Master's Thesis

Computer Science

Minimising the Number of Collision Tests in Probabilistic Road Maps using Approximations in a Binary Connection Strategy

by
Dennis Søgaard
ds@daimi.daimi.au.dk

supervised by
Gerth Stølting Brodal

June 15, 2006

Department of Computer Science
Faculty of Science
University of Aarhus
Denmark

Abstract

In this thesis a short presentation of different solutions to solve the motion planning problem is done. The probabilistic road map approach is investigated further and the learning phase of this approach is spending most of its time on collision tests. An approach is developed to lower the number of collision tests done in this phase. The original approach uses an ε-enlarged object interpolated to connection configurations. The approach suggested uses over-approximations of the movement when connecting configurations in the local planner. A binary connection strategy is developed to do over-approximations at finer degrees to raise the connectivity of the road map. The approximation takes

$$O\left(k \cdot (n+m)\right) \ ,$$

where n is the maximum number of points in the components, m is the maximum number of connection points in the components and k is the number of components. The approach has been implemented to setup experiments that compare the collision tests done for different approaches, parameters of the approaches in multiple environments with several different objects. The experiments have shown several things. That larger search depths in the binary connection strategy increases the number of collision tests. Increasing the number of edges in the work space does not change the relationship in number of collision tests done by the different strategies and parameters. However it is shown that the complexity of the object moved increases the number of collision tests more for the suggested approach than the original approach. The final thing the experiments shows is that suggested approach can lower the number of collision tests for a wide range problems.

Contents

List of Figures

List of Tables

Acknowledgements

I want to thank my supervisor Gerth Stølting Brodal for countless meetings giving several pieces of good advice and proof reading. Furthermore I want to thank Mads Darø Kristensen and Søren Gjellerup Christiansen for the several hours of proof reading done and good advice. Finally I want to thank Jesper Torp Kristensen for proof reading.

Chapter 1

Introduction

Motion planning problem has its origins in robotics, where the ultimate goal is true autonomous robots. Autonomous robots are able to do what they are told to do without having to be told exactly *how* to do it. One of the challenges in creating such robots is the ability to plan movements.

Motion planning is considered with how the robot should get from a start position to a goal position. The robot may not be able to move in a straight line to get to its destination because of obstacles that should be avoided. An example of this could be a robot moving around in a factory, where there are obstacles such as walls and different machines on the factory floor, which the robot should avoid any collision with. These obstacles never moves so they can be predefined as a map to the robot. However obstacles such as humans and other robots can move around and the robot must rely on sensors to detect such obstacles. All information about the environment can be used by a robot to move around while avoiding collisions with obstacles. This is illustrated in Figure 1.1 where a construction robot is trying to get to a goal in a factory.

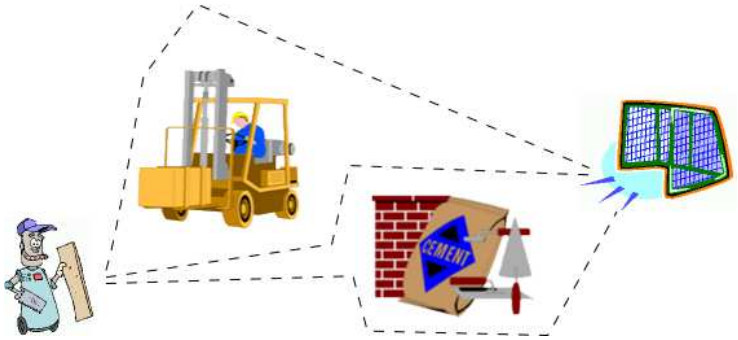


Figure 1.1: A construction robot is planning its motions to get to the goal while avoiding the obstacles.

There are several examples of where the motion planning problem exists. The example above is a rather rare example of real world usage of motion planning. A widely used example could be a robot arm that is used in the production industry. The robot arm consists of several components connected by joints so it can move some tools into correct positions. Another example

is within virtual environments and simulations, where an example could be a prototype of a building. In this building panic simulation could be done which implies that the people are moving around. This simulation is created to give an impression on how the building would work in reality before building it. To make the simulation realistic the people must move around realistically which demands motion planning.

1.1 Motion Planning Problems

When a robot plan its motions it uses a simulated world to make movement choices from. A simulated world is a representation of the real world which can be generated from sensors on the robot or it can be a model created by a human. From the simulated world the robot takes decisions and does actions in the real world.

The simplest problem in motion planning is the *point movement problem*, where the robot is a point in the plane. A shortest path for a point robot can be a path where the robot will slide along an obstacle. For a real robot this might not be convenient because there might be an error margin between the real world and the simulated world, which could result in collisions in the real world. This can be avoided by enlarging the obstacles by some safety margin so that there will be a bigger margin for errors between the real world and the simulated world, hence avoiding collisions in the real world. Figure 1.2 visualise the two different approaches, where the sliding effect of the shortest path can be seen and the keep distance to obstacles longer path. However, the enlargement may cause valid paths in the real world to be illegal in the simulated world. The optimal solution would be a trade-off between keeping distance to the obstacles and not invalidating too many paths. If the strategy is to find any path, and not necessarily the shortest path, the robot keeps as much distance to the obstacles as possible while finding a path. This approach would give longer paths, more safe paths that are more useful for real robots.

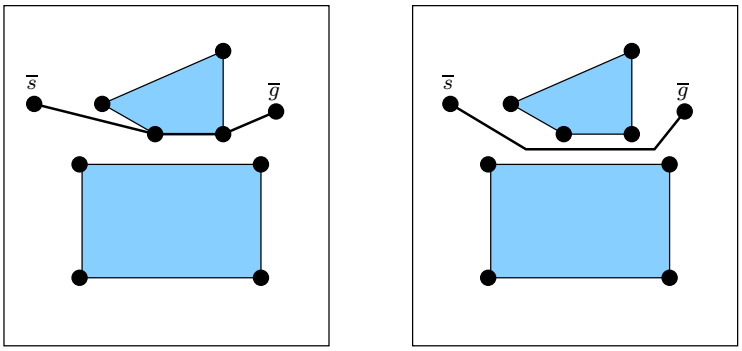
Motion planning of polygon robots that are moving with translations is different. A translation can by formulated as a vector addition :

$$\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x' \\ y' \end{bmatrix},$$

where (x, y) is the point and (x', y') is the translation vector. The *translation polygon movement problem* has a solution that builds on the point robot movement. The solution is to enlarge the obstacles such that the robot can be treated as a point robot. When the point is on the edge of an enlarged obstacle it represents that the robot is touching the edge of the real obstacle. After the enlargement the problem is reduced to solving the point movement problem.

If the robot is allowed to rotate it will make the motion planning problem much harder. The problem is now a *translation and rotation polygon movement problem*, where the robot can be orientated with an angle. A rotation with angle α around origo can be expressed as a matrix multiplication :

$$\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix},$$



(a) The shortest path from $\overline{s}$ to $\overline{g}$ where a point slides along one of the obstacles.

(b) A path from $\overline{s}$ to $\overline{g}$ where a point keeps distance to the obstacles.

Figure 1.2: Two figures show the difference between shortest path and keeping distance to obstacles in motion planning. Figure 1.2(a) shows the shortest path and Figure 1.2(b) show how a path would look if the robot would keep a distance to the obstacles.

where (x, y) is the point rotated. Again there is a solution that builds on the point movement problem by reducing the two dimensional robot to a three dimensional point. This is done by enlarging each obstacle for every possible orientation angle of the robot. After the enlargement the obstacles can be seen as twisted cylinders in three dimensions, where, at some height of the obstacle the two dimensional plane is equivalent to the obstacle enlarged with the robot that has an orientation equivalent to the height. This means that the problem is a three dimensional problem instead of a two dimensional problem, where the third dimension is the orientation of the robot.

Finally there is the *multi joint robot motion planning*, where a practical example could be a robot arm. This problem is a very hard motion planning problem because of the dimensionality of the problem. The robot might be able to translate and rotate which is three dimensions, and then, for each joint of the robot, there are an orientation between two parts of the robot. This means, that the number of dimensions in the problem grows with the number of components in the robot.

A robot may not be able to have joints with an orientation larger than some angle and the parts of the robot may not intersect. This can be represented as constraints so the robot stays in legal positions in the simulated world. A real world example of robots with constraints could be the robot arm. The arm might be mounted on the floor hence it cannot move around. The joints in the arm might only have a certain angle interval that they can move within or else the joint will break. The components of the arm are not allowed to intersect because this would break the robot. From the simple arm robot example there is several possible constraints which must be obeyed in the simulated world.

All examples above ar motion planning for a *single-mover*, which means that the robot is the only robot moving around. However several robots might be moving around in the factory and that is a *multi-mover* problem.

Whether the motion planning is static or dynamic depends on the information about the obstacles. In a *static motion planning problem* all informa-

tion about the obstacles is available and this information will not change over time. The *dynamic motion planning problem* is different because the information about the obstacles is only partial and might change over time. The static model can easily be simulated hence widely used in theory. However the dynamic motion planning is the most realistic model for robotics because information about the obstacles might not be complete. An example could be the factory robot that moves around on the factory floor but has to avoid walls, machines and humans. However, in the robot arm example there is enough information to make a virtual model and solve the motion planning problem in a static environment.

The robot may change geometric shape during the motion planning, which is the *conformable motion planning problem*. The robot arm example is a non-conformable motion planning problem.

Time-varying motion planning problem is when obstacles can change shape, move around, appear or disappear over time. This basically means that there is one more dimension added to the problem, namely time. The non-changeable is a time-invariant motion planning problem.

If the robot can move a subset of the obstacles the problem is called a *movable-object motion planning problem*.

1.2 An Overview

The problem domain in this thesis is a static, non-conformable, time-invariant, single mover motion planning for multi joint objects in two dimensions without constraints. The hardest problem looked into in this thesis is linked polygon movement, which is polygons linked together. The robot is the only moving object, and the environment is a static environment that does not change during the run. The robot is referred as the moving object in the thesis, because it represents a general moving object. There is no constraint handling of the object moved, hence in the linked polygon case it can go into any position, with any orientation between the joints.

In this thesis the standard probabilistic road map approach is used to solve the motion planning problem. The construction of the road map use much time with collision detections in the local planner. To lower the number of collision tests an improvement to the local planner is proposed. Where the local planner uses approximations when attempting to connect configurations. To show the improvement several test are done in multiple environments with several object, to give a good basis for showing the lowering in the number of collision tests.

Background material, such as graph theory, geometric computations, operators, spaces, road map, and more, are defined for later use in Chapter 2. After the background material in, Chapter 3, there is a short summary of different data structures and algorithms that can help solve the motion planning problem, and the probabilistic road map is selected for further investigation. Then, in Chapter 4, the local planner of probabilistic road maps is investigated. The local planner is the module responsible for telling whether the robot can move from a start position to a goal position by using translations and rotations. Here it is interesting to see whether it is possible to approximate the movement of the robot and then only test for intersection on the approximation. This could save many intersection tests, which are very expensive for the algorithm. In

Chapter 5 the implementation of the probabilistic road map and the approximation approach is described. The reason to implement it is to test whether the approximation approach is functional in a real application. Chapter 6 is the description on how the different debugging tests of the implementation are created and executed in this thesis. An experiment comparing the number of collision tests done by different connection strategies is done in Chapter 7. There are two local planners tested: the original local planner from the original article by Kavraki *et al.* [11] and a new local planner developed in this thesis.

Chapter 2

Background Material

This chapter introduces the background material that is used throughout the rest of the thesis. Basic graph theory, geometric shapes are introduced.

2.1 Graphs

Graphs are a big part of motion planning algorithms. Here graphs are used to find paths, finding areas of interest for further investigation. The definition of graph and path is related to chapter 6 in the book by Goodrich and Tamassia[7] which is about graphs.

A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E connecting these vertices. An edge $e \in E$ defined by the two vertices $v_i \in V$ and $v_j \in V$ that it connects and is therefore denoted as $e = (v_i, v_j)$.

There is a distinction between *directed* and *undirected* graphs. In a directed graph the edges are directed. Thus if there exists an edge $e = (v_i, v_j)$ in the graph G it is possible to move from vertex v_i to vertex v_j but not in the opposite direction. In undirected graphs the following must apply $(v_i, v_j) \in E \wedge (v_j, v_i) \in E$ hence an edge must be represented by both edges in the edge set.

A path $P_{i,j}$ in a graph is a sequence of alternating vertices that start at vertex v_i and end at vertex v_j, such that each edge is incident to its predecessor and successor vertex $P_{i,j} = [v_i, \ldots, v_j]$.

Connected components in an undirected graph are sets of n vertices $S = \{v_1, \ldots, v_n\}$ where for each pair of vertices v_i and v_j in the set there exists a path $P_{i,j}$ from vertex v_i to vertex v_j. If a vertex has no edges it is considered in a connected component itself.

2.2 Basic Geometry

Points, vectors, edges and polytopes are basic geometric structures which are defined in n dimensional space. Polygons and polyhedrons are dimensional bounded polytopes, and simplicity and convexity are properties for polytopes.

- *A point p, q or r* in n dimensional space is an n-tuple of real numbers $p = (\beta_1, \beta_2, \ldots, \beta_n)$.

- *An edge* e in n dimensional space is a pair of points, $e = (p, q)$, where $p \in \mathbb{R}^n$ and $q \in \mathbb{R}^n$
- *A vector* $\overrightarrow{v}$ in n dimensional space is an n-tuple of real numbers

$$\overrightarrow{v} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}$$

- *A polytope* P in n dimension space is represented as a set of m points and k edges that defines the boundary of the polytope.

In general when talking about a geometric definition A in a specific dimension n it will be represented as A^n. There are two general definitions of the polytope is, the polygon and the polyhedral, which are defined as :

- *A polygon* P^2 is a polytope in two dimensional space
- *A polyhedron* P^3 is a polytope in three dimensional space

A polytope can have two properties, it can be *simple* or it can be *convex*. These properties are defined as following :

- *A simple polytope* $\widehat{P}$ is a polytope where the boundaries of the polytope are not allowed to intersect, and the polytope is on closed form. By closed form means that there are no holes in the polytope.
- *A convex polytope* $\overline{P}$ is a polytope where there is no intersections at the half planes bounding the polytope.
- *A linked polygon* $\widetilde{P^2}$ is a tree structure, where a tree node n is a three tuple $(n_p, PL, \widehat{P^2})$, where n_p is the parent node, PL is the list of connection points children can be connected to and $\widehat{P^2}$ is a polygon part.

2.3 Operators

The operators used between different geometric structures are defined in this section. Section 2.3.1 contains definitions on addition, subtraction, and division between different basic geometric structures such as points, vectors, and polytopes. The Minkowski Sum operation is defined in Section 2.3.2.

2.3.1 Basic Operators

Simple operators like addition and subtraction can be applied between points, vectors and polytopes. Additionally an angle can be transformed into a rotation matrix and then multiplied to a point or polytope to rotate it around origo.

Addition between two points, p_1 and p_2 is adding the value of each dimension together into a new point

$$p + q = (p_1, p_2, \ldots, p_n) + (q_1, q_2, \ldots, q_n) = (p_1 + q_1, p_2 + q_2, \ldots, p_n + q_n) .$$

This applies to the point subtraction and multiplication operator as well. The vector addition and subtraction operator is also the operator applied between each dimension into a new vector. A vector can be added to a point to translate the point, the result is a new point

$$p+\overrightarrow{v} = (p_1, p_2, \ldots, p_n)+(\overrightarrow{v}_1, \overrightarrow{v}_2, \ldots, \overrightarrow{v}_n) = (p_1+\overrightarrow{v}_1, p_2+\overrightarrow{v}_2, \ldots, p_n+\overrightarrow{v}_n) \ ,$$

which is also shown in Chapter 5 of the book by Foley *et al.*[5]. Subtraction is also possible the operator is replaced with the minus operator instead and the result is also a point.

Operators applied between a polytope and either a point or a vector, are the operator applied on each point in the polygon and the point or vector. Let P be a polytope of n points, where point $q \in P$ and let p be the point added to the polytope, then the addition is

$$P + p = \{q_1 + p, q_2 + p, \ldots, q_n + p\} \ ,$$

where the result is a new polygon. For adding a vector to a polytope it is the same. Let $\overrightarrow{v}$ denote the vector added then

$$P + \overrightarrow{v} = \{q_1 + \overrightarrow{v}, q_2 + \overrightarrow{v}, \ldots, q_n + \overrightarrow{v}\} \ ,$$

defines the addition between a polytope and a vector. Again the result is a new polytope.

Rotating a point p by an angle α returns a point q where the resulting point q is the point p rotated a around origo. In two dimensions it is computed as following

$$\alpha \cdot p = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \times \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \ ,$$

which can be written as

$$(p_1 \cdot \cos\alpha - p_2 \cdot \sin\alpha, p_1 \cdot \sin\alpha + p_2 \cdot \cos\alpha) \ .$$

This is also shown in chapter 5 of the book by Foley *et al.*[5] and the operation can also be applied to a polytope is the same way as the addition and subtraction operator.

2.3.2 Minkowski Sum

The Minkowski Sum is widely used in motion planning, because it has the property to enlarge a polygon with another polygons size. An example of this usage is the construction of a configuration space for a translating polygon. Here each obstacle is enlarged by the polygon size and reducing a motion planning problem to a point movement problem, which is seen in chapter 13 in the book by Berg *et al.* [4].

Formally the Minkowski Sum is defined as the vector sum of two sets of vectors and is denoted $\oplus$. Let S_1 be a set of n vectors and S_2 be a set of m vectors. Then the Minkowski Sum of S_1 and S_2 is :

$$S_1 \oplus S_2 = \{p + q \mid p \in S_1 \wedge q \in S_2\} \ .$$

It is required that, the points of both sets are of the same dimensionality.

2.4 Spaces

In motion planning there is in general three different spaces which are used, these are the *work space*, the *Configuration Space (C-Space)* and the *free space*. The work space represents a real model of the problem while the C-Space is an approximated problem space. The C-Space was introduced in the article by Lozano-Perez [14] and is described in Chapter 13 of the book by Berg *et al.* [4]. The definition of the work space, the C-Space and free space is in Section 2.4.1, Section 2.4.2 and Section 2.4.3, respectively.

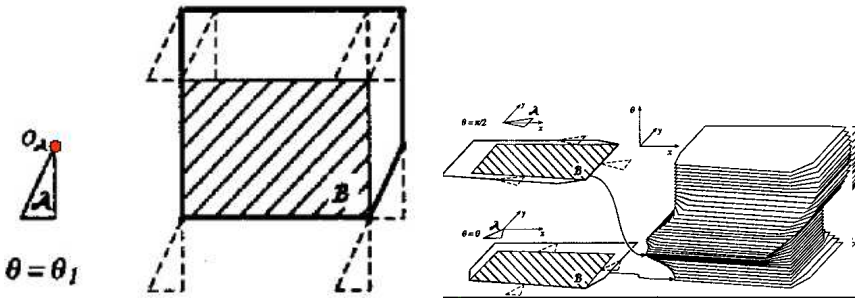
2.4.1 Work Space

The work space is the actual representation of the real world but in the approach used in this thesis there are some simplifications. There are no curves in the workspace, which have the effect that arcs in the real world has to be represented as polygons at some resolution in the work space. There is a limit on the precision of the work space, hence it will not be a precise representation of the real world. A consequence of the representation is that the obstacles in the real world has to be over approximated in the work space to ensure the validity of paths. However this might cause paths in the real world to not be omitted in the work space.

The work space is a polytope in $\mathbb{R}^n$ that contains a set of obstacles $S = \{P_1, P_2, \ldots, P_i\}$ where each obstacle is represented as a simple polytope. The object can be placed into the work space through a reference point R. This reference point is an e-tuple $R = (\gamma_1, \gamma_2, \ldots, \gamma_e)$ where $m = \binom{n}{2}$ and n is the dimensions of the work space. The first n points are the position of the reference point and the rest is the angles specifying the orientation of the object. Hence a reference point is then $R = (\gamma_1, \ldots, \gamma_n) + (\gamma_{n+1}, \ldots, \gamma_m)$ containing both a position and a orientation.

2.4.2 Configuration Space

The C-Space is a transformed work space, by some function or algorithm that takes a work space as input and generates a C-Space as output. A configuration can be seen as a translated reference point for the polytope in the workspace. I.e. a configuration describes how the polytope can be placed in the work space. The definition of a configuration to a workspace in two dimensions with a linked polygon that translates and rotates is as a "polytope" in the non-real space $\mathbb{R}^2 \times [0 : 2\pi[^m$. Here m is the number of orientations which is the number of polygons linked together instead of a polygon in R^2 as the work space. A configuration c in C-Space is a point in the C-Space. The point is a n-tuple on the form $c = \beta_1, \beta_2, \ldots, \beta_n$. As the work space, the C-Space contains a set of obstacles $S = \{P_1, P_2, \ldots, P_i\}$ but each of them is represented as either a polytope or a polytope with curved edges, that can be seen as a sort of "cylinder". The curves comes from the orientation where the obstacles enlarged are rotated along the z-axis. Figure 2.1 shows how translation alone does not increase the dimensionality of the C-Space but the translation and rotations do. Further it shows how the rotation and translation make a cylinder liked obstacle.



(a) The C-Space of a translation object.

(b) The C-Space of a rotating and translating object.

Figure 2.1: Two figures of a configuration space obstacle where Figure 2.1(a) of a translating object and Figure 2.1(b) of translating and rotating object. Both figures are takes from Latombe's lecture notes[13].

The concept cylinder is not entirely correct. It makes more sense in a 2D work space where objects can be translated and rotated. Here the configuration is a three tuple $c = (x, y, \alpha)$ where x and y describes the x- and y-coordinate of the reference point and α describes the orientation. The C-Space is a non-Euclidean space : $\mathbb{R}^2 \times [0 : 2\pi[$. The reason for this is that angles in $]-\infty : 0[$ or $[2\pi : \infty[$ can be described as an equivalent angle in $[0 : \pi[$. Hence, the configuration space of a translating and rotating object has a topology, which is like a cylinder.

Each of the obstacles can be seen as a set of i points $S = \{p_1, p_2, \ldots, p_i\}$, where each point is represented as a n-tuple, $p = (\beta_1, \beta_2, \ldots, \beta_d)$ where n is the dimension of the C-Space.

Configuration Space Obstacle

The Configuration Space Obstacle (C-Obstacle) represents the area which is forbidden to the moved object, if the object is represented as a configuration. This means that the motion planning can be lowered to path planning for points instead of the object. The simplest form for C-Obstacle is an obstacle for a convex translating polygon which is a Minkowski Sum of a work space obstacle and the inverse translating polygon. It is assumed that the obstacles are convex.

The C-Obstacles of convex obstacles for a translating convex obstacle are defined as a set of operations. Let R_o define the object at origo, let P define an obstacle in the work space and finally P_c define the C-Obstacle. To move an object around let $R_o(\beta_1, \ldots, \beta_d)$ define the object at origo. Let r define the position $(\beta_1, \ldots, \beta_d)$ hence R_r is the object at position r. Now the C-Obstacle P_c of obstacle P_x is defined as the set of points where the object P_o and the P_x intersects, $P_c = \{(\beta_1, \ldots, \beta_d) : P_o(\beta_1, \ldots, \beta_d) \cap P_x \neq \emptyset\}$.

Theorem 1 *Let R be a translating object and let P be an obstacle, then the C-Obstacle of P is $P \oplus -R_o$.*

Proof. To prove Theorem 1 the following property has to hold:

$$p \in R_r \cap P \Leftrightarrow r \in P \oplus (-R_o).$$

It is assumed that $R_r \cap P \neq \emptyset$.

To prove $p \in R_r \cap P \Leftarrow r \in P \oplus (-R_0$ let $p \in R_r \cap P$. As $R_r - r = R_o$ the following is obtained:

$$\begin{array}{l} p - r \in R_o \\ \Updownarrow \\ -p + r \in -R_o \end{array}$$

By adding the $p \in P$ the left side becomes a member of the Minowski Sum $P \oplus (-R_o)$:

$$\begin{array}{l} p - p + r \in R_r \cap P \oplus (-R_o) \subset P \oplus (-R_o) \\ \Downarrow \\ r \in P \oplus (-R_o) \end{array}$$

To prove that $r \in P \oplus (-R_o) \Rightarrow p \in R_r \cap P$ let $q \in R_o$ be a point at origo. Then there exist a point $p \in P$ in the obstacle such that $r = q + p$ and it can be determined that $p \in R_r$ as $p = q + p - q = r - q$. As $p \in P$ by assumption it can be concluded that $p \in R_r \cap P$. $\square$

2.4.3 Free Space

The free space can be defined as the space where the moving object is allowed to move within. There are two types of free space: the free space of the workspace and the free space of the C-Space.

Let W_{free} be the set of points that defines the free space in work space, let S be the set of obstacles in the work space and let p be a point in work space. Then the free space of the workspace can be defined as the set of points where

$$p \in W_{free} \text{ iff } \forall P \in S : p \notin P \ ,$$

and where the $p \in \mathbb{R}^n$.

Let C_{free} be the set of configurations that defines the free space in C-Space, let S be the set of C-Obstacles in the C-Space, and let c be a point in C-Space, hence a configuration. Then the free space of the C-Space can be defined as the set of configurations where

$$c \in C_{free} \text{ iff } \forall P \in S : c \notin P \ .$$

There is one difference from the free space of the workspace and that is $c = (\beta_1, \ldots, \beta_n) \times (\beta_{n+1}, \ldots, \beta_{2n}) \times \ldots \times (\beta_{(m-1)n+1}, \ldots, \beta_{nm})$ and $c \notin \mathbb{R}$, unless it is the configuration for a point problem or a translating polygon.

2.5 Road Map

The set C is the set of possible configurations and the set C_{free} is a subset of S with free configurations, e.i. the configurations in C_{free} are collision free. The configuration $\overline{s}$ is the start configuration and the configuration $\overline{g}$ is the goal configuration. The road map is an undirected graph $R = (C, E)$ where the set C is a subset of the set of free configurations C_{free}, and the edges in E are legal paths between configurations. I.e. for a configuration c_1 and a configuration c_2 there is a an edge $(c_1, c_2) \in E$. An edge in E is not just a path between

two configurations. The path has to be a feasible path. A feasible path is a path where the object can go from one configuration to the other configuration without colliding with the surrounding environment and keep its own movement constraints. The set of path E is only a subset of all possible paths between configurations in C.

Chapter 3

Motion Planning Algorithms

In this section several solutions for solving the Motion Planning (MP) problem will be described and discussed. The article of Hwang *et al.*[9] divides the approaches into three groups. At least one approach from each group is investigated further in this chapter. The three groups are: the skeleton approach in Section 3.1, the cell decomposition approach in Section 3.2 and the Potential-Field approach in Section 3.3. The skeleton approach contains the Visibility Graph (V-Graph) structure, the slide structure and the Probabilistic Road Map (PRM) structure. The cell decomposition contains two simple structures, the grid and the quad-tree representation. The potential field approach introduces the original approach. In Section 3.4 the different approaches are compared by how to construct the structure, how to query the structure, and the memory usage.

3.1 Skeleton

There are several different algorithms to solve MP that goes under the term a skeleton approach. The approach is to generate a Road Map (RM) that consists of a set of way points and paths between the way points. This means to generate a graph representing paths of movement. These paths can be generated differently and with different effects. This road map can then be queried to find motions that can be used to get from a start configuration to a goal configuration. In this section three different skeleton structures will be investigated: the V-Graph from Chapter 13 of the book by Berg *et al.*[4], the slide structure by Perez [15][14] that builds on the V-Graph, and the PRM by Kavraki *et al.*[11].

3.1.1 Visibility Graph

The main idea in the V-Graph is to construct a graph G for point movement, where the set of vertices V is the set of all vertices of all obstacles and E is the set of all edges where the two vertices in the edge is visible to each other. Let Os Let a vertex $v_i \in V$ and let a $v_j \in V$ then an edge $e = (v_i, v_j))$ is a straight line between these two points. Two vertices are visible if the edge between them

does not intersect any edges of the obstacles. The query for finding a path for a point is from a start configuration $\overline{s}$ to a goal configuration $\overline{g}$ is connecting both configurations to the graph and then find the shortest path or report failure if none exists. The connection of the configurations is done by adding edges to the vertices that are visible for the node. Finding the shortest path is done with Dijkstra's shortest path algorithm.

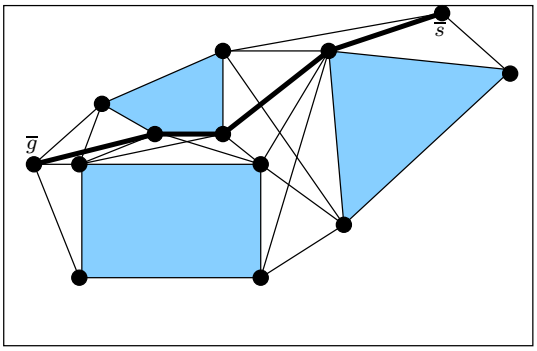


Figure 3.1: Figure of the visibility graph, where the shaded areas are obstacles, the $\overline{s}$ is the start configuration and $\overline{g}$ is the goal configuration. The shortest path is marked with wider edges.

Let n be the number of vertices in the graph, then for point path planning it is trivial to see that the algorithm can run in $O(n^3)$. That is done by connecting each vertex to all the other vertices, each connection is then tested whether it intersects any edges of the obstacles, unless it represents the edge. If the edge intersects then the point is not visible else the point is visible and the edge is added to the edge set E of the V-Graph. However the algorithm can be improved to $O(n^2 \log n)$ by a V-Graph construction algorithm given in the book by Berg *et al.* [4], which was proved in the article by Nilsson[16]. The chapter describes how the V-Graph is constructed with three functions : a construction function "VisibilityGraph" that generates the graph, a function that create a set of visible vertices function "VisibleVertices" which finds visible vertices for a given vertex and finally "Visible" that test whether two vertices are visible to each other. The construction function is the only function looked into in this thesis because the other functions are only interesting if the running time has to be shown. To make the V-Graph work for harder problems than point movement the C-Space is generated for that problem. The C-Space is then used to construct a V-Graph, that can be queried to find the shortest path between configurations. Finding shortest paths in three dimensions or more is known to be a NP-Hard problem.

Construction

The construction part of the algorithm takes a set of disjoint obstacles and it only works for point movement problems. It initializes the V-Graph with an empty edge set and a set of vertices corresponding to all vertices of all the obstacles. Then for each vertex in the set of vertices $v \in V$ it finds the set W of visible vertices for vertex v. Now W contains all vertices visible for V, and

```
VisibilityGraph(S : set of disjoint obstacles)
V ← all vertices in all obstacles
E ← ∅
G ← (V, E) : G is the Visibility Graph
∀ v ∈ V
  W ← VisibleVertices(v, S)
  ∀ w ∈ W
    E ← E ∪ (v, w)
return G
```

Pseudocode 3.1: Pseudo code showing the construction of the V-Graph.

this information must be stored in the graph G, so for each vertex in the set of visible vertices $w \in W$ an edge from the two vertices $e = (v, w)$ is added to the set of edges E in the V-Graph. Pseudo Code 3.1 shows this part of the algorithm.

What is interesting about the construction function is the input. It demands that the input is a disjoint set of obstacles and the construction algorithm only works for point movement problems. Hence something must be done to the input if a V-Graph should be constructed for harder problems than the point movement problem.

Visibility Graph for Translating Polygon

Solving the problem of motion planning for a translating polygon with a V-Graph is done as follow: Create a C-Space for the work space and the polygon translated. This is done by generating new obstacles enlarged by the Minkowski sum between the obstacle and the polygon. The C-Obstacle in C-Space is used as the set of obstacles to the "VisibilityGraph" function and the V-Graph is generated for the translating polygon. The C-Space is still within two dimensions which ensures that the V-Graph can be calculated with $O(n^2 \log n)$ where n is the number of vertices in all the obstacles of the C-Space.

Visibility Graph for Harder Problems

For a polygon in two dimensions that both rotates and translates the C-Space becomes three dimensional. Canny [3] proved in 1987 that the problem of computing a shortest path connecting two points among polyhedral obstacles in three dimensional space is NP-hard. This renders the visibility graph construction for a polygon that both rotates and translates to be NP-Hard. However the slide structure in Section 3.1.2 uses visibility graphs where it approximated the shortest paths for NP-Hard problems.

Conclusion for Visibility Graph

The V-Graph can find the exact shortest path for a point movement and polygon translating in $O(n^2 \log n)$. If the polygon rotates and translates the C-Space is in three dimensions and the problem becomes NP-hard. The conclusion must

be that V-Graphs alone are not viable for harder MP problems such as linked polygon movement.

3.1.2 Slide Structure

The slide structure builds on the V-Graph, by constructing a C-Space and if the C-Space is three dimensional then divide it into n slides, where n is dependent on how great the rotation resolution should be. A slide is a projection of a part of the three dimensional C-Space into the plane. Dividing the C-Space is done by slicing through the z-plane in some step size. Smaller step size results in better resolution of the work space into the C-Space. Each slide is then projected down to (x,y)-plane, where it is the outline of the obstacles that are the obstacles in the slide. When all slides have been created a visibility graph is constructed but the search space has been extended. It is possible to connect a vertex from one slide with a vertex from neighbour slide. If the MP problem is for a translating and rotating polygon then the C-Space generated is three dimensional where the third dimension is the orientation of the object moved. The cost of such an edge might be the Euclidian distance between two configurations and some cost for the rotation. The rotation cost is added to make straight path without rotations a preferred choice. After the construction of the V-Graph the shortest path in the graph can be found by using Dijkstra shortest path algorithm.

Conclusion for Slide Structure

A V-Graph created with the sliding algorithm can handle the case of a multi linked polygon that translates and rotates. The path found may not be the shortest path in workspace, and there might exist a path in workspace that is not in the C-Space. The V-Graph may get very large if the obstacles contain many vertices.

3.1.3 Probabilistic Road Map

A PRM is a RM between different configurations, and the initial phase of the PRM construction is the selection of nodes done with done with *random positions*. The PRM is a graph, where each vertex in the graph represents a configuration, and each edge in the graph is an object that can move by a fixed type of movement, between two different configurations without colliding with the surrounding environment. Further if the object has constrains on its movement and/or has many Degrees Of Freedom (DOF) these constraints has to be kept as well, while the object is moving between two configurations, before an edge is legal.

The PRM algorithm is, in the original article by Kavraki *et al.* [11], divided into two phases: a learning phase where the road map is constructed, and a query phase where the shortest path, if any exists, in the road map between two configurations is found.

Learning Phase

The learning phase is the phase that builds the probabilistic road map, that is used in a later phase. Basically the learning phase is divided into two steps.

```
N ← ∅
E ← ∅
loop
  c ← a randomly chosen free configuration
  N_c ← a set of candidate neighbours of c chosen from N
  N ← N ∪ {c}
    ∀n ∈ N_c in order of increasing D(c,n) :
      if ¬same-connected-component(c,n) ∧ Δ(c,n) then
      E ← E ∪ {(c,n)}
      update R's connected components
```

Pseudocode 3.2: Construction step of the PRM algorithm.

Step one is the construction step and step two is the expansion step. The first step must always be done before step two.

Construction Step

This step constructs the PRM by selecting random configurations in c_{free}. When enough configurations have been selected the configurations are connected if they have feasible paths. To speed up the algorithm only configuration within a certain distance, that are not in the same connected component as the new configuration, are attempted connected. This prevents cycles being created in the construction step, because a configuration is never connected two times to the same connected component. The absence of cycles does not make the road-map worse in the sense of possible reachable configurations, it can only make the paths longer. Shortcuts will be handled in the learning step or in the query phase by a smoothing technique instead of using time in the construction step. Pseudo code 3.2 shows how the construction step works and is taken from the original article by Kavraki *et al* [11].

Creation of random configurations happens by a uniform random sample of c_{free}. Such a configuration can be achieved by uniformly choosing each of its coordinates from an interval of values. The intervals correspond to each DOF. The obtained value is then checked for collisions with the obstacles, and if it is collision free it will be added to N.

The local planner checks the line segment between two configurations for collisions and joint limits. Checking the joint limits is straight forward, but collision detection is somewhat harder to do. Collision detection is done by discretizing the segment into a number of configurations $c_1, c_2, ..., c_m$, such that no pair of consecutive configurations (c_i, c_{i+1}) are further away from each other than some ε, where $\varepsilon > 0$. To contain this movement the moving object is grown with ϵ which gives an enlarged object. Then for each configuration from c_1 to c_m is tested if the enlarged object at the configuration, is collision free. If none of the m configurations yields a collision the path must be collision free. Enlarging the moving object grown by ε only has to happen once because ε is a constant in the local planner.

Selection of node neighbours is done by a distance function. If a distance function D yields a value smaller than some *maxdist* then the node is in the

neighbourhood of the node. This can be defined as

$$N_c \subseteq \{\overline{c} \in N \mid D(c, \overline{c}) \leq maxdist\} ,$$

where nodes in N is considered candidate neighbours of c if they are within the $maxdist$ distance from c.

The distance function reflects the chance that the local planner will fail to compute a feasible path between the pair (c, n) of configurations. The distance function $D(c, n)$ is used to both sort and construct the set N_c for candidate solutions of each new node c:

$$D(c, n) = \max_{x \in object} ||x(n) - x(c)|| ,$$

where x is a point on the object, $x(c)$ is the position of x when the object is at configuration c, and $||x(n) - x(c)||$ is the Euclidean distance between $x(c)$ and $x(n)$.

Expansion Step

In easy scenes the number of nodes, that are generated by the first step, are large uniformly scattered on c_{free} and the connectivity in R is also high. But if c_{free} is more constrained then R contains a few large components and several small component, and there is low connectivity. This does not effectively capture the connectivity of c_{free}.

The idea, is that the expansion step expands the connectivity of R. If the R has no connectivity at some area but c_{free} has then this area is a hard and narrow region, and the connectivity needs to be expanded. The expansion step works by selecting some nodes from N that are likely to be in such an area and expanding them. Expanding a configuration c means adding a new free configuration from the neighbourhood of c, and adding the configuration to N.

For the expansion step a probabilistic scheme can be used. For all node c in N there is associated a positive weight $w(c)$. This weight is a heuristic measure of the hardness of the region around c, the higher the weight the harder the region. The weight is normalised so that they all sum to 1, and then a node is selected from N with the following probability

$$Pr(c\_is\_selected) = w(c) .$$

This node is then expanded. Then the selection and expansion is continued for some iterations.

The definition of the heuristic weight $w(c)$ is defined as follows. If the local planner often fails to connect c to other nodes, then c is in a difficult region. The function should to some extend depend on the input scene.

- The failure ratio function $r_f(c)$ is defined by

  $$r_f(c) = \frac{f(c)}{n(c) + 1} ,$$

  where $n(c)$ is the number of times the local planner has attempted to connect c to another node, and $f(c)$ is the total number of times the local planner failed.

- In the beginning of the expansion step all nodes weight are scaled appropriately so they sum to 1

$$w(c) = \frac{r_f(c)}{\sum_{a \in N} r_f(a)} .$$

For expansion the Random Bounce Walk (RBW) is used. This works by moving in a random direction until a collision occurs, then a new random direction is chosen. There is a limit on how far it can go from its start position by limiting the computation time allowed for each node c, that is chosen for expansion. If an expansion starts in node c and ends in node n, then the edge (c, n) is included in R. The fact that the node c is also in the same connected component as n is recorded and n is attempted connected to the rest of the network as in the construction step.

When the expansion step is over the remaining small components of R are removed. Small means that the component has less nodes than some min-component percent of the total number of nodes.

Query Phase

During this phase paths between arbitrary start and goal configurations are found. A query consists of a start configuration s and a goal configuration g, and these two nodes are attempted connected to two random nodes in R ($\overline{s}$ and $\overline{g}$) with feasible paths P_s and P_g. If no such paths exists the query fails, otherwise the path P that goes from $\overline{s}$ and $\overline{g}$ is computed. Finally a path from s to g is computed by concatenate P_s, P and P_g reversed.

The Paths P_s and P_g is computed in the following way. Try to connect s to R by choosing the node $\overline{s}$ from R in increasing order by the distance function D and using the local planner. This is done until a feasible path is found within the threshold of *maxdist*. If all connection attempts fail one or two random-bounce walks are performed, where instead of adding the node at the end to R, the node is attempted connected with R with the local planner. As soon as s is successfully connected to R the same strategy is applied with g.

It may be that the road map consists of several connected components R_i for $i = 1, 2, ..., p$. This happens when the free C-Space (C_{free}) is not connected or the road map is not dense enough. When there are several components the start configuration s and the goal configuration g is attempted connected with nodes from one of the components, in increasing distance from s and g, one at a time. A component R_i has the distance to s, g defined by the maximum distance between the distance $D(s, R_i)$ and the distance $D(g, R_i)$. This returns failure when it, for all of the components fails to connect s and g to the same component.

If the path planning fails frequently it indicates that too little time is used in the learning phase. To fix this the current run can be extended by extending the current road map to get a more dense road map.

Conclusion for PRM

Generating random initial points is a fast way to divide the space into partitions and the connections of these can be seen as finding the easy paths in the work

space. Configurations that have failed to be connected seem to be a good indicator of where there are hard regions in the workspace. Hard areas can also exist at nodes that have a low failure rate, but these may also be selected for expansion due to the random selection of the node that needs expansion. On the other hand, it might select the wrong nodes for expansion, but this is very unlikely. The road map only uses space linear in the number of configurations, and the number of configurations can be controlled by disallowing the creation of too many configurations in the construction phase. The query phase can be done pretty fast depending on how much time one would use on optimising the path found if such a path exists. However the road map might fail to find a path even though such paths exist.

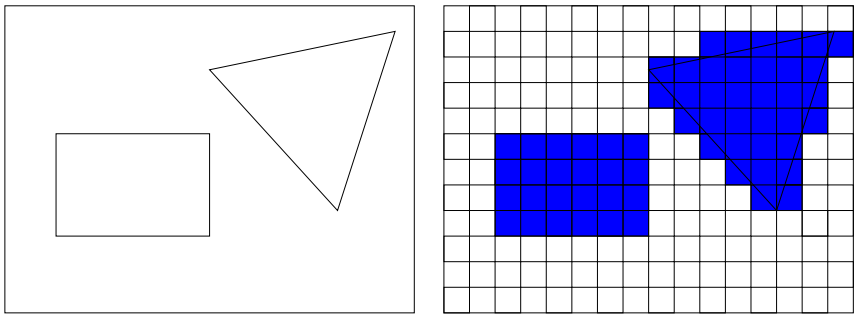
3.2 Cell Decomposition

The main idea in cell decomposition is to decompose the free space into cells and then a search for a path in the cells is done afterwards. The article of Kondo [12] describes an approach to solve motion planning problems with six degrees of freedom using a cell decomposition. A classical example of a cell decomposition technique is to generate a grid. Cells not containing any part of an obstacle is part of the free space and cells that fully contain an obstacle is not a part of free space. How the cells that contain both free space and some part of an obstacle is different from strategy to strategy. Some prefer to look at these calls as not part of the free space while others will investigate these cells more thoroughly. A quad-tree representation of the free space is a strategy where the cells containing both obstacle and free space is divided into four cells to get a higher resolution. However dividing a cell into four cells might produce more cells that need to be divided into four new cells. This continues until a wished resolution is reached, otherwise the algorithm would run forever.

The quad-tree approach is more efficient if you look at the number of cells that is needed to represent the free space. The resolution of the quad-tree is actually a bit greater in Figure 3.2(c) than the resolution of the grid in Figure 3.2(b). The two examples clearly show how memory consuming a grid or quad-tree representation can be. There is only need for seven points to represent the obstacles in the workspace, but there are several more cells representing the free space.

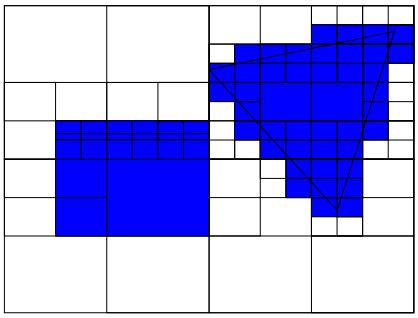
After the generation of the map of cells an A* algorithm can be used find a path from a start position to a goal position. The A* algorithm is a memory heavy path finding algorithm, but is the best option for finding paths in maps of cells compared with other algorithms such as depth-first, breath-first or hill climbing.

The two approaches to cell decomposition investigated in this section, are created for point movement. However, it is possible to extend them to work on harder problems of MP. Instead of feeding the algorithm with the workspace, feed it with the C-Space that represents the corresponding MP problem. When the MP is rotation of polygonal movers or harder, then the C-Space is generated three dimensional. Which is not a problem to make a grid in three-dimensions and instead of using a quad-tree an oct-tree can be used. The search through the grid can still be done by the A* algorithm even though it is a three dimensional problem.



(a) A workspace with two obstacles.

(b) A grid divided space.



(c) A quad-tree divided space.

Figure 3.2: The resulting free space from two types of cell decomposition is generated. Figure 3.2(a) is the space that is being decomposed. Figure 3.2(b) and Figure 3.2(c) is how the space looks after decomposition by grid and quad-tree respectively.

The cell decomposition is not an exact MP algorithm in the form that not all paths in the real world can be found in the represented world. The reason for this is the cells that contains both obstacle and free space, if they are handled as a part of the obstacle a part of the free space will be lost. It is possible to handle these mixed cells separately so the free space in these will not be lost. However if the problem is a MP for a translating and rotating polygon or harder problem it will loose exactness on the C-Space. It is possible to use the two simple cell decomposition approaches to solve a MP problem. There are several other algorithms that are using a form of cell decomposition to solve the MP problem. But the two simple approaches basically explain how the cell decomposition works.

An MP problem can easy be solved by cell decompose the configuration space and the search the free space the for a path. However the problem with this problem is the high memory usage of both the representation of the free space and the search algorithm itself. If the A* search algorithm is used then the path found might not be the shortest, and the cell decomposition is not exact for hard problems.

3.3 Potential-Field

In the potential-field approach, obstacles are assumed to carry electrical charges, and the resulting scalar potential field is used to represent the free space. A repulsive force ensures that no collisions occur between the moving object and the obstacles. This is the definition used by Hwang *et al.* [10], [8].

The MP problem is divided into two stages. The first stage is to find all topological different paths between the start and goal configurations. Then the shortest and most promising candidate path is selected for further investigation. The second stage uses three algorithms that modifies the candidate path to a final path and calculates the orientations of the moving object along the path.

The potential function used to represent the obstacles is one that has its maximum inside the obstacle region and decreases as a function of the distance to the obstacle. When there is multiple obstacles the value of a given point in the space is the largest value of the potential functions of the obstacles represented. Let the potential function in article by Hwang *et al.* [10], [8] be an example. Here they let $g(x) \leq 0, g \in L^m, x \in R^n$ be the set of inequalities describing a convex region of the obstacle where L denotes a set of linear functions and x denotes the location of a point, then the scalar function

$$f(x) = \sum_{i=1}^{no\ of\ bound.\ seg.} g_i(x) + |g_i(x)|$$

is zero inside the region and grows linearly as the distance from the region grows. Figure 3.3 shows the properties of the function $f(x)$.

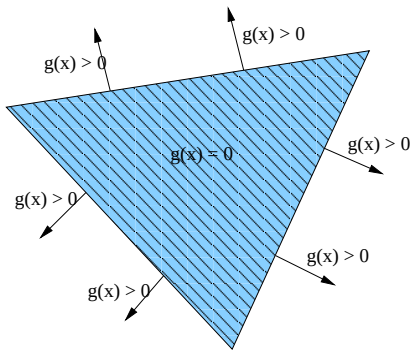


Figure 3.3: The sub function for the potential function. Inside the convex obstacle is the value 0 and outside it grows the further away it is from the obstacle.

Then let the potential function p be defined as $p(x) = [\delta + f(x)]^{-1}$, where δ is a small constant. This function will have a maximum value inside the obstacle and will decrease linearly with the distance to the region outside the region. The Figure 3.4 represents an obstacle potential values, where the obstacle itself have a maximum value and the value decreases with the distance to the obstacle outside it. A Minimum Potential Valley (MPV) is defined as the region between two obstacles where the boundary is defined as the line where they have the same potential value. The partitions of the MPV is similar to the Voronoi Diagram.

Where the Voronoi Diagram uses the Euclidian measure distance instead as a potential function.

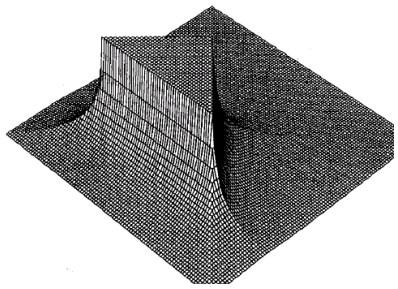
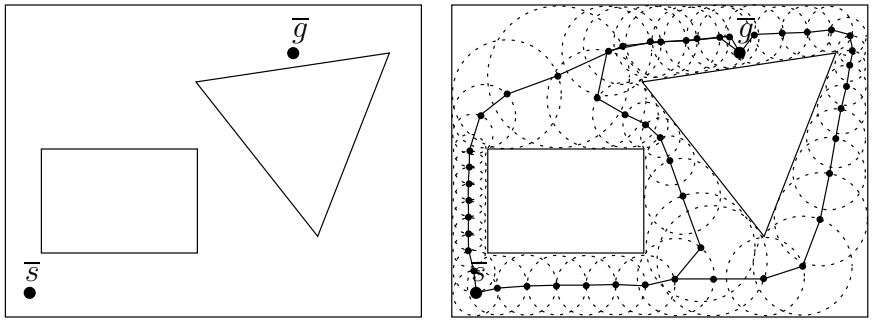


Figure 3.4: The potential field around a triangular obstacle. The figure is taken from the article [10].

The topological structure of the free space is then represented with a graph. The construction of potential paths searches through the MPV to find paths that connects the start and goal position. The search of potential paths starts at the start position and adds this node to the graph. Then a circle of maximum radius possible radius without intersecting any obstacles is generated. The potential along the circumference is calculated at discrete intervals, and the points with locally minimum around the circumference is marked as neighbours of the start node. The neighbours are then again treated as the start configuration $\overline{s}$ to generate neighbours for themselves. The same procedure is used for the goal configuration $\overline{g}$. The graph construction is show in Figure 3.5 where the potential paths from $\overline{s}$ to $\overline{g}$ can be seen. The circles on the picture represents the circles needed to expand the search and the small dots shows the minimum potential around the circles.



(a) The initial start of the potential path. (b) Three potential paths that can be found in this scene

Figure 3.5: Figure 3.5(a) is a scene with a start configuration $\overline{s}$ and a goal configuration $\overline{g}$ and Figure 3.5(b) is three potential paths in the scene going from $\overline{s}$ to $\overline{g}$.

The search for the best path in the MPV starts by selecting the path that is shortest and least likely to cause collisions between the object moved and the obstacles. The chance to collide with an obstacle is estimated with a cost

function that only uses the width and length of the moving object. In the article by Hwang *et al.* [10]. the cost function is defined as

$$C = \int w(x)|dx| \ ,$$

where the weighted factor $w(x)$ is defined as

$$w(x) \begin{cases} maxpotential & \text{for } x \leq \frac{a}{2} \\ z(x) & \text{for } x < \frac{b}{2}, x > \frac{a}{2} \\ 1 & \text{else} \end{cases} \ .$$

Here a is the minimum width and b is the maximum length of the moving object, and the function $z(x)$ is some function that increases in value when x is going towards b. At a and b the value of $z(x)$ is equal to the max potential value and one, respectively. This represents, that it is not possible to pass a region less than half the width of the moving object and if the region is more than half the size of any length of the moving object it can pass the region at any orientation. When the cost of each branch is determined, dynamic programming is used to find the minimum cost path. This path is then used as a potential path.

The potential path is passed through three different algorithms to determine a final collision free path and the orientations along the path for the moving object. The three algorithms are: parallel optimisation algorithm, serial optimisation algorithm and sidetracking algorithm. These algorithms repair a given path, check for collisions and orientate the moving object. This means that they can reject a path if they find that the path is infeasible.

The algorithm takes big steps when possible, in the open regions, and small steps in narrow regions. A potential path is then looked into for further investigation to check for collisions and to find the orientations of the moved object. The algorithm has a weakness when there are many candidate paths and all of them fail, meaning the algorithm might run for a long time to report that no path exists.

3.4 Comparing Approaches

Each approach has been represented in the previous sections but it is not clear which approach, algorithm, or structure that is the most suitable for further investigation for a motion planning problem of a translating, rotating, multi linked polygon. This section will compare the approaches and reason for advantages and disadvantages.

In Section 3.4.1 the exactness of each approach is investigated, where only the V-Graph is exact for simple problems. Another interesting thing is the construction, Section 3.4.2, of each approach, how to query, Section 3.4.3, and the usage of memory, Section 3.4.4. Finally Section 3.4.5 contains the reason why the PRM approach is selected for further investigation.

3.4.1 Exactness

In an exact motion planning approach an existing path between two configurations is always found. For non exact approaches there exist paths that cannot

be found. None of the structures are exact for the motion planning problem of a translating and rotating multi linked polygon.

Table 3.1 indicates whether a structure is exact for each of the MP problems. The problems are abbreviated in the table, the four problems are : P is the point movement, T is the translation of polygons, T+R is the translation and rotation of polygons, and ML is the translation and rotation of multi linked polygons. Most structures are exact for both a point and a translation of polygons with the exception of the cell decomposition. The reason is that it is not possible to make a resolution high enough to contain all information about the obstacles. For motion planning problems harder than translation of polygons none of the structures are exact.

Structure	P	T	T+R	ML
V-Graph	Yes	Yes	No	No
Slide structure	Yes	Yes	No	No
PRM	Yes	Yes	No	No
Cell Decomposition	No	No	No	No
Potential Field	Yes	Yes	No	No

Table 3.1: Table showing whether a structure is exact.

In V-Graphs the edges indicates that a straight line can be drawn between the two vertices. The problem is that the C-Space will contain curves when a polygon is translated and rotated, thus it is not clear how to define visibility along curves. Due to the fact that the slide structure builds on V-Graphs and does not slice up the C-Space before it becomes at least three dimensional. This means the slide structure is precise for the two easiest problems as the V-Graph. The PRM is also exact for the two simplest problems and looses the exactness on the rotations while these has to be approximated to keep straight lines that are easier to test for intersection. For the two simple problems there are no paths that cannot be found in the PRM. If the construction algorithm is given unlimited time. Cell Decomposition is however the worst structure when looking at the exactness. The two proposed algorithms are not exact even for motion planning of a point, while they are making the obstacles discrete and thus loose the exactness. Finally there is the Potential Field which is exact for motion planning of point movement, because the potential value is only maximal at the border and within the obstacle. The translation of a polygon can be a point movement problem instead due to a creation of a C-Space.

Only structure the Cell Decomposition is not exact for any approach. The rest are exact as long the polygon is only translation. However, adding rotations makes all approaches loose exactness due to approximations are used to speed up the search.

3.4.2 Construction

In the construction of the structures there are several parameters that can make it possible to use the workspace otherwise a special C-Space has to be created before the structure is ready.

V-Graphs need a special C-Space, where the obstacles are enlarged, before translation of polygons can be handled. For harder problems there are no solu-

tion for the pure V-Graph, but the slide structure can be used instead, which builds on the V-Graph. This structure also demands a special C-Space for the motion planning problems of polygons or linked polygons. Another problem is that the construction of the V-Graph demands that the polygons are a disjoint set of polygons. This influence how the construction of the C-Space is done. When the obstacles are enlarged several overlaps between obstacles might occur. To make the obstacles a set of disjoint obstacles a union of the obstacles must be created. This can be done with the map overlay algorithm defined in chapter 2 in the book of Berg *et al.* [4] which builds on the original article of Bentley *et al.* [2]. The PRM does not need any specific C-Space to make the construction work. While it checks for intersections by positioning the object moved into the workspace. In Cell Decomposition a C-Space is needed for motion planning for translating and rotating polygons or harder problems. There is no constraints for the input so grids, quad, or oct-trees can be generated from input with curves or joint polygons. The Potential Field approach does not have any constraints on the input, and it is not dependent on a specific C-Space.

There are two approaches that do not need specific C-Space to construct the structures to solve a motion planning problems harder than point movement. The two approaches are the PRM and the Potential Field.

3.4.3 Query

There are different algorithms needed to query the different structures and some of these queries repair the paths found in the structure before outputting them as the result. By repair is meant that the path found might have a shortcut or it can be smoothed. Other algorithms might have illegal paths from the construction that are discarded upon query.

The V-Graph and the slide structure builds up a weighted graph. A query on these approaches is the Dijkstra shortest path algorithm, and there is no need for any form of repair after the search. So all the time used on query is used for shortest path search in the path. But, for the PRM it is quite different because the road map constructed is a graph without cycles so there might be shortcuts in the graph. Additionally the path could contain unnecessary rotations and the path might need smoothing. All of these path optimisations can be done in the query of the PRM or some of the work can be done at the construction. Most of the time will be used for repairing the path rather than searching a graph without cycles. To query a Cell Decomposed structure the A* search algorithm is proposed, but other algorithms such as the breadth first, depth first, or hill climbing can be used. It is possible to do some repairs on the path after the search, but the suggested algorithm does not propose it. I.e. most of the time will be used for the search. The Potential Field approach searches through a set of potential paths, where the paths might not be legal. A candidate path needs to undergo three different algorithms to optimise the length and to orientate the object along the path. The approach is not divided into two phases of construction and query like the others while it generates the potential paths from the start configuration and the goal configuration. This influence the query speed, so a query is very expensive because there is no construction phase to store general information.

The V-Graph and slide structure approach have a very fast query speed, when the path found does not need any repair. Cell Decomposition is also a

fast to query on, when using A* search algorithm, though the path might need repairs. Then there is the PRM which need repairs to optimise the queried path from the structure, though some of the query time can be saved by using more time in the construction phase. Finally there is the Potential Field approach which needs more time in the query phase due to no storage of general information. It seems like the slide structure is best when it comes to the query time, however the PRM only needs to search a non-cyclic graph and then repairs the path. This is good a path from configuration $\overline{s}$ to Configuration $\overline{g}$ is needed.

3.4.4 Memory Usage

It is interesting to make some estimations about the memory usage of the different approaches. If the usage is too big to be in memory it will be cashed onto the hard disk, which is very time consuming. None of the approaches are optimised for I/O so if the structure will be cashed out, all operations on the structure will become very expensive in running time.

The V-Graph and the slide structure are dependent on how many vertices there are in total at the C-Space. The more vertices, the more likely it is to have more visible edges. The worst case memory usage for V-Graphs are $O(n^2)$ given that the graph describes a two dimensional problem where n is the number of vertices. This makes the algorithm dependent on the number of vertices and not the number of obstacles. The memory usage of the PRM can be controlled by the number of configurations which are added to the graph. But the more configurations, the better the RM is to find paths in. I.e. in easy scenes there is no need for a great number of configurations, thus no need for a great amount of memory. In hard scenes there is a need for many configurations and a great deal of memory. However, if the amount of configurations is kept static the PRM will just report failure on the hard scenes if the space is not investigated enough. There are two problems with cell decomposition. At first the cell takes up much space. It consumes $O(n \cdot m)$ amount of space to the resolution in two dimensions, where n and m are the number of cells at the x-axis and y-axis, respectively. For every dimension another factor is multiplied into the memory usage. The quad- and oct-tree will often save memory but in worst case it is the same. Then there is the A*-search algorithm, which can be memory heavy as well. Added together thus makes the cell decomposition approach memory heavy. The potential field method has a weakness towards narrow corridors, where the circles used to generate the path are very small. Imagine a workspace full with narrow corridors. This leads to many small circles, which results in many nodes in the graph for potential paths.

The worst approach seems to be the cell decomposition approach, though it can be controlled how much memory it should use by controlling the resolution of the grid or depth of the quad-tree. The V-Graph and the slide structure can be very memory consuming but it depends on the given scene. A high number of vertices can give a high number of edges, which in worst case will consume quadratic space in the amount of vertices. The potential field approach is also dependent on the input scene, if the input scene contains many tight regions it will consume much space. Finally there is the PRM approach where the memory usage is controllable. Hence it can be kept within memory all the time.

3.4.5 Choice of Approach

The PRM approach will be looked into in thesis because the structure has some very interesting features. The construction and the query phase can be divided into two independent features. When the construction phase is done the RM can be stored on hard disk, and when a query is needed the RM can be loaded into memory again. However, this approach has some challenges in the sense that the algorithm is probabilistic and that it needs many collision detections to connect two configurations. In this thesis an attempt to lower the amount of collision detections by approximating the movement instead of making several steps. The comparison between the original Local Planner (LP) approach and the new approach will happen through an experimental comparison. The experiment will count the number of collision tests required for a wide range of problems to determine the success criteria. I.e. a collision detection only has to happen on the approximation and not for each step when attempting connection between two configurations.

Chapter 4

Probabilistic Road Map Extension

There are several improvements to the original PRM approach given in the article by Kavraki *et al.* [11].

There are some articles describing how to improve the sampling of configurations in the initial phase of the PRM. I.e they are looking into other approaches for the initial step, where the standard is random generated configurations. An approach could be to lay a grid of configurations as in the article of Sanchez [17] or the article of Geraerts *et al.* [6]. The article of Geraerts *et al.* also describes the usage of a halting, a cell-based, a Gaussian and two obstacle based approaches to sample the initial configurations.

However, in the article of Geraerts *et al.* [6] most of the running time for the original construction algorithm by Kavraki *et al.* [11] is used on collision detections. They are writing about how different collision detection algorithms can lower the amount of time used on collision detection, and that a binary search when connecting two configurations is faster than the incremental approach. The article of Sanchez *et al.* [18] considers how to test the connection of two configurations for intersections by making a binary search instead of the standard linear approach in Kavraki *et al.* original PRM.

At first in Section 4.1 the movement of the object is looked into, and a suggestion to approximate the objects movement to save collision detections is proposed. This change of strategy influences the LP so in Section 4.2 a new LP is suggested that uses the approximations and a binary search strategy. The binary search strategy is used because the approximation might be too large for certain movements of the object. The movement is divided into two sub-movements which generate a more precise approximation. This can continue until a desired depth is reached.

4.1 Object Movement

Between two configurations $C_i \rightarrow C_{i+1}$ the problem is to calculate how the object moves between these two configurations. The object can both translate and rotate at the same time. Hence the movement is some curve. The normal approach, e.g used in the original article of Kavraki *et al.*[11], is to use inter-

polation and then for each step test for intersection. This approach needs an enlargement of the object so the path will stay legal. This approach is looked into in Section 4.1.1. However, using an approximation of the area covered by the movement could be used to describe the movement of the object as well. An intersection test of this area will be legal as intersection test for all steps in the interpolation approach. This approach is described in Section 4.1.2.

4.1.1 Interpolation

The original article of by Kavraki *et al.* [11] describes the interpolation approach to connect two given configurations. However, to make sure that the approach does not accept any illegal paths the object is enlarged by some ε where $\varepsilon > 0$ before interpolating. The enlargement has two parameters: a maximum angle a and a maximum translation step t. Let $r(\alpha)$ define the function that generates a rotation matrix for the angle α, let P_0^2 define the original polygon at origo. Then by rotating the original polygon backwards the polygon P_1^2 can be found

$$P_1^2 = r(-\alpha) \cdot P_0^2 \ ,$$

and the P_2^2 is found by a forward rotation

$$P_2^2 = r(\alpha) \cdot P_0^2 \ .$$

Let $f(P_{from}^2, P_{to}^2)$ define a function that generates an approximation polygon over the rotation between the two polygons. The rotation approximation polygon P_x^2 is found by

$$P_x^2 = f(P_1^2, P_2^2) \ .$$

Let polygon P_t^2 be the polygon containing at least the area that the step size can cover. Then the ε-enlarged polygon can be found by a Minkowski Sum, as following

$$P_\varepsilon^2 = P_t^2 \oplus P_x^2 \ .$$

For a linked polygon each of the polygons is ε-enlarged. Figure 4.1 shows a triangle and the enlarged object after the enlargement. In the example a maximum angle rotation of 0.1 radians and a step size of 10 are used. The figure shows how this enlargement makes the polygon much larger and hence it will be able to cover a movement of polygon with a maximum rotation of 0.1 and maximum step size of 10.

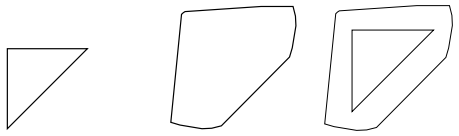


Figure 4.1: The ε-enlargement of a triangle. Where the original object is in Figure 4.1(a) becomes the ε-object in Figure 4.1(b) after the ε-enlargement. Figure 4.1(c) is the object within its enlargement.

The ε-object is then moved from configuration c_i towards configuration c_j in small steps with a step size and maximum rotation smaller than the enlargement. In Figure 4.2 there is an example of an interpolation of the object from

Figure 4.1 that goes from configuration c_i to configuration c_j. The figure clearly shows that the interpolation approach from the original article by Kavraki *et al.* [11] covers a larger area than the real movement. The real movement will be contained in the area covered by the interpolation approach and hence accepting the connection of two configurations as a legal path. However, a vast amount of steps is required to get a resolution close enough, to the real movement.

(a) A triangle moving from a configuration to another configuration

(b) Interpolation movement with an ε-enlarged triangle

Figure 4.2: The movement of a triangle and the interpolated movement of an ε-enlarged triangle.

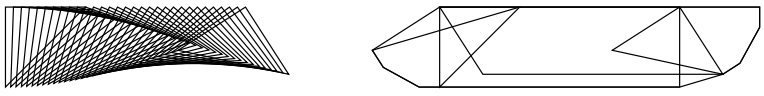
Dividing a movement of an object between two configurations into an interpolation of an ε-enlarged object is a simple approach to connect two configurations in the RM. However, the approach has to make steps and for each step an intersection test against the C-Space or work space has to be done.

4.1.2 Approximation

This section describes how different movements can be estimated. It begins with the point movement, then line movement, polygon movement and finally linked polygon movement. Each movement except the linked polygon movement is divided into translation, rotation, and the combination of translation and rotation. Approximating the objects movement can save intersection tests when attempting to connect two configurations. The approximation needed is assumed to be a conservative over-approximation. Let P^2 be the polygon representing the approximated area and let S be a set containing all configurations in the movement from configuration C_i to configuration C_j then

$$S \subset P^2$$

will describe the conservative over-approximation property of the approximation. Figure 4.3 shows how a given triangle moves from a configuration C_i to a configuration C_j and the polygon describing the approximation of the movement area.



(a) The object moving from a configuration to another configuration

(b) The approximation of the area the movement covers

Figure 4.3: The movement of a triangle and how the area of the movement is approximated.

Point Movement

There are three different methods for moving a point in two dimensional space. The first is a simple translation, the second is a rotation, and the third is a translation and rotation at the same time.

If the movement from C to C' is a translation, then the movement is a line between p to p'. It is trivial that there is no need to approximate anything. This case is shown in Figure 4.4(a).

A rotating movement from C to C' is quite different. This implies that the movement from p to q is an arc with center point p_c, where the lines (p_c, p) and (p_c, p') has the angle α between them. The idea of the approximation is to make a triangle that contains the arc. This is done by first finding the two tangent lines at point p and at point p'. Next the point p_r where these two tangent lines cross is found. Then the three points p, p', and p_r describe the triangle which over-approximates the movement from p to p'. This is shown in Figure 4.4(b). The argument of why this is true is as follow. The two tangent lines will always be above the arc and the line between p and p' will always be below the arc. Hence the triangle's area will cover the arc and the over-approximation will be correct. One assumption needed to make this work is that the angle must be less than π otherwise the vectors will intersect at the wrong side and if the angle is exactly exactly π they will never intersect. However, it is convenient to keep the angle as low as possible to keep the triangle as small as possible. Small triangles make the approximation closer to the real movement.

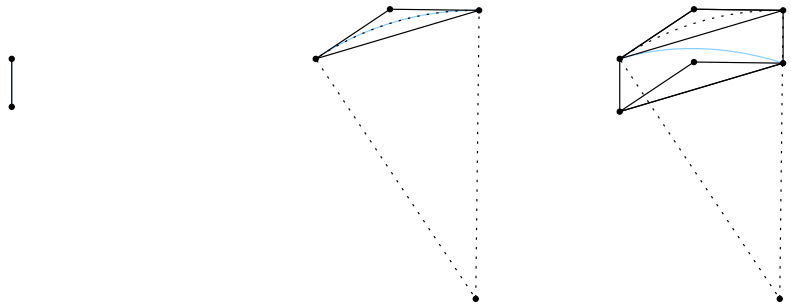
Finally there is a movement that combines a translation and a rotation. This case is harder than the two above, but can be solved by using the rotation case above. Approximate the rotation at C with the angles α and $-\alpha$. Then find the convex hull of the points in the two triangles, and the movement has been approximated. Figure 4.4(c) illustrates the approximation. At configuration C and C' the two triangles' area are large enough to contain a rotation each. Now only the translation needs to be taken care of. The point translation is created be the straight lines between two points. Lines between the six points and a boundary fix would work but a bounding box will contain the same area or more. Hence the area surely can contain the movement.

Line Movement

For approximation of a line movement, bounded between the points p and q there are three cases: Translation, rotation and the combination of translation and rotation.

Translation movement of a straight line is trivial as it was for point movement. Keep the two lines at c, (p, q), and at c', (p', q'), then connect the four points p, q, p' and q' into two lines (p, p') and (q, q'). These four lines describe the area affected by a line movement if it is a translation. Hence there is no need to approximate anything, which can be seen in Figure 4.5(a).

Rotation is harder but is solved in a similar way as a point movement consists of both a translation and a rotation. Approximate the rotation for each of the two points p and q. This gives six points p, q, p', q', p_r and q_r. The determine the convex hull for these points. This yields a simple polygon whose area will contain the rotation of a line from C to C'. Consider the following properties when rotating a line less than π around a point p_c.



(a) Point translation movement approximation.

(b) Point rotation movement approximation.

(c) Point translation and rotation movement approximation.

Figure 4.4: The approximation of a point movement: Figure 4.4(a) is the approximation of a translation of a point, Figure 4.4(b) is the approximation of a rotation of a point and Figure 4.4(c) is the approximation of both a translation and rotation of a point.

- Each endpoint p and q moves like an arc.
- If the endpoints are on each side of the rotation center p_c there is a point on the line that moves like an arc.

Each of these properties makes it impossible to draw lines between the points p, q and p', q', respectively. By approximating p and q as rotating points the movement of these points will be approximated correctly as argued earlier. Now the problem is to determine the rest of the line is within the approximation. The convex hull contains the line movement because it will always take end points from the rotation. If it should not contain the movement there should be a point on the line that should fall outside this area. However, this is not possible.

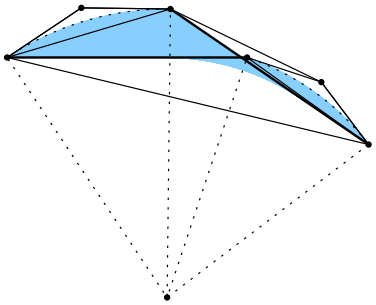
An approximation of a line that translates and rotates at the same time is solved similar to the rotation and translation of a point described earlier, except for a few changes. Instead of approximating rotation with angle α at one line it is done for two lines with angle α at C and angle $-\alpha$ at C'. This will give 12 points p, p_{r1}, p_{r2}, p', p'_{r1}, p'_{r2}, q, q_{r1}, q_{r2}, q', q'_{r1} and q'_{r2}. Finally the Convex Hull of the 12 points will the result in a simple polygon which area covers the movement of the line. Again consider the following properties when rotating and translating a line.

- Each endpoint p and q moves in a curve.
- If the rotation is large and the translation small, and the endpoints are on each side of the rotation center p_c there is a point on the line that moves like a curve.

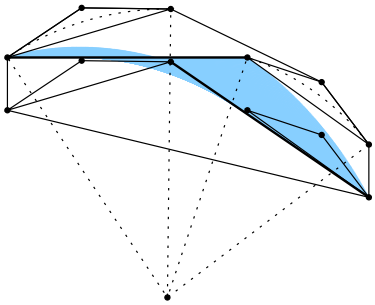
Again each of these properties makes it impossible to draw a straight line between the points like the translation of the line. This time each of the endpoints is approximated as a point that rotates and translates as described earlier, which validates the endpoints movement. The special point of the line only

occurs when the movement is close to being a rotation, hence small translation and "large" rotation. The reason why this movement is within the area of the Convex Hull is same as with line rotation.

(a) Line translation movement approximation.



(b) Line rotation movement approximation.



(c) Line translation and rotation movement approximation.

Figure 4.5: A line going from point p to point q movement approximation : Figure 4.5(a) is the translation movement approximation, Figure 4.5(b) is the rotation movement approximation and Figure 4.5(c) is both translation and rotation movement approximation.

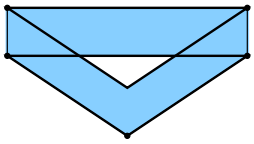
Simple Polygon Movement

There is a simple solution to simple polygon movement this problem that uses line approximation.

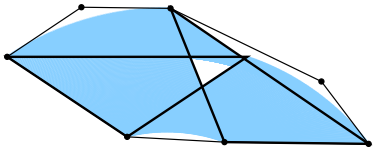
There is a general solution to all three cases of how to approximate the movement of a simple polygon.

The appropriate line approximation is done for each line of the polygon. The approximation of line movements is described earlier.

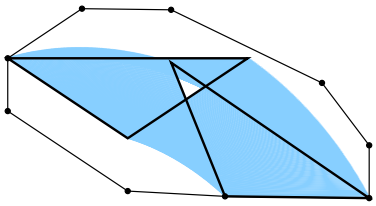
Each approximated polygon from each line approximation is added to a set. This set will in the end contain approximation polygons for all lines and they combined will cover the movement of the polygon.



(a) Polygon translation movement approximation.



(b) Polygon rotation movement approximation.



(c) Polygon translation and rotation movement approximation.

Figure 4.6: The approximation of the different movement types for a simple polygon that is triangle consisting of the points p, q and r. Figure 4.6(a) is the translation movement approximation, Figure 4.6(b) is the rotation movement approximation and Figure 4.6(c) is the approximation of the movement when there is both a translation and a rotation.

Linked Polygon Movement

At first the approximation of the movement of a linked polygon seems like a much harder problem to solve. However this is not the case.

This approximation builds on the simple polygon movement from earlier in this section. Basically this approximation change the translation and the rotation properly for each polygon in the linked polygon. Then each of the polygons in the linked polygon is approximated like a simple polygon movement with the translation and the rotation for the simple polygon. Finally the union of all approximation polygons is the approximation of the linked polygons' movement. The only thing that is not clear, is what happens when the translation and the rotation is changed for each polygon of the linked polygon.

There are different layers of polygons, first there is the root polygon at layer 0. All the polygons connected to layer 0 is in layer 1. Hence, all layers at layer i is connected to layer $i-1$. The translation and rotation of the root or main polygon is first processed. Then each layer is processed iteratively.

The root polygon is the basis of the configuration. Hence it uses the position

and the orientation as the translation vector and a rotation matrix respectively.

The polygons of the first layer are connected to the root polygon through some connection points. Hence the positions of these polygons depend on the root polygon. The root polygon is the parent of the polygons of the first layer, and they are the children of the root polygon.

Let the orientation of a polygon of layer one be static, then the three cases of movement of the object will affect the polygon of the layer one. Any of the three types of movement of the object will be a the same movement of the root and the polygon at layer one. The polygon at layer one with no orientation change can be seen as a part of the root polygon. Hence it will move with the same parameters as the root polygon.

In the second case let the orientation of a polygon of layer one be non-static, hence the orientation changes from configuration c_i to configuration c_j. In the first case the movement of the polygon of layer one dependents on the root polygon and is solved by applying the translation and rotation of the root polygon to the polygon at layer one. Now the polygon of layer one rotate around the connection point to the root. The movement of the polygon at layer one is not a curve from the translation and rotation of the root polygon. Instead it is a curve that contains the change in the orientation between the two polygons, the translation and rotation of the root polygon. Approximating rotation and translation of the connection point with the same parameters of the root polygon gives a convex polygon of that movement. Then the polygon of layer one is approximated as a polygon, described earlier in this section, with a rotation of $\alpha_0 + \alpha_1$ around the connection point and no translation. But at the point approximation of the polygon of layer one is the approximation of a point Minkowski summed with the approximation of the connection point.

At layer i a polygon is connected to its parent polygon in layer $n-1$ through some connection point, which again can be connected to some other polygon. Let $\widehat{P^2}_i$ be a polygon at layer i and let $\widehat{P^2}_0$ be the root polygon. Then the path from the root polygon to the polygon at layer i can be described as

$$Path_i = \{\widehat{P^2}_0, \ldots, \widehat{P^2}_i\} \ .$$

This path is used for both calculating the translation and orientation of polygon i. Let $f()$ define the function that returns the vector describing the movement of the connection point of the simple polygon. Then the translation vector $\overrightarrow{v}_i$ of polygon i can be described as:

$$\overrightarrow{v}_i = \sum_{j=0}^{i} f(\widehat{P^2}_j)$$

Let $g(\widehat{P^2})$ define the function that returns the orientation rotation of the simple polygon in form of an angle. Then the rotation angle α_i of polygon i can be described as:

$$\alpha_i = \sum_{j=0}^{i} f(\widehat{P^2}_j)$$

Let $h()$ define the function that approximates the point movement, let p_i be the connection point of polygon i, let $\overrightarrow{v}_{i-1}$, and α_{i-1} be the vector and angle

found for polygon $i-1$ and let $\widehat{P^2}'$ define the approximation of the connection point movement. Then the movement approximation for the connection point of polygon i is

$$\widehat{P^2}'_i = h(p_i, \overrightarrow{v}_{i-1}, \alpha_{i-1}) \oplus \widehat{P^2}'_{i-1} \ .$$

Let q be a point in polygon i. Then the approximated polygon for the point can be found by using the point approximation from before and the connection point approximation by:

$$\widehat{P^2}'' = h(q, \overrightarrow{v}_0, \alpha_i) \oplus \widehat{P^2}'_i$$

Let m be the points in the approximated polygon, and let f' be a function that takes two simple polygons and returns the convex hull. Then the set of simple polygons S will be the approximation of the simple polygon i and is found by:

$$S = f'(\widehat{P^2}''_1, \widehat{P^2}''_m) \cup \bigcup_{k=1}^{m-1} f'(\widehat{P^2}''_k, \widehat{P^2}''_{k+1}) \ .$$

The total approximation can be expressed as following:

$$S_i = \begin{cases} h(q, \overrightarrow{v}_i, \alpha_i) & \text{if } i = 1 \\ h(q, \alpha_i) \oplus \widehat{P^2}'_i & \text{otherwise} \end{cases} \ .$$

The root polygon is handled as a special case because it needs to rotate, translate, and there is no enlargement from a connection point. Let the set A be the set of approximated polygons. Then:

$$A = \bigcup_{i=1}^{m} S_i \ ,$$

where the approximation of a linked polygon is a set of polygons from each polygon approximations edge approximation. Figure 4.7 is an example of the over-approximation of linked polygon.

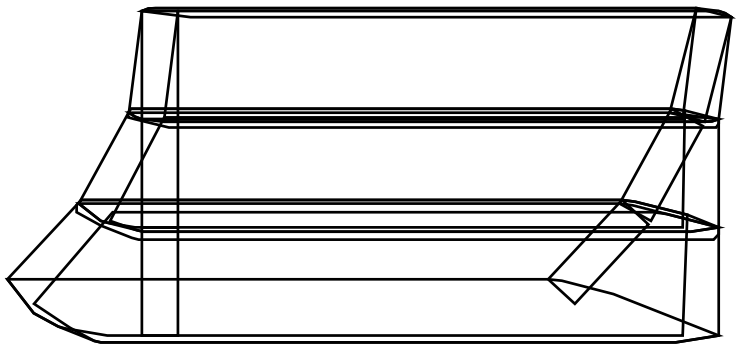


Figure 4.7: The approximation of the movement of three linked polygons, the figure comes from the implemented local planner.

Approximate Movement Algorithm

Using the observations from previous an algorithm that over-approximates movements can be put together. It takes a linked polygon $\widetilde{P^2}$, the center point of the rotation p_c, a translation vector T and a rotation matrix R as parameters and returns a set of approximation polygons that covers the moved area. The linked polygon $\widetilde{P^2}$ is the object whose movement needs to be approximated and it consist of simple polygons $\widehat{P^2}$. The movement of the linked polygon $\widetilde{P^2}$ consist of a translation vector T and a rotation matrix R that rotates the object around center point p_c. The output will be a set of simple polygons with a size equal to the number of simple polygons in the linked polygon. The algorithm will consist of several layers to exploit that linked polygons consists of simple polygons, polygons consists of lines, lines consists of points. The top layer is the linked polygon layer, the next layer is the polygon layer, then there is a line layer and finally the point layer.

Several subroutines are used in the different routines defining the approximations. These subroutines are:

- `parent-index(`$\widehat{P^2}$`)` returns the index of the parent polygon to the given polygon.
- `connection-number(`$\widehat{P^2}$`)` returns the index of the connection point in the parent polygon to the given polygon.
- `connections(`$\widehat{P^2}$`)` returns the list of connection points for a given polygon.

The linked polygon layer calls both the polygon layer and the point layer because it divides the approximation problem into sub-problems. It approximates the movement of each polygon in the simple polygon, which is known as a component of the object moved. When approximating a polygons movement it is dependent on the approximation of the connection point. The double array $ac_{i,j}$ contains approximations, where index one is the component index and the second index is the connection point index at the given component index. For the root polygon there is none. However, it is different for a polygon P_i^2 at layer i, which has a parent polygon P_j^2. It uses the point approximation of the connection point. This point approximation is done before this approximation when approximating the parent polygon p_{j-1}. Meaning that the algorithm depends on that the parent polygons are before the children polygon. The algorithm is outlined in Pseudocode 4.1.

The polygon layer calls the line layer with each edge in the polygon. The approximated movements for each line are seen as simple polygons. Each polygon found at the line approximation is added to a set of simple polygons. The set will in the end represent approximations for each edge in the polygon. The union of the area, which is covered by all the edge approximations covers will be the resulting polygon approximation. For an outline of the algorithm see Pseudocode 4.2.

The next layer is the line layer which calls the point layer with each point in the edge to approximate the movement for them. A set collects the points for each approximation and a bounding box algorithm is executed on the set to find a simple polygon that approximates the movement of the line. The algorithm can be seen in Pseudocode 4.3.

```
function approxLP(P̃² : Linked Polygon,
                  p_c : Center Point,
                  T : Translation Vector,
                  A : Angles)
  S ← ∅ : Set of approximations
  i ← 0
  ∀P̂² ∈ P̃² : in increasing order
    j ← 0
    k ← parent-index(SP)
    a_i ← a_k + A_i
    l ← connection-number(k, SP)
    S ← S ∪ approxSP(P̂², p_c, T, R, ac_{i,j})
    C ← connections(SP)
    ∀p in C : in increasing order
      ac_{i,j} ← approxP(p, ac_{k,l})
      i++
    j++
  return S
```

Pseudocode 4.1: How the linked polygon layer works in the approximate movement algorithm.

```
function approxSP(P̂² : Simple Polygon,
                  p_c : Center Point,
                  T : Translation Vector,
                  R : Rotation Matrix,
                  AP : Approximated Polygon)
  S ← ∅ : Set of simple polygons
  S ← S ∪ P̂²
  ∀e ∈ P̂² :
    S ← S ∪ approxL(e, p_c, T, R)
  return S
```

Pseudocode 4.2: How the polygon layer works in the approximate movement algorithm.

Last layer is the point layer, here each point will be approximated to six point in the approximation. The first three points are a rotation of point p around point p_c with rotation matrix R. Then these three are translated by vector T and finally if there is an approximated polygon it will be Minkowksi summed with the points. This approximated the movement of the point. See pseudocode 4.4 for more details about the algorithm.

These four functions can approximate the movement of a linked simple polygon and the running time for the approximation will be clarified. Let e be the number of edges in the polygon and p be the number of points in the polygon.

```
function approxL(e : Edge,
                 p_c : Center Point,
                 T : Translation Vector,
                 R : Rotation Matrix,
                 A : Approximated Polygon)
  S ← ∅ : Set of Points
  ∀p ∈ e :
    S ← S ∪ approxP(p, PointA_c, T, R, A)
  S ← convexHull(S)
  return S
```

Pseudocode 4.3: How the line layer works in the approximate movement algorithm.

```
function approxP(p : Point,
                 p_c : Center Point,
                 T : Translation Vector,
                 R : Rotation Matrix,
                 A : Approximated Polygon)
  S ← ∅ : Set of Points
  p_r ← R * (p − p_c) + p_c
  p_x ← point-of-intersection(tangent(p),tangent(p_r))
  PointA_t ← T + p
  PointA_{t+r} ← T + p_r
  PointA_{t+x} ← T + p_x
  S ← {p, p_r, p_x, p_t, p_{t+r}, p_{t+x}}
  S ← minkowski-sum(S, A)
  return S
```

Pseudocode 4.4: How the line point works in the approximate movement algorithm.

The number of edges is equal to the number of points in the polygon. Hence $e = p$. Because the number of points and edges are the same this number will be referred as n for the rest of this analysis.

Let n be the maximum number of points in the components, let m be the maximum number of connection points in the components, and let k be the number of components. The linked simple polygon approximation function has two loops. The first loop is for each component of the linked polygon and within this two things are done. The approximation of the simple polygon of the component and a loop going through the connection points of the component. In this loop the connection point is approximated. This give the following running time

$$O\left(k \cdot (approxSP + m \cdot approxP)\right) .$$

The functions running time is dependent on two functions, one that approximates the simple polygon and one that approximates a point. The function that

approximate the movement of a simple polygon has a loop where for each edge in the simple polygon the approximation of the movement of the edge is done. This gives the running time

$$O\left(n \cdot approxL\right) \ .$$

There is only a dependency on the edge movement approximation function. The approximation of an edge is dependent on the two point approximations and the point approximation is a constant time operation. This makes the edge approximation a constant time approximation as well which leads to a total worst case running time of

$$O\left(k \cdot (n+m)\right) \ ,$$

for approximation of linked simple polygons that translates and rotates.

Handeling Large Angles

The approximation can be extended to handle large angles, which is done by dividing the rotation approximation into two separate approximations. The smaller the angle of the rotation is, the better the approximation of the rotation is. The reason is that the third point in rotation approximation is closer to the rotation arc. The dividing of the rotation can continue recursively until a desired angle is reached.

Let α describe the angle which is rotated, let p be the point rotated, let p' be the point rotated to, and let q be the center of rotation. Finally let $f()$ be the function the finds the third point of a rotation approximation. Then the approximation point p_x of the rotation will be

$$p_x = f(p, p', q) \ .$$

Dividing the approximation into two sub-approximations is a new angle and point need to be defined. Let α' be $\frac{\alpha}{2}$ and let p'' be p rotated by α' around point q. Then the two approximation points p_{x1} and p_{x2} are found by:

$$p_{x1} = f(p, p'', q), \ p_{x2} = f(p'', p', q) \ .$$

The division can be done on each of the sub-rotations as well to achieve a better resolution.

4.2 Local Planner

There are two different strategies for the LP that is interesting to look into. The first is the linear interpolation strategy from the original article by Kavraki *et al* [11] and the second is the interpolation binary strategy in the article by of Geraerts *et al.*[6]. In this section there are two subsections: Section 4.2.1 describes the original interpolation LP and Section 4.2.2 describes a LP which is used when approximating the movement.

4.2.1 Interpolation Local Planner

The original interpolation LP uses linear placements of an ε-enlarged object at each step. How this enlargement and placement works is described in Section 4.1.1. The connection strategy of the linear placements is looked into because the approximation approach in Section 4.2.2 needs a different connection strategy.

Connecting Configurations

In a path from configuration c_i to configuration c_j there are k steps. Let a list L contain all configurations on the path from c_i to c_j and in that order. I.e. $L = [c_1, c_2, \ldots, c_k]$. Let $f()$ define a function that tests whether a configuration intersects any obstacles in the environment. Then

$$\forall c \in L : f(c)$$

where the output from the function means

$$f(c) \begin{cases} \textit{not connectable} & \text{if } \textit{true} \\ \textit{maybe connectable} & \text{if } \textit{false} \end{cases} .$$

If all tests return true the two configurations are connectable, but if one of the tests fail they are not connectable. In most failed connection attempts will the obstacle not be near the configuration c_i or the configuration c_j but is will be somewhere in the middle. This is why a binary approach is suggested in the paper by Sanchez *et al.* [18] and the paper by Geraerts *et al.* [6]. However, this approach is not used in this thesis. Because the interpolating approach with a ε-enlarged object is the standard approach and it gives result that are fine to benchmark the new suggested approach.

Conclusion

The LP that uses the interpolation strategy is very simple and the ε enlargement is only required in the initial phase. However, this restricts the possible movement steps, they have a maximum angle and a maximum translation between each step. The approach does not take big steps if possible. I.e. that no matter the distance between two configurations the resolution of the interpolation is the same. This makes the speed of connecting configurations dependent on the distance between them.

4.2.2 Approximation Local Planner

When the LP builds on the approximation approach to connect two configurations. There is need to change the connection strategy. Hence a binary connection strategy is developed. The rest of the LP strategy follows the interpolation LP from the article by Kavraki *et al* [11].

Connecting Configurations

To find a linear path from a configuration c_i to a configuration c_j is different from the linear interpolation approach. The approximation approach is more like the binary approach in Sanchez *et al.*[18] and Geraerts *et al.*[6].

At first an approximation P^2_{approx} of the movement from c_i to c_j is generated. Then the P^2_{approx} is tested whether it intersects any obstacles in the environment. If there are no intersections it is possible to connect the to configurations. If it fails it might be that the approximation needs a better resolution to succeed. To obtain a better resolution the path is divided in the middle at configuration c_k. Then the two paths from c_i to c_k and from c_k to c_j is approximated and tested as the whole path. If both paths are collision free the whole path is collision free,

however if one path is not collision free this path is recursively examined, until it is collision free or a maximum recursion depth is reached. If both paths has a collision both paths needs to be examined, but a depth first search is used here, so if the examination of the first path fails to succeed then it is not possible to connect the two configurations. Pseudocode 4.5 shows a connect function for the approximation LP. It takes three parameters: two configurations, the current and destination, and a search depth. It returns whether the two configurations are connectable within the given search depth.

```
function connect(c_i : from configuration,
                 c_j : to configuration
                 l : search depth)
  if l < 0
    return false

  P^2_approx ← approx(c_i, c_j)
  if collide(P^2_approx)
    c_k ← middle(c_i, c_j)
    if connect(c_i, c_k, l − 1)
      return connect(c_k, c_j, l − 1)
    else
      return false
  return true
```

Pseudocode 4.5: Pseudocode of how the approximation local planner attempts to connect two configurations together.

This approach to connect two configurations does not have any constraints on the given configurations. I.e there is no maximum angle or maximum step size. The approximation has a constraint saying that it cannot take a angle larger than $\frac{\pi}{2}$ but the approximation can be improved to take larger angles as shown in the earlier Section 4.1.2. By keeping a maximum search depth independent of the length of the path between c_i and c_j the small paths will have a better resolution than the long paths.

Conclusion

In general the PRM approach needs to take large imprecise steps when possible. However in narrow regions there is a need to make the resolution as close to the real movement as possible. The approximation LP attempts to do exactly this by going into the same search depth independent of the length of the path.

Chapter 5

Implementation

The motion planning program has been implemented in C++ and uses a few libraries such as the Standard Template Library (STL) and the OpenGL Utility Toolkit (GLUT). The STL is used for sorting and data structures such as vectors, lists, and maps. The GLUT library is used to visualize a generated RM. The program is compilable on different Operating Systems (OSs) with different versions of GNU Compiler Collection (GCC) compilers which is described in Section 5.1. Several programs were created in this thesis and these are shortly explained in Section 5.2. Representations of different structures such as points, vectors, polygons, etc. are describe in Section 5.3. Section 5.4 follows with details about how configurations are generated in the program. Section 5.5 and Section 5.6 describe the implementation details on the Convex Hull and Minkowski Sum algorithms. Section 5.7 follows with details about how the approximation is implemented. Section 5.8 follows with details about the different local planner implementations. Finally Section 5.9 shortly describes how the structure and the different phases of the PRM is implemented. The code to the program can be found on the attached CD or at http://www.daimi.au.dk/ ds/?page=6.

5.1 Operating System and Compiler

The program has been compiled on three different OSs with each their version of the GCC compiler. On all OSs the program works as expected according to the performed tests. The program is compiled with a GCC compiler, and it has been compiled on the following systems : A GCC compiler version 4.0.1 on Mandriva Linux 2006 64bit, a GCC compiler version 3.3.2 on a Fedora Core 3 Linux and finally on a GCC compiler version 4.1.1 on a fedore Core 5 Linux. These can be seen in Table 5.1.

5.2 Programs

There are several programs constructed to test, implement, experiment, visualising the PRM algorithm, the work spaces, the output of the PRM and counting collisions tests. In Section 5.2.1 three different programs are described, which

OS	GCC version
Fedora Core 3	3.3.2
Fedora Core 5	4.1.1
Mandriva 2006 64 bit	4.0.1

Table 5.1: The different operating systems and compilers the implementation has been debugged on.

are closely related to the PRM algorithm. In Section 5.2.2 a secondary program for generating work spaces is described.

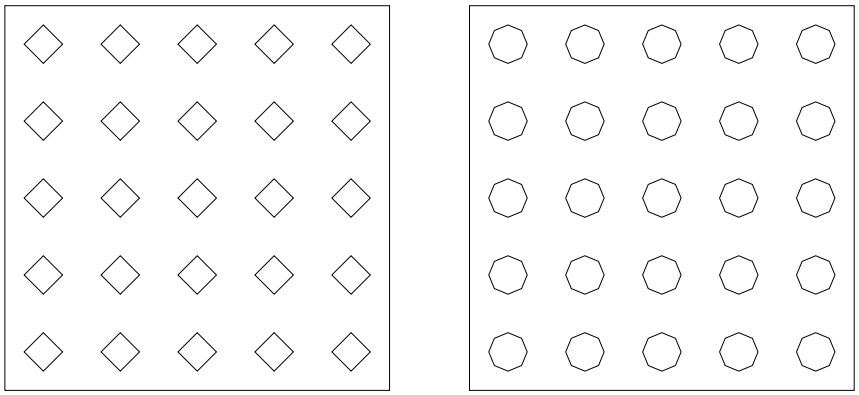
5.2.1 Main Programs

There are three different main programs: `motionplanning-testrun`, `motionplanning-prompt` and `motionplanning-opengl`. The `motionplanning-testrun` is a program that tests the different algorithms and structures needed in the PRM algorithm. How these test works can be seen in Chapter 6. The `motionplanning-prompt` is the program which can generate the PRM but has other uses as well. This program can generate output in the fig format as well. The fig format is specified in the Xfig manual [1]. Different scripts are constructed to generated different outputs of the different features of the algorithm. The data generated from the scripts can then be viewed in Xfig, which is a vector graphic editing tool in linux.Another feature of this program is that it can count the number of collision tests made during the construction of a PRM and output this value. This is used in the count of collision tests described in Chapter 7. Finally the `motionplanning-opengl` program can visualize a PRM through OpenGL. To visualise through OpenGL the construction of the PRM is divided into two separate programs. The test of the program is a separate program because it only needs to be executed when something has been changed in the program. Another thing is that one may want to save the time it takes to compile the test cases.

More details on the different parameters are described in Appendix A.

5.2.2 Sub Program

A small program was created to generate the circle work spaces which are used in the comparison of collision test counts. It places an obstacle at some interval and generates the circle-like obstacle with a given amount of points. For an example take a look at Figure 5.1 where Figure 5.1(a) is a generated work space with a resolution of four. I.e. the “circles” consist of four points. Figure 5.1(b) uses a resolution of eight. Hence the “circles” consist of eight points. This doubling ends at a resolution of 128 where each “circle” has 128 points. This feature is used to show the how different LP approaches perform collision test counts in different work spaces with increasing edges. The results from this experiment can be seen in Appendix F and the experiments are described in Chapter 7. There are more figures of the different resolutions in Appendix C.



(a) The circles workspace with a resolution of four.

(b) The circles workspace with a resolution of eight.

Figure 5.1: Two circles work spaces with a resolution of four and eight.

5.3 Representations in the Program

There are several structures implemented in the program, they are divided into groups of geometric structures, collections, and PRM specific structures. The geometric structures are angles, points, edges, vectors, and simple polygons. The collections are arrays, sets, lists, vectors, maps, and graphs. The PRM specific structures are the Road Map, the Object, the Configuration, and the Work Space.

Efloat

To be able to control the precision of the floating point type used in the program the type efloat is used. Efloat is defined in the file `Typedefs.hpp`, currently as a double precision floating point. Originally a struct was representing the floating point, which could be expanded with extra features. However the price for this structure was that the motion planning used more than fifty percent of the time on allocating, assignment, comparison of the structure. The information about the time usage was gathered with compiling the program with the -gp flag and using the gprof after the execution of the program. To save running time a native type is used.

Angle, Point, Edge, Vector, and Simple Polygon

An efloat is used to represent the angle in radian. A point is represented as two efloats. One for each dimension of the point. A vector is represented in a similar way as the point namely with two efloats. Both the edge and the simple polygon are represented through points in the plane. An edge is represented as two points. A simple polygon is represented as a list of n points $\{p_1, \ldots, p_n\}$, where the point pairs $\{(p_1, p_2), \ldots, (p_n, p_1)\}$ represent the edges of the border of the simple polygon. The definition of the structs can be found in the files `Angle.hpp`, `Point2D.hpp`, `Edge2D.hpp`, `Vector2D.hpp` and `SimplePolygon.hpp`. The implementation can be found in the following files `Angle.cpp`, `Point2D.cpp`, `Edge2D.cpp`, `Vector2D.cpp`, and `SimplePolygon.cpp`

Arrays

An array is represented as a template with both the data and the length of array. The reason for representing an array this way is that C++ does not store the length of an array, and this information is very useful. I.e. let a function calculate the convex hull over an array given as input and let it return the convex hull as an array. Whatever called this function will now loose the information about how big the returned array is, because the length of the input and output might not be the same. The array implementation eliminates this problem and arrays with their size can be passed easily between functions. The array is a template, hence it must be implemented in the header due to GCC restrictions. The header file of the array can be seen in file Array.hpp.

Sets, Lists, Vectors, and Maps

For collections like sets, lists, vectors and maps the STL implementation are used. The program does not need any special implementations of these and the STL implementation performs fine as long the program can fit in memory.

Graphs

Only a smaller functionality of the graph data type is needed in the program. A graph is implemented with a vector of vertices of the graph. This collection is used to sort vertices after some comparison functor or to apply a functor to each vertex in the graph. The edges between the vertices are represented with incident list at each vertex and by an edge list. The only needed functionality of the graph is traversal and inserts. The delete functionality is not needed in the graph because no configuration will never have to be removed from the graph again and no edges never needs to be removed as well. This have the effect that the graph can be optimized for insertion and traversal. I.e. if there is a space problem then the edge list can be removed from the graph but it is very convenient to have when saving the graph to disk. A connected component is represented as with a vertex and a size. The vertex is a vertex in the connected component and hence the connected component can be traversed from this. The size is how many vertices there is in the current connected component, which is used when inserting a new edge in the graph, this might result in two connected components that is merged into one. This merge is done by traversing the vertices in one connected component, updating their pointer to the other connected component, and end the end the connected component is removed. To optimise the update of connected components the smallest connected component is selected. The graph, the vertices, the edges and the connected component is templates so the implementation is in the header file, due to GCC restrictions. The code can be seen in the following files: `UndirectedGraph.hpp`, `UndirectedGraphVertex.hpp`, `UndirectedGraphEdge.hpp` and `ConnectedComponent.hpp`.

Road Map

The RM representation is a part of the PRM class and it uses a graph for representation. Each node in the graph is a configuration in the road map and each edge in the graph is a collision free path between two configurations. More information about how to construct, expand and query the RM is in Section 5.9

where the three phases are documented. The definition of the RM is in the file `PRM.hpp` and the implementation is in the file `PRM.cpp`.

Object

The object moved around in the work space is represented as a list of components. Each component is represented by a shape, parent, connection point. The shape of the component is represented as a simple polygon. The parent component is represented as the index of the parent component at the object and the connection point is represented as the index of the point at the parent component. The connection points are represented as a list of points where sub components can be connected to the current component. The object and component definition can be seen in the files `Object.hpp` and `Component.hpp` and the implementation is in the files `Object.cpp` and `Component.cpp`.

Configuration

A configuration is represented with the following three parameters : the object that the configuration is a configuration of, the position of the object and the orientations of the components. The object is a pointer to the object and the position is a vector. The orientations is a list of angles, where the angle at index i the list is the orientation of component i of the object. The definition and implementation of the configuration can be seen in the files `Configuration.hpp` and `Configuration.cpp`, respectively.

Work Space

The work space is contains a vector with polygons that represent the obstacles of the of workspace. The border is two points, a minimum point and a maximum point, which defines a rectangle. The minimum point represents the corner that has both minimum x- and y-coordinate and the maximum point has both maximum x- and y-coordinate. The definition and implementation of the workspace can be seen in the files `WorkSpace.hpp` and `WorkSpace.cpp`, respectively.

5.4 Creating Configurations

There is two ways that the program generates configurations. The first type is random configuration generation where a random free configuration is generated, this is used in the construction step of the PRM algorithm. In the expansion step free configurations are generated by a RBW.

In the learning phase the creation of random free configurations for the construction of the road map happens by creating random values for each parameter of the configuration within a legal interval. These random numbers are generated by calling the *rand* function in the C library call. However this only generates pseudo random values from a seed, but is faster than generating "real" random numbers. The current time on the computer is used as seed value to add extra randomness to the generation. When a configuration is generated it might not be a legal configuration. Each generated configuration is tested for intersections, and if this test fails the configuration is discarded, else it is used a new configuration in the PRM. For each failed configuration a counter is

incremented and while this counter is less than some maximum number of fails the program continues to generate configurations. As soon the counter reach the limit the program terminates, which makes the program terminable even though there is no free configurations in the configuration space.

The RBW is implemented by generating a random configuration and then move from the start configuration in direction of the generated configuration. Moving the configuration is done by making a linear function with a step size. Then for each configuration step the next configuration step is tested whether it is a free configuration, if not another random configuration is generated to find a new direction to move in. This continues until a certain number of steps is reached, and the configuration that is current as configuration step is the generated configuration of the RBW.

The definition and implementation of both configuration creation approaches is in the files `ConfigurationGenerator.hpp` and `ConfigurationGenerator.cpp`.

5.5 Minkowski Sum

The Minkowski Sum consists of two functions the sum function which generates the Minkowski Sum and it uses a help function to sort the two given arrays of points. They are sorted so the first point is the point with minimum y-value and if there is two it is the one with the minimum x-value. Then it is made sure the points go Counter Clock Wise (CCW) around in the polygon. If they go Clock Wise (CW) around in the polygon the inverse order is CCW. That is what the help function does and the resulting ordered polygon is returned as a new point array. The sum function builds on the pseudo function from chapter 13 in the book by Berg *et al.* [4]. Where it generates a point array that contains the Mikowski Sum of the two point arrays it takes as a parameter. The maximum number of points in the resulting polygon of a Minkowski Sum of two convex polygons is the sum of the number of points in the two convex polygons. An vector could be used instead, but due to the nature of the vector when inserting points an array of the maximum size is used. This saves the double effect of the vector, though amortised it is linear but the Minkowski Sum is called often in the movement approximation of a linked polygon. Using more ram for processor time seems like a good trade off because it is only local in each connection attempt there will be larger usage of memory and that usage is not that great comparing to storing information about the road map. The definition and implementation of the Mikowksi Sum is in the files `MikowskiSum.hpp` and `MinkowskiSum.cpp`, respectively.

5.6 Convex Hull

The Convex Hull is implemented as the algorithm given in Chapter 1 in the book by Berg *et al.* [4], which is based on the Grahams Scan. Normally a STL vector can be used to contain the points approximated but an array is used with the size of the input. This can be done because the maximum size of a convex hull of a polygon is maximum the size of the polygon. The array is chosen to avoid the doubling effect of the vector and save computing time. There is only one usage of the Convex Hull algorithm in the motion planning

algorithm and that is the line approximation. The result of approximating each end point in the edge/line is a constant size. This leads to a need for faster computation time with the cost of more memory which is achieved by using the array type instead of the vector. To achieve even more speed up the algorithm has been implemented within a single function to save time to call sub functions. The files `ConvexHull2D.hpp` and `ConvexHull2D.cpp` contains the definition and implementation of the convex hull algorithm.

5.7 Approximation

The approximation is divided into two: movement approximation of everything up to a simple polygon and movement approximation of linked polygons.

The first half of the approximations is build by a function that approximates point a movement, with translation, rotation and enlargement polygon. The translation is a vector representing the translation of the point, if there is no translation the vector is a zero vector. The angle rotated is an angle representation and a rotation matrix is generated to make the rotation. If there is no rotation there will be no generation of a rotation matrix. If there is a polygon of which the movement must be enlarged with, this is needed in the approximation of linked polygons, then a Minkowski Sum is done. The line movement approximation calls the point approximation for each endpoint of the edge. A convex hull is then found on the points generated from approximation of the points. The convex hull is returned as the result of the line movement approximation. Finally there is the simple polygon movement approximation which for each edge in the polygon generates a line movement approximation and returns the collection of all line approximations in a simple polygon array. The number of polygons generated is equal to the number of edges in the polygon approximated. This makes an array an obvious representation instead of the vector to save computation time.

The second half of the approximations is build on the function that approximates the movement of a linked polygon. It takes two configurations, that describes where the movement goes from and to. The approximation will for each component approximate the movement of the component. The function follows the theory from Section 4.1.2 but it uses point arrays instead of polygons within the function. In the end each approximation is converted to a polygon and added to an array of polygons which is returned.

The files that defines the first half of the approximation is in `ApproximateMovement2D.hpp` and the implementation is in `ApproximateMovement2D.cpp`. The definition of the second half is in `ApproximateMovement.hpp` and the implementation is in `ApproximateMovement.cpp`. The split was made to prepare the algorithm for three dimensional motion planning and the ApproximateMovement was supposed to handle both cases. Then whether it was two dimensional approximation or three dimensional approximation it could use the approximation for two dimensions or three dimensions, respectively.

5.8 Local Planner

All LP takes a pointer to a collision detector so the same collision detector can be used throughout the program. Another advantage of this is that the number of collision tests can be counted. Three LPs are implemented: the first one is the original LP, the second is the interpolation LP and the third is the approximation LP. There are more about the implementation of these in Section 5.8.1, Section 5.8.2 and Section 5.8.3.

5.8.1 Original Local Planner

The original LP attempts to connect two configurations by moving the configuration in small steps from one configuration to the other configuration. There is generated a current configuration that starts in the position of one of the configurations and a configuration that represents the size of the step is also generated. For each step that needs to be taken the step size configuration is added to the current configuration. The current configuration is then for each step tested for intersections with any obstacle in the workspace. If there is an intersection the connection failed, else if the current configuration reaches the second configuration then the connection is successful. The definition and implementation of this LP can be seen in the files `OriginalLocalPlanner.hpp` and `OriginalLocalPlanner.cpp`, respectively.

5.8.2 Interpolation Local Planner

There is a small difference between the original LP and the interpolation LP and that is the object moved. In the original LP the original object is moved around and in the interpolation LP it is the ε-enlarged object. The ε-enlargement happens in a different class and a pointer is stored in the interpolation LP to the ε-enlarged object for futher usage. The files `InterpolationLocalPlanner.hpp` and `InterpolationLocalPlanner.cpp` contains the definition and implementation.

5.8.3 Approximation Local Planner

The movement LP approximates the movement between two configurations, which give a polygon array with approximation of each edge. Then the LP test each approximation for intersection. If an approximation intersects an obstacle the LP attempts to divide the movement into two movements generating a temporary configuration midway between the two configurations. Then each of the two new sub movements are approximated and tested for collisions with any obstacles in the work space. The LP is implemented in such way that as soon it knows that it cannot connect two configurations it will report failure, it does not continue the search. When the LP can report failure or success it cleans the memory for the used approximations and the temporary configurations. For further information the definition and implementation is in the files `ApproximateLocalPlanner.hpp` and `ApproximateLocalPlanner.cpp`.

5.9 PRM

The PRM is implemented as one structure that contains a graph to represent the RM. Further it contains three functions that implement the construction, the expansion and the query of the RM. The implementation of the construction phase, the expansion phase and the query step is described in Section 5.9.1, Section 5.9.2 and Section 5.9.3, respectively. The definition and the implementation of the PRM are in the files `PRM.hpp` and `PRM.cpp` respectively.

5.9.1 Construction

The construction phase of the PRM algorithm generates random free configurations and attempts to connect it to configurations in the RM. How the connections are attempted is handled by the LP described in Section 5.8. The construction of the road map continues until the number of wished configurations have been added or until the maximum number of failures have been reached. The failures in mind are failures of generating a free configuration, not failures of connecting configurations.

5.9.2 Expansion

At first the expansion phase of the PRM algorithm need to calculate the probabilities of selecting each configuration for expansion. This is done by a vector the for the failure rate $r_f(c)$ rounded to the nearest integer, inserts a corresponding number of references to the configuration. Figure 5.9.2 shows an example of the vector to select a configuration for expansion can look like.

1	2	3	4	5	6	7	8	9	10
C_1	C_1	C_2	C_2	C_2	C_3	C_4	C_5	C_5	C_5

Figure 5.2: The vector to select configuration for expansion in a thought situation.

When all configurations failure ratio $r_f(c)$ have been calculated, then an index is selected randomly in the interval $\{1, \ldots, n\}$ where n is all the total number of references. This index reference is then followed to the configuration which have been selected for expansion. Then the expansion generates a configuration with the RBW and the generated configuration is then added to the RM and attempted connected to the other configurations in the RM. The expansion continues to select new configurations for expansion until the number of wished configurations has been generated. There is just a single but in the expansion step, the RBW might not be able to find a new configuration, if this is the case the number of found configurations is still increased to make sure that the expansion phase does terminate.

5.9.3 Query

The query phase of the algorithm is not like the original query phase described in the article by Kavrak *et al.* [11]. Here they describe a query phase that if it fails it can do some RBWs to attempt connection and the path found is

put through some repair functions. These things are not implemented because the focus in this master thesis is lowering the number of collision tests by using approximations to connect configurations in the LP. However it does attempt to find a path by connecting the start configuration and the goal configuration to the same connected component. It attempts to connect to the closest connected component and if it fails it attempts to connect to other connected components in increasing distance from the start and goal configuration. If the start and goal configuration is successfully connected to the same connected component then the rest of the path is a graph path search.

Chapter 6

Debug

This section describes the debugging of the implemented PRM. Both the original interpolation strategy by Kavraki *et al.* [11] and the new approximation strategy is tested to ensure they are correct. It is vital that the implementation is correct so the experimental comparison between the two approaches is reliable. There is two different types of debugging tests executed in this thesis. The first type of test is the automatic test, which is executed automatically. If a test case fails the program will terminate the test with an error report describing the problem. The second type of test is the visual test, here each test case outputs a fig figure that is viewed in xfig. Each test in the visual test has to be manually approved to ensure that the output generated is correct. The visual tests has the advantage that humans easily can see errors when geometric structures are visualised.

6.1 Automatic Tests

To debug and ensure that the algorithms and the structures that the PRM builds on are correct an automatic test was constructed. This ensures that a given test case satisfy its constrains when the test is done. In this way if an optimization is implemented in an algorithm it will still output the same result to get through the test case. There have been implemented a large number of test on the different structures and algorithms in the program. However this form of automatic test is slow to construct compared to visual debugging. This has the consequence that the tests are not all throughout complete as they could be while most tests are done visually. As an example the convex hull test case is described in Section 6.1.1. In Section 6.1.2 the output from the test program is explained.

6.1.1 Convex Hull

When analysing the Grahams Scan algorithm there are five interesting sizes of the input that needs to be tested. These are shortly listed in Table 6.1

The first case is the zero case where the convex hull algorithm has to return an empty point array.

The one point and two point case does in both cases return a point array

$\emptyset$	The empty case
$\{p\}$	The one point case
$\{p_1, p_2\}$	The two points case
$\{p_1, p_2, p_3\}$	The three points case
$\{p_1, \ldots, p_n\}$	n points case

Table 6.1: A short overview of interesting input sizes to the Grahams scan algorithm.

with the same points. However what if the two points in the array are the same point ? Should the algorithm terminate the program because there is a constraint telling that there cannot be two points that are equal in the same list. By equal it is meant that the two points, p and q, are equal $p = q$ which means each coordinate set is equal $p_x = q_x \wedge p_y = q_y$. In this thesis the convex hull algorithm cleans the input list of equal points, before the algorithm starts. This answers the question from before, the algorithm must return a point array with one of the points.

Now to the three point case where it often returns a list of three points, due to the fact that three points represents a triangle, which is the most basic convex structure in two dimensions. Like the two point case there can be equal points which are removed. Finally there is a problem when the three points are aligned, what should the algorithm do ? Return an array with two points or three points ? This has the origin of whether a points orientation comparison with two other points is strictly. If the comparison is strictly points on line with others are not considered as a part of the convex hull. Points that are on line are not considered as a part of the convex hull in this thesis. The implemented convex hull algorithm will in this given test case return a list with the two end points as a result.

Finally there is the n point case, which is by far the most complex one to test. Some base cases whether it can find the convex hull of a simple number of points is done. After that several list of several points has to be tested such that the Grahams Scan algorithm has to remove points from the internal convex border. By that is meant that when it is about to add a new point the convex border will be broken if the point is unless points are removed from the current border. If the algorithm is correct this will be done correctly and these points will not be in the convex hull returned by the algorithm.

These test cases has been implemented in the `TestConvexHull.cpp` file which tests the implemented Grahams Scan algorithm in the `ConvexHull2D.cpp` file.

6.1.2 Test Program Output

When running the motionplanning-testrun program it outputs the different headings for a specific test. The convex hull is one of the heading meaning that all the test of the convex hull happens within the start heading `Testing Convex Hull started` and the end heading `Testing Convex Hull completed`. After all the tests are executed the program test a PRM construction of 100 configurations. Some of the output when executing motionplanning-testrun is listed below.

```
Testing Array started
Testing Array completed
Testing object started
  Testing Test Component started
  Testing Test Component completed
Testing object completed
Testing UndirectedGraph started
Testing UndirectedGraph completed
Testing geometry started
  Testing Angle started
  Testing Angle completed
  Testing Test Point2D started
  Testing Test Point2D completed
  Testing Vector2D started
  Testing Vector2D completed
  Testing RotationMatrix2D started
  Testing RotationMatrix2D completed
  Testing SimplePolygon started
  Testing SimplePolygon completed
Testing geometry completed
Testing Convex Hull started
Testing Convex Hull completed
Testing MinkowskiSum started
Testing MinkowskiSum completed
Testing Approximate Movement started
  Testing Approximate Movement 2D started
  Testing Approximate Movement 2D completed
  Testing Approximate Movement started
  Testing Approximate Movement completed
Testing Approximate Movement completed
Testing LocalPlanner started
  Testing Original Planner started
  Testing Original Planner completed
Testing LocalPlanner completed
loading workspace
workspaced loaded, loading object
loaded object, init prm
Random() constructor
construction started
...
```

6.2 Visual Tests

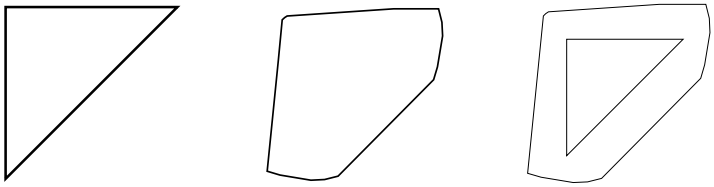
To ensure that the different aspects of the algorithm is correct several visual tests have been done. Visual tests are faster to implement than the automatic tests from Section 6.1 but they need visual verification. However, a human can easier verify whether some output is correct from a geometric algorithm or structure when it is visualised rather than looking at some numbers. The reason for doing visual tests is to ensure that the different algorithms and structures

are correct when experimenting with the collision tests count.

6.2.1 ε-enlargement

The ε-enlargement is tested with visual debugging on several convex objects was tested. The test of the enlargement is important to ensure that the interpolating LP used to as comparator to the new approximation LP is correct. In this section the triangle object will be used as an example but all object ε-enlargements can be seen in Appendix B.

The original triangle in Figure 6.1(a) is used to verify that the object used is interpreted correctly by the program. The ε-enlargement of the triangle using a step size of 10 and a rotation of 0.1 is in Figure 6.1(b) This figure is used to see whether the polygon is still convex after the enlargement. Finally Figure 6.1(c) is the triangle within its ε-enlargement. This figure is used to two things: first to determine whether the resulting polygon from the ε-enlargement is larger than the original polygon and secondly verify that the polygon is centred within its enlargement.



(a) The original triangle.

(b) The ε-enlarged polygon of the triangle.

(c) The original triangle within its ε-enlarged polygon.

Figure 6.1: Example on visual representation of the object, where the three figures are; Figure 6.1(a) is the original object, Figure 6.1(b) is the polygon from the ε-enlargement of Figure 6.1(a) and Figure 6.1(c) is the original object within it's enlargement.

6.2.2 Local Planner

There are three different LPs implemented and all of them is visual debugged with a wide range of different objects and work spaces. The visual debugging of the LPs is crucial to ensure the correctness of the output from PRM learning phase. The reason is that the LPs performance must be reliable to compare the results of the experiments. In this section a triangle is moving from a configuration $c_1 = (100.0, 100.0, 0.0)$ to a configuration $c_2 = (250.0, 100.0, 1.0)$. There are several other visual tests of the LP and these can be seen in Appendix D.

The debug has been divided into five different steps; the two configurations, the real movement (Original LP), the interpolating LP, the approximating LP and the binary spilt in the approximating LP.

The Two Configurations

This visual test is generated to ensure that the program represent the object correct at a given configuration. Another detail is that it helps to understand the output from the other LP tests when it is clear where the object is moving from and to. Figure 6.2 is the visual output of a triangle at each configuration.

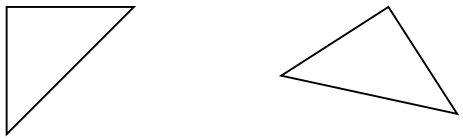


Figure 6.2: The triangle at from configuration $(100.0, 100.0, 0.0)$ and at to configuration $(250.0, 100.0, 1.0)$.

The Original Local Planner

The original LPs output is generated as a reference of the real movement when debugging the two other LPs. A feature of the original LP is that it is using the interpolation strategy to connect two configurations as the interpolating LP but with the original object instead of the ε-enlarged object. This feature is convenient to visual debug the interpolation connection strategy as well. The visual output from the original LP for a triangle can be seen in Figure 6.3.

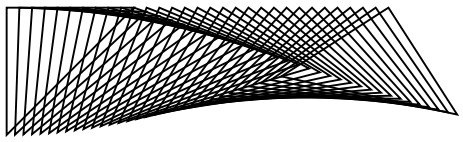


Figure 6.3: The original movement of the triangle between two configurations.

The Interpolating Local Planner

The interpolating LP is the LP strategy used in the original article by Kavraki *et al.* [11]. This strategy must with the placements of the ε-enlarged object contain at least the area the real movement covers. Comparing the original movement in Figure 6.3 and the output from this LP in Figure 6.4 confirms that the interpolating LP is correct.

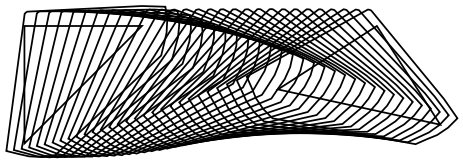


Figure 6.4: The interpolating local planners output for a triangle.

The Approximating Local Planner

In this thesis an approximating LP is proposed to lower the number of collision tests done when connecting two configurations. To validate the LP contains at least the real movement of an object visual tests has been done. The output of the approximating LP is hold against the output from the original LP to check if the approximation covers at least the area of the real movement. An example is the triangle and the output when connecting the two configurations $(100.0, 100.0, 0.0)$ and $(250.0, 100.0, 1.0)$ is shown in Figure 6.5. Then by looking at the original movement at Figure 6.3 it can be concluded that the approximation holds.

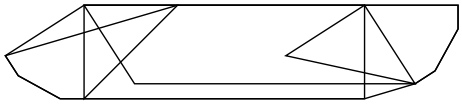


Figure 6.5: The approximation local planner output for a triangle.

The Binary Spilt in the Approximating Local Planner

The final test of the LPs is the test of the binary split in the approximating LP. The visual debug of this case is important to ensure that the binary connection strategy works as intended. The test is done by using the approximating LP and let it fail the first connection attempt between the two configurations c and c'. When failing the connection attempt the approximation LP will split the search into two sub problems. At first the configuration c'' describing the middle of the movement is found and the search continues for two new connection attempts. The first is connection is the from configuration c to configuration c''. The second connection is from configuration c'' to configuration $Configuration'$. Figure 6.6 is an example on the output from this scenery. Again it is crucial that the approximation is at least covering the area of the original movement, seen in Figure 6.3. Another issue is to ensure that the approximation looks somewhat smooth such that there is no edges or polygons sticking out from the rest of the approximation.

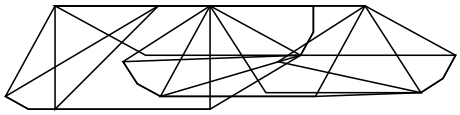


Figure 6.6: The approximation local planners output during a single binary split of the approximation for a triangle.

6.2.3 Road Map

To ensure that the generated RM is correct they are visually debugged. There are three constraints that must be obeyed by the configurations and paths representing the RM. These constraints are as following: the configurations must be free configurations and paths must be both non cyclic and legal.

Configurations must be free configurations, hence they are not allowed to intersect any of the obstacle edges. This is controlled by outputting both the object at all configurations and the obstacles. Then a visual check is performed ensuring that no configurations intersects any of the obstacles in the workspace. In this example a triangle moving in the labyrinth workspace is considered. Figure 6.7 is output generated to visualise the configurations placements in the work space. Triangles are placed in the labyrinth work space and it can be seen that no configuration intersects any obstacle.

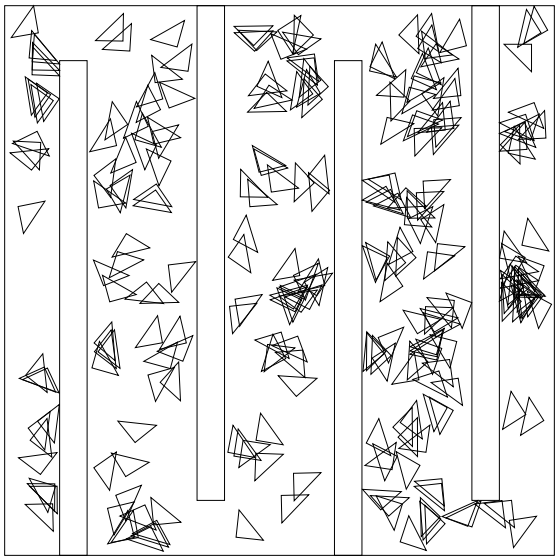


Figure 6.7: Placement of a triangles for each configuration in the road map. The work space used is the labyrinth work space.

The RM must be a Directed Acyclic Graph (DAG), this means cyclic paths are not allowed in the RM. All edges in the RM is outputted for visual confirmation that there is no cycles in the graph. Figure 6.8 is the triangle in the labyrinth work space, and for each edge in the generated road map an edge is visualised. It is clear that the graph in non-cyclic as the constraint was.

Paths must be legal paths, meaning that along a path the object is not allowed to intersect an obstacle. The first control is to check whether an edge going from the reference points of the two configurations is collision free. The program outputs these edges and the obstacles so it can be visual confirmed that there is not intersection between the edges and the obstacles. To confirm no intersections happens in the paths, each path is printed out as a interpolation of placed objects with a small step size. These outputs can then visually be confirmed to be collision free with the obstacles. An example of this is Figure 6.9 where the triangle is placed in the labyrinth workspace several times along each edge in the road map. The area the movements each edge represent is covered with triangles and it can be confirmed visually whether the paths are legal path. This given clearly shows that the paths found are legal in the current example.

There are more examples of these visual views for other object types in the labyrinth work space that can be seen in Appendix E.

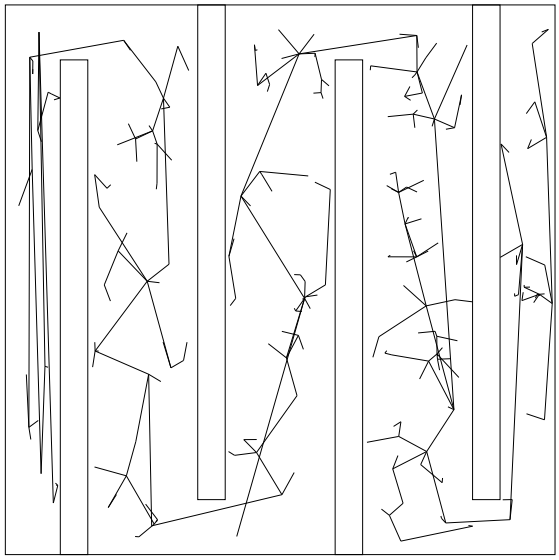


Figure 6.8: Placement of an edge for edge in the road map. The work space used is the labyrinth work space.

Figure 6.9: Placement of a triangles showing the movement between each edge in the road map represents. The work space used is the labyrinth work space.

Chapter 7

Results

A new connection strategy is developed to lower the number of collision tests when constructing the RM. The collision tests count is the count of how many times two arbitrary edges are tested for collision during the construction.

The experiment is executed on seven different work spaces to ensure that a general conclusion can be draw from the data generated. Another reason for all these tests is to see how the complexity of the work space effects the number of collision tests. There are seven work spaces: the empty work space, the cross work space, the labyrinth work space, the circles004, the circles008, the circles016 and the circles032, these are visual shown in Appendix C.

Four different object types are tested on each work space to test whether the complexity of the object has any affect on the number of collision tests. The four objects are: the triangle, the square, the two small sticks and the three small sticks. The objects are shown in Appendix B.

Section 7.1 describes the setup of the experiments, which is what machines are used, what scripts that are created to execute the experiments, etc. Then there are separate sections for each type of work space: Section 7.2 is the empty work space, Section 7.3 is the cross work space, Section 7.4 is the labyrinth work space and Section 7.5 is the circles work spaces. Finally in Section 7.6 the result from the experiments is concluded.

7.1 Test Setup

This section contains a short description of the different setup parameters set when collecting the data.

To make the program return the number of collision tests a counter was placed in the collision detector that counts every time two edges are tested for collision. Then the test suite calls the program from a perl script, which collects the data and stores it in a data file. This file is then used by gnuplot to generate graphs that can be closer examined. The files `CCCcircles004.pl`, `CCCcircles008.pl`, `CCCcircles016.pl`, `CCCcircles032.pl`, `CCCempty.pl`, `CCCcross.pl` and `CCClabyrinth.pl` contains the perl scripts that selects the work spaces used to generate data from. However each of them uses the two files
`CreateCollisionsCountData.pl` and `CreateData.pl` which contains the gen-

eral approach to generate data and all the different lists of work space, object, etc. Each setup is executed ten times and the result stored is the average of these ten executions. Another detail one should be aware of is that both the initial step and the expansion step of the construction phase is executed. The effect of this is when two configurations is set as parameter it will attempt to find two configurations in the initial step and two others in the expansion step, which is four generated configurations. The rest of the parameters are static to ensure trustworthy data. The parameter of the maximum number of failed attempts to find a free configuration is 1000. The RBW takes between 10 and 100 steps and the search radius to connect configurations is 30.

The data collection has been divided out on a range of machines to speed up the process. There are two types of machines used to collect the data and the specifications of these two can be seen in Table 7.1.

CPU	Intel Xeon CPU 3.00GHz	Intel Pentium 4 CPU 3.00GHz
Memory	1Gb	1Gb
OS	Fedora core 3	Fedora core 3
Compiler	GCC 3.3.2	GCC 3.3.2

Table 7.1: The two machine types the collision tests count experiments are executed on.

There are two different LP approaches compared in this thesis, the interpolating LP and the approximating LP. The interpolating LP is using a rotation angle of 0.1 radians per step and a step size of five. There are four different search depths used with the approximating LPs, these are; one, two, four and eight. The naming on the graphs is approximation 1 for the approximating LP with search depth of one and approximation 2 for search depth two and so on.

7.2 The Empty Work Space

This workspace shows the performance of the approximation LP in large open spaces. It should be here that the approximation LP has it strength in saving collision tests compared with the interpolating LP. The collision tests in the empty work space comes from testing for collision between the object and the border of the work space. Another detail about the empty work space is that it is a very large work space going from (-10000, -10000) to (10000,10000) where the others goes from (0,0) to (1000,1000).

The graph in Figure 7.1 clearly indicates that the number of collision tests is lowered by the new connection strategy. There is barely no difference in the number of collision test no matter the depth of the approximation LP. The reason is the there is barely any collisions when connecting configuration and hence there will be no difference in the performance on the search depth. The object used as an example here is the object, Figure B.1(g), where three small sticks that are linked together.

There are graphs of all object types which can be seen in Figure F.1 in Appendix F. They all clearly shows that the approximating LP lowers the number of collision tests for the empty environment.

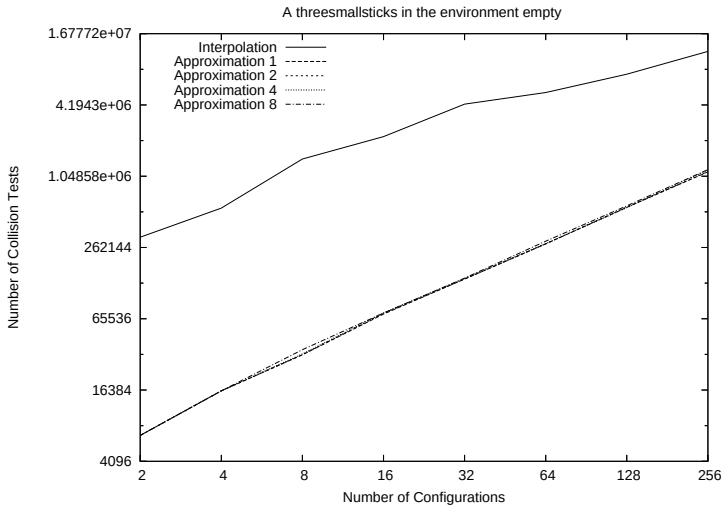


Figure 7.1: Collision tests in the empty work space with the three small sticks linked together. It is clear from the graph that approximations lowers the number of collision tests.

7.3 The Cross Work Space

The reason to test the number of collision tests counts on the cross work space is to show the difference in the number of collision tests in a simple work space. The cross work space can be seen in Figure C.1(a) in Appendix B. The results of the collision tests count for the object, Figure B.1(g) in Appendix B in the cross workspace can be seen in Figure 7.2. The order between the different search spaces are as expected with the smallest search depth as the best and the largest search depth is worst. The approximation strategy seems to come out as a winner, except when using a search depth of eight. When using a search depth of eight the number of collisions tests will exceed the interpolating LP. Using such high a search depth gives a potential of 2^8 of sub connection attempts, which gives the approximation a very fine resolution when the distance between the two configurations is small. Another detail is that the interpolating LP is using a step size of five which is very large steps, maybe a step size of one or even less should have been selected. If a smaller step size is selected the number of collision tests will go up with a reasonable factor for the interpolating LP.

The result of the other three figures reasons for the same conclusion and these can be seen in Figure F.2 in Appendix F. However the number of collision tests for the triangle is different, while here is the interpolating LP the approach having most collision tests. This can be due to the fact that the triangle only consists of three edges which results in fever polygons representing the approximation, hence fever edges to test for collision in each step.

7.4 The Labyrinth Work Space

This is a different type of workspace where long obstacles from each side ensures that the object must zigzag through the scene. Another detail is that this

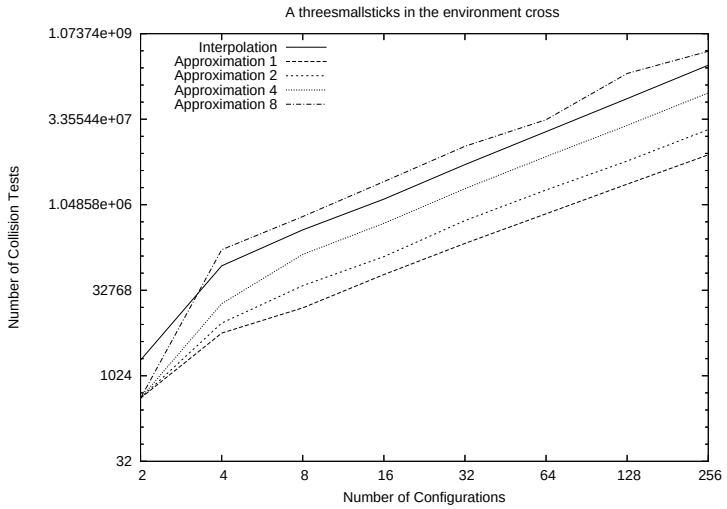


Figure 7.2: Collision tests in the cross work space with the three small sticks linked together.

approach have the potential of generating many failed connection attempts due to the closest configuration might often be on the other side of or inside an obstacle. The labyrinth workspace can be seen in Figure C.1(b) in Appendix B. The graph for a triangle, in Figure 7.3, shows that the approximation with search depth eight is cheaper than the interpolation. However the graph for three small sticks, in Figure 7.4 imply otherwise. The reason for this is the number of edges in the triangle are fewer than the number of edges in the three sticks, hence less complex approximation. Another thing that tributes to the complexity of the result from the approximation of three sticks is that they are linked, this can be seen in Figure D.19, Figure D.20 and Figure D.21 in Appendix D.7.

Again there is a reasonable factor in the number of collision tests between the different search depths. Where the larger search depth the more collision tests are done. All approaches seems to follow each other nicely when increasing the number of configurations which implies that there is a constant factor between them.

7.5 The Circle Work Spaces

The circle works space is generated in several resolutions, these are then tested to see if the relation between the different LP approaches and settings holds no matter the resolution. The work spaces used are: circles004 Figure C.2(a), circles008 Figure C.2(b), circles016 Figure C.2(c) and circles032 Figure C.2(d).

The first relation to be confirmed is testing with the triangle object, Figure B.1(a) in circle work spaces of different resolutions. The graph in Figure 7.5 and the graph in Figure 7.6 agree that the worst is the interpolating LP, then the approximating LPs. Where the order is from the best to the worst, resolution on one, two four and eight. This relation is as expected but both graphs are a

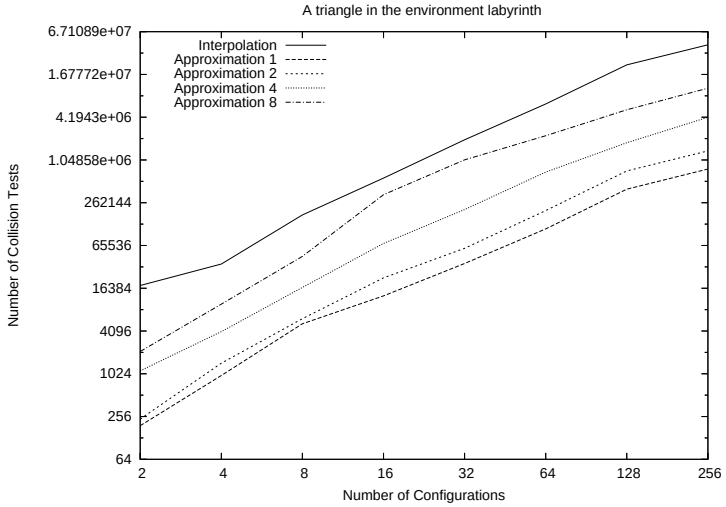


Figure 7.3: The number of collision tests for the triangle object in the labyrinth work space.

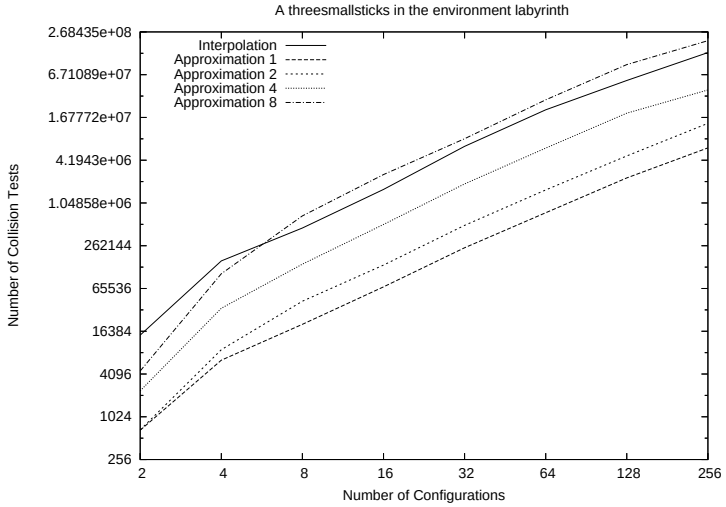


Figure 7.4: The number of collision tests for the three sticks object in the labyrinth work space.

bit unclear when counting collision tests for two configurations. The reason is that some configurations can be placed close together or almost together. If this happens it will have a huge effect on the total number of collision tests while there is only two configurations.

The second relation to be confirmed is that linked polygons increases the number of collision tests for the approximating LP compared with the interpolating LP. But the relation between the approximating LPs search depths should continue to be the same. Again the tree small sticks object from Figure B.1(g) in Appendix B is compared in circle work spaces of different resolutions.

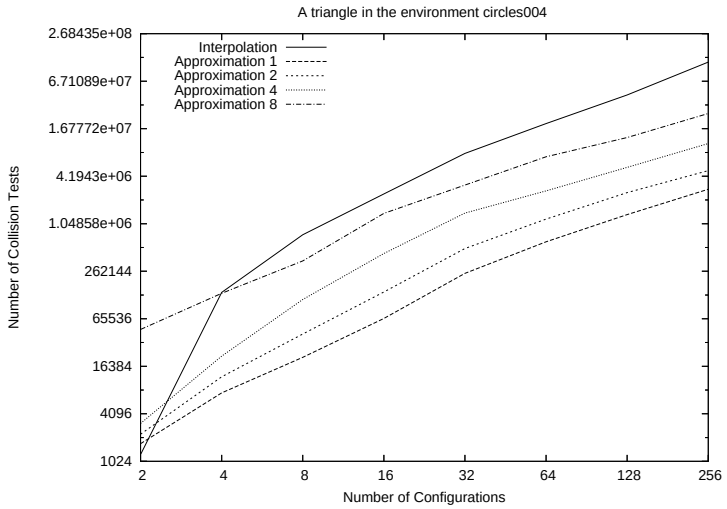


Figure 7.5: Number of collision tests for a triangle in the circles004 work space.

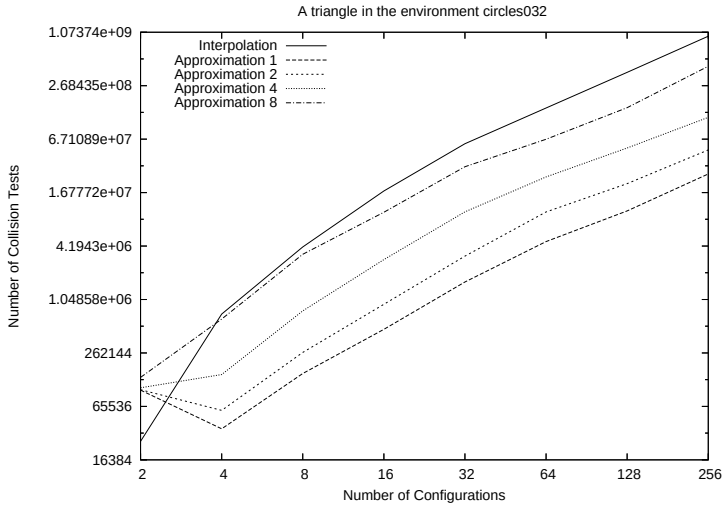


Figure 7.6: Number of collision tests for a triangle in the circles032 work space.

Only the circles004 work space Figure C.2(a) and the circles032 work space Figure C.2(d) is shown in this section but there are tests of the two work spaces with a resolution between these two. Appendix F contains the graphs of all experiments executed. The graph in Figure 7.7 and the graph in Figure 7.8 confirms this assumption.

The number of edges in the work space does not change the relation between different approaches, but the number of edges in the object does. This is shown with the circles work spaces of different resolution with four different object types. Another thing is that the step size of the interpolating LP is set to five and that is a large step size. For example if the step size was lowered one the interpolating LP is expected to do five times as many collision tests as it does

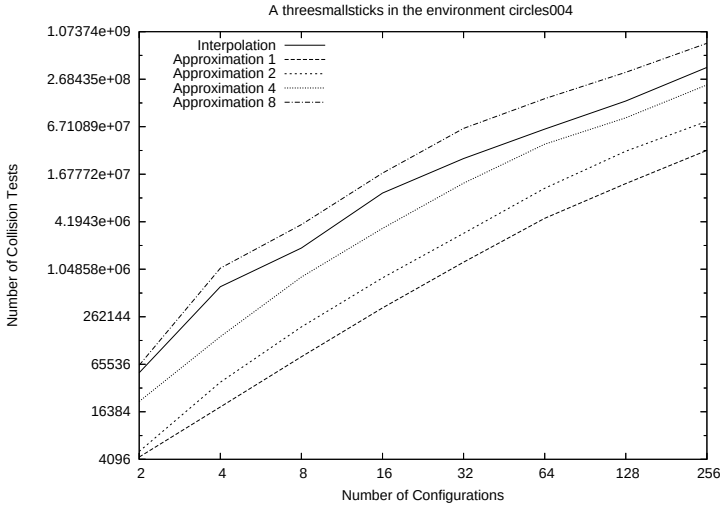


Figure 7.7: Number of collision tests for three small sticks in the circles004 work space.

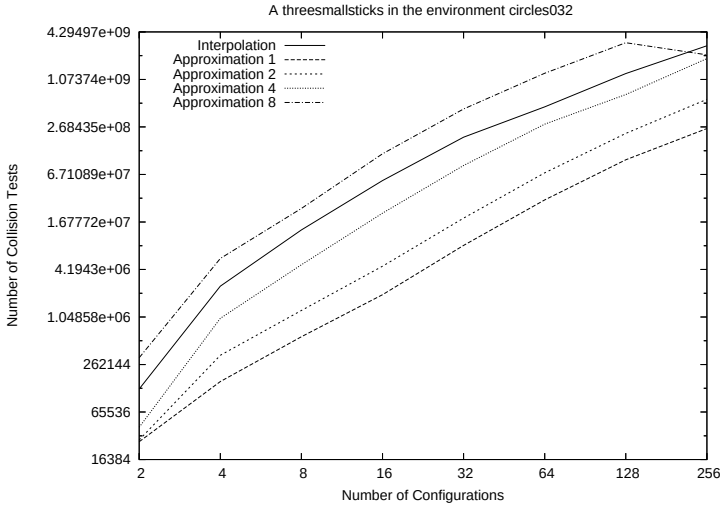


Figure 7.8: Number of collision tests for three small sticks in the circles032 work space.

with step size five. With this in mind though the interpolating LP is better than the approximating LP with search depth right now a factor five would shift the order of the two.

Table 7.2 is a table over the data extracted from collision tests of the circles004 work space. Here the interpolating LP for 256 configurations does in average 378815600 collision test and the approximating LP with search depth of eight does 764518276.6 collision tests in average. This concludes the interpolating LP beats the approximating LP by a factor 2. However if the step size in the interpolating LP is lowered to one then approximately a factor five will

by multiplied to the current number. This results in approximately 1894078000 collisions tests instead and then the factor will be 2.4 in advantage for the approximating LP for search depth eight. This concludes if a step size of less than 2.5 is selected for the interpolating LP the approximating LP of search depth eight will come out on top in number of collision tests for the three small stick object. However for more complex object the border line of which is best will be lower. The more complex object the smaller step size in the interpolating LP is the border between which is best. The best factor is between the approximating LP with search depth of one and the interpolating LP where the approximating LP wins with a factor ten at least.

2	51108.5	4370	5145.1	22170.6	63824
4	630511.4	18857.7	38871.2	146784.7	1079760.1
8	1947877.6	81656.9	194891.5	834440.4	3857795.7
16	9681739.7	339534.9	815170.3	3450741.8	17319765.9
32	26518302.9	1289542.2	2984967	12923930.3	63909735.9
64	62631332.2	4636797.1	11121848.6	40244252.6	151916681.8
128	142193193.1	12746687	32859587.2	86924281.6	328584182.2
256	378815600	33589710.5	78606879.2	227335106.5	764518276.6

Table 7.2: The number of collision tests for three small triangles in the circles004 work space. First column is the number of configurations, then the collision tests for the interpolating local planner and finally collision tests for the approximating local planner with search depth one, two, four and eight.

Again it is confirmed that the lowest search depth in the approximation LP yields the lowest number of collision tests and it increases with the search depth.

7.6 Conclusion

The new LP approach lowers the number of collision tests when constructing the RM.

The different work spaces shows that the complexity of the work space does not effect the number of collision tests done by the different approaches. This is clearly shown in Section 7.5 where the number of edges is increased and the order of the tested approaches are still the same. The different search depth in the approximation LP is ordered as expected in all work spaces where a search depth of one is the best, second best is two and third is four and the worst is a search depth of eight. The only shifting happens between the interpolating LP and the approximating LP where the interpolating LP is better than the approximating LP with a search depth of eight from time to time. This is not due to the work space but is the object used.

The four different objects clearly shows that the approximating LP is more dependent on the complexity of the object used than the interpolating LP. The reason is that more complex objects yields a number of extra edges while the approximating LP does not generate any extra edges. In Section 7.5 it is stated that the step size of the interpolating LP can be lowered and that would effect the result in such way that the approximating LP will come out on top. But for arbitrary complex objects will make the approximation LP loose to the

interpolating LP.

Chapter 8

Conclusion

In this thesis a new local planner strategy for the probabilistic road map is suggested. This strategy uses approximations when attempting to connect two configurations. Further more it uses a binary connection strategy to do finer approximations when attempting connecting two configurations fails. The new strategy was created to lower the number of collision tests done in the learning phase of the probabilistic road map.

The approximation used when connecting two configurations is an over-approximation of the movement of the object between the two configurations. The approximation has the a worst case running time of

$$O\left(k \cdot (n+m)\right) \ ,$$

where n is the maximum number of points in the components, m is the maximum number of connection points in the components, and k is the number of components.

The binary connection strategy attempts to connect two configurations by over-approximating the path between them. If the connection attempt fails the binary connection strategy continues the search by dividing the path into two sub-paths. This has the effect that each of the two sub-paths has a finer over-approximation, and hence may be able to succeed in finding a path. The search depth can be adjusted to control how fine the approximations should be. The larger search depth that is used the finer the approximations will be.

Both the original approach and the new local planner have been implemented in C++. A great deal of effort has been placed into ensuring that different approaches generate correct output, so reliable test data can be generated from the implementation.

Several experiments have been performed to generate reasonable data shows the number of collision tests done under different circumstances. The experiments has been executed on different work spaces with different objects to show different dependencies. The experiments shows that:

- Larger search depths increases the number of collision tests.
- More edges in the work space increases the number of collision tests with the same factor for both the original and the new local planner.

- The new local planner strategy is more dependent on the complexity of the object than the original.
- The new local planner strategy saves collision tests compared to the original. But this is dependent on the complexity of the object moved, the search depth and the step size.

The proposed approach saves collision tests in the learning phase of the probabilistic road map. But the cost is more dependency on the complexity of the object moved compared with the interpolating connection strategy.

Chapter 9

Future Work

9.1 Removing non-important Edges

The approximations in the given strategy gives a polygon for each edge in the object moved. There are several edges in this approach that are duplicates. Removing these duplicates will lower the number of collision tests done by the approximation approach further. There are several edges of the polygons generated by the approximation that is within the outer boundary of the total approximation. It is suggested that a map overlay algorithm can be used to make a union of all the polygons. The union would remove inner edges as well as duplicate edges, hence lowering the total number of edges outputted by the approximation. This would lower the number of collision tests further but on the cost of more running time done approximating movements. But this trade off should be fine for work spaces with a high number of edges.

9.2 Comparing more Approaches

There is other connection approaches used to construct probabilistic road maps. There is a another binary approach where it uses binary division of the path instead of the interpolation. It could be interesting to compare this approach with the original and the new approach from this theses.

9.3 Connectivity Comparison

The number of collision tests have been lowered for some problems by the suggested local planner in this thesis. But how does it perform when focusing on the connectivity. Does the lower number of collision tests have a price of lower connectivity ? or can the suggested local planner have a lower number of collision tests while having the same connectivity as the original local planner. Several experiments can be setup by using the implementation from this thesis to solve these question.

9.4 Learning Phase Speed Comparison

Another interesting parameter that can be tested is how does the new approach perform when comparing the time usage in the learning phase. Does the suggested approach use to much time on approximation compared with what can be saved by lowering the number of collision tests. However the construction speed is not that an interesting parameter alone. If the speed of the learning phase is great but the connectivity is too poor it would be a bad strategy. What is interesting is a trade off between both speed and connectivity.

9.5 Combining Approaches

The suggest approach is cheap when connecting configurations that are far way from each other. At low distances the original approach seem better. The idea is to combine these two types of connection strategies. Where configurations over a certain distance is attempted connected with the suggested approach. While configurations under the distance is attempted connected with the original approach. This seems like a good combination between the two strategies that can yield even better results than the current.

9.6 Three Dimensions

Applying the suggested approach to three dimensional problems would be interesting. Would it still lower the number of collision tests done. Does the extra dimension have some cost that are too high when approximating the movements. The algorithms used to generate approximations in two dimensions is the convex hull and the Minkowski sum. Both the convex hull and the Minkowski sum are solvable in three dimensions within reasonable running time. The only factor could be that the number of generated facets from the approximation would be to great. But it should be possible to apply the connection suggested here in this thesis to three dimensions as well.

Appendix A

Program

A.1 Parameters

A.1.1 run

There are different run types of the program, meaning the program can print out data to fig format, generate road maps and show different features used in the complete algorithm. The syntax for running a certain run type is run=parameter, where the parameters are shown below.

parameter	description
roadmap	starts the construction, followed by an expansion
construction	starts the construction of a road map
localplanner	starts a local planner
approximation	stats an approximation
printenlarge	prints an ε enlargement of an object
printobject	prints an object
printobjenl	prints both the object and the related ε enlargement.
printprm	print a probabilistic road map
printcollisionstests	print the collision test count
printconfigurations	print the configurations
printworkspace	print the workspace

A.1.2 print

When printing some data from the program the wanted information can be set, an example can be that one might want to print obstacles and edges only. The syntax is print=parameter and the parameters are shown below.

parameter	description
obstacles	obstacles print is activated
border	border print is activated
edges	edges print is activated
configurations	configurations print is activated
originalmovement	original movement of the object is activated

A.1.3 numof

When generating a PRM the number of configurations generated or the maximum number of failures etc. can be set by using the following syntax numof=parameter=number. The different parameters are shown below.

parameter	description
configurations	number of maximum configurations
failures	number of maximum failures
closestsconnect	maximum number of attempts to connect configurations
minsteps	minimum steps in the RBW
maxsteps	maximum steps in the RBW
searchdepth	search depth in the Approximation LP

A.1.4 localplanner

To control what LP strategy is used to connect configurations the localplanner parameter can be set. This is done by the following syntax localplanner=parameter. The different LP types are the parameters shown below.

parameter	description
original	use the original LP
interpolation	use the interpolating LP
approximate	use the approximating LP

A.1.5 loading data files

There are several parameters needed to load data files into the program. To load in a data of a given type use the following syntax parameter=location and below are the parameters shown.

parameter	description
object	loads an object file
configurations	loads a configurations file
workspace	loads a workspace file
prm	loads a probabilistic road map

Appendix B

Objects Used

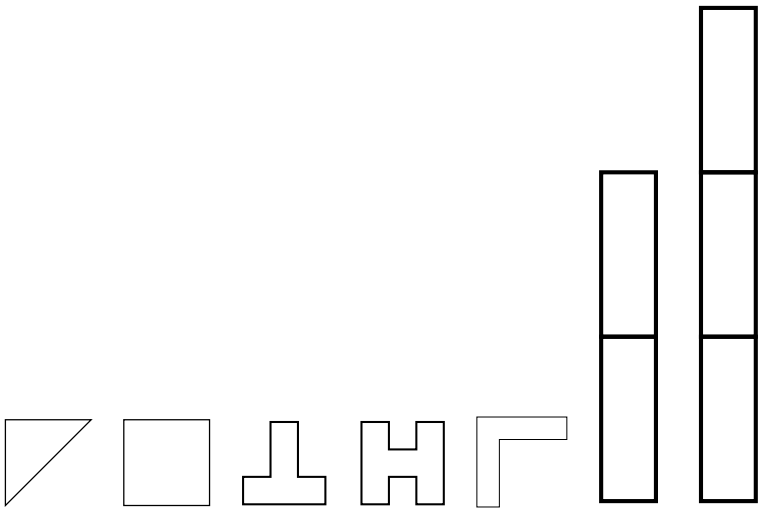


Figure B.1: The seven objects used in the program, the objects are a triangle, square, T structure, H structure, V structure, and two linked polygons. The linked polygons are two sticks and three sticks, respectively.

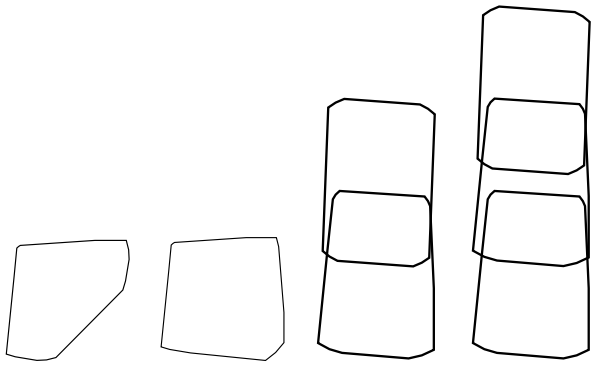


Figure B.2: The objects used in the program that are ε enlarged. The T, H and V are non-convex figures and there are no ε-enlargement for these figures.

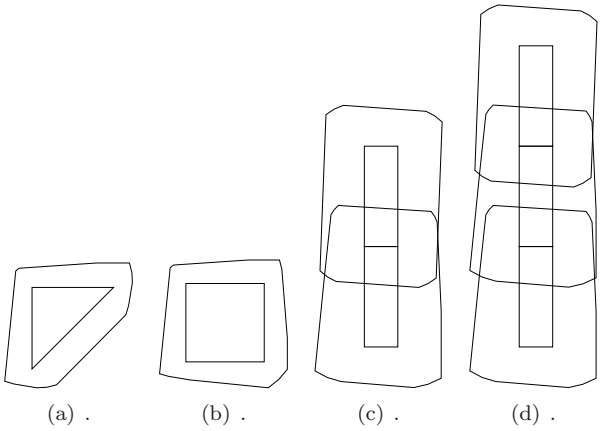
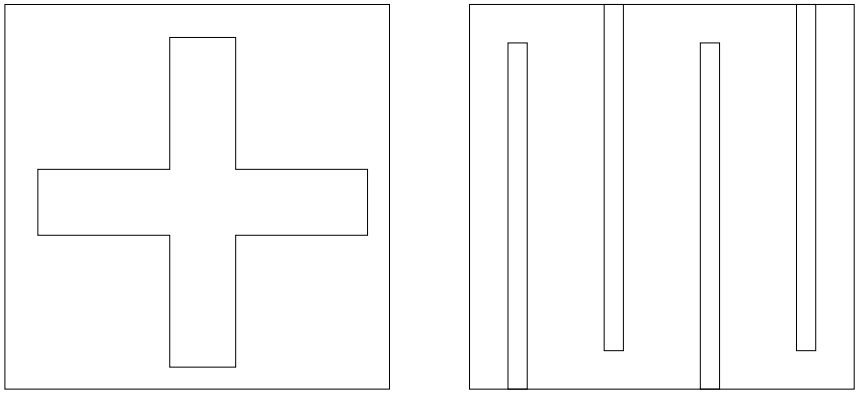


Figure B.3: The objects used in the program that are ε enlarged, where the original object is within the ε enlargement. The T, H and V are non-convex figures and there are no ε-enlargement for these figures.

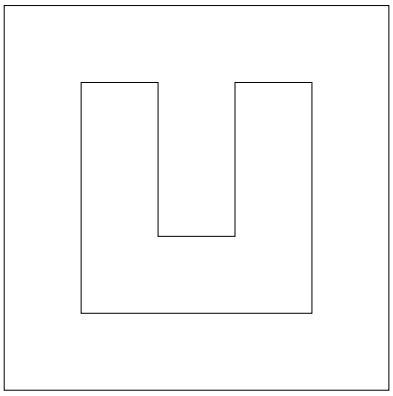
Appendix C

Work Spaces Used



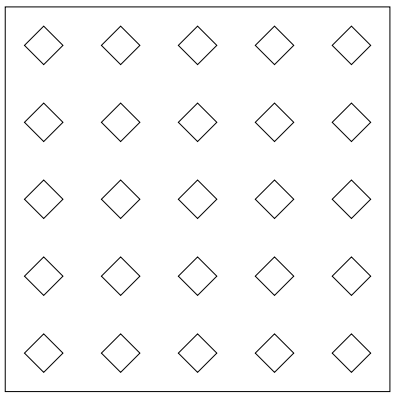
(a) The cross workspace.

(b) The labyrinth workspace.

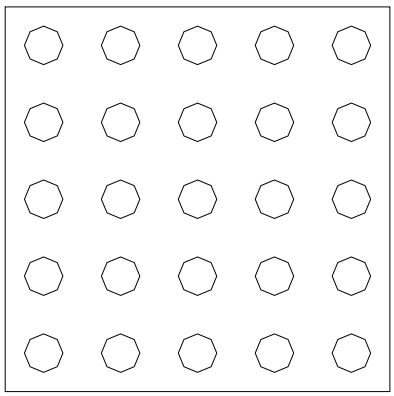


(c) The corridors workspace.

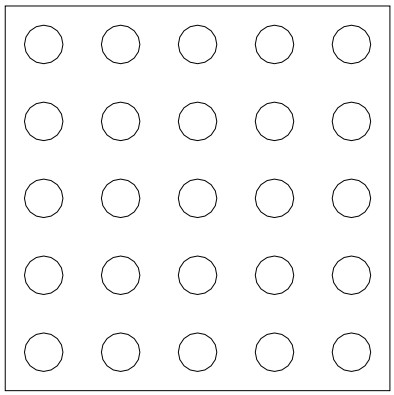
Figure C.1: Three workspaces used to test the motion planning algorithm. There is no dump of the empty workspace as it does not have any obstacles.



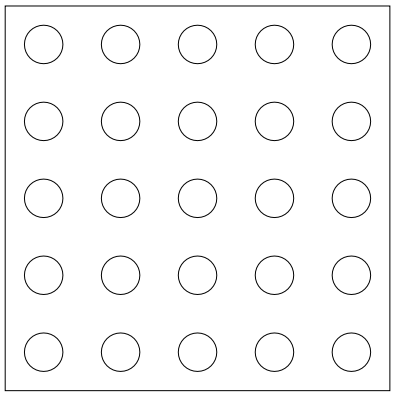
(a) The circles workspace with a resolution of four.



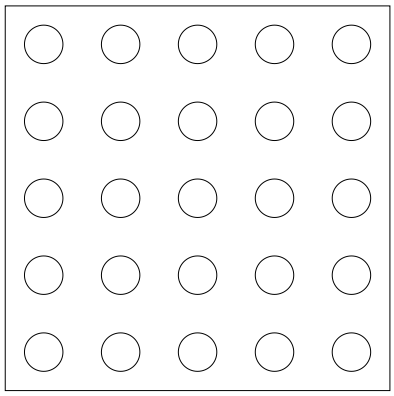
(b) The circles workspace with a resolution of eight.



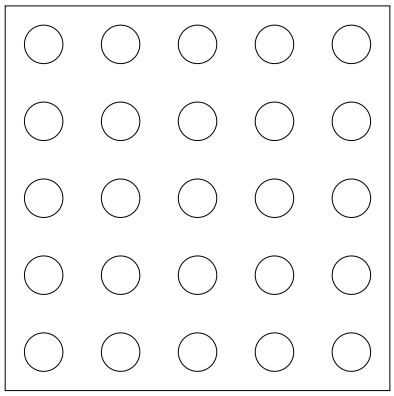
(c) The circles workspace with a resolution of 16.



(d) The circles workspace with a resolution of 32.



(e) The circles workspace with a resolution of 64.



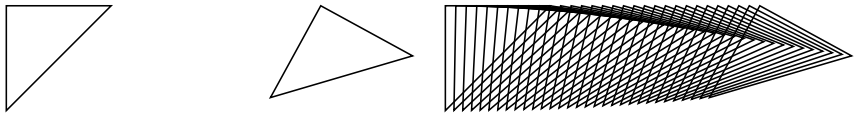
(f) The circles workspace with a resolution of 128.

Figure C.2: The six cirles work spaces used to test the number of collisions done by different local planner strategies. They are with a resolution of 4, 8, 16, 32, 64 and 128.

Appendix D

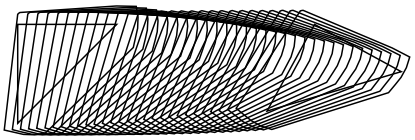
Local Planner Figures

D.1 Triangle

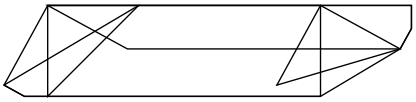


(a) The from- and to-configuration of the triangle.

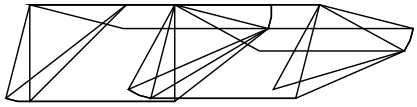
(b) The real movement of the triangle.



(c) The ε enlarged triangle movement.

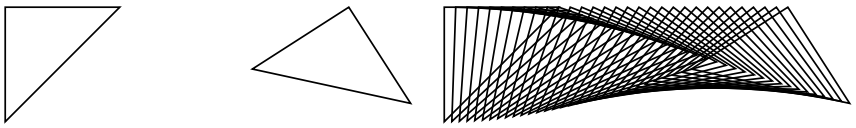


(d) The approximation of the movement.



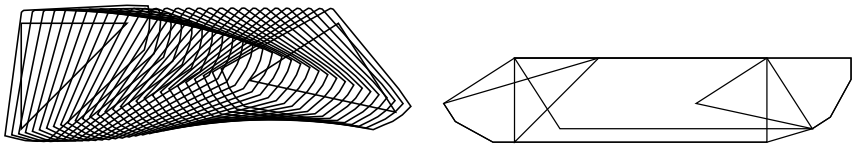
(e) The approximation of the movement splitted into two.

Figure D.1: The three different local planners connection strategies for a triangle between two configurations the translation is (150,0) and the rotation is 0.5.



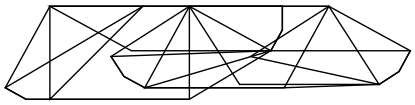
(a) The from- and to-configuration of the triangle.

(b) The real movement of the triangle.



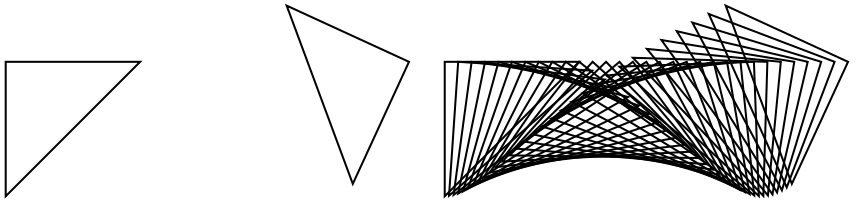
(c) The ε enlarged triangle movement.

(d) The approximation of the movement.



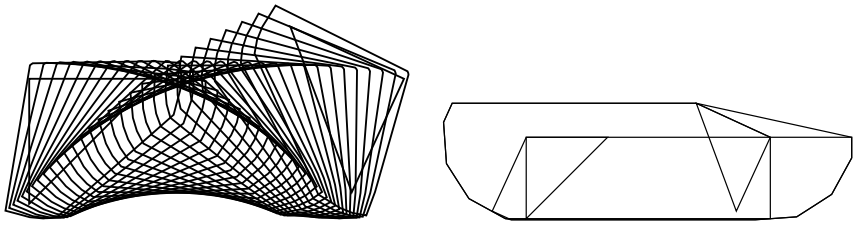
(e) The approximation of the movement splitted into two.

Figure D.2: The three different local planners connection strategies for a triangle between two configurations the translation is (150,0) and the rotation is 1.0.



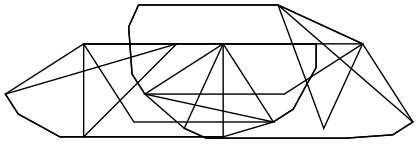
(a) The from- and to-configuration of the triangle.

(b) The real movement of the triangle.



(c) The ε enlarged triangle movement.

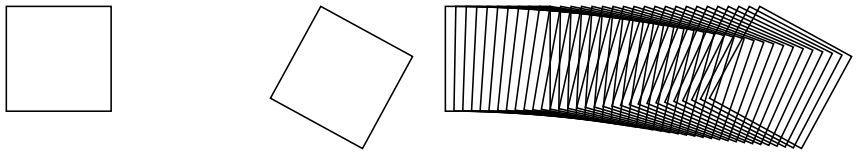
(d) The approximation of the movement.



(e) The approximation of the movement splitted into two.

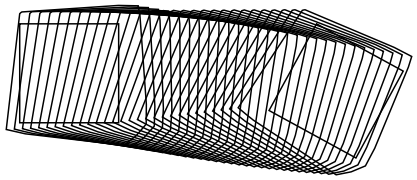
Figure D.3: The three different local planners connection strategies for a triangle between two configurations the translation is (150,0) and the rotation is 2.

D.2 Square

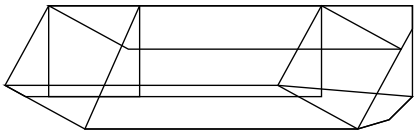


(a) The from- and to-configuration of the square.

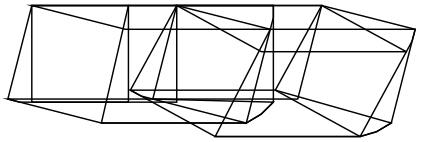
(b) The real movement of the square.



(c) The ε enlarged square movement.

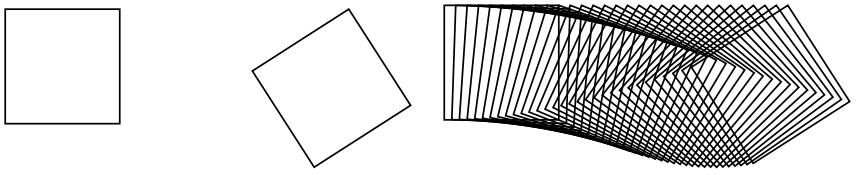


(d) The approximation of the movement.



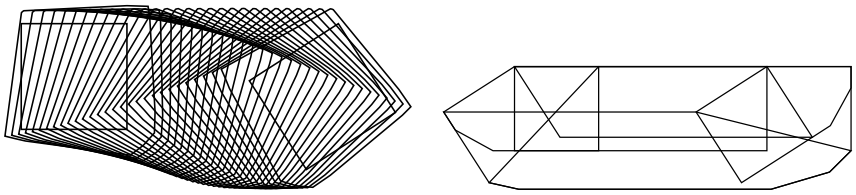
(e) The approximation of the movement splitted into two.

Figure D.4: The three different local planners connection strategies for a square between two configurations the translation is (150,0) and the rotation is 0.5.



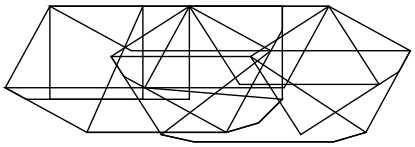
(a) The from- and to-configuration of the square.

(b) The real movement of the square.



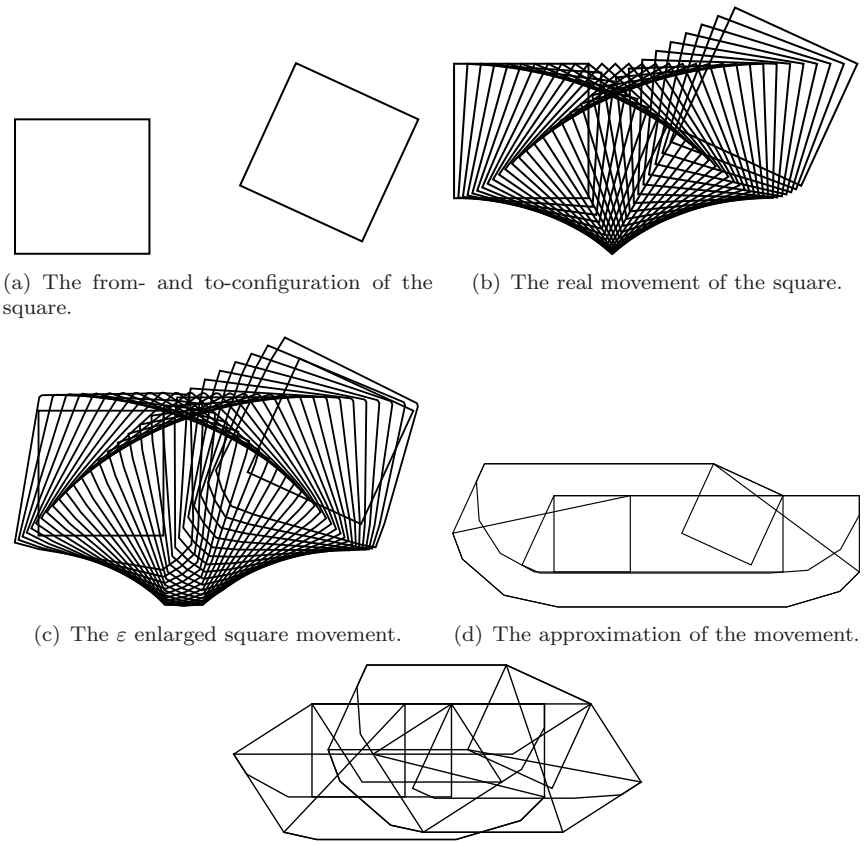
(c) The ε enlarged square movement.

(d) The approximation of the movement.



(e) The approximation of the movement splitted into two.

Figure D.5: The three different local planners connection strategies for a square between two configurations the translation is (150,0) and the rotation is 1.



(a) The from- and to-configuration of the square.

(b) The real movement of the square.

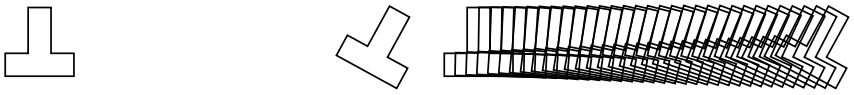
(c) The ε enlarged square movement.

(d) The approximation of the movement.

(e) The approximation of the movement splitted into two.

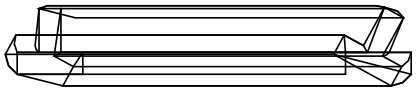
Figure D.6: The three different local planners connection strategies for a square between two configurations the translation is (150,0) and the rotation is 2.

D.3 T

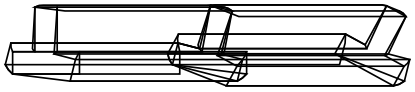


(a) The from- and to-configuration of the T.

(b) The real movement of a T structure.

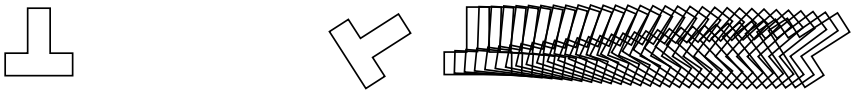


(c) The from- and to-configuration of the T.



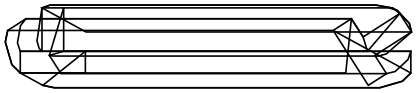
(d) The approximation of the movement splitted into two.

Figure D.7: The two different local planners connection strategies for a T structure between two configurations the translation is (150,0,0.5) and the rotation is 0.5.



(a) The from- and to-configuration of the T.

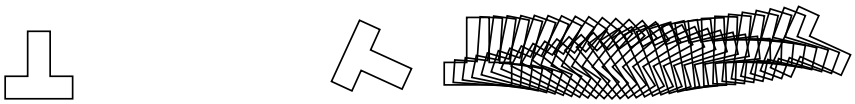
(b) The real movement of a T structure.



(c) The approximation of the movement.

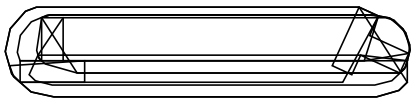
(d) The approximation of the movement splitted into two.

Figure D.8: The two different local planners connection strategies for a T structure between two configurations the translation is (150,0) and the rotation is 1.

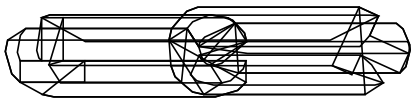


(a) The from- and to-configuration of the T.

(b) The real movement of a T structure.



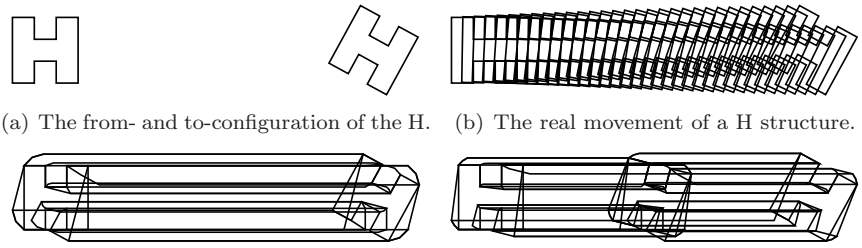
(c) The approximation of the movement.



(d) The approximation of the movement splitted into two.

Figure D.9: The two different local planners connection strategies for a T structure between two configurations the translation is (150,0) and the rotation is 2.

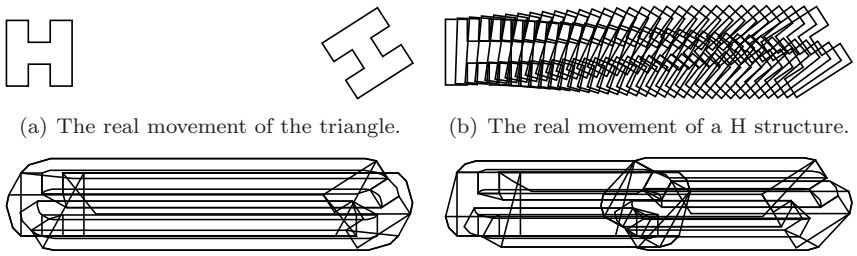
D.4 H



(a) The from- and to-configuration of the H. (b) The real movement of a H structure.

(c) The approximation of the movement. (d) The approximation of the movement splitted into two.

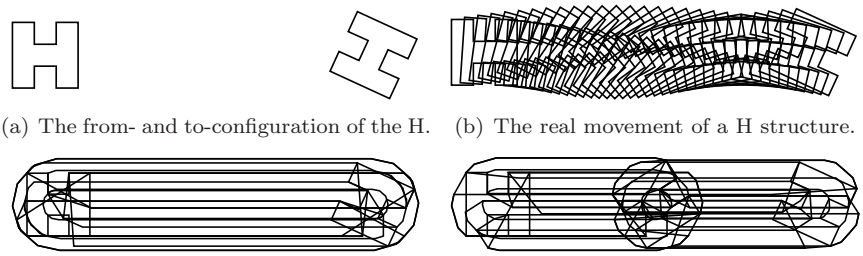
Figure D.10: The two different local planners connection strategies for a H structure between two configurations the translation is (150,0) and the rotation is 0.5.



(a) The real movement of the triangle. (b) The real movement of a H structure.

(c) The approximation of the movement. (d) The approximation of the movement splitted into two.

Figure D.11: The two different local planners connection strategies for a H structure between two configurations the translation is (150,0) and the rotation is 1.

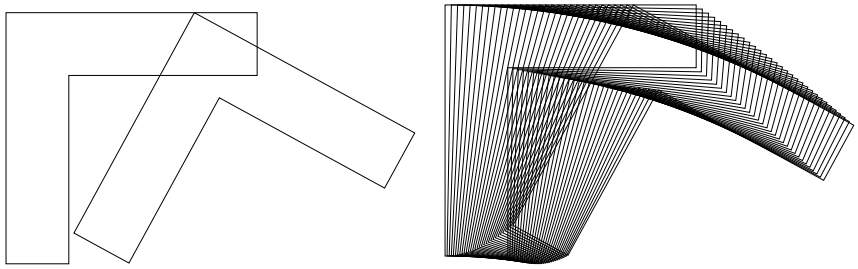


(a) The from- and to-configuration of the H. (b) The real movement of a H structure.

(c) The approximation of the movement. (d) The approximation of the movement splitted into two.

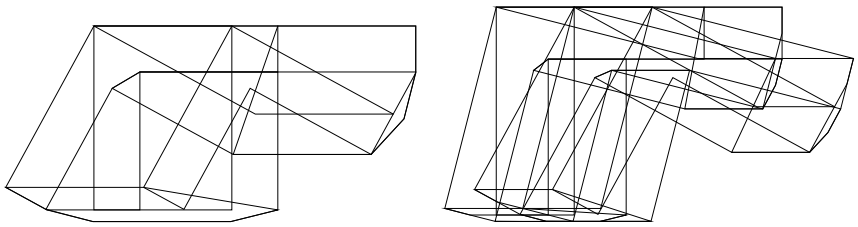
Figure D.12: Shows the three different local planners connection strategies for a H structure between two configurations the translation is (150,0) and the rotation is 2.

D.5 V



(a) The from- and to-configuration of the V.

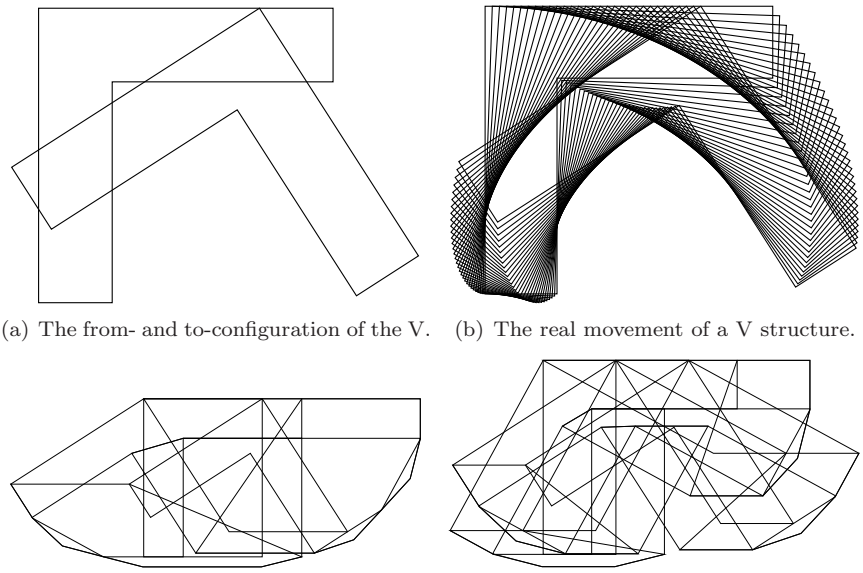
(b) The real movement of a V structure.



(c) The approximation of the movement.

(d) The approximation of the movement splitted into two.

Figure D.13: The two different local planners connection strategies for a V structure between two configurations the translation is (150,0) and the rotation is 0.5.



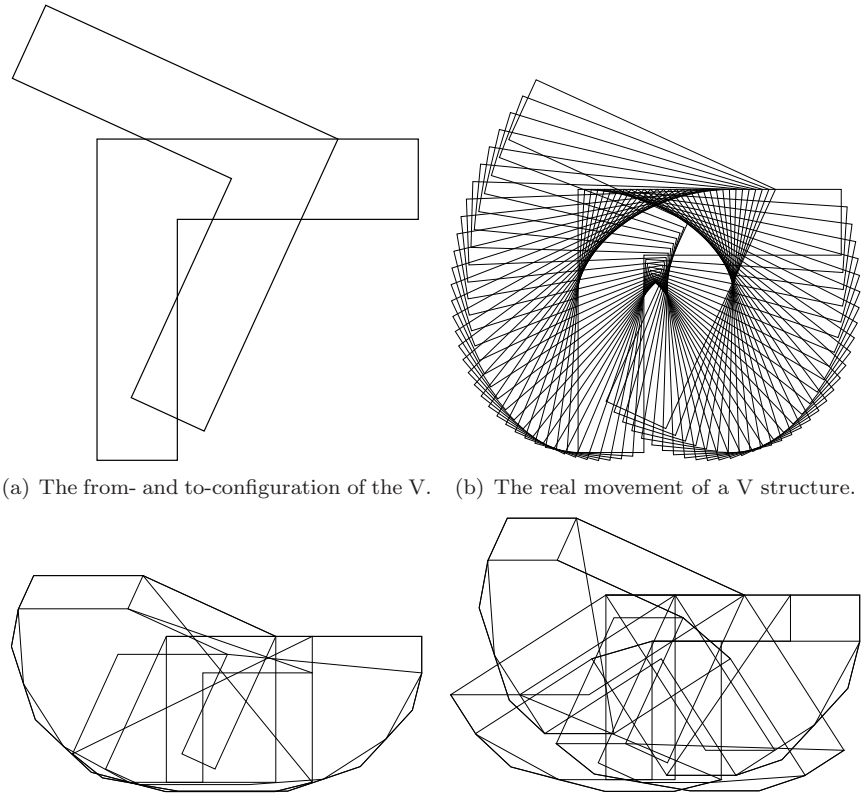
(a) The from- and to-configuration of the V.

(b) The real movement of a V structure.

(c) The approximation of the movement.

(d) The approximation of the movement splitted into two.

Figure D.14: Shows the three different local planners connection strategies for a V structure between two configurations the translation is (150,0) and the rotation is 1.



(a) The from- and to-configuration of the V.

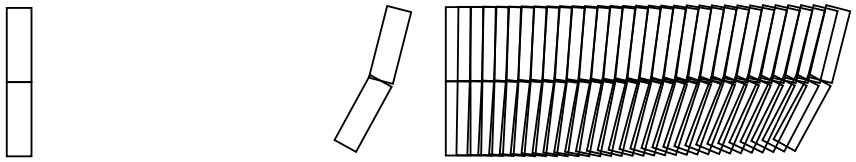
(b) The real movement of a V structure.

(c) The approximation of the movement.

(d) The approximation of the movement splitted into two.

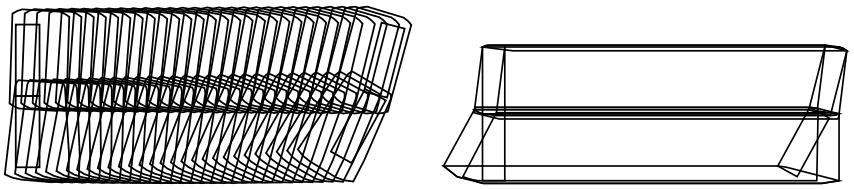
Figure D.15: Shows the three different local planners connection strategies for a V structure between two configurations the translation is (150,0) and the rotation is 2.

D.6 Two Small Sticks



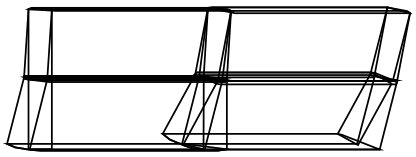
(a) The from- and to-configuration of the two small sticks.

(b) The real movement of linked object that consists of two sticks.



(c) The ε enlarged linked object of two sticks movement.

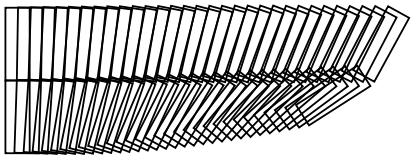
(d) The approximation of the movement of the two sticks.



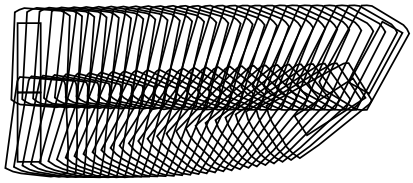
(e) The approximation of the movement splitted into two.

Figure D.16: The three different local planners connection strategies for a linked polygon between two configurations the translation is (150,0) and the rotation is 0.25 for each orientation. The linked polygon consists of two sticks.

(a) The from- and to-configuration of the two small sticks.

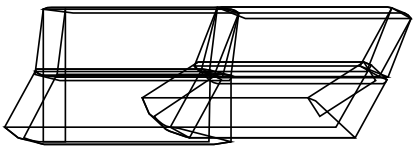


(b) The real movement of linked object that consists of two sticks.



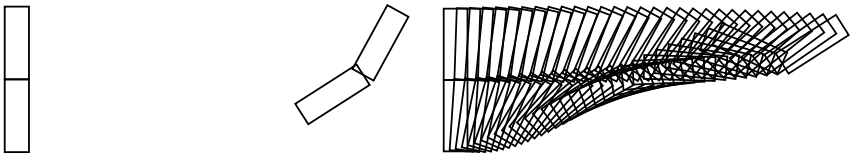
(c) The ε enlarged linked object of two sticks movement.

(d) The approximation of the movement of the two sticks.



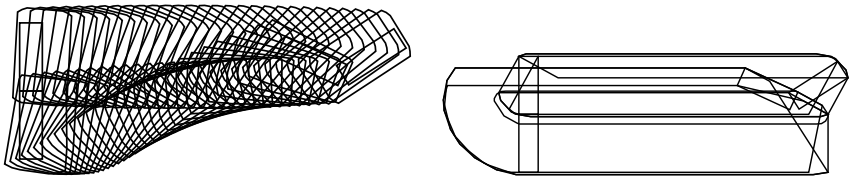
(e) The approximation of the movement splitted into two.

Figure D.17: The three different local planners connection strategies for a linked polygon between two configurations the translation is (150,0) and the rotation is 0.5 for each orientation. The linked polygon consists of two sticks.



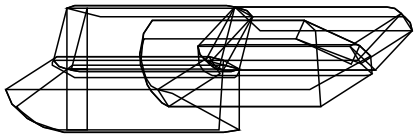
(a) The from- and to-configuration of the two small sticks.

(b) The real movement of linked object that consists of two sticks.



(c) The ε enlarged linked object of two sticks movement.

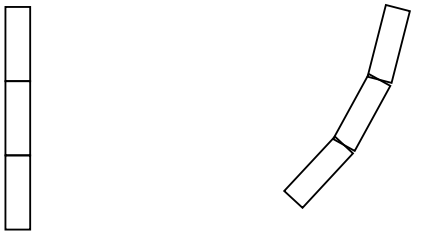
(d) The approximation of the movement of the two sticks.



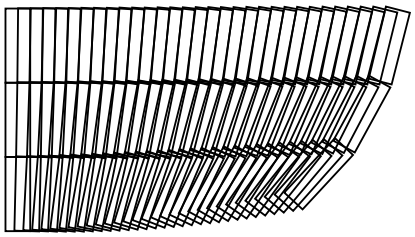
(e) The approximation of the movement splitted into two.

Figure D.18: The three different local planners connection strategies for a linked polygon between two configurations the translation is (150,0) and the rotation is 1 for each orientation. The linked polygon consists of two sticks.

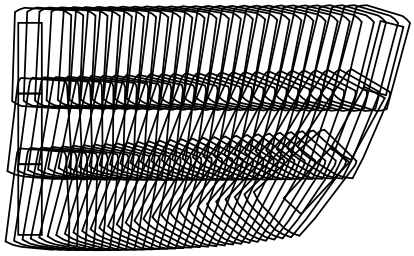
D.7 Three Small Sticks



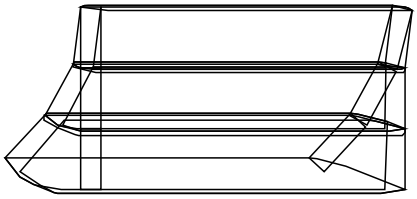
(a) The from- and to-configuration of the three small sticks.



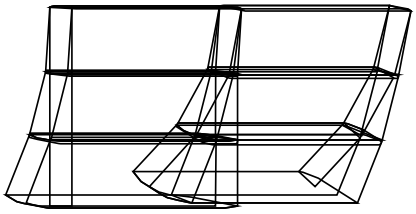
(b) The real movement of linked object that consists of three sticks.



(c) The ε enlarged linked object of three sticks movement.

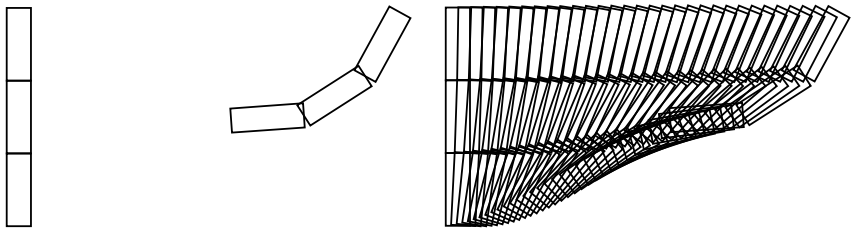


(d) The approximation of the movement of the three sticks.



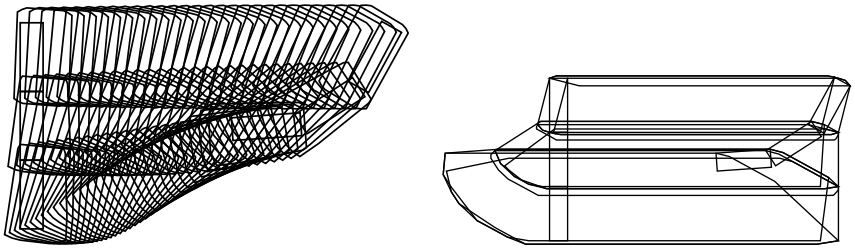
(e) The approximation of the movement splitted into two.

Figure D.19: The three different local planners connection strategies for a linked polygon between two configurations the translation is (150,0) and the rotation is 0.25 for each orientation. The linked polygon consists of three sticks.



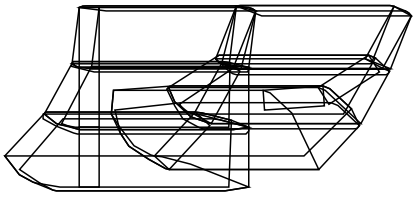
(a) The from- and to-configuration of the three small sticks.

(b) The real movement of linked object that consists of three sticks.



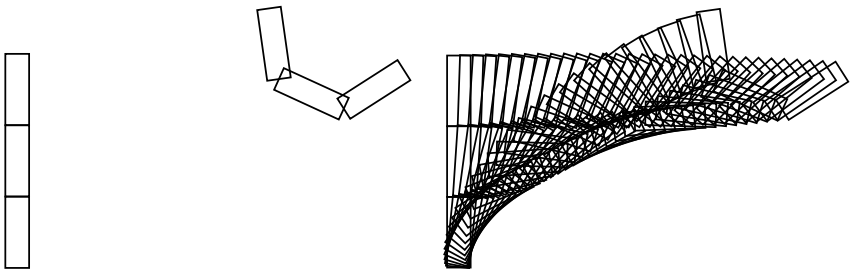
(c) The ε enlarged linked object of three sticks movement.

(d) The approximation of the movement of the three sticks.



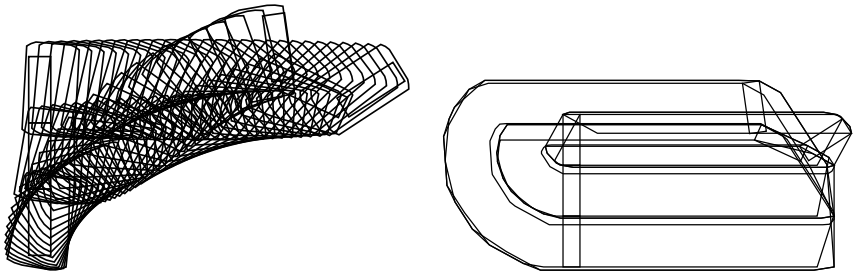
(e) The approximation of the movement splitted into two.

Figure D.20: The three different local planners connection strategies for a linked polygon between two configurations the translation is (150,0) and the rotation is 0.5 for each orientation. The linked polygon consists of three sticks.



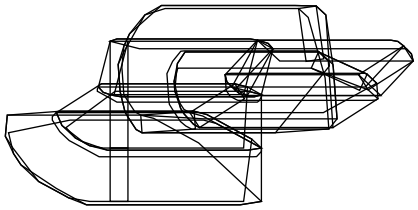
(a) The from- and to-configuration of the three small sticks.

(b) The real movement of linked object that consists of three sticks.



(c) The ε enlarged linked object of three sticks movement.

(d) The approximation of the movement of the three sticks.

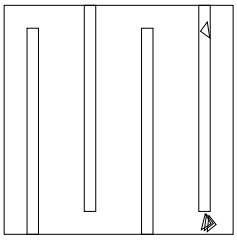


(e) The approximation of the movement splitted into two.

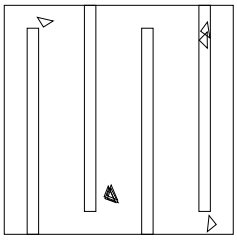
Figure D.21: The three different local planners connection strategies for a linked polygon between two configurations the translation is (150,0) and the rotation is 1 for each orientation. The linked polygon consists of three sticks.

Appendix E

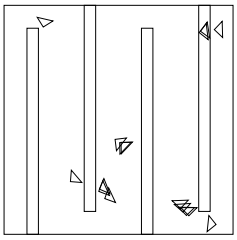
Road Maps



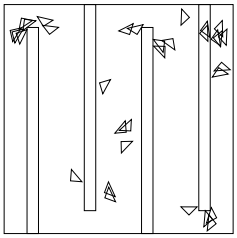
(a) Two configurations.



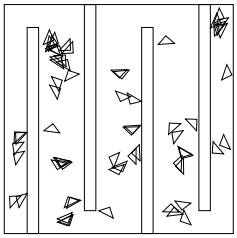
(b) Four configurations.



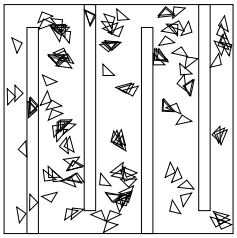
(c) Eight configurations.



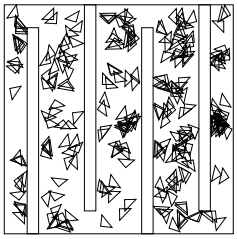
(d) 16 configurations.



(e) 32 configurations.

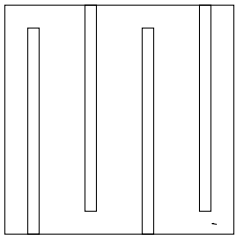


(f) 64 configurations.

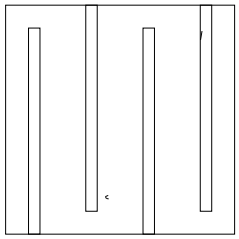


(g) 128 configurations.

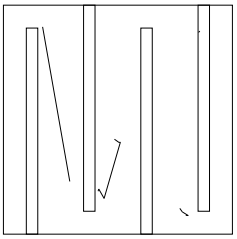
Figure E.1: Placements of a triangle for each configuration in the road map. Figure E.1(a), Figure E.1(b), Figure E.1(c), Figure E.1(d), Figure E.1(e), Figure E.1(f) and Figure E.1(g) is the road map generated with the number of configurations parameter set to 2, 4, 8, 16, 32, 64 and 128, respectively.



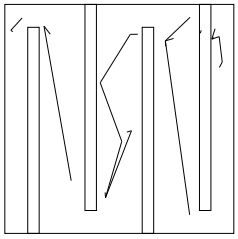
(a) Two configurations.



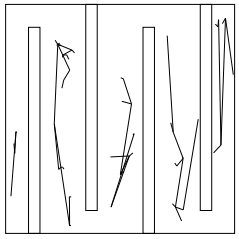
(b) Four configurations.



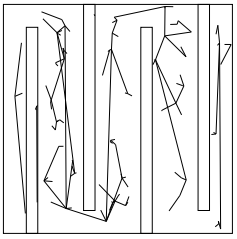
(c) Eight configurations.



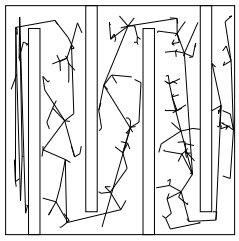
(d) 16 configurations.



(e) 32 configurations.

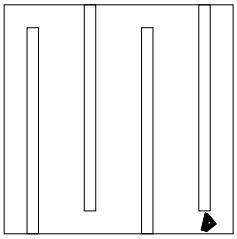


(f) 64 configurations.

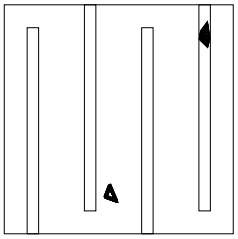


(g) 128 configurations.

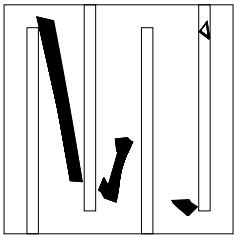
Figure E.2: Placements of an edge for each edge in the road map. Figure E.2(a), Figure E.2(b), Figure E.2(c), Figure E.2(d), Figure E.2(e), Figure E.2(f) and Figure E.2(g) is the road map generated with the number of configurations parameter set to 2, 4, 8, 16, 32, 64 and 128, respectively.



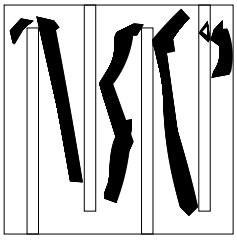
(a) Two configurations.



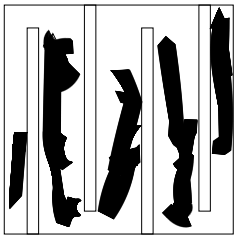
(b) Four configurations.



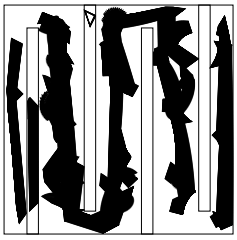
(c) Eight configurations.



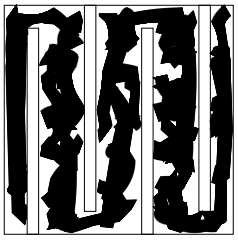
(d) 16 configurations.



(e) 32 configurations.

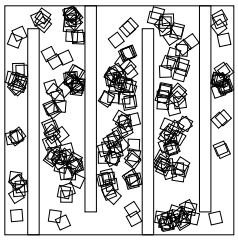


(f) 64 configurations.

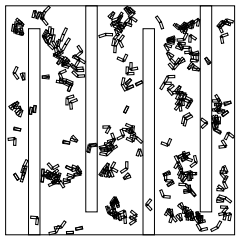


(g) 128 configurations.

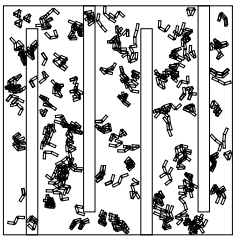
Figure E.3: Placements of a triangles to show the movement for each edge in the road map. Figure E.2(a), Figure E.2(b), Figure E.2(c), Figure E.2(d), Figure E.2(e), Figure E.2(f) and Figure E.2(g) is the road map generated with the number of configurations parameter set to 2, 4, 8, 16, 32, 64 and 128, respectively.



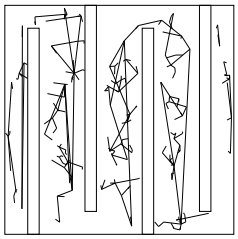
(a) Placement for 128 configurations of a square.



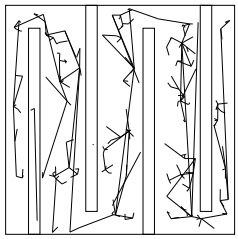
(b) Placement for 128 configurations of two small sticks.



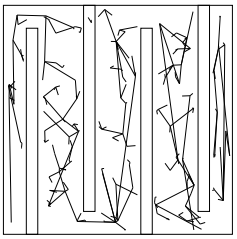
(c) Placement for 128 configurations of three small sticks.



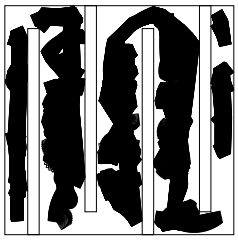
(d) Edges for 128 configurations of a square.



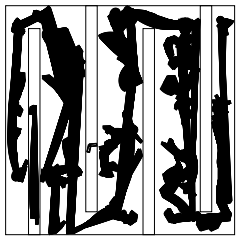
(e) Edges for 128 configurations of two small sticks.



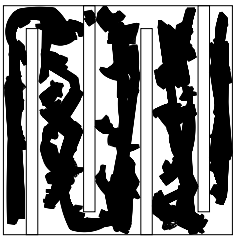
(f) Edges for 128 configurations of three small sticks.



(g) Real movement for 128 configurations of a square.



(h) Real movement for 128 configurations of two small sticks.

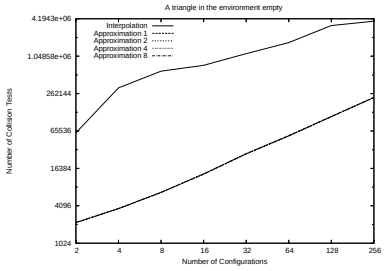


(i) Real movement for 128 configurations of three small sticks.

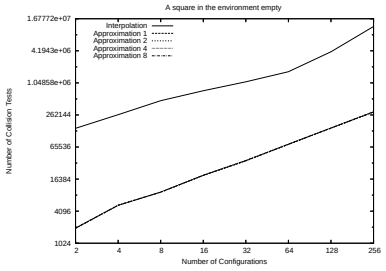
Figure E.4: .

Appendix F

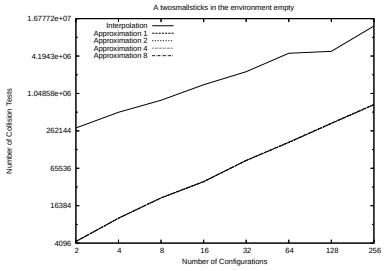
Collision Tests Graphs



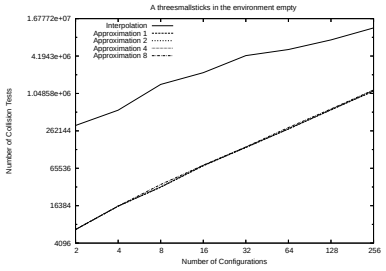
(a) Collision tests on the empty work space with a triangle.



(b) Collision tests on the empty work space with a square.

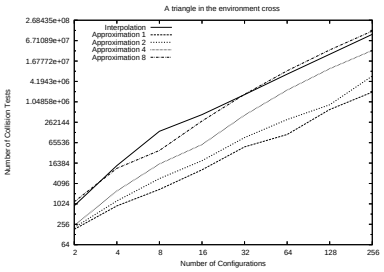


(c) Collision tests on the empty work space with two small sticks linked together.

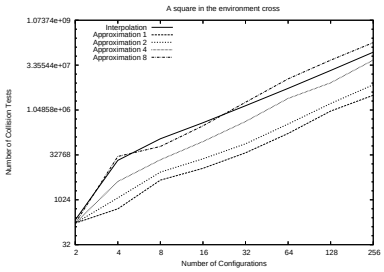


(d) Collision tests on the empty work space with three small sticks linked together.

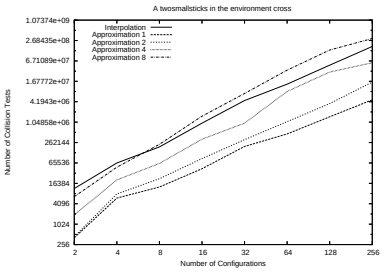
Figure F.1: Collision tests on the empty work space with the four different objects. The graphs clearly shows how the approximation lowers the number of collision tests for any object in the empty work space.



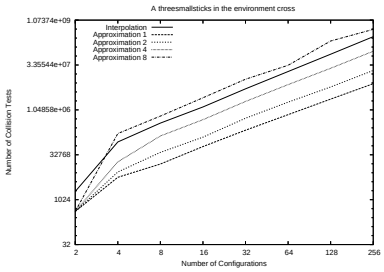
(a) Collision tests on the cross work space with a triangle.



(b) Collision tests on the cross work space with a square.

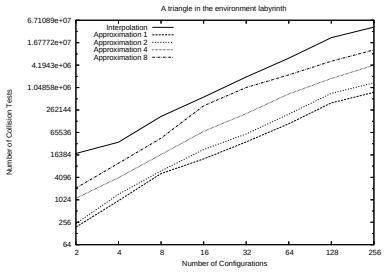


(c) Collision tests on the cross work space with two small sticks linked together.

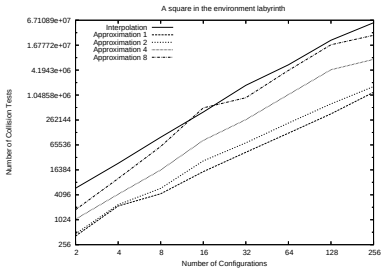


(d) Collision tests on the cross work space with three small sticks linked together.

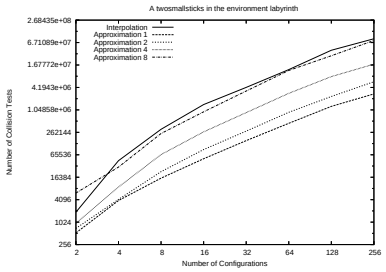
Figure F.2: Collision tests on the cross work space with the four different objects. The graphs shows how the approximation lowers the number of collision tests except for the search depth of eight.



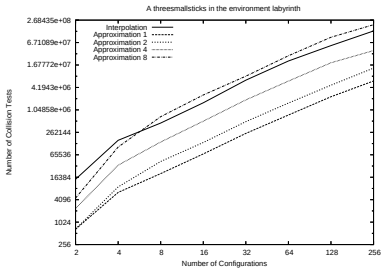
(a) Collision tests on the labyrinth work space with a triangle.



(b) Collision tests on the labyrinth work space with a square.

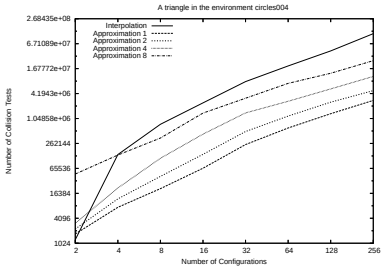


(c) Collision tests on the labyrinth work space with two small sticks linked together.

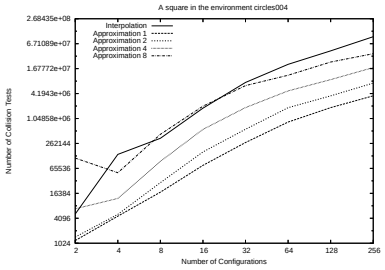


(d) Collision tests on the labyrinth work space with three small sticks linked together.

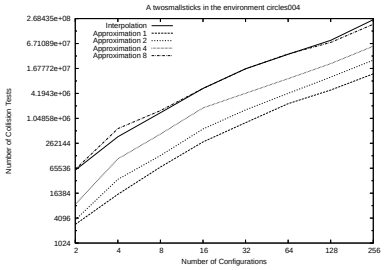
Figure F.3: Collision tests on the labyrinth work space with the four different objects. The graphs shows how the approximation lowers the number of collision tests except for the search depth of eight.



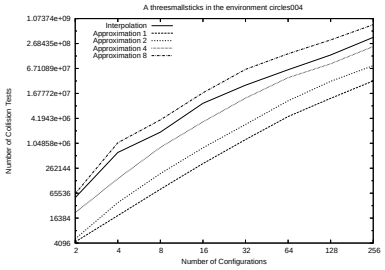
(a) Collision tests on the circles work space with a resolution of four with a triangle.



(b) Collision tests on the circles work space with a resolution of four with a square.

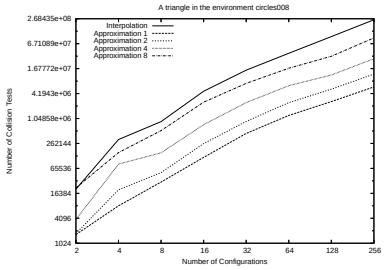


(c) Collision tests on the circles work space with a resolution of four with small sticks linked together.

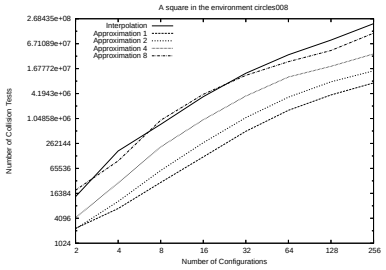


(d) Collision tests on the circles work space with a resolution of four with three small sticks linked together.

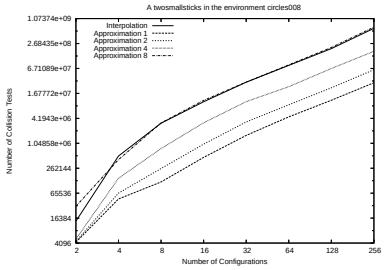
Figure F.4: Collision tests on the circles work space with a resolution of four with the four different objects. The graphs shows how the approximation lowers the number of collision tests except for the search depth of eight.



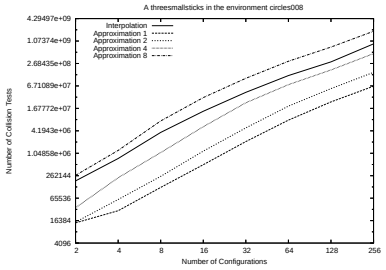
(a) Collision tests on the circles work space with a resolution of eight with a triangle.



(b) Collision tests on the circles work space with a resolution of eight with a square.

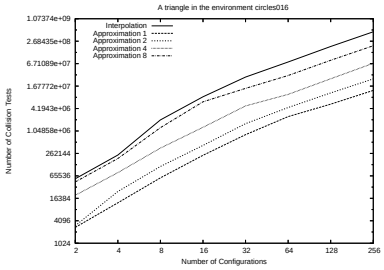


(c) Collision tests on the circles work space with a resolution of eight with small sticks linked together.

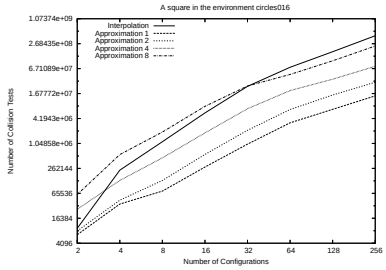


(d) Collision tests on the circles work space with a resolution of eight with three small sticks linked together.

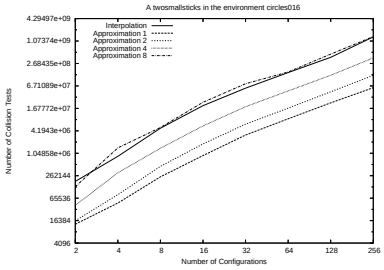
Figure F.5: Collision tests on the circles work space with a resolution of eight with the four different objects. The graphs indicates that the approximation lowers the number of collision tests except for the search depth of eight.



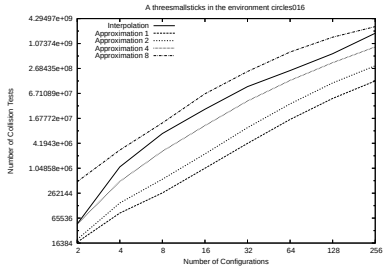
(a) Collision tests on the circles work space with a resolution of 16 with a triangle.



(b) Collision tests on the circles work space with a resolution of 16 with a square.

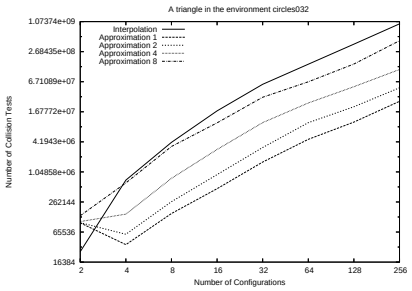


(c) Collision tests on the circles work space with a resolution of 16 with two small sticks linked together.

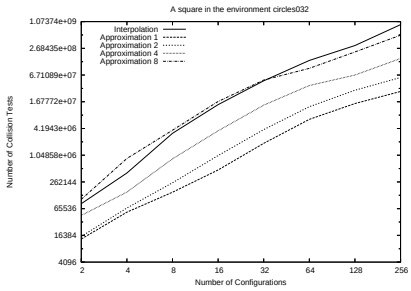


(d) Collision tests on the circles work space with a resolution of 16 with three small sticks linked together.

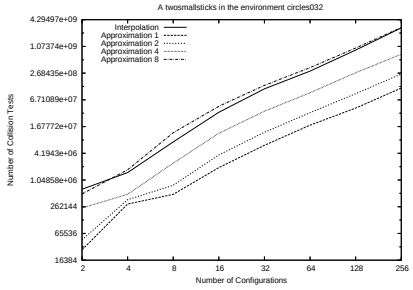
Figure F.6: Collision tests on the circles work space with a resolution of 16 with the four different objects. The graphs shows how the approximation lowers the number of collision tests except for the search depth of eight.



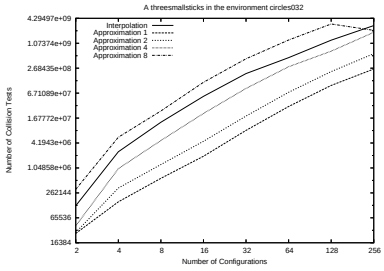
(a) Collision tests on the circles work space with a resolution of 32 with a triangle.



(b) Collision tests on the circles work space with a resolution of 32 with a square.



(c) Collision tests on the circles work space with a resolution of 32 with two small sticks linked together.



(d) Collision tests on the circles work space with a resolution of 32 with three small sticks linked together.

Figure F.7: Collision tests on the circles work space with a resolution of 32 with the four different objects. The graphs shows how the approximation lowers the number of collision tests except for the search depth of eight.

Appendix G

Notations

There will be a notation reference here

β and γ	A real number
α	Angle in radians.
p, q and r	Point in $\mathbb{R}$.
e	Edge in $\mathbb{R}$.
P^2	Polygon in the plane.
P	Obstacle
c	a configuration
c_{free}	free configuration
$\overline{s}$	start configuration
$\overline{g}$	goal configuration
C	set of configurations
C_{free}	set of free configurations

Appendix H

Acronyms

C-Obstacle Configuration Space Obstacle

C-Space Configuration Space

CCW Counter Clock Wise

CW Clock Wise

DAG Directed Acyclic Graph

DOF Degrees Of Freedom

GCC GNU Compiler Collection

GLUT OpenGL Utility Toolkit

RBW Random Bounce Walk

LP Local Planner

MP Motion Planning

MPV Minimum Potential Valley

OS Operating System

PRM Probabilistic Road Map

RM Road Map

STL Standard Template Library

V-Graph Visibility Graph

Bibliography

[1] *Xfig User Manual, version 3.2.4*, December 2002.

[2] J. L. Bentley and T.A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28:643–647, 1979.

[3] John Canny. *The Complexity of Robot Motion Planning*. Cambridge, 1987.

[4] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computation Geometry Algorithms and Applications*. Springer, second edition, 1998.

[5] James D. Foley, Andies van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics Principles and Practice*. Addison-Wesley, 2000.

[6] Roland Geraerts and Mark H. Overmars. A comparative study of probabilistic roadmap planners. 2002.

[7] Michael T. Goodrich and Roberto Tamassia. *Algorithm Design : Foundations, Analysis, and Internet Examples*. John Wiley & Sons, Inc., 2002.

[8] Yong Hwang and Narendra Ahuja. A potential field approach to path planning. 1992.

[9] Yong K Hwang and Narendra Ahuja. Gross motion planning - a survey. *ACM Computer Surveys*, 24(3), September 1992.

[10] Young Hwang and Narendra Ahuja. Path planning using a potential field representation. 1989.

[11] Lydia Kavraki, Petr Svestka, Jean-Cleaude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. 1994.

[12] Koichi Kondo. Motion plannig with six degrees of freedom by multistrategic bedirectional heuristic free-space enumeration. *IEEE Transactions on Robotic and Automation*, 7(3):267–277, 1991.

[13] Jean-Cleaude Latombe. c-space. Lecture slides about configuration space, April 2006. http://ai.stanford.edu/ latombe/cs26n/slides/c-space.ppt.

[14] Tomas Lozano-Perez. Spartial planning: A configuration space approach. 1980.

[15] Tomas Lozano-Perez. A simple motion-planning algorithm for general robot manipulators. *Robotics and Automation, IEEE Journal*, 1988.

[16] N. Nilsson. A mobile automaton: An application of artificial intelligence techniques. *Proc. IJCAI*, pages 509–520, 1969.

[17] Abraham Sanchez. A deterministic sampling approach to robot motion planning. 2003.

[18] Gilardo Sanchez and Jean-Claude Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. *Springer Tracts in Advanced Robotics*, 2001.

Index