KUNGL
TEKNISKA
HÖGSKOLAN

Royal Institute of Technology
Dept. of Numerical Analysis and Computer Science

Intelligent Motion Planning for a Multi-Robot System

by
Ronnie Johansson

TRITA-NA-E0133

NADA

Nada (Numerisk analys och datalogi)
KTH
100 44 Stockholm

Department of Numerical Analysis
and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, SWEDEN

Intelligent Motion Planning for a Multi-Robot System

by
Ronnie Johansson

TRITA-NA-E0133

Master's Thesis in Computer Science (20 credits)
at the School of Computer Science and Engineering,
Royal Institute of Technology year 2000
Supervisor at Nada was Dr. Henrik I. Christensen
Examiner was Prof. Jan-Olov Eklundh

Abstract

Multi-robot systems of autonomous mobile robots offer many benefits but also many challenges. This work addresses collision avoidance of robots solving continuous problems in known environments. The approach to handling collision avoidance is here to enhance a motion planning method for single-robot systems to account for auxiliary robots. A few assumptions are made to put the focus of the work on path planning, rather than on localization.

A method, based on exact cell decomposition and extended with a few rules, was developed and its consistency was proven. The method is divided into two steps: path planning, which is off-line, and path monitoring, which is on-line. This work also introduces the notion of path obstacle, an essential tool for this kind of path planning with many robots.

Furthermore, an implementation was performed on a system of omni-directional robots and tested in simulations and experiments. The implementation practices centralized control, by letting an additional computer handle the motion planning, to relieve the robots of strenuous computations.

A few drawbacks with the method are stressed, and the characteristics of problems that the method is suitable for are presented.

Intelligent rörelseplanering för flera robotar

Sammanfattning

System med flera autonoma, mobila robotar har många fördelar, men de ger även upphov till en del besvärliga problem. Det här examensarbetet inriktar sig på att få robotar, som arbetar med fortlöpande problem i kända arbetsutrymmen, att undvika att kollidera. Här undviks kollisioner genom att utöka en befintlig rörelseplaneringsmetod för en ensam robot till att kunna hantera flera robotar. Några antaganden är gjorda för att lägga arbetets tonvikt på vägplanering snarare än på lokalisering.

En metod, baserad på ”exact cell decomposition” och utökad med några regler, har utvecklats och dess korrekthet har visats. Metoden kan delas in i två steg: dels vägplanering, vilket sker innan roboten rör på sig, dels vägövervakning, vilket sker under robotens rörelse. Begreppet ”path obstacle” presenteras också, vilket är ett viktigt verktyg vid den här typen av vägplanering för flera robotar.

Metoden implementerades för att användas i ett system med fysiska robotar och implementationen testades både i simuleringar och robotexperiment. Central kontroll utövas av implementationen, genom att låta en särskild dator hantera vägplaneringen, för att avlasta robotarna.

Några nackdelar med metoden belyses och faktorer som karaktäriserar problem som metoden lämpar sig för presenteras.

Preface

Project Information

This thesis concerns the Master's project in Computer Science of Ronnie Johansson, student at the *School of Computer Science and Engineering*, *The Royal Institute of Technology* (KTH). The project was approved and monitored by staff of the *Department of Numerical Analysis and Computer Science* (NADA), KTH. The work was performed at *The Institute of Physical and Chemical Research* (RIKEN) in Saitama prefecture, Japan, and supervised by Dr. Igor Paromtchik, *Advanced Engineering Center*, RIKEN, and Dr. Henrik Christensen, *Centre for Autonomous Systems*, KTH. The Master's project was performed in the years 2000 and 2001.

Thesis Structure

This thesis is divided into five chapters and three appendices. Two of the appendices offer clarifications to some of the material in the chapters.

Acknowledgments

My first thanks go to Prof. **Funakubo** of *Shibaura Institute of Technology*, not only for his initial work with finding me an accepting laboratory, but also for his struggle to find sponsors for me. My contact with Prof. Funakubo was effectively maintained by Ms. **Ann Hellner**, employee at the Department for External Relations, KTH.

Prof. **Endo** and Prof. **Asama** of *The Institute of Physical and Chemical Research* (RIKEN) kindly accepted me and Prof. **Eklundh** (KTH) helped me with my application to RIKEN.

Without the help of my sponsors, I had probably never even made it to Japan. *The Sweden-Japan foundation* in Sweden generously offered me financial support for everyday and travel expenses. My second sponsor is the *Precision Measurement Technique Promoting foundation* in Japan, which kindly supported me with my apartment rent.

My supervisors, Dr. **Paromtchik** and Dr. **Christensen**, are thanked for their scientific support. Dr. Paromtchik particularly for sharing his experience and knowl-

edge of scientific work and for many comments and suggestions.

In my RIKEN laboratory a few people should be mentioned: Dr. **Suzuki** for never hesitating to answer my many questions, for repairing and preparing robots for my experiments, and for assisting me on weekends and sometimes even late nights. I thank Dr. **Kurabayashi** for offering scientific advice and fruitful discussions at several occasions.

My practical tasks were often made simple by the accurate and swift help from my secretary, Ms. **Takahashi**.

Other people who have helped me and brightened my stay in Japan are Mrs. **Uchiyama**, Mrs. **Okada**, Dr. **Takamatsu**, Dr. **Hong**, Dr. **Tsujimura**, Dr. **Odaka**, Dr. **Kawabata**, Dr. **Yamamoto**, Mr. **Kaetsu**, Mr. **Murakami**, Mr. **Uehara**, Mr. **Okina**, Mr. **Akamatsu**, Mr. **Kaneda**, Mr. **Sugimoto**, Mr. **Noda**, Mr. **Fujimoto**, Mr. **Motohashi**, and Mr. **Aoyama**.

I would finally like to mention some people who supported me with encouragements and friendship during my stay in Japan. I would like to thank **Alexander Kirchner**, **Mircea Giurgiu**, **Peter Ghoroghchian**, and **Atsusa Kanno** for being good friends in Japan; **Mattias Ånstrand** and **Torbjörn Bäck** for sharing their previous experiences of performing Master's projects in Japan; **Svante Hellstadius**, **Patrik Rundström**, and **Sofie Carlsson** for their many suggestions and advice. Last but not least, I would like to thank members of my family: my sister, my parents, my aunts, and my cousin Viveca, who were all very supportive despite the distance.

Contents

Chapter 1

Introduction

In this chapter the field of mobile robotics is outlined and the problem addressed in this thesis is presented. In Section 1.1 the work of the thesis is situated. In Section 1.2 and subsections the problem is described. The introduction is ended in Section 1.3 with an overview of the rest of the thesis.

1.1 Research Context

The research field of this Master's project, intelligent motion planning of a multi-robot system, aims at the development of methods which provide coherent actions of mobile robots in systems with more than one mobile robot. This issue has attracted much attention due to the advantages which can be obtained with a multi-robot system compared to a single-robot system (e.g., [Parker, 1995]).

This section introduces mobile robots and especially multi-robot systems. Advantages and challenges connected to these issues are presented and relevant terms defined.

1.1.1 Mobile Robots

Robot manipulators (first and foremost the popular stationary robot arms) work fine for instance in assembly applications in factories. However, mobile robots offer some very important advantages, for instance

Reach Mobile robots are necessary if the problem the robot should solve is not restricted to some sufficiently small area.

Flexibility If the position of the problem to be solved is not static, the mobile robot has the ability to pursue it.

Mobile robots can roughly be classified into two groups depending on the balance between their number of controllable *degrees of freedom* (DoFs)[1] and actual

[1] A degree of freedom is normally associated with an *actuator*. An actuator is typically an electric motor or hydraulic or pneumatic cylinder. Hence, the number of degrees of freedom of a robot is

degrees of freedom. The two groups are called *holonomic* and *non-holonomic* respectively. A strict definition in [Norvig, Russell, 1995] says that non-holonomic robots are robots which have fewer controllable DoFs than actual DoFs. A car for instance is non-holonomic because it has three actual degrees of freedom, two because it can move in two directions in the plane, and one for its direction (see Figure 1.1). A car only has two controllable DoFs, though. It can move in the direction it points, or it can make a turn. For an example of work with non-holonomic robots see [Laugier et al, 1998].

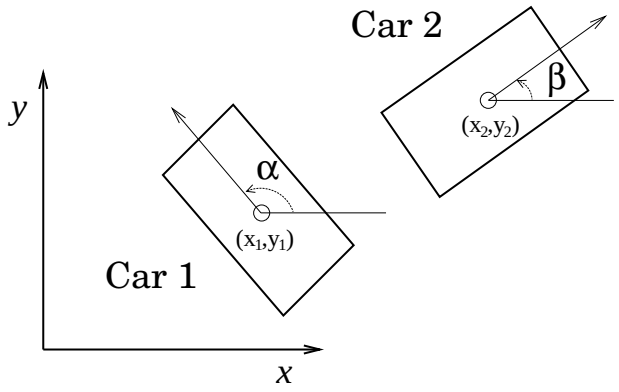


Figure 1.1. A car has three DoFs, two for its coordinates and one for its orientation.

A non-holonomic robot is normally more difficult to control than a holonomic one. In the case of a holonomic robot, it has the same number of controllable DoFs as it has actual DoFs. The *omni-directional* robot (meaning that it can move in any direction in the plane and rotate on the spot) at RIKEN [Asama et al., 1995] is an example of a holonomic robot (Figure 1.2).

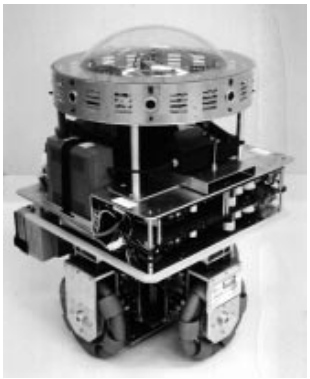


Figure 1.2. An omni-directional robot (With courtesy of the *Advanced Engineering Center* at RIKEN).

normally equal to its number of actuators [Norvig, Russell, 1995].

Motion Planning

The process of generating a sequence of actions that has to be performed in order for a robot to move through its *environment* (also called *workspace*, see Figure 1.3) autonomously and without collisions is called *motion planning*. Even though only mobile robots are discussed in this thesis, similar motion planning methods can also be used for manipulator robots.

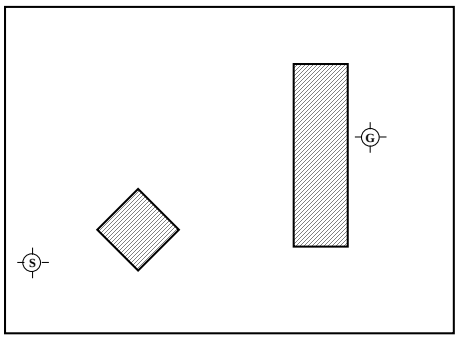


Figure 1.3. A simple workspace with two obstacles and an enclosing workspace boundary. The symbols *S* and *G* denote start position and destination respectively for a robot.

Methods for motion planning, in their most simple form, fall into one of two categories, *deliberative* or *reactive*, depending on the kind of control they exhibit[2].

Characteristics of the two categories, as presented in [Arkin, 1998], are summarized in Table 1.1.

Table 1.1. Characteristics of deliberative and reactive planning

Deliberative	**Reactive**
Dependent on representation	Representation-free
Slow response	Real-time response
High-level intelligence	Low-level intelligence

Pure reactive methods offer real-time motion response provided by simple computations and do not use any representation of its workspace (i.e., it has no understanding of the layout of the workspace, e.g., where obstacles are situated). Pure deliberative methods, on the other hand, are highly representation dependent and express a higher level of intelligence.

The *potential field* method with only local knowledge[3] is an example of a reactive method. It generates a virtual motion force for the robot by adding repulsive

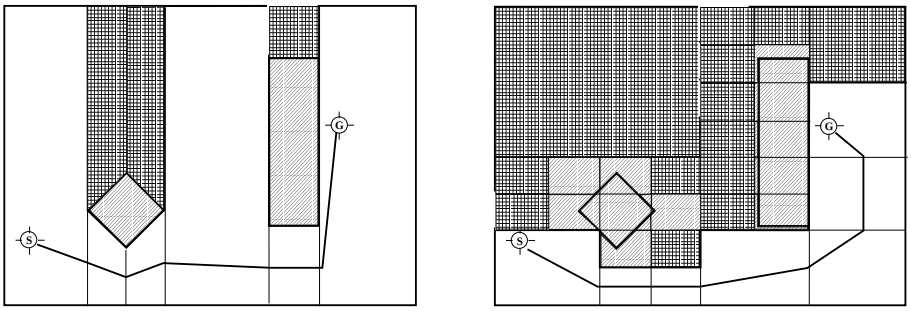
[2]Note however that it is possible to combine methods from both categories into *hybrid* methods.

[3]Meaning that it has not global knowledge about the workspace, rather is only aware of its immediate surroundings.

forces from obstacles with an attractive one from the destination of the robot.

Unlike reactive methods, deliberative ones use a priori knowledge to pre-calculate robot motion. A typical way to utilize this knowledge is to plan a path through the workspace for the robot, so called *path planning*. There are three main approaches of deliberative methods:

Cell decomposition The *free space*[4] of the workspace is decomposed into a set of cells. The cells must be simple so that a path easily can be planned through each cell (a suitable cell is typically a convex polygon). A *channel* of free cells (i.e., a sequence of contiguous cells) is subsequently constructed starting with the cell which contains the current position of the robot and ending with the cell that contains its destination. Finally, a path can be planned from the start position through the channel to the destination. Cell decomposition is further divided into *exact* and *approximate cell decomposition*. The main differences between them are that approximate cell decomposition is normally easier to implement, but is more coarse and can not always guarantee to find a free path even if one exists. Figure 1.4(a) shows how an exact cell decomposition method (in this case trapezoidal cell decomposition) may work with the workspace in Figure 1.3. Figure 1.4(b) shows what the result of an approximate cell decomposition method (in this case quadtree) may be.

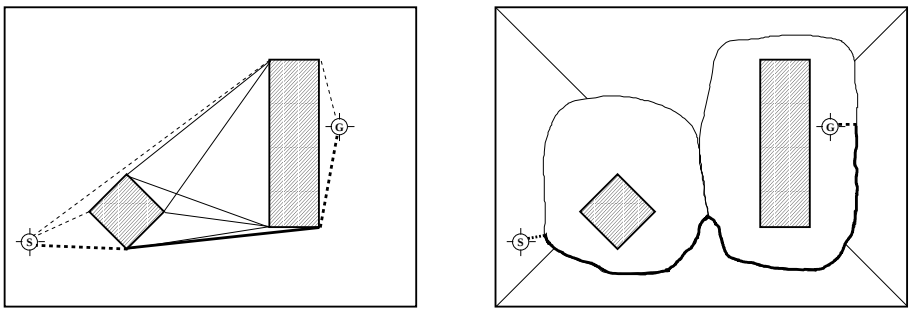


(a) (b)

Figure 1.4. (a) The free space of the workspace has been decomposed into a set of cells using the trapezoidal cell decomposition method. The white cells indicate a channel for a robot starting in position *S* with its destination in *G*. Finally, a path through the channel is planned. **(b)** The approximate cell decomposition method called quadtree has simplified the decomposition of the free space by discretizing the obstacles. The white cells constitute a channel in which a path for the robot subsequently is planned.

[4] Space which a robot can occupy, typically space which is not already occupied by some obstacle.

Roadmaps There are many different roadmap methods, but one thing they all have in common is that they try to convert the free space of the workspace into a graph representation (a roadmap). A collision-free path can now be constructed (if one exists) by connecting the start position and destination to the roadmap. The roadmap method called *visibility graph* constructs a shortest path, but it is only *semi-free*[5]. The method called *voronoi diagram* on the other hand maximizes the distance between robot and obstacles. Examples of how the workspace in Figure 1.3 is treated by these methods is shown in Figure 1.5(a) and Figure 1.5(b).



(a) (b)

Figure 1.5. (a) In the visibility graph method, all pairs of vertices of the obstacles in the workspace are connected. A connection between a pair of vertices is called an edge and all edges form a possible path segment on an optimal path. In the figure the thick path segment and the dotted thick lines from *S* to *G* constitute a path. **(b)** In the voronoi diagram method, a path is constructed by connecting *S* and *G* with the roadmap, which consists of positions that are on a maximum distance from the obstacles and workspace boundary.

Potential field The notion of potential field may also be used in a deliberative way. A potential field may be calculated for the whole workspace using knowledge about obstacles and destination. The field determines the path for the robot from its current position to its destination.

Please notice that all of these methods, in their most fundamental form, only are applicable to a point robot (i.e., a robot with the shape of a point). Extra measures are required to handle robots of other shapes. A tool, called configuration space, that makes path planning for robots with a more complex shape possible, is presented later.

The theory and ideas concerning cell decomposition and roadmap methods are thoroughly explained in [Latombe, 1993]. These methods are well established, but

[5]Let p_r be a semi-free path for a robot r, then r might "touch" obstacles as it travels along p_r

limited in the sense that they are only applicable to *static environments* (where obstacles do not move). Environments which have moving obstacles are called *dynamic* (motion planning in dynamic environments is discussed in [Fujimura, 1991]).

Sometimes deliberative and reactive methods are combined to achieve motion plans that both have an optimal, coarse path and that is insensitive to uncertainties in the environment (e.g., unknown obstacles or erroneous workspace representation).

An alternative way of classifying motion planning methods is to say whether they are *on-line* or *off-line*[6]. On-line planning is performed in real-time, i.e., at the same time the robot is moving, and is exceptionally useful when the environment is not known. Off-line planning is performed before any robot motion and is not useful unless the workspace is known. While reactive methods are inherently on-line, deliberative methods may be classified as either on-line or off-line, depending on the need for a rapid response.

A comparison between on-line and off-line planning, originally from [Arai, Ota, 1992], can be found in Table 1.2.

Table 1.2. Characteristics of off- and on-line planning

Off-line	**On-line**
Sensitive to differences between the representation of the workspace (which is used during planning) and the world itself.	Can compensate for deviations between the knowledge of the world and the world itself.
Needs only small computation power (and only once).	Needs relatively large computation power to achieve real-time computation
The whole workspace can be considered.	The planning may be local, and there is therefore a risk that robots get stuck in a *local minimum*.

Configuration space

An interesting tool which can be used for motion planning is *configuration space*. When using this tool, planning is not performed in the physical space of the environment, rather in the possible configurations of the robot. The configuration space

[6] In the same way as with deliberative/reactive methods it is possible to combine on-line and off-line methods.

of a robot r is denoted C_r (or just C when there is no ambiguity) and has as many dimensions as the robot has DoFs. The concept of configuration space is probably best illustrated by a simple manipulator robot as in Figure 1.6(a). The figure shows a workspace with an obstacle and a manipulator robot r with two joints and two links. Figure 1.6(b) shows the corresponding configuration space of r. Configuration space can also be used with mobile robots, an omni-directional robot for instance has three DoFs (it can translate in two directions and rotate[7]).

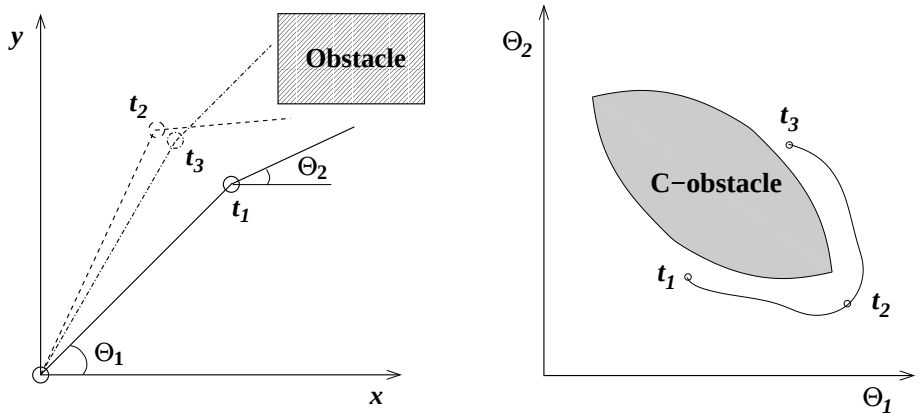


Figure 1.6. (a) A configuration q of the manipulator robot arm r, which can move in the x-y-plane, is defined by the angles, θ_1 and θ_2, of the two links. An obstacle occupies a part of the workspace, space which may not be shared with the arm. The configurations of the arm are shown at three moments, at t_1 with the start configuration, at t_2 with an intermediate configuration, and ultimately at t_3 with its final configuration. **(b)** In the configuration space, r is represented as a point. The obstacle in (a) is represented as a region of forbidden configurations (C-obstacle) of r. The figure shows the configurations of r at the moments t_1, t_2, t_3 and a possible "path" through them which corresponds to a collision-free movement of r in the workspace.

As suggested in Figure 1.6(b), planning in configuration space is performed for a point (the configuration q of a robot) instead of a, perhaps very complex, robot shape. This makes it a useful tool for all of the path planning methods previously explained (which in their most simple form assume that the robot has the shape of a point). The main problems however, as pointed out in [Gill, Zomaya, 1998] is the difficult mapping from Cartesian space (i.e., the spatial space or workspace) to configuration space and the complexity of handling it (since it often has many more dimensions than the physical space).

[7] If collision of a robot is not dependent on its rotation, the orientation dimension may be left out. Hence, in this case $C \subset \mathbf{R}^2$ instead of $C \subset \mathbf{R}^3$.

The motion planning approaches presented here are summarized in Table 1.3. The table uses the terms *sound* and *complete*. A method is said to be complete if it guarantees to yield a solution (i.e., a collision-free path) if one exits and sound if it guarantees that all its solutions are correct (i.e., collision-free).

Table 1.3. Summary of motion planning methods (Type: R = reactive, D = deliberative)

Method	Type	Advantage	Disadvantage
Potential field (local)	R	Real-time	Not complete
Exact cell decomposition	D	Complete	Heavy computations
Appr. cell decomposition	D	Sound and useful when only a coarse representation (e.g., a digital image) of workspace is available	Not complete
Visibility graph	D	Complete and yields minimum length path	Generates semi-free paths
Voronoi diagram	D	Complete and generates roadmap with maximum distance to obstacles	Possibly inefficient resulting path

Previous work

Over the years, many different motion planning problems for a single robot have been studied. Some researchers have developed more efficient methods for motion planning in static environments (for instance [Habib, Asama, 1991]), while others have addressed variations of the problem. In [Choset, Pignon, 1998] the authors modify the general cell decomposition method for a coverage application, in which the robot must visit all the free space (which is useful for e.g., vacuum cleaning or inspection).

The authors of [Kant, Zucker, 1988] approach the problem of collision avoidance for a single robot in dynamic environments with complete knowledge of the

environment, including the trajectories[8] of all moving obstacles. The authors suggest that geometric algorithms can be used in conjunction with a local avoidance strategy. To accomplish this they first subdivide the collision avoidance into two steps by using complete knowledge of the environment:

1. the path of the robot is planned to avoid collisions with static obstacles,

2. the velocity along this path is planned to avoid collisions with obstacles moving across it.

The two steps are performed off-line and result in a coarse trajectory for the robot. To account for uncertainties (e.g., unknown obstacles), they suggest an on-line strategy based on sensor data.

The authors of [Tsoularis, Kambhampati, 1999] use acceleration and deceleration to avoid collisions along a path planned off-line. However, due to for instance the hardware limitations of a robot, it might not always be possible for it to accelerate or decelerate in the required way to avoid collisions, so as a last resort they allow it to deviate from its path.

In [Fiorini, Shiller, 1995], the concept of *velocity obstacle*, which is the mapping of the dynamics of the robot workspace into the robot velocity space, is utilized. This approach allows any number of moving obstacles, it unifies the avoidance of both moving and static obstacles, and it also allows robot velocity constraints to be considered.

In [Ghosray, Yen, 1996], an approximate cell decomposition method, called quadtree, for collision avoidance between a single robot and obstacles is modified. It also lets a single quadtree represent the set of obstacles in the workspace. Each time an obstacle changes its position in the workspace the quadtrees have to be updated accordingly. The authors claim that their algorithm is especially well suited for problems with frequently rotating obstacles and robot.

1.1.2 Localization

It is crucial for a robot to know its position in the workspace in order for it to execute its tasks properly, because if the robot can not determine its own position it will be hard for it to perform a useful interaction with its environment. This is especially important if a map of the workspace is used and it is critical for the operation of the robot that it knows its location on this map. The process of determining a robot's position is called *localization*. Related to localization is the concept of *navigation*. A system that can help a robot to establish its location or by some means help it to find its way through its workspace correctly is called a *navigation system*.

There are quite a few techniques available for localization. Most of them are described in [Borenstein, 1996].

[8] Trajectory means path and the velocity along it.

Odometry and *inertial navigation*, so called *relative position measurements*, are both localization techniques which can be handled by a robot itself without external help.

A robot, which is aware of its original position in space, can after some movement utilize odometry to estimate its new position. The idea of odometry is to use information already available to the robot, such as wheel rotation, to estimate its current position. However, due to for instance tire slippage and irregularities in the surface of the workspace, the position estimation of the robot will quickly deviate from its actual position.

To sum up, both of these techniques will accumulate errors in a real environment and can not be used on their own for correct localization.

A method for correcting odometry error is described in [Borenstein, 1996]. The method, called *internal position error correction* (IPEC), utilizes two mobile robots that correct each other's odometry errors. By cleverly connecting two mobile robots, which both apply the IPEC method, a single robot with considerable error correction skills is achieved. The commercial omni-directional robot *OmniMate* [Borenstein] developed at the Mobile Robotics Lab at the University of Michigan is such a robot.

External measurement techniques, so called *absolute measurement techniques*, are often used in conjunction with the relative measurement techniques to improve localization estimates. Examples of absolute measurement techniques are

- Guidepath
- Active beacons
- Artificial landmarks
- Natural landmarks

Guidepath is one of the simplest forms of robot navigation. The guidepath can for instance be a wire (which transmits audio or radio signals) or a magnetic stripe [Everett, 1995], which a robot can follow. Guidepaths indicate static paths in an environment and are not suitable in applications where mobile robots should be allowed to move flexibly.

The method of *active beacons* uses a set of transmitters (beacons), which locations in the workspace are known in advance. The transmitters, in most cases, use light or radio signals to announce their presence. For a robot to be able to use the active beacons for its localization, at some position in the workspace, at least three of the beacons have to be "visible" (detectable) at that exact position. This technique has mostly been utilized in large-scale out-door environments. One successful implementation in a small-scale environment is [Kleeman, 1992].

In the method of *artificial landmark recognition*, objects or images with a distinctive shape (for recognition to be performed easily) are placed in the workspace. The position of the landmarks are known, and if three or more landmarks are detectable at a certain position, an estimate of the position can be calculated.

The difference between artificial and *natural landmark recognition* is that the landmarks used in the latter case are not placed in the workspace for the robotics application, they exist in the workspace already. Ironically, natural landmarks are mostly man-made since suitable landmarks must appear as structured to stand out in the environment. A suitable "natural landmark" is therefore for instance the edge formed by the joint between a wall and the floor in a straight corridor.

The landmark recognition methods rely on odometry for an approximate position in the workspace. Using the approximate position, the robot can look for landmarks which it expects to find in the part of the workspace in which it believes it currently is. With the help of the landmarks, the robot is able to refine its position estimate.

1.1.3 Multi-Robot Systems

Some advantages with robot systems containing more than one robot, so called *multi-robot systems*, some of them mentioned in [Parker, 1995], are:

Task enablement The solutions to some problems are inherently parallel and may require a number of robots. Other solutions require robotic capabilities which can not all be (efficiently) implemented on a single robot.

Improved system performance Time and efficiency can be gained by using multiple robots.

Distributed sensing Sensing robots can acquire sensor data from different positions simultaneously, yielding results which a single robot can not produce (consider for instance a set of robots surrounding and investigating some object).

Fault tolerance If one robot fails it does not necessarily mean that the whole system of robots fail.

However, adding more robots to solve a task does not only yield advantages, it also introduces some new design considerations, for instance:

1. How should tasks be distributed among the robots?

2. How should shared resources be used by the robots?

The answer to the first consideration depends on the composition of the set of robots used. If the set of robots is *homogeneous*, it means that the robots have identical capabilities (in this thesis the definition is extended to include also the shape and size of the robot). If it on the other hand is *heterogeneous*, it means that the robots have different capabilities (which is necessary for some applications). Task distribution in a heterogeneous set of robots becomes more complex than in a homogeneous set since some tasks may only be suitable for some robots in the set.

A resource, in the second consideration, can for instance be a tool (which robots need to use to fulfill their tasks), communication media or space. In practice, to share space means to avoid collisions among the robots in the set. With the sharing of resources, the deadlock problem[9] also arises. Robots, which are "polite" and wait for other robots to release their resources, might have to wait forever. Cooperation between robots in multi-robot systems is desired to handle these problems.

In [Uny Cao et al., 1997], the authors make an attempt to define "cooperative mobile robotics". They define *collective behavior* to be any behavior of a set of agents, containing at least two agents. They further consider *cooperative behavior* to be a subclass of collective behavior and provide a definition: 'Given some task specified by the designer, a multiple-robot system displays cooperative behavior if, due to some underlying mechanism (i.e., the "mechanism of cooperation"), there is an increase in the total utility of the system.' A mechanism of cooperation may in this context for instance be a control architecture for the robot system, which yields a performance gain over (naive) collective behavior.

One aspect of cooperation is *control architecture*. Control architectures of a multi-robot system can be divided into two groups: *centralized* and *decentralized*. In centralized planning, the motion planning of all robots is handled by a single planner, a supervisor agent. In decentralized planning, however, each robot plans its motion individually. Some characteristics of the two groups, provided by [Arai, Ota, 1992], can be found in the Table 1.4.

The most common is the decentralized approach. Decentralized architectures are often said to have several advantages over centralized architectures, e.g., fault tolerance, reliability and scalability. However, [Uny Cao et al., 1997] claims that very little work, if any, has focused on comparing centralized to decentralized architectures, and "it is therefore not clear whether scaling properties of decentralization offset the coordinative advantage of centralized systems".

Latombe suggests two approaches to deal with motion planning in multi-robot systems ([Latombe, 1993]). One approach is to treat all robots as one in the configuration space (i.e., treating the set of robots as one robot with many degrees of freedom), but this requires that both the initial configuration and the final configuration of the complete multi-robot system is known. Another and more common approach is *decoupled planning*, in which case planning for each robot is performed in two steps: first a coarse plan for each robot is prepared (more or less independently of the other robots), which in a second step can be corrected along the way if it was not accurate enough.

In [Liu et al, 1989], a two-level planner is used to avoid collisions in a multi-robot system. The high level of the planner finds a collision-free path (if one exists) for each robot, and the lower planner coordinates the motions of the robots in intersecting parts of the planned paths. The algorithm constructs a Petri-net, based

[9] A deadlock is a situation which involves at least two robots. Each robot waits for some other robot to release a resource (e.g., a tool or space) it needs before it can continue. In the deadlock none of the robots are willing to release the resources they have and have to wait forever.

Table 1.4. Characteristics of control mechanisms

Centralized	**Decentralized**
Optimization of the motion of all robots is possible since the supervisor can take all of them into consideration at the same time.	Is inherently uncoordinated to some extent because no robot (normally) enjoys global knowledge.
The computational capacity of the supervisor must grow with the number of robots.	An increase in the number of robots does not imply any extra computational load on each of the robots.
The fact that only one agent is engaged in motion planning makes the multi-robot system vulnerable. If the supervisor cease to function properly, so will the whole system.	The motion planning of each robot will probably not cease to function, if just one robot does.

on a quadtree decomposition of the workspace, which is used by the robots for local collision avoidance (such as moving forwards/backwards or even moving away from the pre-planned path).

In [Yuta, Premvuti, 1992], a set of polite robots, in which all robots yield for each other, with decentralized control is studied. The authors claim that even detecting a deadlock is a difficult problem. In their experiment the authors design a environment with a few restrictions to make it easier to detect deadlocks:

- The free space consists of straight roads
- The width of each road is fixed so that no two robots can go pass each other on the same road
- Only three branches are allowed at an intersection in the network

Each robot broadcasts information about its status to all others. This way all robots can enjoy global knowledge, but only as long as the communication between the robots is reliable. The robots use a resource handling custom which the authors call “modest cooperation”, which means that a robot will not try and take a resource which is owned by another robot. This is especially reasonable when the resources in question are space, since trying to take such a resource would lead to a collision. If a robot wants to enter a part of the road which is already occupied it will have to stop. A robot that stops at a crossroads will check, with the help of its knowledge of the positions of the other robots, how many other robots that have stopped at this

crossroads and the direction of each of the other robots. By that means, a deadlock can be detected.

To handle the deadlock, one of the robots achieves leader status. The state of the robots involved in the deadlock changes temporarily from being fully autonomous to being either leader or follower. There is a simple technique to resolve the deadlock at the crossroads and the leader directs the followers where to go. After the deadlock is resolved all the involved robots restore their autonomous control.

The environment described is called road network and is used by many researchers (e.g., [Rausch, Levi, 1996]). It is useful because it is a common environment (e.g., in car traffic or supermarket applications) that often simplifies the problem. In systems which allow deadlocks to occur the most common solution is to let one agent in the system temporarily take charge of the motion planning even if the system under normal operations is decentralized [Qutub et al, 1997, Alami et al, 1997].

In [Noborio, Yoshioka, 1994], an on-line and decentralized path-planning algorithm for multi-robot systems is proposed. It avoids deadlocks instead of trying to handle them as they occur. The algorithm utilizes a set of rules to ensure deadlock avoidance. It, however, assumes a workspace that is infinite in all directions in the plane.

1.2 Problem Specification

Previous works highlight the usefulness of multi-robot systems. Some problems are not suitable for a single robot, and even for those who are, performance gain can be achieved by adding more robots. The difficulties with having more than one robot have been noticed, including the resource sharing problem of collision avoidance.

1.2.1 Aim

The aim of this Master's project was to "Develop a method for collision-free motion for mobile robots in a multi-robot system". The developed method should be integrated into an existing multi-robot system with omni-directional mobile robots, and target problems such as those presented in Section 1.2.2. To make the aim even more clear, a few assumptions are introduced in Section 1.2.3.

1.2.2 Problem Domain

The notion of *scenario* is introduced in order to specify which problems the method presented in this thesis can be applied to. A scenario is an abstract description of a problem and consists of two parts:

- Contents

- Interaction and dependencies

In this project, the first part involves a workspace containing some objects. The workspace is an area where mobile robots are allowed to operate and contains apart from robots also *request generators* and *forbidden areas*. Request generators, as their name suggests, generate requests whenever they are in need of service. A forbidden area is a part of the workspace in which mobile robots are not allowed to enter (entering such areas may result in damage to either the robot or some object occupying the area). Note that the definition of forbidden area is more general than that of obstacle (forbidden area also comprises, e.g., holes in the ground or areas which the robot should not enter for other reasons).

The second part of the scenario describes how the objects interact and depend on each other. In this case, requests are decomposed into tasks for mobile robots. It is assumed that mobile robots are equipped in such a way that they can provide the request generators with suitable service. Mobile robots will have to travel to request generators without entering forbidden areas, exiting the workspace or colliding with other robots. The described scenario is quite general and is realized in various “real problems”, such as:

- An assembly line with a number of assembly machines (request generators). Each machine performs its specific assembly operation. It requires simple components to assemble them. A machine must be supplied with new simple components and relieved of its assembled composites. Mobile robots can assist them with these two tasks (a similar example is displayed in Figure 1.7).

- A nuclear power plant or a chemical factory that is composed of many different machines. In this context, machines in the areas which are unsuitable for humans can be considered as request generators, and mobile robots, will perform inspection or maintenance tasks.

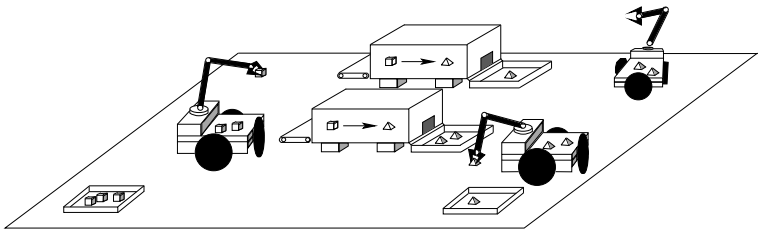


Figure 1.7. Some robots transporting components to and from manufacturing machines in a shared workspace.

Notice about these problems that they are continuous in the sense that there is no obvious end-state. Rather, the systems may run for an indefinite amount of time.

1.2.3 Assumptions

The problem presented here in Section 1.2 is indeed very general and it is not possible to include all aspects in this project. Therefore, for the theoretical work of this thesis, the following assumptions are made:

- perfect localization,
- robots are fully holonomic,
- request generators, forbidden areas and workspace boundary are static,
- the system enjoys perfect knowledge of the world,
- robots are fully equipped to handle requests,
- inter-robot communication is reliable.

Consequently, the focus of this project is on motion planning for holonomic robots sharing the same known workspace.

1.3 Thesis Overview

These are the contents of the rest of the chapters and appendices of the thesis.

Chapter 2 describes different motion planning approaches and highlights one called *exact cell decomposition*, which is described in detail. Furthermore, Chapter 2 explains how exact cell decomposition can be extended to account for other known and possibly moving robots. The vital concept of *path obstacle* is presented in order to achieve this motion planning. A motion planning method (`MPM`) is then created by adding a few rules which dictates the behavior of robots at paths' intersections. `MPM` is composed of two parts, a *path planner* and a *path monitor*.This chapter is supported by Appendix A, in which the validity of the rules is proven.

Chapter 3 explains how `MPM` was implemented on a robot system. At first some simplifications of the problem are introduced to set a reasonable level of difficulty on the implementation task and then the experiment hardware (robots and support computer) is presented. The architecture of the software developed for the hardware is presented. The software includes a path generation algorithm which is discussed thoroughly in Appendix B.

Chapter 4 presents the results of a few simulations and one experiment. The purpose of the first simulations is to show that the path planning works as intended. The final simulations show that robots on a collision course will take the necessary precautions to avoid the collision. The chapter is ended with a presentation of a real robot experiment.

Chapter 5 summarizes the work of the project. It points out deficiencies with `MPM`, but also the characteristics of the problems it is particularly suitable for. Finally, the chapter discusses future directions of work and research for `MPM` and its implementation.

Appendix A contains two proofs. The first one proves, using natural deduction, that it is sufficient for robots to obey the rules from Section 2.4 to avoid collisions. The second one is an induction proof that proves that no deadlock will be planned.

Appendix B discusses an algorithm for path planning given a channel and start and end positions. The algorithm is analyzed and its time complexity determined.

Appendix C provides a glossary of terms that are used in the thesis.

Chapter 2

Development of a Motion Planning Method

This chapter explains how a conventional path planning method for single robot-systems can be extended to account for multi-robot systems.

2.1 Review of Motion Planning Methods

Relatively few previous works concentrate on multi-robot systems in workspaces with static obstacles where the only moving objects are the robots themselves. Some of the works mentioned in Chapter 1 assume knowledge of the trajectories of all moving objects (e.g., [Kant, Zucker, 1988]). In this work such an assumption is not made since the trajectories of the robots are certainly not known, they are calculated when they are needed. There are also works that assume that the destinations of all robots are known ([Latombe, 1993]). In this work such an assumption can not be made either, since the robot destinations are not known in advance, they arise dynamically and depend on requests from request generators.

Since this work is restricted (see Section 1.2.3) to focus on workspaces that are known (rather than workspaces which have to be explored), it is natural to look for ideas among the available methods which deal with this condition for a single robot (e.g., [Latombe, 1993]). A rough subdivision of the motion planning methods is shown in Figure 2.1.

The motion planning approach called cell decomposition is quite attractive. It allows generation of collision-free paths (whereas, e.g., visibility graph only guarantees semi-free paths), it is practical (compared to, e.g., voronoi diagram which appears more difficult to implement) and it takes global knowledge into consideration (unlike local potential field).

Furthermore, the path generation of cell decomposition has a channel[1](read more about channels and cell decomposition in Section 2.2) as an intermediate

[1] A channel is simply a connected region of free space which includes both the start position and the goal position of the robot.

result if one exists. This yields at least two advantages:

1. The channel is an efficient way to answer the question if a collision-free path exists or not,

2. There are no restrictions on the generated path other than that it must be inside the channel (in contrast to voronoi diagram and visibility graph where the path is immediately generated).

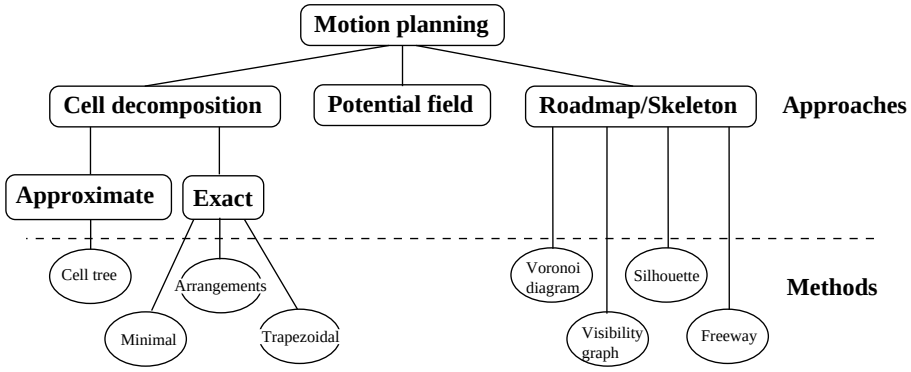


Figure 2.1. Motion planning for a single robot can roughly be divided into three main approaches: cell decomposition, potential field and roadmap. Most methods for motion planning can be derived from these approaches or hybrids of these approaches. Some of the methods are shown in this diagram.

The cell decomposition approach can further be divided into two groups: exact and approximate. The latter is more coarse as its name suggests. It is not complete, i.e., it does not guarantee to find a path if one exists, unlike exact cell decomposition, and is most suitable when the knowledge of the workspace is represented by an image (in which case the input data is already coarse to begin with). In this project it is assumed (see Section 1.2.3) that the workspace is known so there is no need for approximate cell decomposition. Hence, exact cell decomposition constitute a good foundation for a motion planning method.

In the following sections, it is explained how the ordinary exact cell decomposition works (Section 2.2), how it can be extended to be applicable to multi-robot systems (Section 2.3), and finally how it can be integrated into a complete motion planning method (Section 2.4).

2.2 Exact Cell Decomposition Explained

The aim of exact cell decomposition is to generate a collision-free path for a robot r, from its current position to its destination in a known workspace. This aim is achieved by first decomposing the free space of the workspace into non-overlapping regions (called cells) and constructing a connectivity graph which have

the cells as nodes. The next step is to search the graph for a sequence of contiguous cells, including the cells that hold the current position and destination of r, and finally generate a path through the cells. It is natural to divide this approach into these three steps:

1. decomposition of free space,
2. connectivity graph search, and
3. path generation.

Each step requires a few inputs and generate a certain output as depicted in Figure 2.2.

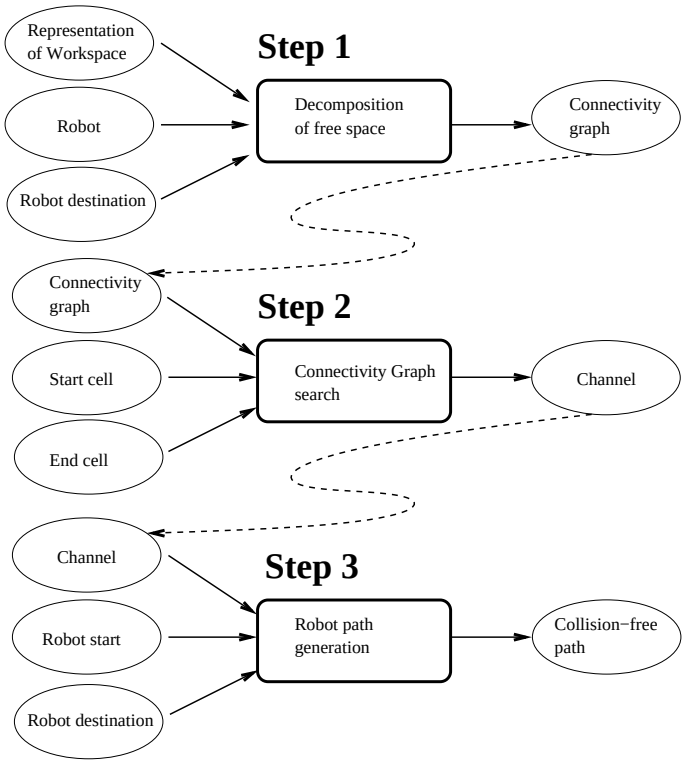


Figure 2.2. Steps of the Exact cell decomposition method

2.2.1 Decomposition of Free Space

The first consideration to deal with is how a robot should be represented. The exact cell decomposition approach is suitable for path planning for a point object. However, in most cases a robot can not be approximated with a point, rather a polygon would be much more suitable. A second consideration is how the free space should be decomposed.

Configuration Space

In [Lozano-Pérez, 1983], an idea to handle the first consideration was presented. It suggested that instead of handling a complex geometrical representation of a robot in the Euclidean representation of the workspace, the robot can be treated as a point in its configuration space (simply denoted C when there is no ambiguity).

The configuration space has as many dimensions as the robot has degrees of freedom (DoFs). This project deals with omni-directional robots. A robot of that kind normally has three DoFs, two for translation and one for rotation. E.g., a robot r with position (x,y) and angle θ (relative to some axis in the Euclidean space) corresponds to a point (x,y,θ) in its configuration space. If there are obstacles or forbidden areas in the Euclidean space, then correspondingly there are configurations (points in configuration space) which are not allowed (remember the introduction to configuration space in 1.1.1).

In the special case that the robot is symmetric in its z-axis (i.e., it is rotation invariant) the θ-coordinate may be ignored[2] and C will be two-dimensional, the same as the Euclidean workspace. In this special case the configuration space of r (denoted C_r) will resemble the workspace (denoted $\mathcal{W}$) a lot. In fact, we say that an obstacle (and forbidden area) O_w in the workspace "grows" with the size of r in C_r into a configuration space obstacle $C_r\{O_w\} = O_c$ (see Figure 2.3).

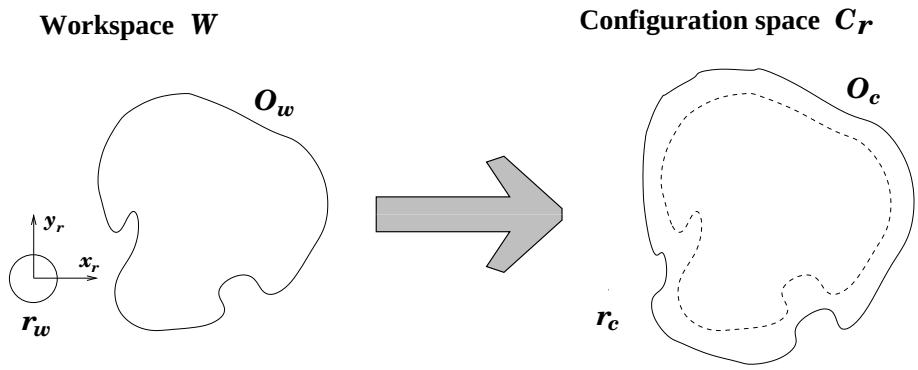


Figure 2.3. A workspace $\mathcal{W}$, containing a robot r_w and obstacle O_W, is being mapped into the configuration space C_r for r_w. r_w, which is symmetric in the z_r axis (which is perpendicular to the x_r and y_r axis), is directed out of the paper. In C_r r_c will simply be a point and O_c (approximately illustrated here) is O_w grown with the radius of r_w (O_w is drawn inside O_c for comparison). r_c touching the boundary of O_c in C_r is equivalent to r_w touching O_w in $\mathcal{W}$.

This project uses rotation invariant robots and enjoys the property that the configuration spaces for these robots are comparatively easy to generate and visualize.

[2] This fact can be expressed as
$\forall x \forall y \forall \theta_1 \forall \theta_2 (allowed\_configuration(x,y,\theta_1) \leftrightarrow allowed\_configuration(x,y,\theta_2))$
in mathematical terms.

Decomposition

The free space C_{free} of the configuration space C (depicted in Figure 2.4) is the space which is inside the C and not inside any of the configuration space obstacles $C\{O_i\}, i \in [1,k]$, for some k [3].

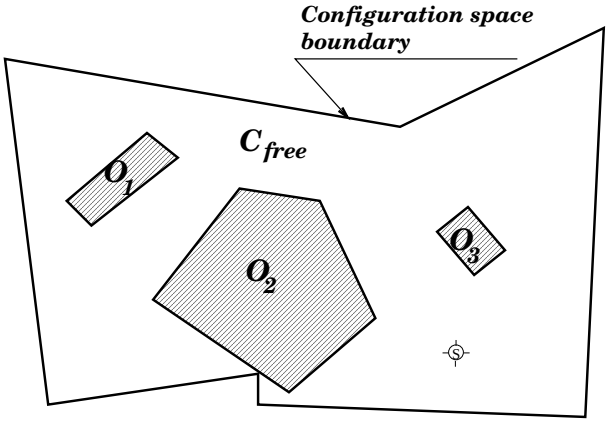


Figure 2.4. The empty region outside some configuration space obstacles O_1, O_2, and O_3 in C, but within the configuration space boundary is called free space C_{free}.

There is not one unique way to decompose C_{free}, but there are a few general rules common to all decompositions (according to [Latombe, 1993]):

1. The cells $K_i, i \in [1,n]$, of the decomposition must be non-overlapping and their union equal to C_{free}, i.e., $\bigcup_{i\in[1,n]} K_i = C_{free}$,
2. "The geometry of each cell should be 'simple' enough to make it easy to compute a path between any two configurations in the cell."
3. "It should not be difficult to test the adjacency of any two cells and to find a path crossing the portion of boundary shared by two adjacent cells."

Convex regions, which meet the requirements of Rule 2 and Rule 3, are suitable as cells and are often used. There are however various algorithms for generating convex cells. A minimal decomposition yields a minimal number of convex cells, but it was shown in [Lingas, 1982] that it is a NP-hard problem[4]. Fortunately, there are efficient non-minimal decompositions, and it is not even clear if the minimal decomposition is always the most suitable for path planning. Trapezoidal decomposition (which gives the name to the trapezoidal cell decomposition method) is one of those non-minimal decomposition algorithms; it divides the free space into

[3] Free space is defined as $C_{free} = C \setminus \bigcup_{i\in[1,k]} C\{O_i\}$

[4] NP is short for "Non-deterministic Polynomial time". A problem is in NP if you can quickly (in polynomial time) test whether a solution is correct. Some problems are at least as hard to solve as any problem in NP. Such problems are called NP-hard.

cells that are either trapezoids or triangles. A trapezoidal decomposition is constructed by sweeping a vertical line over C_{free} from, say, left to right. Whenever the line encounters a vertex of an obstacle, a vertical line is inserted at that vertex (which divides C_{free} into more and more cells). Another example of a non-minimal decomposition is decomposition of arrangements which is used in the implementation of the method (Chapter 3).

The following simple example[5] illustrates how Exact cell decomposition works. Once again, have a look at the simple configuration space C_r with a few configuration space obstacles (from now on referred to simply as "obstacles") in Figure 2.5. Path planning is going to be performed for a robot *r*, which is currently in position *S* and has the destination *G*.

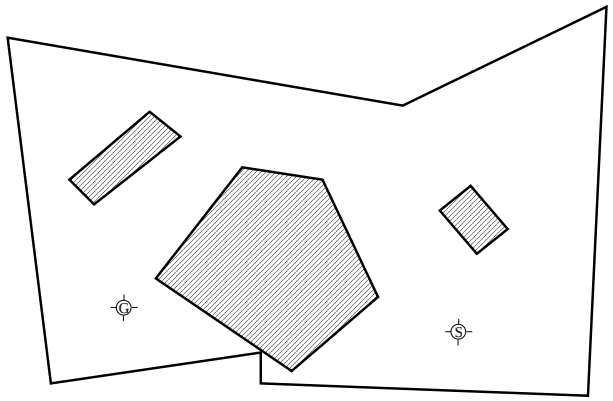


Figure 2.5. The simple configuration space includes a boundary and three obstacles. Path planning is going to be performed for a robot *r* (which is a point since path planning is performed in its configuration space C_r), which has a current position *S*, and a destination *G*.

Let us briefly return to Figure 2.2 and begin *Step 1*. In this example trapezoidal decomposition is used to decompose C_{free} and the result is shown in Figure 2.6.

2.2.2 Graph Search

After the decomposition, C_{free} has been sliced into a set of cells $K = \{K_i\}_{i\in[1,n]}$. A graph *CG*, a so called connectivity graph, is constructed. The nodes of the graph are the cells of *K* and an edge between a pair of nodes, e.g., K_l and K_m, indicates adjacency between K_l and K_m in C_r. In Figure 2.7 the corresponding connectivity graph of the decomposition in Figure 2.6 is depicted. Nodes 17 and 5 are marked in the figure. They contain the current position of the robot and its destination respectively and may be called *start node* and *goal node*.

[5]Please note that to keep this example simple and possible to illustrate, a significant difference between Euclidean space and configuration space is not made. However, keep in mind that in general, the configuration space will look quite different from the corresponding Euclidean workspace (and possibly have many more dimensions).

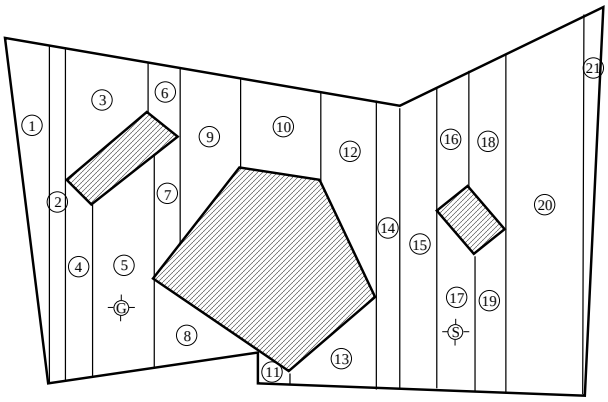


Figure 2.6. C_{free} is decomposed into cells by inserting a vertical line at every vertex of the obstacles (including the configuration space boundary). Each cell has been given a unique number.

The object of the next step, *Step 2*, is to search the *CG* for a path from the start node to the goal node. Any conventional search algorithm may be chosen for the search (e.g., breadth-first or A*). Weights may be assigned to the edges of *CG* (where the weight may be defined as the Euclidean distance between the centroids of the two cells connected by the edge) to improve the result of the search.

The result of the second step is a path through the graph (if there exists a path between start and goal nodes), a so called channel. A channel is a sequence of contiguous cells such that the cells of every pair of cells in the sequence are adjacent. A possible channel for our example is shown in Figure 2.8.

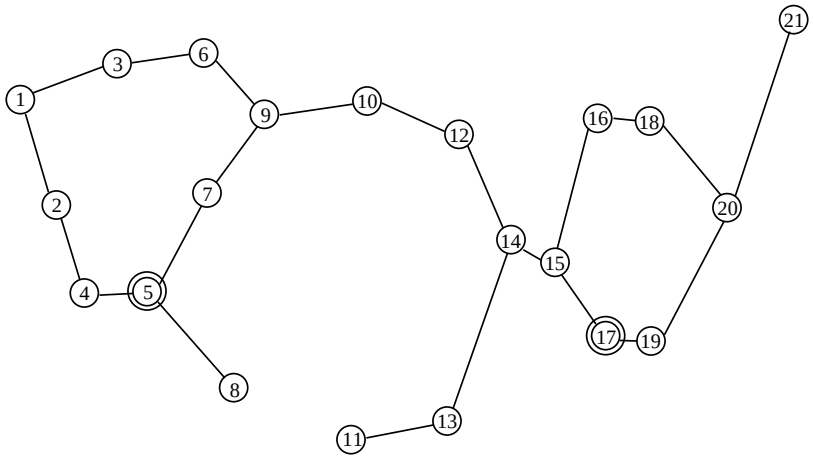


Figure 2.7. The nodes of the connectivity graph are the cells of C_r and the edges indicate adjacency between cells (compare to Figure 2.6). The nodes that hold the current position for r and its destination, 17 and 5 respectively, are marked in the figure.

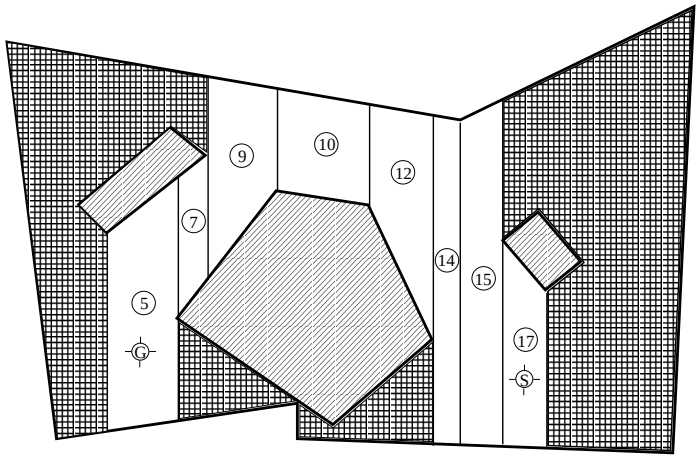


Figure 2.8. If there exists a robot path from current position to destination, then a path will be found in *CG*. This path corresponds to a channel in C_{free}.

2.2.3 Path Generation

Given a channel we can continue with *Step 3*, generating a robot path. As long as the path we plan is inside the channel, we can be sure that it is collision-free. When a path is created, qualities such as shortest path and all kinds of robot and workspace constraints may be considered. A simple path can be constructed by connecting start and goal positions through the centroids and edge points of the cells in the channel. Remember that the generated path is a path through the possible configurations of the configuration space C_r and not through the "real" Euclidean workspace of $\mathcal{W}$. However, in this example they happen to coincide very well. A possible path through the channel in this example is shown in Figure 2.9.

2.3 Exact Cell Decomposition for Multi-Robot Systems

When workspaces with more than one robot are studied, the path planning problem is changed from planning in a static workspace to planning in a dynamic one. Fortunately, all moving objects are known (because they are all known robots).

2.3.1 Decomposition of Free Space with Many Robots

Let us say that we are planning a path for a robot r_1 in a workspace $\mathcal{W}$. $\mathcal{W}$ contains some obstacles $O = \{O_i\}_{i\in[1,k]}$ and some robots $r = \{r_j\}_{j\in[1,l]}$. While planning for r_1, each of the other robots $r\backslash\{r_1\}$ are in exactly one of these two states

1. non-moving
2. moving

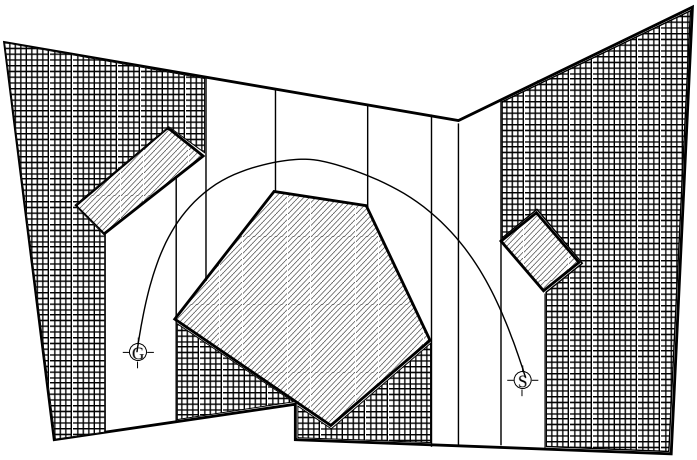


Figure 2.9. When a channel is found, a path can be planned for the robot.

A non-moving robot, e.g., r_2, is nothing more than an obstacle like any other as far as r_1 is concerned. So a reasonable approach to handling non-moving robots is to treat them just like any other forbidden area (see Figure 2.10). When considering

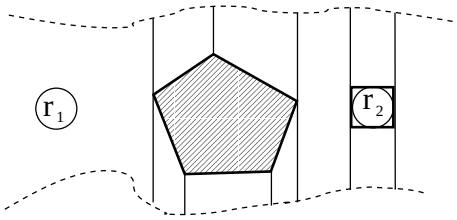


Figure 2.10. When decomposing the space for a robot r_1, treat other non-moving robots, e.g., r_2, as ordinary obstacles. This figure shows that decomposition works the same for an obstacle (the polygon in the middle) and the robot r_2 (the shape of r_2 has here been simplified to a square).

a non-moving robot r_2, while planning a path for r_1, the idea here is to treat the space occupied by r_2 on the rest of its path as an obstacle. Let us call this obstacle for *path obstacle* (see Figure 2.11).

This idea will yield a safe path for r_1 since it will avoid all the space that r_2 will occupy along the rest of its path. On the other hand, it may also be very inefficient since the path of r_2 may be long, yielding big path obstacles for r_1 and an unnecessarily poor path. To improve the efficiency, a special property is added to the path obstacles: paths may be planned across path obstacles.

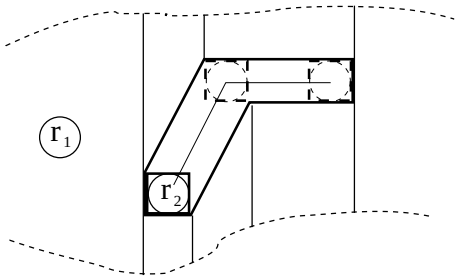


Figure 2.11. When considering a moving robot r_2, while planning a path for r_1, the space occupied by r_2 on the rest of its path is considered to be an obstacle, a so called *path obstacle*.

2.3.2 Example

Consider the decomposition of C_{free} in Figure 2.12, where path planning is performed for a robot r_1. It has its current position in *S* and its destination in *G*. On the left there is a non-moving robot r_2, and on the right there is a moving robot r_3 with its corresponding path obstacle. Notice how r_2 and r_3 are taken into consideration during the decomposition.

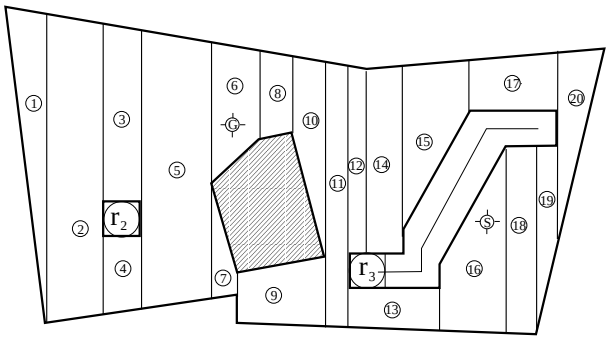


Figure 2.12. C_{free} is decomposed for a robot in position *S* with destination *G*. Notice how r_2 and r_3 are taken into consideration during the decomposition.

Just as with ordinary cell decomposition, a connectivity graph *CG* is constructed (see Figure 2.13). Notice that nodes which correspond to cells that are on either side of a path obstacle are connected by an edge.

The graph *CG* is in itself an ordinary graph and may be searched using any graph search algorithm. In the example, a path can be found through the graph from the start node to goal node. Just as in the case of ordinary cell decomposition, the path corresponds to a channel in C_{free} through which a path can be generated (see Figure 2.14).

Until now it has only been explained when to plan a path. What has not been shown is how this will be achieved in practice. The observant reader has

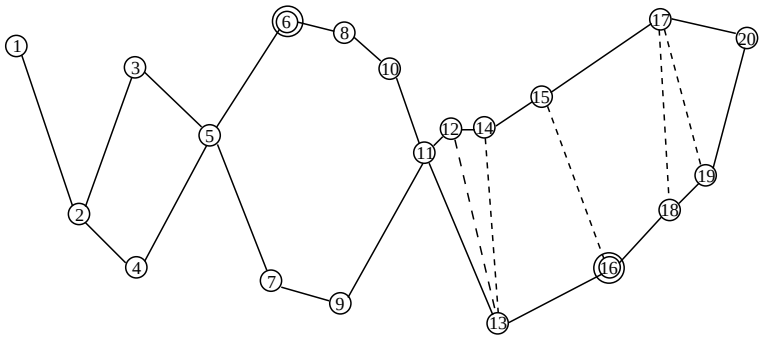


Figure 2.13. The graph for this method is similar to one constructed in ordinary cell decomposition. Notice, however that cells that are on either side of a path obstacle are connected by an edge (dashed in the figure). The start and goal nodes of the robot r_1, 16 and 6 respectively, have been marked in the figure.

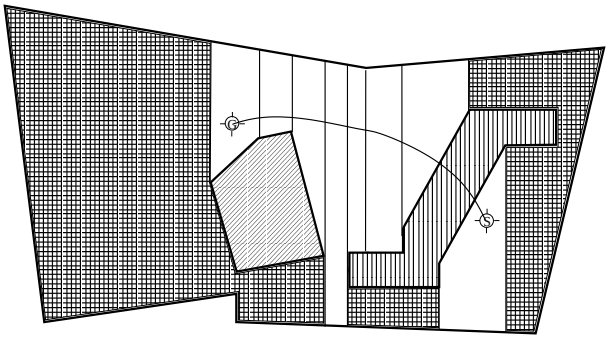


Figure 2.14. In this case there are actually two channels separated by a path obstacle. A path, which is collision-free everywhere except where it is crossing the path obstacle, is planned.

already noticed that the planned path is collision-free almost everywhere except at the paths' intersection. How paths' intersections are handled will be explained in Section 2.4, where the path planning method is extended into a motion planning method.

2.4 Motion planning for Multi-Robot Systems

So far a method to plan paths for robots in a shared workspace has been presented. To complete the motion planning method, it will also be explained how paths' intersections, the only region where robots stand a risk of colliding with each other, should be handled.

Two rules are introduced to assure that collisions do not occur:

Rule 1: A robot r_1 that enters and travels a paths' intersection has reserved the

region of that intersection (Figure 2.15). r_1 has *allocated* the intersection.

Rule 2: Furthermore, a paths' intersection may only be allocated by one robot at a time (Figure 2.16).

That Rule 1 and Rule 2 are sufficient for collision avoidance is proven in Appendix A.1.

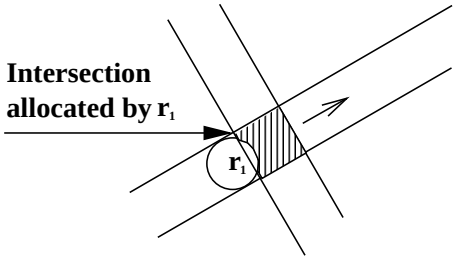


Figure 2.15. A robot traveling a paths' intersection has allocated the intersection.

Rule 1 can be realized just the way it is stated, i.e., r_1's allocation is noted and this piece of information made globally available. Rule 2 can be realized by not allowing a robot r_2 to enter the paths' intersection if it is currently allocated by some other robot r_2.

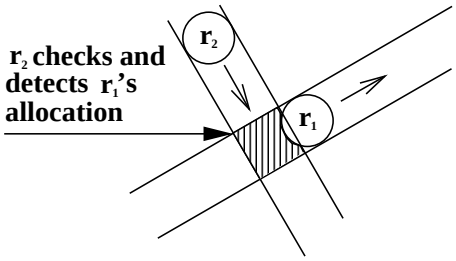


Figure 2.16. A robot r_2 may not enter a paths' intersection if some other robot r_1 has allocated it already.

Rule 2 raises the question what a robot r_2 should do if it wants to enter a paths' intersection that is currently allocated by another robot r_1. The most simple approach is just to wait until the allocation has been released by r_1. Simple solutions are normally good since they keep the complexity of a method low, but since robots may have to wait for each other, the problem of deadlock arises.

To avoid deadlocks, it is sufficient to violate at least one of the four required conditions for deadlock ([Coffman et al, 1971]). One of the four is the "hold and wait"-condition which states that a necessary requirement for deadlock is that "each agent holds resources while waiting for other agents to release theirs". A translation (of the opposite, i.e., how to violate the condition) into the terms of robots in this method is formulated in Rule 3.

Rule 3: A robot is not allowed to allocate a new paths' intersection without first releasing a previously allocated one.

It is not obvious how to realize Rule 3, but since a robot that can release an allocation never ends up in a deadlock it is suitable to plan paths that only cross one path obstacle at a time. In other words, there will be no edge in the connectivity graph between two cells in C_{free} which are separated by two path obstacles (Figure 2.17 illustrates this). A proof is presented in Appendix A.2.

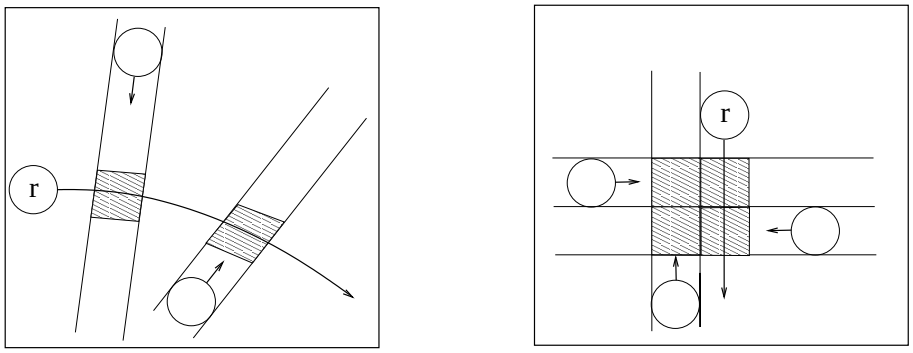


Figure 2.17. The left figure shows a planned path for a robot r that only crosses one path obstacle at a time. On the right is a robot r which has a path planned across two adjacent path obstacles. This leads to a potential deadlock situation.

In order for the method to work properly, correct global information must always be available. Problems arise if path planning is performed for two robots in parallel. We might for instance end up with paths that are overlapping a lot and difficult to handle, as depicted in Figure 2.18. In order to guarantee the global knowledge, planning must be performed in sequence. I.e., let us say that paths are going to be planned for some robots r_1 at time t_1, r_2 at time t_2 and r_3 at time t_3, with $t_1 < t_2 < t_3$. If we start to plan for r_1 at t_1 and the planning has not been finished by t_2, which is the first time we can plan for r_2, we can not plan for r_2 before a path has been generated for r_1. In the mean time, the state of the system can be said to be *diffuse*, not reliable and no planning may be performed during this time.

Note that paths' intersections also have to be handled in sequence to prevent two robots from allocating the same intersection at the same time.

2.5 Method Summary and Evaluation

It is natural to divide the motion planning method (`MPM`) into two separate parts:

1. Path planning

2. Path monitoring

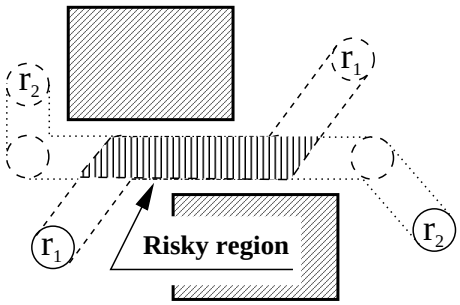


Figure 2.18. Path planning has been performed for robots r_1 and r_2 in parallel. Their paths overlap in the narrow passage, and if they both occupy the "risky region" at the same time, some extra functionality has to be added to solve the problem (otherwise they will collide or be deadlocked).

Path planning can be said to be off-line, it is performed before any motion has been executed. The path planning is performed using a modified exact cell decomposition method, obeying Rule 3 in Section 2.4. The main difference from the ordinary method is that robots are considered to be obstacles in the workspace (non-moving robots as ordinary obstacles and moving as path obstacles which may be crossed). A side-effect is that connectivity graphs are constructed differently (nodes corresponding to cells that are on either side of a path obstacle have an edge between them).

Path monitoring is an on-line process. It monitors a robot as it travels along its path and makes sure that paths' intersections are handled correctly (implementing Rule 1 and Rule 2 in Section 2.4).

Both parts handle their respective input in sequence to assure that sufficient information for the motion planning is available, and they are both deliberative since they rely heavily on the workspace representation.

`MPM` has been proven to yield collision-free motion for mobile robots sharing the same workspace. The robots can be applied to continuous problems of the type mentioned in Section 1.2.2, since paths may be planned for robots at any time. Furthermore, the method lacks static priorities which would assign unjust privileges to the robots (in the sense that robots with high priority would get more work done than robots with low priority).

Additionally, `MPM`, when discussing multi-robot control architectures, is inherently more centralized than decentralized because of the need for global knowledge. It is possible, though, to "decentralize" it in the sense that each robot may plan for itself (independent of a supervisor) by acquiring essential global information when needed ([Qutub et al, 1997]). This, however, does not offer all the advantages, in terms of for instance scalability, that could be expected of a decentralized control architecture ([Arai, Ota, 1992]).

Chapter 3 explains how `MPM` can be implemented to bring the theory into practice.

Chapter 3

Implementing the Motion Planning Method

In Chapter 2, a method for motion planning for a multi-robot system (MPM) was presented. In this chapter, the implementation of MPM on a system containing two robots is described.

3.1 Restrictions

The implementation of MPM was affected by the available experiment equipment. For the experiments, two robots could be used. This is unfortunately the least number of robots to use when experimenting with collision avoidance, but at the same time the use of more robots would imply an added complexity that is unnecessary to illustrate the basic principles of MPM. Furthermore, using MPM would introduce some indeed interesting, yet difficult geometrical problems which can not be considered to be within the scope of this project.

To make the mapping from workspace to configuration space less complicated it is assumed that

- workspace, robots and forbidden areas all have polygonal shape,
- the robots are rotation invariant[1], which is a realistic approximation for the robots used.

Additionally, even though MPM allows robots themselves to be responsible for the planning work, as global information is required for the planning it is easier to implement it in a centralized architecture. Since the implementation is a demonstration of feasibility, a centralized architecture with a single planner (a supervisor) was chosen.

[1] "Rotation invariant" is explained in Section 1.1.3.

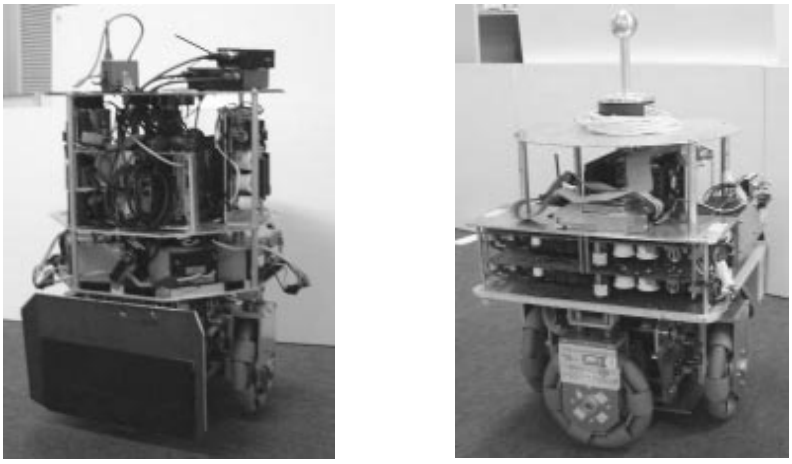
3.2 Equipment

For the experiments two omni-directional[2] robots (shown in Figure 3.1(a) and Figure 3.1(b)) of the *Instrumentation Project Promotion Division* at RIKEN were used. They both run the VxWorks operating system[3] on their on-board computers and are connected to a wire-less Ethernet LAN. More details on these robots can be found in [Asama et al., 1995].

The supervisor agent mentioned in Section 3.1 was implemented and run on a Toshiba Satellite 2180 CDT laptop computer with a 475 MHz AMD-K6 CPU and 64 MB RAM.

Table 3.1. Robot features

Feature	**Figure 3.1(a)**	**Figure 3.1(b)**
Omni-directional	Yes	Yes
Dims (bxdxh)	$45x50x62cm^3$	
CPU	Pentium 200 MHz	Pentium MMX 133 MHz
RAM	32 MB	64 MB



(a) (b)

Figure 3.1. Experiment robots

[2] Explained in Section 1.1.1.

[3] VxWorks is a real-time operating system (RTOS).

3.3 Software Architecture

Programming on the robots, which was performed in the C programming language, constitutes only a small part of the entire programming work. The C programming involved invocation of low-level robot motion commands, establishing and maintaining TCP/IP network connection, parsing received messages and sending status messages. The purpose of this part of the programming was to set up server processes that provide interfaces to external programs for sending requests (e.g., "Move to location x,y") and inquiries (e.g., "What is your speed?") to robots.

The major part of the programming efforts was devoted to the supervisor agent, which was implemented in the JAVA2 programming language on the laptop computer (described in Section 3.2). The program architecture is depicted in Figure 3.2 as a UML deployment diagram. The supervisor agent[4] maintains the state of the robot system (i.e., data about for instance robot and obstacle positions and robot paths); it also encompasses the path planner and path monitor explained in Chapter 2. This way, the robots are relieved of a lot of the, sometimes strenuous, computation work regarding path planning and path monitoring.

In the UML diagram (Figure 3.2), the boxes with two smaller boxes overlapping the left edge are computer processes (e.g., **Move**, **OperatorGUI** or **MainControl**). Some of the process symbols have a line ending in a circle extending from their bottom edges (e.g., **Request**). The circles denote interfaces towards other programs. The plain boxes (**EnvironmentModel** and **CellDecompositionModel**) are just class instances. Dashed arrows indicate dependencies. A dashed arrow from a process or instance c_1 to another c_2 means that c_1 is aware of c_2 and that changes in c_2 might lead to changes in c_1.

The rest of this section deals just with the software running on the laptop computer (if nothing else is stated).

3.3.1 Model-View-Control

The implementation on the laptop is based on the *Model-View-Control* (MVC) programming architecture. MVC encourages the application developer to subdivide a program into objects of three different types:

Model keeps the data or state of the program

View displays data

Control may access and alter data

A model accepts *subscribers* (objects which claim to be interested in the data stored in the model) which it will inform when its contained information gets updated.

[4]The supervisor agent is composed of the **MainControl** and **PathMonitor** processes and the **EnvironmentModel** and **CellDecompositionModel** classes in the diagram.

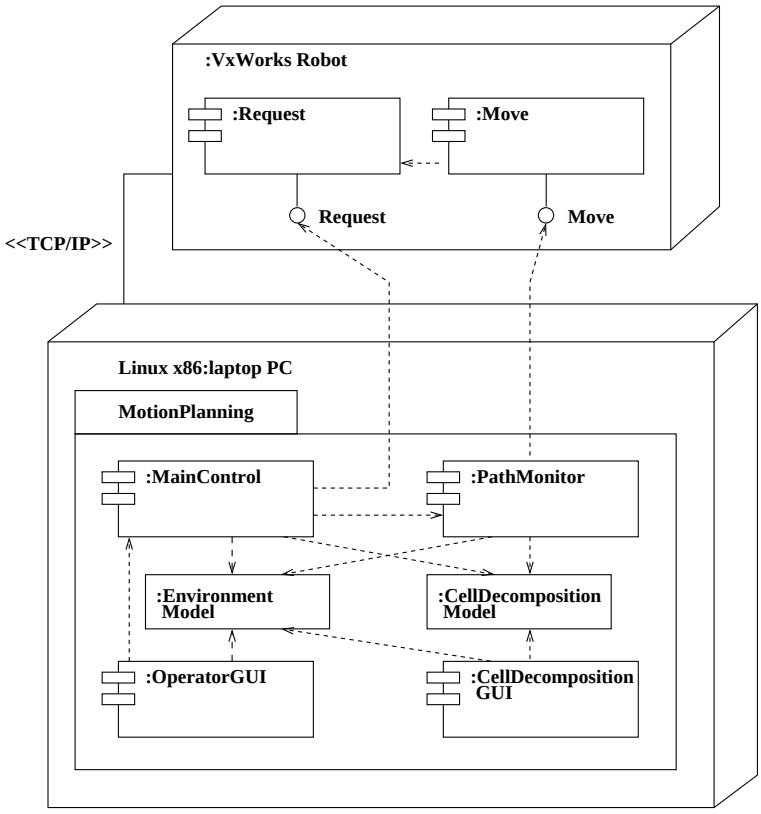


Figure 3.2. UML Deployment Diagram

The MVC architecture is flexible and allows for instance additional Views to easily be added. Figure 3.3 explains how MVC objects depend in this implementation.

3.3.2 Models - data containers

The supervisor has two models (data containers):

EnvironmentModel keeps information about the workspace, robots and forbidden areas

CellDecompositionModel keeps configuration space obstacles and cells of a decomposition

It is reasonable to make this division; the **EnvironmentModel** contains information that must be available to the entire application (i.e., supervisor and GUIs[5]), while the **CellDecompositionModel** contains information from the latest cell de-

[5]GUI is an acronym for Graphical User Interface

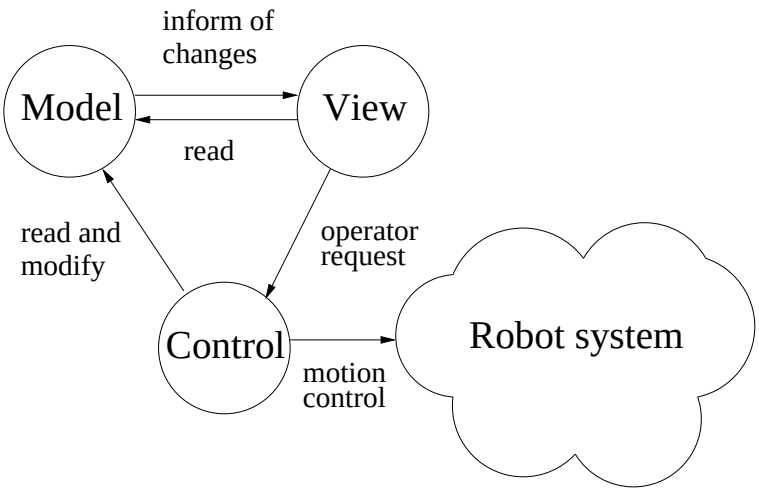


Figure 3.3. This figure explains how the MVC objects in the supervisor agent and GUIs depend on each other. An arrow from an object *A* to another object *B* in the figure expresses that *A* has a reference to *B* and a text string close to the arrow explains what the reference is used for (e.g., read).

composition and its contents are only used to be displayed and to generate a robot path.

3.3.3 Views - data displays

A few Views (or GUIs) were developed to make the testing of the method simpler.

The Operator GUI (Figure 3.4) displays the data of the **EnvironmentModel** (Section 3.3.2). It also allows a (human) *operator* to mark a spot in the workspace and command a robot to go to the spot.

The cell decomposition GUI (Figure 3.5) displays data from both the **EnvironmentModel** and **CellDecompositionModel**. The operator can choose which data should be displayed.

3.3.4 Controls - data manipulators

A process called **MainControl** encompasses both the Path Planner and the Path Monitor. The normal operation of the **MainControl**, when it receives a request for robot movement (the request includes data about the robot and its destination) for a robot *r*, is to use the Path Planner to plan a path for the robot. If a path exists, a **PathMonitor** process is started that interacts with the robot when it travels along its path and makes sure it will not collide.

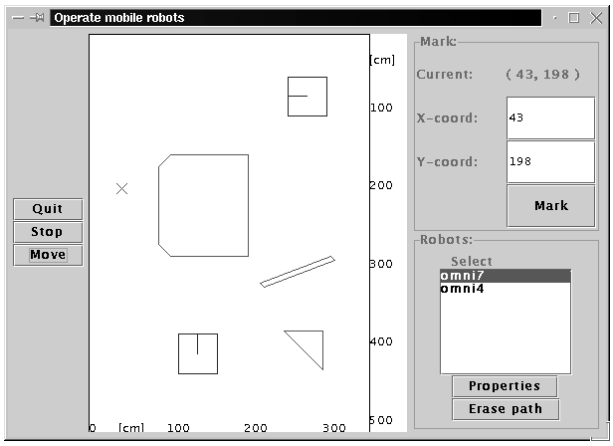


Figure 3.4. This GUI displays data contained in the **EnvironmentModel**. The workspace is in the middle of the window. Besides from the boundary of the workspace it also displays three obstacles (polygons) and two robots (square with a line that indicates current orientation).

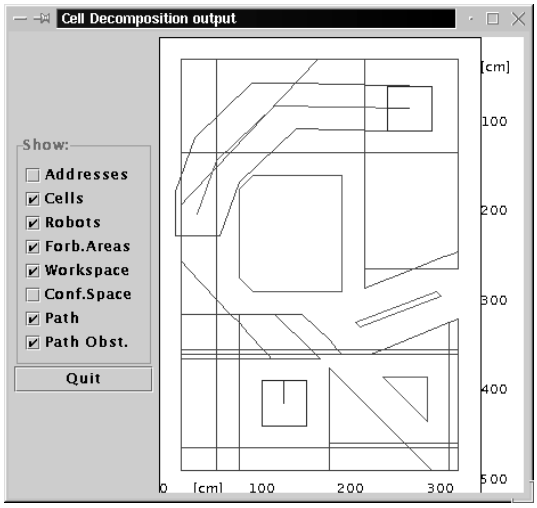


Figure 3.5. This GUI displays the workspace objects and the result of the latest cell decomposition (e.g., configuration space obstacles, cells and paths). The operator of the GUI has the possibility to choose which information he or she wants to see.

Path Planner

The Path Planner constructs the configuration space C given the workspace and the robot which it plans for. The Path Planner implements the method for cell decomposition for multi-robot systems explained in Chapter 2. The decomposition of space is based on a method called “Cell Decomposition of Arrangements” (presented in [Sleumer, Tschichold-Gürman, 1999]). This method differs a bit from the

trapezoidal decomposition. Instead of using a sweep-line, all edges (i.e., edges of obstacles and workspace boundary) in the workspace are extended from both endpoints until they reach another workspace edge or the boundary of the workspace. This way the free space C_{free} is decomposed. A comparison between trapezoidal and arrangements decomposition is shown in Figure 3.6.

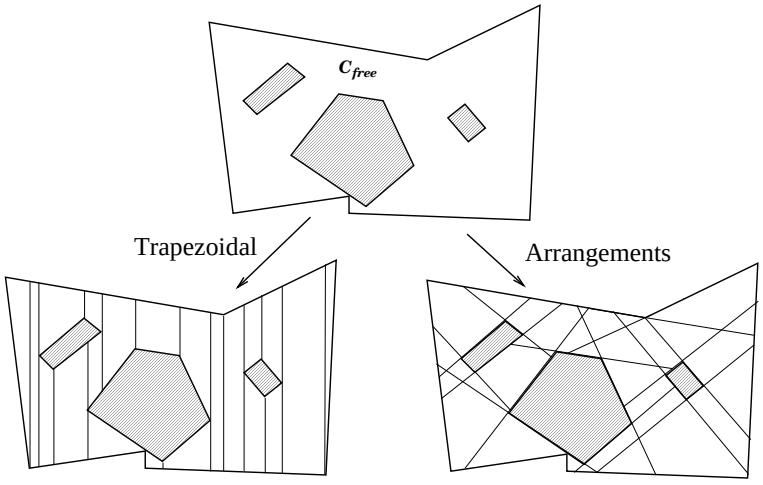


Figure 3.6. There is not a single unique way to decompose free space. C_{free} in the top of the figure is here decomposed using Trapezoidal (lower left) and Arrangements (lower right) decompositions.

Once a channel has been found through the cells of C_{free}, a path can be planned. A greedy and heuristic algorithm which takes paths' intersections into consideration has been developed for this purpose. The resulting path is represented as a sequence of vertices (i.e., intermediate workspace positions and the destination). The path generation algorithm is described in Appendix B.

Path Monitor

When a path *p* has been planned for a robot *r*, **MainControl** starts up a **PathMonitor** process which interacts with *r* as it travels along *p*. The Path Planner sends a message to *r* to go to the next intermediate position (vertex) *pos* on *p*. *r* goes to *pos* using its own odometry. When it has reached *pos* it stops and sends a message back to its **PathMonitor** which replies with the next intermediate position, unless the destination has been reached in which case the **PathMonitor** terminates.

3.3.5 Software interaction

The sequence diagram in Figure 3.7 depicts the interaction between processes and class instances when an operator uses the **OperatorGUI** to move a robot *r*.

1. **MainControl** invokes the **PathPlanner**.

2. When the **PathPlanner** returns, the result of the cell decomposition is stored in the **CellDecompositionModel**.

3. If **PathPlanner** succeeded in planning a path, a **PathMonitor** process will be started, which interacts with the **Move** process of the robot *r*.

4. **PathMonitor** sends a position (which is the next vertex in the path) to **Move** which the robot should travel to.

5. **Move** replies with an **arrived** message when the robot has reached the position.

6. **PathMonitor** removes the last vertex from the path and asks the **EnvironmentModel** to update to the new information about robot *r*.

Steps 4 to 6 are repeated until there are no more vertices left in the path and then the **PathMonitor** terminates.

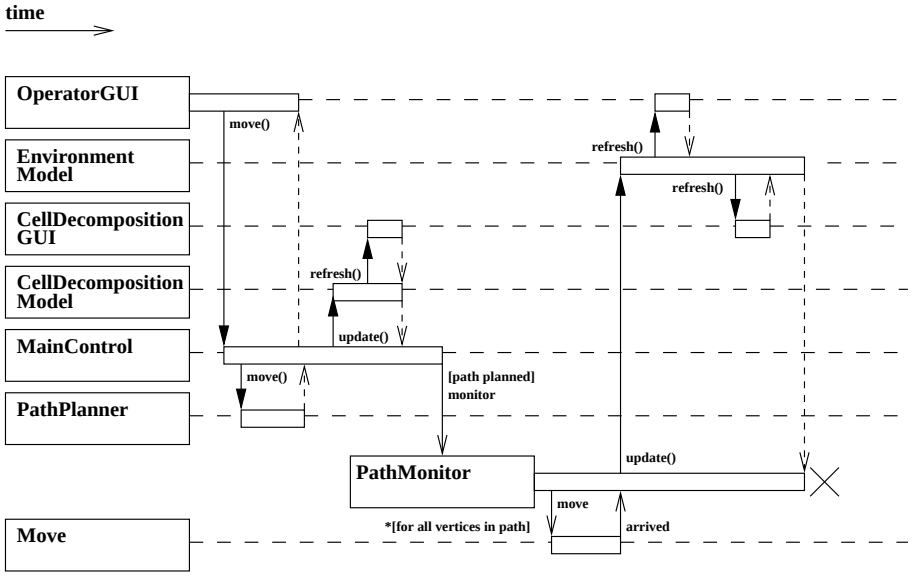


Figure 3.7. This diagram describes how the processes and class instances in Figure 3.2 interact when an operator issues a valid move-request from the **OperatorGUI**. The move-arrived message passing between the **PathMonitor** and **Move** is performed repeatedly until the path has no more vertices and the robot has reached its destination.

3.4 Summary

The implementation of MPM in this project provides a centralized control architecture where most of the computations are concentrated to a supervisor agent on a special computer (which is not on board any robot). The supervisor agent care

for path planning for all robots, as well as path monitoring. The programming structure of the supervisor is based on the MVC programming architecture which encourages separation of data, data display and data manipulation. MVC furthermore allows future data manipulator and data display software to easily be added.

Each time a new destination for a robot r is set, the supervisor plans a path (path planning) for it and finally starts a separate process which interacts with r as it travels to the destination to instruct it where to go and when to halt (path monitoring).

Chapter 4

Collision-Free Motion in Simulations and Experiments

This chapter contains the results of some simulations and real robot experiments. The purpose of the simulations and experiments is to show that the implementation works as planned and that the motion planning method (MPM) can be integrated with a set of real robots.

4.1 Path Planning Simulations

In Chapter 2, a few extensions to the path planning method exact cell decomposition was presented. The extensions describe how robots in a workspace are handled during path planning.

In a multi-robot system, when planning for a robot r, treat other robots as either

- ordinary obstacles, if they are non-moving (1), or
- path obstacles, if they are moving (2).

Furthermore, allow paths to be planned across path obstacles (3).

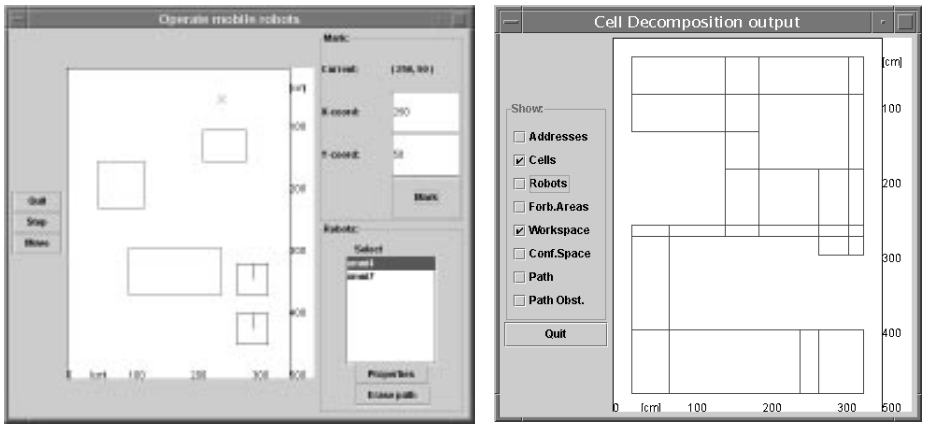
The following simulations show that these three extensions have been implemented correctly.

4.1.1 Considering Non-Moving Robots

Figure 2.10 illustrates the idea of (1). If a path is planned for a robot r_1, treat a non-moving robot r_2 as an obstacle in order to avoid collision between the two robots.

A reasonable way to show that the implementation works correctly is to compare the decompositions of two workspaces $\mathcal{W}_1'$ and $\mathcal{W}_2'$, where the difference between the two is that $\mathcal{W}_1'$ has a robot which has been replaced by an obstacle with the same shape and position in $\mathcal{W}_2'$.

Figure 4.1(a) shows the workspace $\mathcal{W}_1$, which includes three obstacles and two robots (the robots are the rectangles that contain a small line, which indicates the orientation of the robot, from center to border) and Figure 4.1(b) the decomposition of the free space of $\mathcal{W}_1$.



(a) (b)

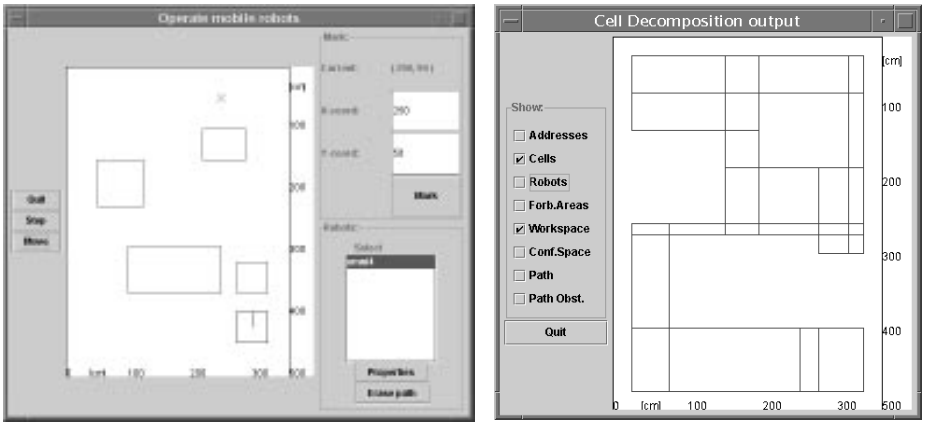
Figure 4.1. (a) Workspace $\mathcal{W}_1$ including two robots. A destination for the lower robot is marked with a cross in the workspace. **(b)** The rectangles are the cells of the decomposition.

Figure 4.2(a) shows the workspace $\mathcal{W}_2$ and Figure 4.2(b) the decomposition of its free space. Compare Figure 4.1(b) with Figure 4.2(b). They are identical and consequently a non-moving robot has the same impact on the decomposition as an obstacle of the same size and shape. Hence, both $\mathcal{W}_1$ and $\mathcal{W}_2$ yield the same robot path (see Figure 4.3) if the same graph search and path generation algorithms are used.

4.1.2 Considering Moving Robots

To handle robots that are moving during path planning, the idea of path obstacle is used (illustrated in Figure 2.11). When planning for a robot r_1, the space occupied by a moving robot r_2 on its known path, from its current position to the end of its current path, is considered to be an obstacle. Using this idea, r_1 will not be assigned a path which leads to a conflict (i.e., a deadlock or collision situation) with r_2.

The following example will show that moving robots are taken into consideration as intended during path planning. Figure 4.4(a) shows a workspace $\mathcal{W}_3$ that contains two robots r_1 and r_2, two long vertical obstacles and a cross which indicates the intended destination of robot r_1. r_1 is the robot in the lower right corner.



(a) (b)

Figure 4.2. (a) Workspace $\mathcal{W}_2$ including two robots. A destination for the lower robot is marked with a cross in the workspace. **(b)** The rectangles are the cells of the decomposition.

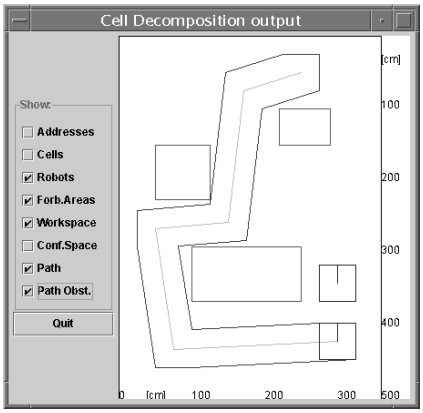
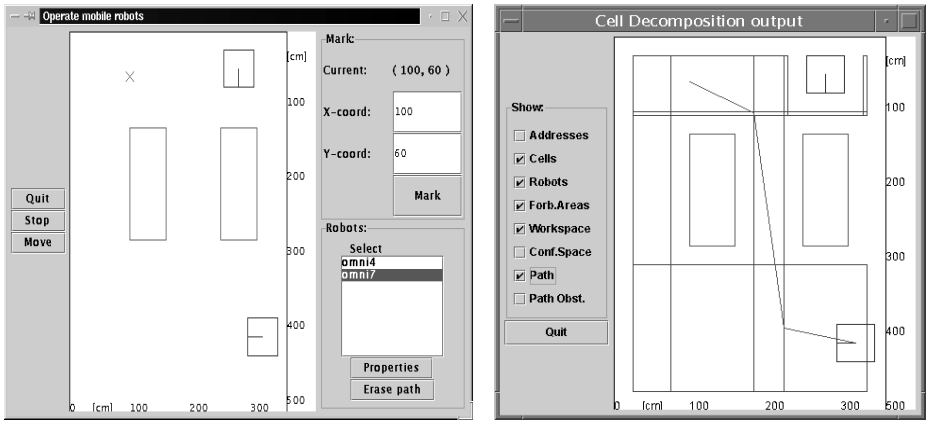


Figure 4.3. Both workspaces $\mathcal{W}_1$ and $\mathcal{W}_2$ yield the same robot path.

In the first case r_2 is non-moving when a path is planned for r_1 to its destination. The result of the path planning is illustrated in Figure 4.4(b). The free space has been decomposed into a set of cells which are displayed with lines that surround the robots and obstacles in the figure. The planned path of r_1 is the polyline that extends from its center.

In the second case, which is illustrated with workspace $\mathcal{W}_4$ in Figure 4.5(a), r_2 has just started to move along a path that goes between the two obstacles. When path planning for r_1 is started, r_2's path makes up a path obstacle. The path obstacle



(a) (b)

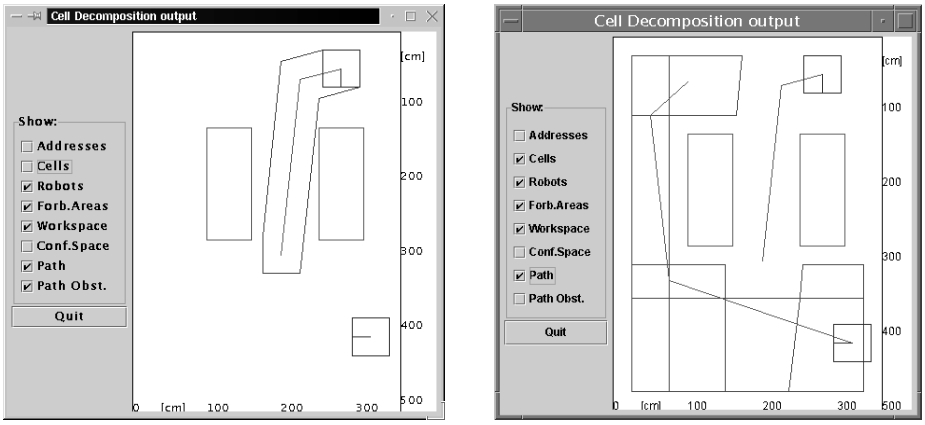
Figure 4.4. (a) Workspace $\mathcal{W}_3$ with two robots and two long vertical obstacles. A destination for the lower robot r_1 is marked with a cross in the workspace. **(b)** The cells of the decomposition are the lines that surround the robots and obstacles. The path of robot r_1 extends from its center to its destination.

is taken into consideration during path planning for r_1 and the result is displayed in Figure 4.5(b). Compare this result with the result from the previous case (displayed in Figure 4.4(b)). Now there are no cells in the narrow passage between the two obstacles which is blocked by r_2 and hence the path obstacle has been taken into consideration and the resulting path (for r_1) differs from the previous case.

4.1.3 Planning Paths' Intersections

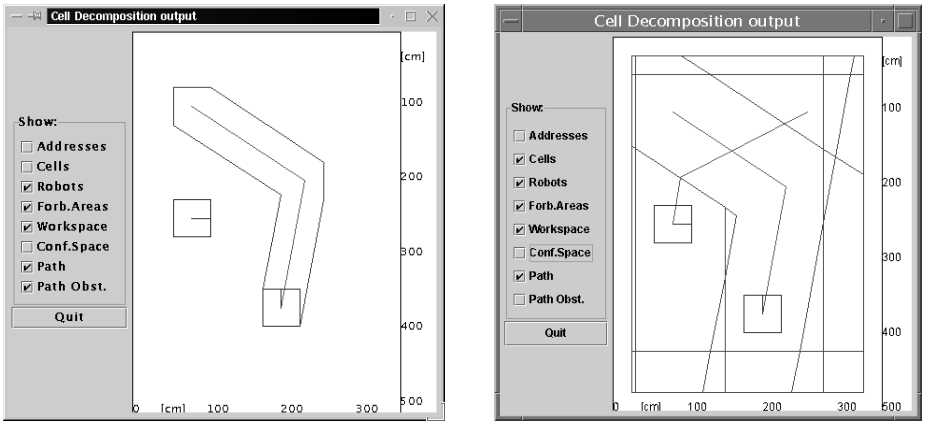
Path obstacles are suitable to use during path planning, but if paths are long path obstacles will be large and consequently the free space will be small. The impact of this problem is reduced by allowing paths to be planned across path obstacles.

The following example shows that paths may be planned across path obstacles. Figure 4.6(a) displays a workspace containing only two robots, r_1 (on the upper left) and r_2 (on the lower right), and the path obstacle of r_2 that extends to the left side of the workspace. Now, r_1 wants to move from its current position to the right side of the workspace and a path is planned for it. As can be seen in Figure 4.6(b), the result of the path planning for r_1 is a path that is planned to cross r_2's path. It is not easy to see in the figure that the path of r_1 is actually crossing a path obstacle, but the bordered area surrounding r_2 in Figure 4.6(b) is actually its path obstacle and not a cell (remember that cells are convex).



(a) (b)

Figure 4.5. (a) Workspace $\mathcal{W}_4$ with two robots and two long vertical obstacles. A destination for the lower robot r_1 is marked with a cross in the workspace. Robot r_2 makes up a path obstacle. **(b)** The cells of the decomposition are the lines that surround the robots and obstacles. The paths of the robots are the polylines that extend from their centers.



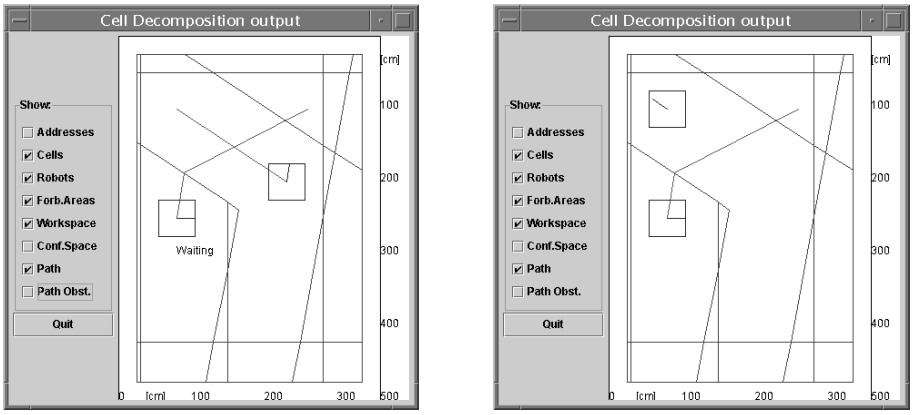
(a) (b)

Figure 4.6. (a) The lower robot r_2 has a planned path to its destination. **(b)** The free space has been decomposed and the robot on the left r_1 has a path planned across the path obstacle of the lower robot r_2.

4.2 Path Monitoring Simulations

MPM does not assign static priorities to robots, i.e., the robot that first reaches a paths' intersection may allocate it and pass. The following example shows that the robot which should stop and wait for another robot at a paths' intersection is not predefined.

Remember the example in Section 4.1.3 with two robots r_1 and r_2 with crossing paths in an empty workspace. Figure 4.7(a) shows a simulation where r_2 has reached the paths' intersection before r_1 can allocate it. Before r_1 is allowed to cross the intersection, r_2's allocation is detected and r_1 has to wait (which is indicated by the "waiting" message in the figure). Figure 4.7(b) shows that r_1 will no longer have to wait when r_2 has released its allocation.

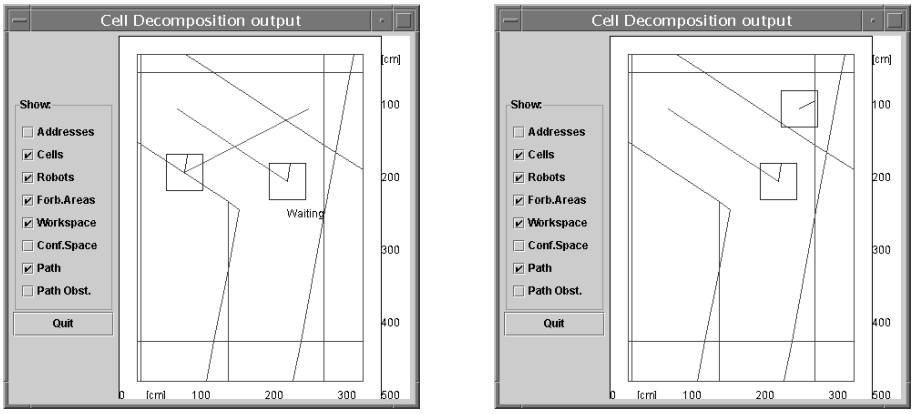


(a) (b)

Figure 4.7. **(a)** The robot on the left r_1 has to wait while the other robot r_2 has allocated the paths' intersection. **(b)** The paths' intersection is no longer allocated by r_2 and r_1 may pass.

By delaying the start of r_2 or lowering its speed, r_1 may reach the common paths' intersection before r_2. In Figure 4.8(a), r_1 is the first robot to reach the intersection and in this case r_2 will have to wait. Figure 4.8(b) shows that r_2 is allowed to cross the intersection when r_1 has left the paths' intersection and released the allocation.

These two simulations show that static priorities, which may have bad side-effects, are not used in the implementation.



(a) (b)

Figure 4.8. (a) The left robot r_1 has allocated the paths' intersection and robot r_1 has to wait. **(b)** Robot r_1 has released its allocation and the other robot r_2 may cross the intersection.

4.3 Real Robot Experiments

Until this point, the chapter has only dealt with simulations. To show that the implementation of both path planning and path monitoring works with the available robot hardware, some experiments with physical robots were performed. As pointed out in Section 1.2.3, neither localization nor robot communication is in the focus of this project, so the same assumptions were made for the real robot experiments as for the simulations. This means that it was assumed that the communication between robots and laptop computer, over the wireless Ethernet LAN in the experiment environment, worked reliably and that the robot odometry worked perfectly (no navigation system was used).

The experiments realize the simulation displayed in Figure 4.8(a) and use the two robots presented in Section 3.2. The experiments' workspace is of the same shape and size as the corresponding simulated workspace, i.e., rectangular and approximately three times five meters. This particular simulation is interesting because it exposes the implementation of `MPM` to the circumstance of both robots being subjects to an imminent collision, and therefore tests how the software handles robot-travel along path segments, synchronization of path planning, allocation of paths' intersection, detecting allocation, halting and releasing allocation.

Figure 4.9 shows the real world representation of Figure 4.8(a) where r_1 has allocated the paths' intersection. Consequently, Figure 4.10 shows the real world representation of Figure 4.8(b).

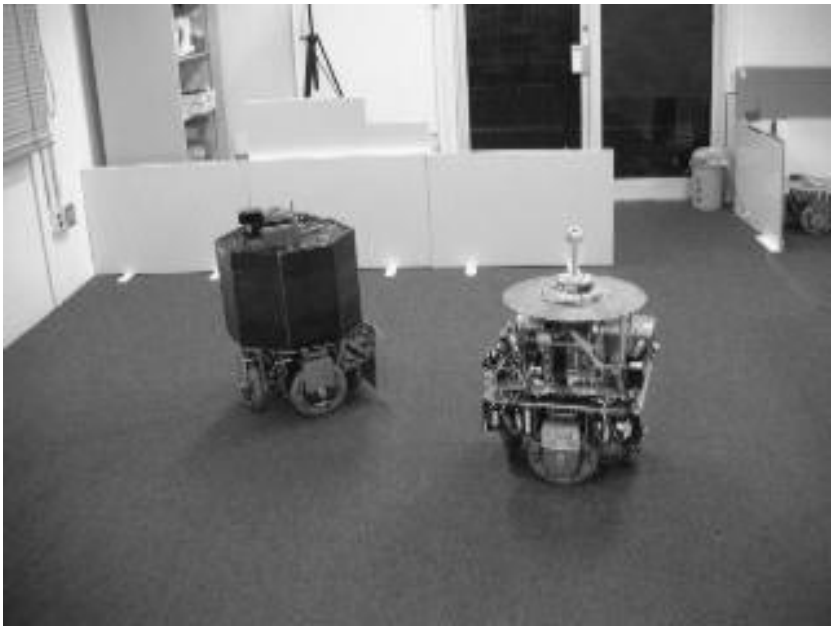


Figure 4.9. The black robot has allocated the paths' intersection and the white one has to wait.

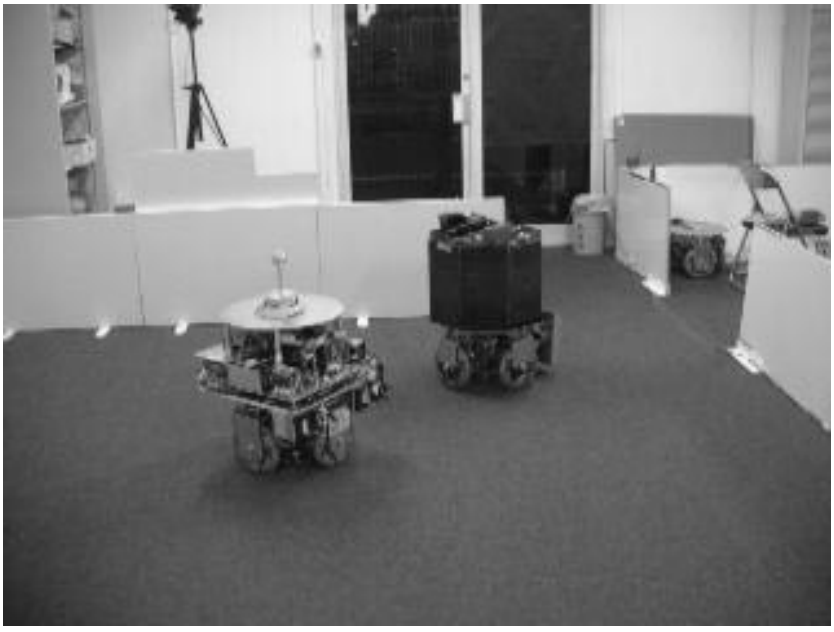


Figure 4.10. The black robot has released its allocation and the white one may cross the intersection.

Chapter 5

Summary and Discussion

This chapter summarizes the work presented in this thesis and highlights the advantages of the developed motion planning method (MPM). It also suggests suitable applications, improvements of and extensions to MPM.

5.1 Summary

The problem presented in Section 1.2 concerns the development of a motion planning method for control of robots in a multi-robot system. There are a few restrictions contained within the problem. The following research topics, for instance, are not discussed:

- task allocation,
- navigation system,
- non-holonomic robots,
- unknown workspaces.

A method for motion planning (MPM) was developed by extending the path planning method exact cell decomposition and adding rules for how robots cross paths' intersections. MPM is naturally subdivided into two parts: path planning and path monitoring.

Path planning includes decomposition of free space into cells taking special care of robots in the workspace, treating moving ones as path obstacles. Paths are allowed to be planned across other paths.

Path monitoring includes monitoring robots as they travel along their planned paths making sure that no collision will occur at paths' intersections.

The method was implemented on a set of two robots. Some restrictions were introduced to simplify the implementation, e.g.,

- robots are rotation invariant[1],
- obstacles and robots are spatially modeled as polygons.

A few simulations have been performed on the software and experiments on the robots. They show that the implementation works as intended and that one of the simulations can be realized with the robots.

5.2 Conclusions

A method, MPM was devised, implemented, and tested with satisfying results. Note that in the path planning part of MPM, only step 1 (in Figure 2.2) of the original cell decomposition method is altered. Hence, the contribution of this work to path planning is how to generate connectivity graphs (the crossing of path obstacles is optional), and subsequently channels, for multi-robot systems. As mentioned in Section 2.1, channels are useful both for deciding if a path exists and for path generation.

Unfortunately, MPM has a few drawbacks. Firstly, it requires global knowledge about robots and workspace and, although they can be made to plan for themselves, robots need to acquire information about all other robots occasionally to maintain the motion planning. This has a negative effect on the scalability (the efficiency when the number of robots increases) of a robot system using MPM[2].

Secondly, path obstacles eat up a lot of the free space (especially if paths are long) and will possibly prevent paths from being planned. This problem, illustrated in Figure 5.1, is yet another threat to scalability.

Thirdly, even though a deadlock will not occur for moving robots, it is still possible that a deadlock arises when two or more robots try to plan paths and they are blocking each other (see Figure 5.2(a) and Figure 5.2(b) for examples). However, such a deadlock situation is easily detected (since global information is at hand) and there are already methods available for solving it[3].

MPM is particularly applicable to problems which lack an end-state (or end-configuration, e.g., end positions of all robots) and value safety higher than efficiency.

No end-configuration Some motion planning methods for multi-robot systems presented in previous works have the drawback that they require a known

[1] This assumption eliminates the orientation dimension of the configuration space and makes calculations simpler.

[2] When the number of robots increases, if the robot system is centralized, i.e., has a single agent devoted to motion planning, this agent will have to be improved. On the other hand, if the system is decentralized, communication resources between robots will have to be improved.

[3] In [Latombe, 1993] a motion planning method is described which plans collision-free motion for robots of a multi-robot system given known start and end configurations. In this case the start configuration is the configuration of all robots in the deadlock situation and the end configuration is the configuration of all robots when the robots of the deadlock have been moved out of the deadlock.

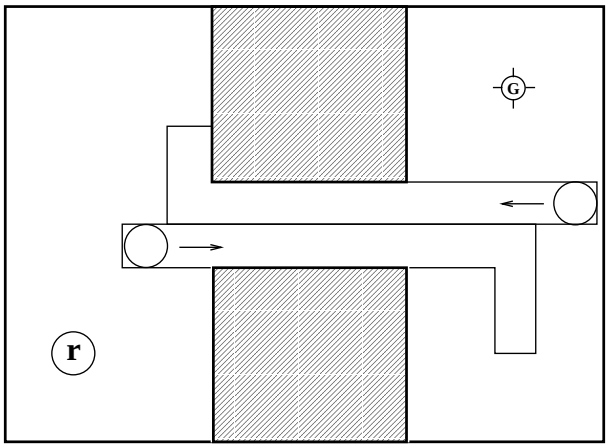
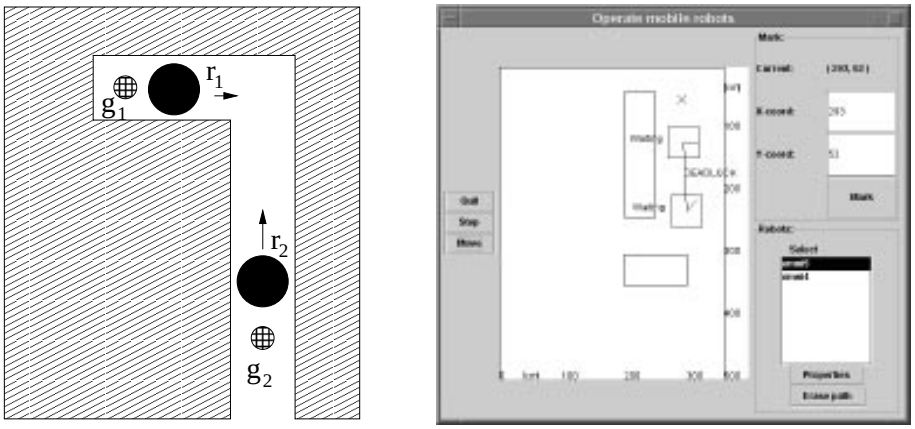


Figure 5.1. In this case the use of path obstacles prevents *r* from planning a path to its destination *G*.



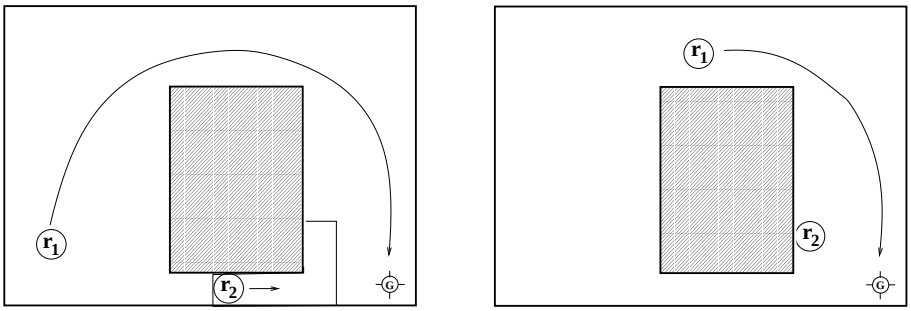
(a) (b)

Figure 5.2. (a) This is a deadlock situation since if r_1 wants to go to g_2 and r_2 wants to go to g_1, then r_1 and r_2 are blocking each other. **(b)** In this simulation a line between the two robots and the DEADLOCK message next to it indicate that a (path planning) deadlock has been detected.

end-configuration of the system. One important property with MPM is that it has no such requirement. This makes it applicable to continuous problems that lack an end-configuration, e.g., those mentioned in Section 1.2.2.

Safety Robots will have to stop sometimes to let other robots pass, and planned paths may be inefficient since robots taken into consideration during path

planning for a robot *r* may be long gone when *r* eventually reaches the region where the other robots were when its path was planned (as illustrated in Figure 5.3(a) and Figure 5.3(b)). On the other hand, *r* is safe on its path and knows that other robots are aware of its path and try to avoid it.



(a) (b)

Figure 5.3. (a) r_1 has a path planned, through a workspace also containing an obstacle and another robot r_2, to its destination G. If it were not for r_2, a shorter path, stretching below the obstacle, could have been planned. **(b)** r_2 is not blocking the path below the obstacle anymore. r_1 could have used the other path and reached its destination in less time (but there would have been trouble if r_2 halted in the narrow passage).

5.3 Future Work

When discussing future work, it is useful to divide the work of this thesis into two separate parts: *method* and its *implementation*. While the method encompasses the sub-methods and rules used and presented in Chapter 2, the implementation encompasses the software and hardware used to implement `MPM`. It is natural to separate them since the method is not dependent on its implementation (it can be implemented in many different ways).

5.3.1 Implementation

As mentioned in Section 5.1, a few simplifications have been introduced to the implementation of `MPM`. What first of all would be interesting is to allow more robots to be used in simulations and experiments for further testing of the utility of `MPM` in practice. It would also be interesting to study how to handle overlapping path obstacles and forbidden areas. This, however, requires the development of some fairly complicated geometrical algorithms.

At present the robot-system used in Section 4.3 only relies on odometry for its navigation. This makes it useful only for a short while since the robots' position estimations will soon start to deviate from the actual positions. In order to assure correct navigation, which is necessary for MPM to work reliably, an absolute measurement technique (see Section 1.1.2) will have to be integrated into the robot system.

When it comes to path generation, it would both be interesting to generate the shortest paths from start to destination and to optimize the path to suit the hardware of the robot system in use. To give an example of the latter wish, it may, for instance, be useful to increase or decrease the number of stops on the path if this is necessary for the use of a possibly employed navigation system.

The inefficient stops along planned paths are slightly annoying and should be eliminated. This however is somewhat complicated and requires that the dynamics of robots (e.g., its speed) and speed and reliability of communication links are taken into consideration.

5.3.2 Method

The work in this thesis only presents a foundation for a task solving multi-robot system. Future work may progress in many directions, but the most important motivation is to make MPM applicable to more problems. Here are a few examples of possible future fields of research:

Scalability As pointed out in Section 5.2, MPM offers poor scalability. In order to make it useful in applications with a fixed, large number of robots or varying number of robots, this issue has to be addressed.

Partial models One of the assumptions in this work is that the environment is completely known. For real problems, though, this is rarely the case. Many problems include unknown non-moving and moving objects. It is therefore essential to add mechanisms to MPM that can manage such objects. One obvious measure to achieve this goal is to employ sensors (for instance sonar and IR) on board the robots.

Dynamic constraints If robots are given dynamic constraints, for instance velocity constraints (e.g., maximum or minimum speeds) this can be taken into consideration during path planning and especially during path monitoring.

Non-holonomic robots At present only holonomic robots are considered. It would be interesting to extend MPM to include even non-holonomic ones.

Tasks Specific tasks might impose non-geometric constraints that ought to be considered. In a transportation setup, it is of interest that robots which carry objects have precedence over robots that do not, as delivery time might be an objective. In addition, the relation between robots might be a task constraint to be considered.

Appendix A

Proofs of Motion Planning Consistency

A.1 Collision-Free Paths' Intersections

A proof method called natural deduction is here used to prove that it is sufficient to obey Rule 1 and Rule 2, from Section 2.4, to avoid collisions.

Rule 1: A robot r_1, which enters and travels a paths' intersection, has reserved the region of that intersection. r_1 is said to have allocated the intersection (Figure 2.15).

Rule 2: Furthermore, a paths' intersection may only be allocated by one robot at a time (Figure 2.16).

Table A.1. Definitions

Sets R (robots), I (paths' intersections), T (continuous time)
Variables $r, r_1, r_2 \in R$, $i \in I$ and $t \in T$
Constants $a, b \in R$, $c \in I$ and $d \in T$
Predicate1 $Inside(r,i,t)$: robot r is inside paths' intersection i at time t
Predicate2 $Alloc(r,i,t)$: robot r has allocated paths' intersection i at time t
Predicate3 $Collision(r_1,r_2,t)$: robots r_1 and r_2 are colliding at time t. $Collision(r_1,r_2,t) \leftrightarrow \exists i(Inside(r_1,i,t) \wedge Inside(r_2,i,t) \wedge \neg(r_1 = r_2))$

In Table A.1 some symbols that are useful for the proof are defined. A collision is here, a bit sloppily, defined to be equivalent to two (unique) robots being inside the

same paths' intersection at the same time. In fact, if this happens, collision might be avoided (by pure luck), but $Collision()$ is a condition that is required for collision (i.e., we will not have a collision at any time t such that $\neg Collision(r_1, r_2, t)$).

With the definitions in Table A.1, Rule 1 and Rule 2 can be expressed as

Rule 1: $R1 \leftrightarrow \forall r \forall i \forall t (Inside(r,i,t) \rightarrow Alloc(r,i,t))$

Rule 2: $R2 \leftrightarrow \forall r_1 \forall r_2 \forall i \forall t (Alloc(r_1,i,t) \wedge Alloc(r_2,i,t) \rightarrow (r_1 = r_2))$

It will be proven, given Rule 1 and Rule 2, that no robot will collide. This can be expressed in logic using the symbols now defined as

$$R1, R2 \models \neg \exists r_1 \exists r_2 \exists t Collision(r_1, r_2, t) \tag{A.1}$$

where $Collision(r_1, r_2, t)$ may be replaced with $\exists i (Inside(r_1, a, t) \wedge Inside(r_2, i, t) \wedge \neg(r_1 = r_2))$ according to its definition.

By assuming the opposite and deduce a contradiction A.1 will be proven.

1		$\forall i \forall t \forall r (Inside(r,i,t) \rightarrow Alloc(r,i,t))$	premise
2		$\forall r_1 \forall r_2 \forall i \forall t (Alloc(r_1,i,t) \wedge Alloc(r_2,i,t) \rightarrow (r_1 = r_2))$	premise
3	+	$\exists r_1 \exists r_2 \exists t \exists i (Inside(r_1,i,t) \wedge Inside(r_2,i,t) \wedge \neg(r_1 = r_2))$	assumption
4	++	$\exists r_2 \exists t \exists i (Inside(a,i,t) \wedge Inside(r_2,i,t) \wedge \neg(a = r_2))$	assumption($r_1 = a$)
5	+++	$\exists t \exists i (Inside(a,i,t) \wedge Inside(b,i,t) \wedge \neg(a = b))$	assumption($r_2 = b$)
6	++++	$\exists i (Inside(a,i,d) \wedge Inside(b,i,d) \wedge \neg(a = b))$	assumption($t = d$)
7	+++++	$Inside(a,c,d) \wedge Inside(b,c,d) \wedge \neg(a = b)$	assumption($i = c$)
8	+++++	$Inside(a,c,d)$	$\wedge$E(7)
9	+++++	$Inside(b,c,d)$	$\wedge$E(7)
10	+++++	$\forall t \forall r (Inside(r,c,t) \rightarrow Alloc(r,c,t))$	$\forall$E(1, $i = c$)
11	+++++	$\forall r (Inside(r,c,d) \rightarrow Alloc(r,c,d))$	$\forall$E(10, $t = d$)
12	+++++	$Inside(a,c,d) \rightarrow Alloc(a,c,d)$	$\forall$E(11, $r = a$)
13	+++++	$Inside(b,c,d) \rightarrow Alloc(b,c,d)$	$\forall$E(11, $r = b$)
14	+++++	$Alloc(a,c,d)$	$\rightarrow$E(8,12)
15	+++++	$Alloc(b,c,d)$	$\rightarrow$E(9,13)
16	+++++	$Alloc(a,c,d) \wedge Alloc(b,c,d)$	$\wedge$I(14,15)
17	+++++	$\forall r_2 \forall i \forall t (Alloc(a,i,t) \wedge Alloc(r_2,i,t) \rightarrow (a = r_2))$	$\forall$E(2, $r_1 = a$)
18	+++++	$\forall i \forall t (Alloc(a,i,t) \wedge Alloc(b,i,t) \rightarrow (a = b))$	$\forall$E(17, $r_2 = b$)
19	+++++	$\forall t (Alloc(a,c,t) \wedge Alloc(b,c,t) \rightarrow (a = b))$	$\forall$E(18, $i = c$)
20	+++++	$Alloc(a,c,d) \wedge Alloc(b,c,d) \rightarrow (a = b)$	$\forall$E(19, $t = d$)
21	+++++	$a = b$	$\rightarrow$E(16,20)
22	+++++	$\neg(a = b)$	$\wedge$E(7)
23	+++++	$\perp$	$\neg$E(21,22)
24	++++	$\perp$	$\exists$E(6,7-23)
25	+++	$\perp$	$\exists$E(5,6-24)
26	++	$\perp$	$\exists$E(4,5-25)
27	+	$\perp$	$\exists$E(3,4-26)
28		$\neg \exists r_1 \exists r_2 \exists t \exists i (Inside(r_1,i,t) \wedge Inside(r_2,i,t) \wedge \neg(r_1 = r_2))$	C(3-27)

■

A.2 Avoid Planning Deadlocks

In Section 2.4, it is claimed that `MPM` avoids planning deadlocks, if "paths being planned are allowed to cross only one path obstacle at a time" (a consequence of

Rule 3 from the same section). This means, in other words, that there will be no edge in the connectivity graph between two cells in C_{free} which are separated by two path obstacles.

It is important to notice that even though a robot will never have its path planned (say at time t_0) across two adjacent path obstacles (i.e., it will never have a path planned that has two contiguous paths' intersections), it may subsequently end up with that because of paths planned later (at some time $t > t_0$) for other robots.

Also notice that for n robots the worst case is when all robots have paths (i.e., all robots are moving), since a non-moving robot will not contribute with even a single paths' intersection.

An induction proof proves this fact. Let n be the number of robots and $P(n)$ be the predicate $P(n)$: path planning with n robots is deadlock-free (see Figure 2.17).

The base case of the proof is $P(3)$[1]. Assume the worst case, i.e., that all three robots r_1, r_2 and r_3 are moving (have paths). Since robots are handled in sequence (Section 2.4), paths will be planned first for r_1 and subsequently for r_2. r_1 and r_2 may have many intersections, but at this point in time, the system is deadlock-free (since deadlock is not possible for just two robots). Now, even if the path for the third robot r_3 will be planned in such a way that both r_1 and r_2 get paths with contiguous intersections ($i_{1,3}$ and $i_{1,2}$, for instance, in Figure A.1), no deadlock will arise. The reason for this is that r_3 can not end up in a deadlock (it path was planned that way) and hence r_1 and r_2 will have no problems with $i_{1,3}$ and $i_{2,3}$ respectively.

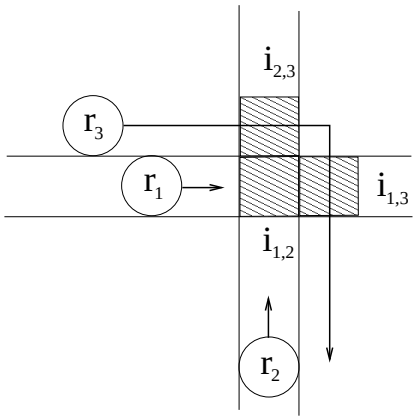


Figure A.1. Paths are planned for r_1 and r_2 first and then finally for r_3. r_3's path makes r_1's and r_2's paths have two contiguous intersections each. This however poses no problem since r_3 can not end up in a deadlock.

The induction step, which will show $P(n) \rightarrow P(n+1)$, concludes the proof. Assume $P(n)$, which means that paths have been planned for n robots without caus-

[1] $P(1)$ is not interesting since there can be no deadlock with just one robot. $P(2)$ can not have deadlocks either since a robot can not have two contiguous paths' intersections (which would require three robot paths for two robots which, of course, is impossible).

ing a potential deadlock. Let us now plan for a robot r_{n+1}. Its path may intersect many other paths and may result in all n other robots having two or more contiguous intersections, but it may not cross two adjacent path obstacles. Once again, at least robot r_{n+1} is deadlock-free and may travel to its destination, and since the other n robots also are deadlock-free (through the assumption in the induction step) $P(n+1)$ also holds. ■

Appendix B

Path Generation

In Chapter 2 and Chapter 3, a method MPM for motion planning is described. However, the method only reveals how a set of channels[1] is generated, it does not say how to generate a path for a robot.

For the experiments in this project, Algorithm 1 below was used. The algorithm assumes that a set of channels *chnls*, a start vertex v_s, and a destination vertex v_d are available. The cells are ordered as illustrated in Figure B.1. Let us furthermore assume that *chnls* is a data structure in which all cell edges $\{e_{i,j}\}_{i,j}$ (i.e., edge j in cell c_i) that have a *corresponding edge* have a reference to it. All edges also know which cell it belongs to.

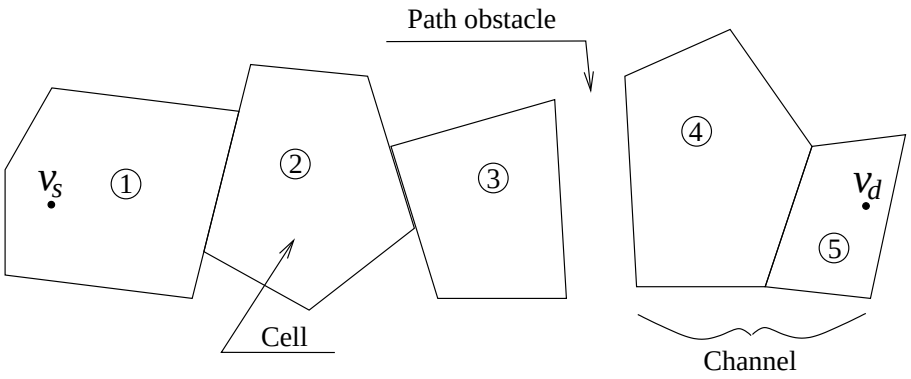


Figure B.1. The *chnls* variable contains a channel (or possibly many channels if there are path obstacles intersecting). The cells of *chnls* are numbered from the the cell containing v_s to the one containing v_d.

Corresponding edge Edge e_1 is said to have a corresponding edge e_2 if e_1 and e_2 either intersect or if they are on either side of a path obstacle. It is also required that e_1 and e_2 belong to different consecutive cells. Furthermore, e_1 is

[1] A channel is an ordered set of contiguous cells

also the corresponding edge of e_2. Hence, they are mutually corresponding. Figure B.2 illustrates the concept of corresponding edges.

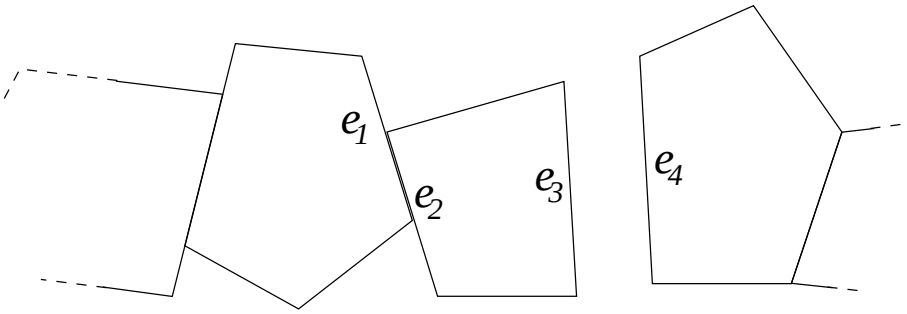


Figure B.2. Edges e_1 and e_2 are corresponding since they belong to two different and contiguous cells and intersect. Cells e_3 and e_4 are also corresponding since they are on either side of a path obstacle.

Algorithm 1 is greedy in the sense that it tries to make a path from v_s to v_d by connecting them with a straight line l. A variable *path* keeps the vertices of the path that is generated. If it is not possible to generate a straight path between v_s and v_d (i.e., if l intersects the channel boundary) the first cell *cell* which l exited the channel will be noted (line (6)). The intersection between the cell boundary between *cell* and $cell-1$ and l will be stored if $cell > currentcell$ (line (12)), where *currentcell* keeps the cell number of the vertex last stored in *path*. Also add the midpoint on the edge between the cells *cell* and $cell+1$ (line (14)). If however $cell \leq currentcell$ just add the midpoint between the cells *currentcell* and $currentcell+1$ (line (17)). If the last vertex added is on an edge next to a path obstacle, then the midpoint on the edge on the corresponding edge is also added (line (24)). Now, the algorithm once again uses the greedy strategy to connect the last vertex stored in *path currentpos* with v_d. This is repeated until v_d has been reached (line (4)). Figure B.3 displays an example of how the algorithm works.

Let's analyze the time complexity of Algorithm 1.

- INCELL$(chnls, v)$ in line (3) returns the number of the cell that contains v. Given a cell, it can be determined in $O(E)$, where E is the number of edges, if v is inside the cell. In the worst case we have to iterate through all cells C and so the total time complexity is $O(C \cdot E)$.

- INTERSECTEDCELL$(chnls, line)$ in line (6) returns the number of the first cell in which *line* intersects the channel boundary. In the worst case this function iterates through all cells. Checking if a cell is intersected by *line* is performed in $O(E)$ and so the total time complexity is $O(C \cdot E)$.

- CLOSESTINTERSECTION$(cell, line, v)$ in line (12) returns the vertex of the intersection between *cell* and *line*. If there is more than one intersection,

Algorithm 1: Heuristic path generation algorithm
Input: An ordered set $chnls = \{c_1, c_2, \cdots, c_n\}$, a start vertex v_s, and a destination vertex v_d.
Output: An ordered set of path vertices *path*.

```
PATH(chnls, v_s, v_d)
(1)   path ← {v_s}
(2)   currentpos ← v_s
(3)   currentcell ← INCELL(chnls, v_s)
(4)   while currentpos ≠ v_d
(5)       line ← (currentpos, v_d)
(6)       cell ← INTERSECTEDCELL(chnls, line)
(7)       if cell = NOCELL
(8)           path ← path ∪ {v_d}
(9)           currentpos ← v_d
(10)          continue
(11)      else if cell > currentcell
(12)          path ← path ∪ CLOSESTCROSSING(cell, line, currentpos)
(13)          currentpos ← MIDPOINT(cell, cell + 1)
(14)          path ← path ∪ currentpos
(15)      else
(16)          currentpos ← MIDPOINT(currentcell, currentcell + 1)
(17)          path ← path ∪ currentpos
(18)      if cell > currentcell
(19)          currentcell ← cell
(20)      else
(21)          currentcell ← currentcell + 1
(22)      if SEPARATECHANNELS(currentcell, currentcell + 1) = true
(23)          currentpos ← MIDPOINT(cell + 1, cell)
(24)          path ← path ∪ currentpos
(25)          return path
```

the one with the least distance to v will be returned. It is sufficient to iterate through all edges of *cell* to determine the closest intersection vertex and so the time complexity is $O(E)$.

- MIDPOINT(c_1, c_2) in lines (13), (16), and (23) returns the vertex that is the midpoint of the edge that separates c_1 and c_2. If c_1 and c_2 are separated by a path obstacle then the midpoint of c_1 will be returned. It is sufficient to iterate through all edges of c_1 to find the right edge (since it is assumed about the data structure that each edge knows its corresponding edge and which cell it belongs to). Hence, the time complexity is $O(E)$.

- SEPARATECHANNELS(c_1, c_2) in line (20) returns **true** if two cells c_1 and

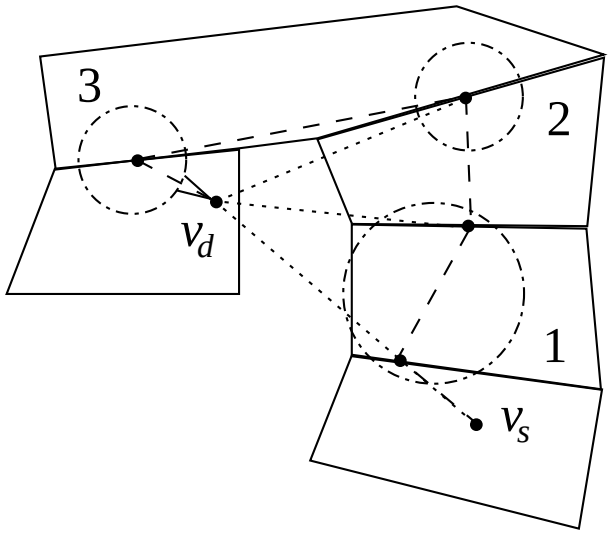


Figure B.3. In the first loop of Algorithm 1, the two vertices inside circle 1 are the result. In the second loop, the vertex in circle 2 is added to the path. In the fourth loop, v_d is reached and the path completed. The dotted lines illustrate the greedy strategy of the algorithm (that it wants to find a straight path to the destination v_d with each iteration). The dashed line show the resulting robot path.

c_2 are separated by a path obstacle (and only by the path obstacle). It is sufficient to iterate through all edges to determine this, and hence the time complexity is $O(E)$.

To complete this time complexity analysis, it has to be determined how many times the while-loop in lines (4)-(23) is run. It is run until $currentpos = v_d$. Since *currentpos* is updated with each iteration so that it is at least on the border of the next cell, no more than C iterations are necessary.

The lines (1)-(3) are run only once and contribute with $O(1+1+C\cdot E) = O(C\cdot E)$. One loop in the while-loop contributes with $O(1+C\cdot E+1+1+1+E+E+1+E) = O(C\cdot E)$. Since the while loop is iterated $O(C)$ times the total time complexity is $O(C\cdot E+C\cdot(C\cdot E)) = O(C^2\cdot E)$.

It is a bit awkward to have C in the time complexity expression since it is a variable that is hard to determine at the time of workspace design. E, on the other hand, directly relates to the workspace design, since it denotes the number of edges in the workspace. According to [Sleumer, Tschichold-Gürman, 1999], the number of cells in a decomposition of arrangements is E^2. Hence, C may be replaced with E^2 in the time complexity expression resulting in $O(E^5)$. Not surprisingly it is useful to minimize the number of edges used to represent robots and obstacles in order to improve the running time.

The generated path can be improved further. Figure B.4 shows that the path straight from vertex v_k to v_{k+2} is shorter than the path that goes through v_k, v_{k+1}

and v_{k+2}. Hence, the path can be improved by removing v_{k+1}. Special care has to be taken at paths' intersections.

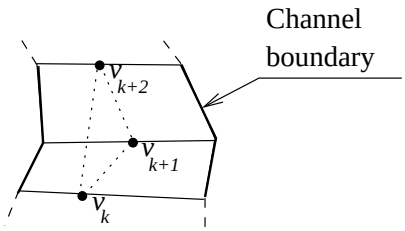


Figure B.4. The intermediate vertex v_{k+1} on the robot path $\{v_k, v_{k+1}, v_{k+2}\}$ may be removed, since the straight path from v_k to v_{k+2} does not exit the channel boundary and is shorter.

Appendix C

Glossary

This chapter explains some of the terms used in this thesis. Some explanations include words with *italic* format, which can also be found in this glossary.

Channel In the path planning method cell decomposition, a channel is a sequence of contiguous cells of *free space*.

Complete A method is said to be complete if it always produces a solution when one exits.

Configuration space A tool used to simplify path planning. Instead of planning for a robot with a (possibly complex) shape in a *workspace*, a path can be planned through a space of valid configurations of the robot.

Connectivity graph In the path planning method cell decomposition, a connectivity graph is a computational graph, which has cells of a decomposition as nodes, and where an edge between two nodes indicates that the two corresponding cells are consecutive.

Deadlock A situation which involves at least two robots. Each robot waits for some other robot to release a resource (e.g., a tool or space) it needs before it can continue. In the deadlock, none of the robots are willing to release the resources they have, and the robots wait forever.

Forbidden area A forbidden area is a part of the *workspace*, in which mobile robots are not allowed to enter. The word "obstacle" is sometimes used in this thesis to denote forbidden area.

Free space The space of a *workspace* or a *configuration space* which is not occupied by some obstacle.

GUI Acronym for "Graphical User Interface". A GUI is an interface to a program for an operator.

Holonomic A holonomic robot has the same number of controllable degrees of freedom as it has actual degrees of freedom. A holonomic robot is normally more easy to control than a non-holonomic robot, for which the numbers of degrees of freedom differs. A car-like robot is an example of a non-holonomic robot.

Localization The process of determining a robot's position.

Motion planning The process of generating a sequence of actions, that has to be performed in order for a robot to move through its *workspace* without collisions. *Path planning* may be a part of the motion planning.

Multi-robot system A system of more than one robot.

MVC Acronym for "Model-View-Control". A programming architecture, which encourages programmers to separate data, display of data and manipulation of data in their applications.

Omni-directional A robot which can move in any direction in the plane and rotate on the spot.

Path obstacle A special type of obstacle which belongs to a robot *r* (with a planned path), and which shape is equal to the space occupied by *r* on the rest of its planned path. Since a path obstacle is not solid another, robot may cross it.

Path planning The process of planning a path for a robot through a known *workspace*.

Request generator Generates requests for robots. A robot has to go to the request generator to handle the request.

Rotation invariant In this motion planning context, a robot *r* is rotation invariant if a decomposition of *r*'s free space is independent of *r*'s orientation (rotation).

Semi-free path A path which is collision-free, but a robot which travels on it may stand the risk of touching obstacles along the way.

Sound A method is said to be sound if all its produced solutions are correct.

Supervisor agent In this context, a computational agent that is responsible for *motion planning*.

Workspace Some space in which a robot may move and operate. A workspace may, in this project, contain robots, *request generators* and *forbidden areas*. In this thesis, the more general word "environment" is also used to denote workspace.

Bibliography

Alami et al, 1997. ALAMI, R., INGRAND, F., QUTUB, S., 1997, "Planning Coordination and Execution in Multi-robots Environments", *8th International Conference on Advanced Robotics, ICAR '97*, pp. 525-530.

Arai, Ota, 1992. ARAI, T., OTA, J., 1992, "Motion Planning of Multiple Mobile Robots", *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1761-1768.

Arkin, 1998. ARKIN, R.C., 1998, *Behavior-Based Robotics*, MIT Press, England, ISBN 0-262-01165-4.

Asama et al., 1995. ASAMA, H., SATO, M., BOGONI, L., KAETSU, H., MATSUMOTO, A., ENDO, I, 1995, "Development of an omni-directional Mobile robot with 3 DOF Decoupling Drive Mechanism", *IEEE International Conference on Robotics and Automation 1995*, Vol. 2, pp. 1925-1930.

Borenstein. BORENSTEIN, J., *The OmniMate: A Guidewire- and Beacon-free AGV for Highly Reconfigurable Environments*, `http://www-personal.engin.umich.edu/~johannb/OmniMate.htm`, latest visit in May 2000.

Borenstein, 1996. BORENSTEIN, J., EVERETT, H.R., FENG, L, 1996, *Navigating Mobile Robots - Systems and Techniques*, A K Peters, Wessley, MA, ISBN 1-56881-058-X.

Choset, Pignon, 1998. CHOSET, H., PIGNON, P., 1998, "Coverage Path Planning: The Boustrophedon Cellular Decomposition", *Field and Service Robotics*, Springer-Verlag, pp. 203-209.

Coffman et al, 1971. COFFMAN, E.G., ELPHICK, M.J., SHOSHANI, A., 1971, "System Deadlocks," *ACM Computing Surveys*, Vol. 3, pp. 67-78 (1971).

Everett, 1995. EVERETT, H.R., 1995, *Sensors for Mobile Robots - Theory and application*, A K Peters, USA, ISBN 1-56881-048-2.

Fiorini, Shiller, 1995. FIORINI, P., SHILLER, Z., 1995. "Motion planning in Dynamic Environments using Velocity Obstacles", *International Journal of Robotics Research*, Sage Publications, Vol. 17, No. 7, July 1998, pp. 760-772.

Fujimura, 1991. FUJIMURA, K., 1991, *Motion Planning in Dynamic Environments*, Springer-Verlag, Hong Kong.

Ghosray, Yen, 1996. GHOSRAY, S., YEN, K.K., 1996, "A Comprehensive Robot Collision Avoidance Scheme by Two-Dimensional Geometric Modeling", *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, IEEE Comput.Soc.Press, 1996, pp. 1087-1092.

Gill, Zomaya, 1998. GILL, M., ZOMAYA, A., 1998, *Obstacle Avoidance in Multi-Robot Systems*, World Scientific Publishing Co., Singapore, ISBN 9810234236.

Habib, Asama, 1991. HABIB, M., ASAMA, H., 1991, "Efficient Method to Generate Collision Free Paths for Autonomous Mobile Robot Based on New Free Space Structuring Approach", *IEEE/RSJ International Workshop on Intelligent Robots and Systems 1991*, pp. 563-567.

Kant, Zucker, 1988. KANT, K., ZUCKER, S., 1988, "Planning Collision-Free Trajectories in Time-Varying Envrionments: A Two-Level Hierarchy", *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, IEEE Comput.Soc.Press, 1988, pp. 1644-1649.

Kleeman, 1992. KLEEMAN, L., 1992, "Optimal Estimation of Position and Heading for Mobile Robots Using Ultrasonic Beacons and Dead-Reckoning", *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, IEEE Computer Press Society, Vol. 3, pp. 2582-2587.

Latombe, 1993. LATOMBE, J.C., 1993, *Robot Motion Planning*, Kluwer Academic Publishers, USA, ISBN 0-7923-9206-X.

Laugier et al, 1998. LAUGIER, C., GARNIER, PH., FRAICHARD, TH., PAROMTCHIK, I., SCHEUER, A., "Motion Planning and Sensor-Guided Maneuvre Generation for an Autonomous Vehicle", *Field and Service Robotics*, Springer Verlag, 1998, pp. 60-67.

Lingas, 1982. LINGAS, A., 1982, "The power of non-rectilinear holes", *Proceedings of the 9th International Colloquium on Automata, Languages and Programming*, LNCS Springer-Verlag, pages 369-383.

Liu et al, 1989. LIU, Y.H., KURODA, S., NANIWA, T., NOBORIO, H., ARIMOTO, S., "A Practical Algorithm for Planning Collision-Free Coordinated Motion of Multiple Mobile Robots", *Proceedings 1989 Conference on Robotics and Automation*, IEEE Comput.Soc.Press, Vol. 3, pp. 1427-32.

Lozano-Pérez, 1983. LOZANO-PÉREZ, T., "Spatial planning: A configuration space approach", *IEEE Tr. Computers*, C-32(2), 108-120.

Noborio, Yoshioka, 1994. NOBORIO, H., YOSHIOKA, T., 1994, "On a Deadlock-free Characteristic of the On-line and Decentralized Path-planning for Multiple Automata", *Distributed Autonomous Robotic Systems*, Springer-Verlag, pp. 111-122.

Norvig, Russell, 1995. NORVIG, P., RUSSELL, S., 1995, *Artificial Intelligence - a modern approach*, Prentice Hall, USA, ISBN 0-13-360124-2.

Parker, 1995. PARKER, L.E., 1995, "Multi-Robot Team Design for Real-World Applications", *Distributed Autonomous Robotic Systems 2*, Springer Verlag, pp. 91-102.

Qutub et al, 1997. QUTUB, S., ALAMI, R., INGRAND, F., 1997, "How to Solve Deadlock Situations within the Plan-Merging Paradigm for Multi-robot Cooperation", *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'97)*, Vol. 3, pp. 1610-1615.

Rausch, Levi, 1996. RAUSCH, W.A., LEVI, P., 1996, "Asynchronous and Synchronous Cooperation - Demonstrated by Deadlock Resolution in a Distributed Robot System", *Distributed Autonomous Robotic Systems 2*, Springer-Verlag, pp. 245-256.

Sleumer, Tschichold-Gürman, 1999. SLEUMER, TSCHICHOLD-GÜRMAN, 1999, *Exact Cell Decomposition of Arrangements used for Path Planning in Robotics*, http://www.inf.ethz.ch/publications/abstract.php3?no=tech-reports/3xx/329, latest visit in October 2000.

Tsoularis, Kambhampati, 1999. TSOULARIS, A., KAMBHAMPATI, C., 1999, "Avoiding Moving Obstacles by Deviation from a Mobile Robot's Nominal Path", *The International Journal of Robotics Research*, Sage Publications, Vol. 18, No. 5, pp. 454-465.

Uny Cao et al., 1997. UNY CAO, Y., FUKUNAGA, A.S., KAHNG, A.B., "Cooperative Mobile Robotics: Antecedents and Directions", *Autonomous Robots*, *Kluwer Academic Publishers*, Vol. 4, 1997, pp. 7-27.

Yuta, Premvuti, 1992. YUTA, S., PREMVUTI, S., "Coordinating Autonomous and Centralized Decision Making to Achieve Cooperative Behaviors Between Multiple Mobile Robots", *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1566-1574.