science Stanford University, 1987.

[11] D. Nieuwenhuisen. Callisto. `http://nieuwenhuisen.nl/callisto/callisto.html`, 2007.

[12] F. Reuleaux. *Lehrbuch der Kinematik*. Braunschweig: F. Vieweg und sohn, 1876.

[13] R. Sedgewick. *Algorithms in C++*, volume 5 - Graph algorithms. Addison Wesley, third edition, 2002.

[14] T. Siméon, J. Cortés, A. Sahbani, and J. Laumond. A manipulation planner for pick and place operations under continuous grasps and placements, 2002.

[15] A. Sudsang and T. Phoka. Regrasp planning for a 4-fingered hand manipulating a polygon. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference*, volume 2, pages 2671– 2676, 2003.

[16] R. Wein. Exact and approximate construction of offset polygons. *Comput. Aided Des.*, 39(6):518–527, 2007.

[17] M. Yashima and H. Yamaguchi. Control of whole finger manipulation utilizing frictionless sliding contact-theory and experiment. In *Mechanism and Machine Theory*, volume 34, pages 1255–1269. Elsevier, November 1999.

Appendix A

General Mathematics

A.1 Graphs

In mathematics graphs [13] are used to represent the relations between pairs of objects. Grasps consist of vertices and edges. The vertices are the representation of the objects and the edges are the relations between the objects. Edges always connect two vertices. In some graphs it is allowed that an edge starts and ends at the same vertex. Whether or not this is possible depends on the relationship an edge represents.

This relationship also determines if the edges are symmetrical or not. If they are symmetrical a connection from vertex I to vertex II means there is also a connection from II to I. If the edges are not symmetrical the way to represent the above relation is to create two separate edges, one from I to II and another one from II to I.

When a graph is drawn the vertices are represented by circles and edges as lines between these circles. In Figure A.1 the vertices are marked with the roman numbers and the edges are marked with the lower-case letters. Despite the fact that it is not possible to traverse the graph from vertex I to VII, all vertices and edges shown are part of the same graph. This graph can be split in three separate subgraphs without altering the connectedness of the graph. These subgraphs are called connected components.

Connected components

A connected component is a maximal connected subgraph of the whole graph. Maximum connection of a subgraph means that it is not possible to add edges or vertices from the original graph to the subgraph while preserving its connectivity. In Figure A.1 a graph consisting of three connected components can be seen. The first component consists of vertices I, II, III and IV. The second consists only of vertex V and the third constsist of the remaining vertices, VI, VII and VIII.

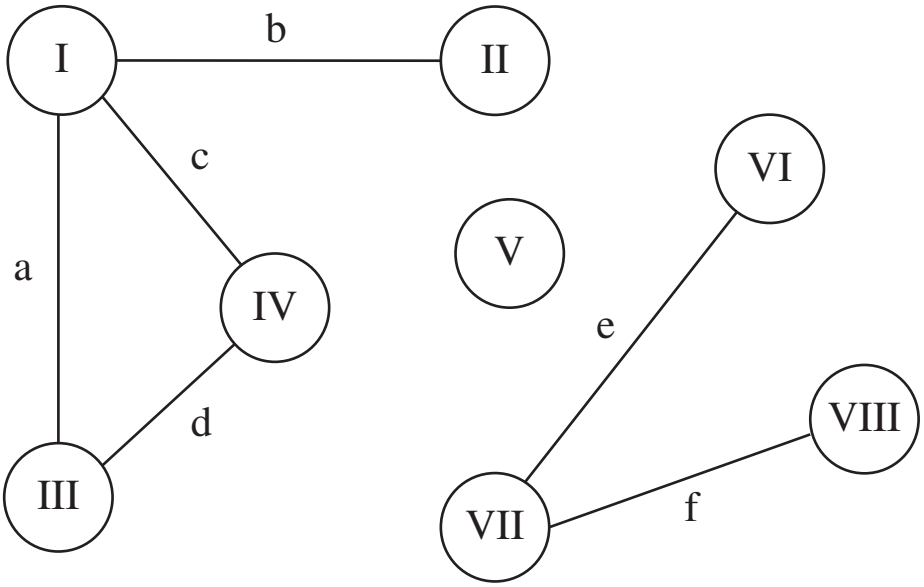


Figure A.1: A simple graph with three connected components.

Creating and counting connected components

Algorithm 2 is used to create and count the connected components in a graph. A queue q, on which the default operations can be preformed, is used to store which nodes n need to be processed in the current connected component of the road map G. To simultaneously determine the number of connected components the counter c_{cc} is used. When the active node is used the superscript a is used, while the subscript cc is used to denote the number of the connected component the node for which the subscript is used. The boolean *new_component* indicates whether or not a new connected component is found.

Algorithm 2 Building the connected components from a graph

```
c_cc ← 0
for all v ∈ G do
  new_component ← false
  if !v_cc then
    q.push(n)
    new_component ← true
  end if
  size ← 1
  while !q.empty() do
    v^active ← q.front()
    q.pop()
    if !n^a_cc then
      n^a_cc ← c_cc
      for all neighbor ∈ neigbors of n^a do
        if !neighbour_cc then
          q.push(neigbour)
          size ← size + 1
        else
          size ← size − 1
        end if
      end for
    end if
  end while
  if new_component then
    c_cc ← c_cc + 1
  end if
end for
```

Appendix B

Software

The software used to test the proposed methods uses the Callisto framework, developed by Dennis Nieuwenhuisen [11] for loading environments and collision checking.

B.1 Input

The following variables are used as input for the software. For some of these values it is interesting to see what their effect is on the results, others are just there to tune the algorithms for the given input.

Number of nodes in the grasp graph This number determines the number of grasps that are used to create the regrasp graph.

Number of nodes in the roadmap This is a variable that can be used to determine the filling of the environment with roadmap nodes. A low value will result in large distances between nodes and thus a greater chance of obstacles between the nodes. This will then result in a less strong connected roadmap.

Maximum overlap of the fingers between two samples The higher the overlap of the samples used, the lower the chance of incorrectly found regrasp possibilities. The aim for this variable is to get correct result with as less overlap and thus samples as possible.

Maximum sample distance while moving This sets the maximum distance between two samples in the roadmap building phase.

Finger radius factor Determines the radius of the finger by dividing the diameter of the polygon by the given value.

Polygon The passive object that will be manipulated by the four fingers. For testing purposes, four different polygons are used.

The environment The environment determines the obstacles amongst which the object has to be moved.

The regrasping method There are two methods of regrasping, moving the fingers simultaneously or sequentially while regrasping.

The method for constructing the roadmap This variable has got four possible values:

1. Regrasping at the start position, then moving to the goal position.
2. First moving to the goal position, then regrasping.
3. Moving until a collision occurs, then backtracking to a position where the regrasping can be completed and finally moving to the goal position while holding the object with the grasp corresponding to the goal position.
4. Simultaneously moving and regrasping between the start and goal positions.

These four methods must be combined with one of the two available methods for regrasping. Therefore the total number of ways to connect the roadmap nodes is eight.

B.2 Output

The following variables are all results from calculations done in the software and are used to get the results.

Number of tries needed to genrate a valid grasp This gives an indication of how hard it is to generate a valid grasp for the object.

The number of edges in the grasp graph The more edges that are connected, the more options there are to regrasp. For each regrasping method the number of edges is given. This gives an indication of the differences between the polygons.

The connected components of the grasp graph For each graph the number and sizes of the connected components are given. For both regrasping methods and the combination of the two separate results are given.

The number of edges in the roadmap This represents the connectedness of the roadmap. For all used methods of eight available ones the number of edges in the roadmap are given.

Connection failures when creating the roadmap For each of the used methods for creating the roadmap the number of failed connections between nodes are given, divided in the cause for the failure. The two causes can be failure to regrasp and a collision with one of the obstacles.

List of Figures

List of Tables

List of Algorithms