Thesis for the Degree of Doctor of Philosophy

Robot Path Planning

Robert Bohlin

CHALMERS | GÖTEBORG UNIVERSITY

Department of Mathematics
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden
Göteborg, May 2002

Robot Path Planning
Robert Bohlin
ISBN 91-7291-168-9

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 1850
ISSN 0346-718X

Department of Mathematics
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31-772 1000

Abstract

This thesis consists of three papers concerned with the basic path planning problem for robots moving in a known static environment. Our main interest has been industrial robots, but the methods are general and apply to a wide range of robots. The path planning problem is to find a sequence of configurations that moves a robot from an initial configuration to a goal configuration without colliding with obstacles in the environment.

The first paper presents a variation of the Probabilistic Roadmap Method (PRM). The new planner is called Lazy PRM and is tailored for single query path planning. By introducing a scheme for lazy evaluation, the pronounced multiple query planner is converted into an efficient single query planner.

The second paper presents a resolution complete, de-randomized version of Lazy PRM. The planner uses an implicit, non-uniform grid that allows local refinement to represent the configuration space.

The third paper presents a novel potential field method for free-flying rigid bodies. The planning is performed directly in the group $SE(3)$ and is reinforced by a potential function in the workspace. Thus, the planner benefits from the explicit representation of the workspace obstacles at the same time as the planning takes place in the 6-dimensional configuration space. The potential function is harmonic and is composed of translates of the Green kernel in $SE(3)$.

Experimental results provided show that the planners are capable of solving relevant problems in various environments.

Keywords: Collision avoidance, Green kernel, fundamental solution, harmonic function, motion planning, path planning, potential field, probabilistic roadmap, randomized algorithm, robotics.

AMS 2000 Mathematics Subject Classification: 22E70, 34B27, 58J90, 65C99, 68W20

This thesis consists of an introduction and the following papers

Paper I: R. Bohlin and L.E. Kavraki. A lazy probabilistic roadmap planner for single query path planning
This paper is a combination of the papers

- R. Bohlin and L.E. Kavraki. Path Planning Using Lazy PRM. In Proc. *IEEE International Conference on Robotics and Automation*, 2000
- R. Bohlin and L.E. Kavraki. A Randomized Algorithm for Robot Path Planning Based on Lazy Evaluation. *Handbook on Randomized Computing*, S. Rajasekaran and P. Pardalos and J. Reif and J. Rolim Editors, Kluwer Academic Publishers, 2001

Paper II: R. Bohlin. Path planning in practice; lazy evaluation on a multi-resolution grid. In Proc. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001

Paper III: R. Bohlin. Rigid body path planning using the Green kernel in SE(3)

Acknowledgements

First and foremost, I thank my advisor Bo Johansson for all the support and encouragement he has given me throughout the work. Without his enthusiasm, comments and patience when listening to my sometimes confused ideas, this work would not have been completed.

I also express my gratitude to the people at ABB Robotics who supported this research. Special thanks goes to Henrik Berlin, Pernilla Johansson, Pär Oskarsson, Lars Östlund, Anders Ekelund and Martin Härnquist.

During my studies I had the great opportunity to visit the Physical Computing Group at Rice University, and I am deeply indebted to Lydia Kavraki. Working with Lydia has been a pleasure, and her comments on various manuscripts have been invaluable for the development of Lazy PRM.

I would also like to thank the Robotics Group at Oxford University Computing Laboratory, with whom I had the opportunity to spend three months. Particularly, I give my gratitude to Stephen Cameron and Joe Pitt-Francis for many fruitful discussions.

Many colleagues at the department have made me enjoy my time as a Ph.D. student. In particular my thanks go to Tobias Adolfsson, Henrik Berlin, Jan Rohlén and Magnus Oskarsson, but many others have contributed to a nice working environment. I am also grateful to Peter Sjögren, Grigori Rozenblioum and Bo Berndtsson for their help on Green kernels.

Finally, I wish to thank my wife Petra for her continuous support during this work, and for being an unlimited source of joy and inspiration.

Göteborg, May 2002

Robert Bohlin

Robot Path Planning
An introduction

Robert Bohlin
Department of Mathematics
Chalmers University of Technology
SE-412 96 Göteborg, Sweden

This thesis deals with the basic path planning problem for robots moving in a known static environment. Our main interest has been industrial robots, but the methods are general and also apply to a wide range of other robots. The path planning problem is to find a sequence of configurations that moves a robot from an initial configuration to a goal configuration without colliding with obstacles in the environment. Our aim has been to develop algorithms capable of solving path planning problems that frequently occur in industry. Automatic path planners are essential to autonomous robot systems and increase the efficiency and applicability of off-line programming systems.

The mathematical aspects of path planning are of great variety. The relationship with optimization comes immediately into one's mind. There are, however, major differences. One way to see the path planning problem is to consider the initial configuration as a starting point, and then try to reach a global minimum located at the goal configuration. In robotics, such techniques are known as potential field methods, but in contrast with traditional optimization our interest is not the optimum solution itself, but rather the path that leads to the optimum.

Another way to formulate the path planning problem is to optimize over all paths that connects the initial and the goal configurations. The objective function should penalize colliding paths and promote short, smooth paths, see [2]. Such a variational formulation generally leads to a highly non-convex optimization problem. Owing to the complexity of the path planning problem, our primary interest is to find *some* feasible path. Finding an optimal solution is far too difficult in most cases.

To deal with the complexity issue, approximate methods are most useful in practice. Randomized techniques have gained much interest, and in particular the Probabilistic Roadmap Method (PRM), see [23, 24, 36]. Paper I of this thesis further develops this method by introducing a planner called

Lazy PRM. The planner uses a lazy evaluation technique in order to reduce costly collision detection.

Paper II presents a de-randomized version of Lazy PRM. Instead of using an explicit representation of a random network, the planner uses an implicit grid. The grid is not necessarily uniform, but can be refined locally as the planner proceeds. The work in Paper II was done independently of the work presented in [6], although the methods have strong similarities. The planners in [6] use quasi-random numbers to create lattices. Depending on what quasi-random numbers are used, different lattices are generated. The lattices can be represented implicitly and, in a special case, forms a regular grid. A deeper discussion of randomized versus deterministic techniques can be found in [7].

Paper III introduces a potential field method for planning for a rigid body moving freely in space. The novel approach uses Green kernels in order to create a harmonic potential function that guides the robot towards the goal. Mathematical tools for a theoretical foundation are developed that open up for further development in various directions. The technique is still premature, but our implementation and test examples show good performance and looks promising for future improvements.

The remaining pages of this introduction give a short background of robotics, describe some applications and different kinds of robots, and motivate the use of automatic path planners. Section 3 introduces some notations and defines the problem. Some variations of the problem are given in Section 2.1.

1 History and applications

The use of robots in a large scale started in industry in the 1960's and has grown continuously ever since. Initially, robots were mainly used for handling materials, positioning, and for simple assembly tasks. The main objectives were to relieve humans from repetitive or tedious tasks, work in hazardous environments, and physically exacting work.

The development was fast and soon robots started to replace humans where extreme precision or high repeatability was needed. Robots also became easier to program, became more flexible, and could solve more complex tasks. Spray painting and spot welding, for instance, were applications well suited for being taken over by robots, see [34].

Mobile robots, known as automated guided vehicles (AGVs), were developed in the 1970's in order to transport material in factories. By following guide wires buried in the floor, the robots were able to navigate and position themselves. A decade later wireless navigation systems started to appear. Laser sensors and beacons increased the flexibility and made it easier to alter the map of predetermined paths.

Today robot applications have spread far beyond manufacturing industry. Modern forestry, agriculture and construction industry have all been influenced by robotics. See for example [34] for a sheep-shearing robot! Medical robots assist surgeons and even perform remotely controlled operations, see [21]. Also in our homes one can find autonomous cleaning robots and robots cutting the lawn, see [37]. As can be seen from these applications, a robot can be almost anything that moves in a controlled manner.

The examples until now have been robots that exist in the physical world. However, extending our view a bit more, there is an endless number of virtual robots with similar properties, see [29]. Movies and commercials with animated characters are getting more and more common. These characters, as well as vehicles, humans and animals in computer games, can be seen as robots moving along specified paths and acting according to a certain pattern. See for example [18, 22, 25, 26] for articles related to path planning. Moreover, in pharmaceutical drug design, interactions between molecules are essential. For instance, the docking of ligands inside protein cavities can be simulated; see [11] for an overview of computer-aided drug design.

Closing the circle by going back to where we started – to manufacturing industry – we find still more virtual robots. Long before a product is being manufactured, its assemblies are designed with computer aid, see [20]. Virtual prototyping and assembly planning shortens the development time for new products, and production planning and simulation increase the efficiency in factories. All parts that move when a process is simulated can be seen as robots. Furthermore, an increasing number of industrial robots are being programmed *off-line*. That is, instead of programming the physical robot in a noisy workshop, programs are written in a comfortable office by using a virtual copy of the robot and its environment. Programs can be edited, simulated and verified before they are transfered to the physical robot. All this can be done without interfering the flow in the production line.

In the examples mentioned so far, the robots have been single rigid objects or a collection of rigid objects, called links, that are connected to each other by joints. The atoms and bindings of molecules, for instance, can be modeled by rigid pieces connected to each other by joints. However, robotics also concerns robots that have an infinite number of joints, see [13] for an overview. Flexible parts like sheet metal, clothing, and human tissues are interesting to consider, see [1, 27]. Recent work in [12] also treats snake-like robots.

In spite of the great variety of robots and applications, the basic concept is the same; program some robot (real or artificial) to act such that a certain task is accomplished. On a lower level, the essential operation it all comes down to is to find a sequence of robot motions such that the task is solved. During the motions certain constraints must be satisfied.

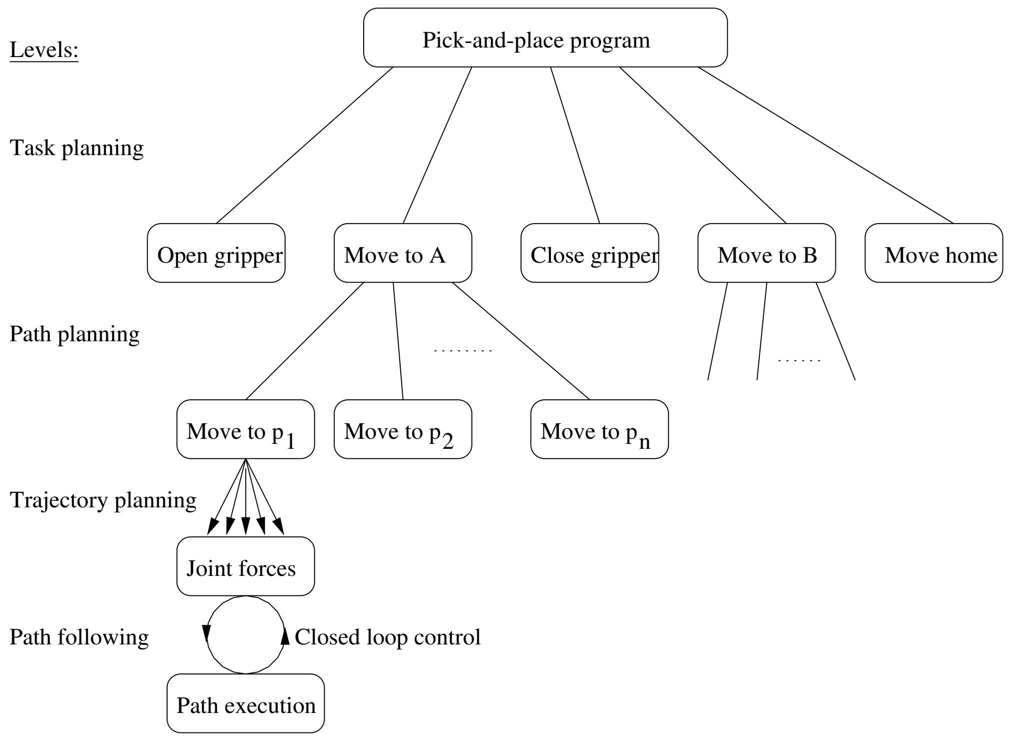


Figure 1: A simple pick-and-place program and its levels.

To illustrate the components of a robot program and give an idea of how a task is divided into pieces, we will use a simple example. Consider a robot with a gripper that shall move an object from location A to location B. On the task level, the program is divided into a sequence of actions, see Figure 1. Some of these actions involve motions from an initial configuration to a goal configuration. Typically, the procedure from planning to execution of a motion looks as follows. Presumably, the straight-line path from start to goal is impossible to follow due to obstacles. Then a detour, specified by a sequence of via-points ($p_1, \ldots, p_n$ in Figure 1), must be found in order to pass the obstacles. This step is called *kinematic path planning* or just path planning. Kinematic constraints, like how robot links are related to each other and minimum turning radius for mobile robots, must also be taken into account.

The second step is to calculate a *trajectory*, or a motion plan, and involves the equations describing the dynamics of the robot. A desired velocity along the path is determined and a plan is calculated that specifies the forces and torques of the actuators. If limits of joint forces are exceeded, the velocity profile along the path needs to be reconsidered. The trajectory may be pre-calculated or determined in real time.

The third step is the execution of the trajectory and is sometimes called path following. This is handled by a control unit or a simulation device and is performed without human interaction. In general closed loop control with sensor feedback is needed. If the trajectory is calculated in real time, the second and third steps may be merged together.

Optimal control theory has been extensively used in the second step in order to minimize the execution time for a path while the constraints on joint forces are satisfied, see [34]. Time-optimal trajectories has, however, limited practical use. Typically the solution requires bang-bang control, that is, at all time during execution, the force on at least one actuator is maximal. The input switches instantaneously between the limits, hence there are discontinuous jumps in the actuator forces. In practice such ideal forces cannot be produced, and even getting close to such a control scheme would prematurely wear out the robot.

2 Automatic path planning

The human brain is extremely talented regarding spatial path planning. Nevertheless, a tremendous amount of time is spent on manual path planning. In many cases the programmer must specify complicated paths with long detours containing a large number of via-points. This makes robot programming time consuming and sometimes tedious.

Although robot programming is performed off-line in modern industry, the programming procedure is basically the same as it was 30 years ago – to manually teach the robot how to move by specifying all locations along a path. The difference is that a computer model is used instead of the real robot. Thus, programmers still need to focus on the low level *motions* of the robot. It would be more natural to raise the level of abstraction and focus on *what* the robot shall do instead of *how* to do it. An essential component in software that better supports the robot programmer is a *path planner*. Off-line programmers and users in many other applications mentioned in previous section could benefit a lot from automatic path planning.

Automatic path planning deals with the problem of finding a robot path from an initial configuration to a goal configuration, or determining that no such path exists. Depending on the robot and the environment, various constraints may impose additional requirements on the motion. The most obvious constraints come from obstacles that possibly are located in the workspace. The path must be selected to assure that the robot does not collide with the obstacles.

Kinematic constraints must also be satisfied. A robot may consist of several links that are joined together. If the links are formed in a loop, the loop must be maintained all along the path. Kinematic constraints are called holonomic if they can be eliminated in some way by changing the parameterization the robot. On the contrary, non-holonomic constraints cannot be eliminated and typically occur for mobile robots. Some vehicles may not be able to move sideways, but only drive forward or backwards and turn with limited radius. Tractor-trailer systems may have even more complicated non-holonomic constraints.

In the last decades, the path planning problem has been extensively studied, and a large number of different approaches have been proposed. The introductory parts of Paper I, Paper II, and Paper III give brief overviews of randomized planners, grid-based planners, and potential field planners respectively. For other techniques, we refer to [14, 17, 28].

2.1 Variations of the path planning problem

A challenging task is to coordinate several robots moving and possibly interacting in the same workspace, see [31, 39, 41, 43, 44]. Naturally the complexity grows rapidly with the number of robots. To reduce the complexity, various levels of decentralized planning are currently the methods of most interest. The general idea is to first plan for each robot separately and independently of the other robots. Then the paths must be modified and coordinated in order to avoid collisions and to optimize some global performance measure.

A variation of robot coordination is assembly planning, which deals with the problem of finding motions that put an assembly together, see [15]. Typically, fine tolerances are involved, and a solution may require that the parts are in contact with each other during the motion. Another special case of robot coordination is known as manipulation planning. Then some of the obstacles are movable and may be manipulated in order to accomplish a task, see [10, 47].

The environment is sometimes unknown and needs to be explored in order to detect the obstacles. The obstacles may also change or move in an unpredictable way, see [33]. Uncertainties in positioning and sensing are other aspects that must be considered in practice. A unifying framework that covers several of these variations of the path planning problem is studied in [30].

3 Configuration space

A robot may consist of one or more rigid bodies moving in a workspace denoted by $\mathcal{W}$. Each rigid body is called a *link* and has a coordinate system attached to it. The placement of each link is described by a rigid body transformation with respect to a fixed coordinate frame $F_{\mathcal{W}}$ in $\mathcal{W}$. In the case of n links, their locations are given by a point in the n-fold product of the space of rigid body transformations.

If some of the links are connected by joints, or if other kinematic constraints must be satisfied, the possible locations of the links are restricted. The subset $\mathcal{C}$ of the n-fold product space for which all these constraints are satisfied is called the *configuration space* of the robot. A point in $\mathcal{C}$ is called a *configuration.* The dimension d of $\mathcal{C}$ is equal to the number of degrees of freedom (dof) of the robot. The use of configuration space for path planning was originally introduced in [32].

With the definition just given, the configuration space is not necessarily connected. That is, there may be configurations in $\mathcal{C}$ that are not reachable without first violating some constraints. See Figure 2 for a simple example of a planar linkage with two separate components of $\mathcal{C}$. For further reading, see for example [35, 45]. The set of unreachable configurations will not be

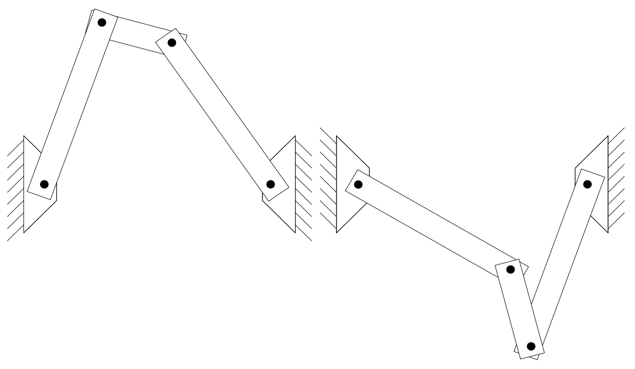


Figure 2: An example of a planar robot whose configuration space have two disjoint components.

of our interest, so we simply exclude it from the configuration space. Thus, from now on, the configuration space $\mathcal{C}$ is one connected component.

For many robots it is possible to parameterize $\mathcal{C}$ in a convenient way. Consider an articulated robot arm. Given the position of the first link, the position of the second link can be determined from the state of the joint connecting them, and so on. Thus, a configuration of a robot arm with d distinct joints can be specified with d joint values. The space of joint values, typically a subset of $\mathbf{R}^d$, can be seen as a local coordinate system on $\mathcal{C}$.

It is often convenient to identify the configuration space with a subset of $\mathbf{R}^d$. However, this must be done carefully since the standard topology in $\mathbf{R}^d$ may be completely different from the topology of $\mathcal{C}$. A single rigid body, for instance, has six degrees of freedom but the space of rigid body transformations is not homeomorphic to $\mathbf{R}^6$, see Paper III. Consequently, $\mathbf{R}^d$ is not always a good choice for parameterizing $\mathcal{C}$. See also [46].

Consider the end-effector of an articulated robot arm. It may reach the same position in $\mathcal{W}$ in several ways; in an ”elbow up” or in an ”elbow down” pose. Moreover, when two joint axis coincide, the same position can be reached in an infinite number of ways. Thus, to to avoid ambiguity, a path must be specified in the configuration space. Therefore it is in general preferable to consider the planning problem in the configuration space, where each robot configuration is uniquely defined by a point.

The mapping from the configuration space $\mathcal{C}$ to the work space $\mathcal{W}$ is called forward kinematics and the mapping in the other direction is called inverse kinematics. Owing to the redundancy, there may be singularities in the multi-valued mapping from $\mathcal{W}$ to $\mathcal{C}$. As a result, a smooth path in $\mathcal{C}$ generally gives a smooth path in $\mathcal{W}$, whereas the converse is not always true. Typically, the forward kinematics is easy to calculate whereas the inverse mapping is more difficult and, as just pointed out, may give several solutions. There are, however, robots for which the inverse kinematics is easy to calculate and the forward kinematics is more intricate, see e.g. [42].

3.1 Complexity

Several important results regarding the complexity of the path planning problem have been derived. As in many other disciplines, it is very large step from a 2-dimensional workspace to a 3-dimensional workspace. For example, the shortest path for a point among polygonal obstacles can be found in $O(n \log n)$, where n is the number of vertices of the obstacles, see [16]. This seemingly easy problem is much more complex in three dimensions; the shortest path for a point in a polyhedral world is proven to be NP-hard, see [8, 9].

However, not all problems in 2-dimensional workspaces with polygonal obstacles are easy to solve. The path planning problem for a robot arm with polygonal links connected by revolute joints is PSPACE-hard, see [19]. In this case, as in most path planning situations, the essential measure of the size of the problem is the number of degrees of freedom of the robot.

One family of path planning problems is called the *generalized mover's problem*. This family includes, for example, planning for one or more rigid bodies moving freely in space, and for robot arms. Reif showed in [38] that also this family belongs to the class of PSPACE-hard problems. Complete algorithms do exist, but they are rarely used in practice due to their computational complexity, see [8, 40]. It is believed that any complete algorithm requires time exponential in the number of degrees of freedom of the robot. This complexity bound is obtained by the complete algorithm in [8].

4 Outline and contributions of the thesis

This thesis consists of three papers concerning the basic robot path planning problem in a known static environment. We restrict our attention to robots with a finite number of degrees of freedom and holonomic kinematic constraints. In this short introduction we do not want to go too deep into the details of the separate papers, but we wish to briefly outline their contents and point out the main contributions of this work.

Paper I is a combination of the two papers [4] and [5], and presents a variation of the Probabilistic Roadmap Method (PRM). The new planner is called Lazy PRM and is tailored for single query path planning. The main contributions of the first paper are:

- The concept of lazy evaluation in order to reduce collision checking
- Further development of node enhancement techniques for PRM
- A proof of probabilistic completeness for Lazy PRM.

By introducing lazy evaluation, the pronounced multiple query planner is converted into an efficient single query planner. Experimental results provided show that the planner is capable of solving relevant problems in an industrial environment.

Paper II presents a resolution complete, de-randomized version of Lazy PRM. The main contributions of the second paper are:

- An implicit representation of the roadmap
- A non-uniform grid that allows local refinement.

The planner is compared with Lazy PRM and the experiments indicate that the grid representation gives better performance.

Paper III presents a novel potential field planner that uses Green kernels. The planner is tailored for rigid body path planning. The main contributions of the third paper are:

- Path planning directly in the group $SE(3)$
- Explicit formulas for the Green kernel in $SE(3)$ and $S^3 \times \mathbf{R}^3$
- Application of harmonic functions in high-dimensional configuration spaces.
- Combination of sampling and potential field methods.

References

[1] E. Anshelevich, S. Owens, F. Lamiraux, and L. Kavraki. Deformable volumes in path planning applications. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 2000.

[2] J. Barraquand and P. Ferbach. Path planning through variational dynamic programming. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1839–1846, San Diego, CA, 1994.

[3] R. Bohlin. Path planning in practice; lazy evaluation on a multi-resolution grid. In *Proc. IEEE/RSJ Int. Conf. on Int. Rob. and Syst.*, 2001.

[4] R. Bohlin and L.E. Kavraki. Path planning using Lazy PRM. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 2000.

[5] R. Bohlin and L.E. Kavraki. A randomized algorithm for robot path planning based on lazy evaluation. In S. Rajasekaran, P. Pardalos, J. Reif, and J. Rolim, editors, *Handbook on Randomized Computing*, pages 221–249. Kluwer Academic Publishers, 2001.

[6] M.S. Branicky, S.M LaValle, K. Olson, and L. Yang. Quasi-randomized path planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 2001.

[7] M.S. Branicky, S.M. LaValle, K. Olson, and L. Yang. Deterministic vs. probabilistic roadmaps. *submitted to IEEE Transactions on Robotics and Automation*, 2002.

[8] J.F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, MA, 1988.

[9] J.F. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *Proc. 28th IEEE Symp. on Found. of Comp. Sci.*, pages 49–60, 1987.

[10] P. Ferbach and J. Barraquand. A method of progressive constraints for manipulation planning. *IEEE Tr. on Rob. & Aut.*, 13(4), 1997.

[11] P. Finn and L. Kavraki. Computational approaches to drug design. *Algorithmica*, 25:347–371, 1999.

[12] I.A. Gravagne and I.D. Walker. Ellipsoid analysis for planar continuum robots. *IEEE Tr. on Rob. & Aut.*, to appear.

[13] K. Gupta. Motion planning for flexible shapes (systems with many degrees of freedom): a survey. *Visual Computer*, 14(5-6):288–302, 1998.

[14] K. Gupta and A.P. del Pobil. *Practical Motion Planning in Robotics.* John Wiley, West Sussex, England, 1998.

[15] D. Halperin, J.C. Latombe, and R.H. Wilson. A general framework for assembly planning: The motion space approach. *Algoritmica*, 26:577–601, 2000.

[16] J. Hershberger and S. Suri. Efficient computation of Euclidean shortest paths in the plane. In *Annual Symp. on the Foundations of Computer Science*, 1993.

[17] Y.K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Comp. Surveys*, 24(3):219–291, 1992.

[18] J.J. Kuffner, Jr., K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots under obstacle and dynamic balance constraints. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 2001.

[19] D.A. Joseph and W.H. Plantinga. On the complexity of reachability and motion planning questions. In *ACM Symp. on Computational Geometry*, pages 62–66, 1985.

[20] L. Joskowicz and E. Sacks. Computer-aided mechanical design using configuration spaces. *Computing in Science & Engineering*, 1, 1999.

[21] L. Joskowicz and R.H. Taylor. Computers in imaging and guided surgery. *Computing in Science & Engineering*, 3, 2001.

[22] M. Kalisiak and M. van de Panne. A grasp-based motion planning algorithm for character animation. *Journal of Visualization and Computer Animation*, 12(3):117–129, 2001.

[23] L.E. Kavraki and J.C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1994.

[24] L.E. Kavraki, P. Švestka, J.C. Latombe, and M. Overmars. Probabilistic roadmaps for fast path planning in high dimensional configuration spaces. *IEEE Tr. on Rob. & Aut.*, 12:566–580, 1996.

[25] Y. Koga, K. Kondo, J. Kuffner, and J.C. Latombe. Planning motions with intentions. *Computer Graphics (SIGGRAPH'94)*, pages 395–408, 1994.

[26] J.J. Kuffner, Jr. *Autonomous Agents for Real-time Animation.* PhD thesis, Stanford University, Stanford, CA, 1999.

[27] F. Lamiraux and L.E. Kavraki. Planning paths for elastic objects under manipulation constraints. *Int. J. of Rob. Research*, 20(3), 2001.

[28] J.C. Latombe. *Robot Motion Planning.* Kluwer, Boston, MA, 1991.

[29] J.C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *Int. J. of Rob. Research*, 18(11):1119–1128, 1999.

[30] S.M. LaValle. Robot motion planning: A game-theoretic foundation. *Algorithmica*, 26, 2000.

[31] S.M. LaValle and S.A. Hutchinson. Optimal motion planning for multiple robots having independent goals. *IEEE Tr. on Rob. & Aut.*, 14(6):912–925, 1998.

[32] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Tr. on Computers*, 32:108–120, 1983.

[33] J.B. Mbede, X.H. Huang, and M. Wang. Fuzzy motion planning among dynamic obstacles using artificial potential fields for robot manipulators. *Robotics and Autonomous Systems*, 32(1):61–72, 2000.

[34] Phillip J. McKerrow. *Introduction to Robotics.* Addison-Wesley, 1991.

[35] R.J. Milgram and J.C. Trinkle. The geometry of configuration spaces for closed chains in two and three dimensions. *Homology Homotopy and Applications*, 2002. To appear.

[36] M. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In K.Y. Goldberg, D. Halperin, J.C. Latombe, and R.H. Wilson, editors, *Algorithmic Foundations of Robotics*, pages 19–37. A K Peters, 1995.

[37] E. Prassler, A. Ritter, C. Shaeffer, and P. Fiorini. A short history of cleaning robots. *Autonomous Robots*, 9(3):211–226, 2000.

[38] J. Reif. Complexity of the mover's problem and generalizations. In *Proc. 20th IEEE Symp. on Found. of Comp. Sci.*, pages 421–427, 1979.

[39] M. Rude. Collision avoidance using space-time representation of motion processes. *Autonomous Robots*, 4(1), 1997.

[40] J.T. Schwartz and M. Sharir. On the 'piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4:298–351, 1983.

[41] T. Simeon, S. Leroy, and J.P. Laumond. Path coordination for multiple mobile robots: A resolution-complete algorithm. *IEEE Tr. on Rob. & Aut.*, 18(1):42–49, 2002.

[42] S.K. Song and D.S. Kwon. Efficient formulation approach for the forward kinematics of the 3-6 Stewart-Gough platform. In *Proc. IEEE/RSJ Int. Conf. on Int. Rob. and Syst.*, 2001.

[43] P. Švestka and M. Overmars. Coordinated path planning for multiple robots. *Robotics and autonomous systems*, pages 125–152, 1998.

[44] P. Švestka and M. Overmars. Probabilistic path planning. In J-P. Laumond, editor, *Robot Motion Planning and Control*, pages 255–304. Lecture Notes in Control and Information Sciences, Springer, NY, 1998.

[45] J.C. Trinkle and R.J. Milgram. Motion planning for planar n-bar mechanisms with revolute joints. In *Proc. IEEE/RSJ Int. Conf. on Int. Rob. and Syst.*, 2001.

[46] K.D Wise and A. Bowyer. A survey of global configuration-space mapping techniques for a single robot in a static environment. *Int. J. of Rob. Research*, 19(8), 2000.

[47] J.H Yakey, S.M. LaValle, and L.E. Kavraki. Randomized path planning for linkages with closed kinematic chains. *IEEE Tr. on Rob. & Aut.*, 17(6), 2001.

Paper I

A lazy probabilistic roadmap planner for single query path planning

Robert Bohlin
Department of Mathematics
Chalmers University of Technology
SE-412 96 Göteborg, Sweden

Lydia E. Kavraki
Department of Computer Science
Rice University
Houston, TX 77005, USA

Abstract

Autonomous path planning addresses the problem of finding collision-free paths for moving objects – robots – among obstacles. In this paper we consider robots operating in workspaces occupied by stationary, completely known obstacles. We describe a new approach to probabilistic roadmap planners (PRMs). The overall theme of the algorithm, called Lazy PRM, is to minimize the number of collision checks performed during planning. Our algorithm builds a roadmap in the configuration space, whose nodes are the user-defined initial and goal configurations and a number of randomly generated configurations. Neighboring nodes are connected by edges representing the straight line path between the nodes. In contrast with PRMs, our planner initially assumes that all nodes and edges in the roadmap are collision-free, and searches the roadmap at hand for a shortest path between the initial and the goal node. The nodes and edges along the path are then checked for collision. If a collision with the obstacles occurs, the corresponding nodes and edges are removed from the roadmap. Our planner either finds a new shortest path, or first updates the roadmap with new nodes and edges, and then searches for a shortest path. The above process is repeated until a collision-free path is found.

Lazy PRM is tailored to efficiently answer single planning queries in standard industrial applications. Experimental results presented in this paper show that our lazy method is very efficient in practice.

Keywords: Collision avoidance, motion planning, path planning, probabilistic roadmaps, robotics.

1 Introduction and motivation

Autonomous path planning addresses the problem of finding collision-free paths for moving objects – robots – among obstacles. In this paper we consider robots operating in workspaces occupied by stationary, completely known obstacles. Our main concern is path planning in industrial environments, where the need for automatic solutions is huge. A car body may have thousands of spots to weld, and a ship may have miles of seams to arc weld. Programming by hand is tedious and, in comparison to manual programming, a powerful planner enables more complex motions to be executed and increases the quality of the paths.

The position of a robot is described by a *configuration*, which is a set of independent parameters such that the position of every point of the robot can be determined relative to a fixed frame in the workspace. We let $\mathcal{C}$ denote the set of all configurations – the *configuration space* – and $\mathcal{C}_{\mathcal{F}}$ denote the open subset of collision-free configurations. The dimension of $\mathcal{C}$ equals the number of degrees of freedom (dof) of the robot. A *path* is simply a continuous curve in $\mathcal{C}$. Given an initial configuration $\boldsymbol{q}_{init}$ and a goal configuration $\boldsymbol{q}_{goal}$ in $\mathcal{C}_{\mathcal{F}}$, the basic path planning problem is to find a path in $\mathcal{C}_{\mathcal{F}}$ connecting these points, or determine that none exists [28].

A workspace in industry is designed for a specific task and for the robot of interest to move as freely as possible. This typically means that the configuration space is relatively uncluttered, i.e. a relatively large fraction, say more than 15%, is collision-free. Unfortunately this does not imply that planning becomes easy. Industrial applications are characterized by robot systems with many dof and complex geometry of the robot and the obstacles. Existing algorithms applicable in high-dimensional configuration spaces either heavily rely on fast collision checking or require long preprocessing. Due to the complex geometry, which makes collision checking time consuming, these planners are too slow for single planning queries in the industrial applications we would like to cover.

1.1 Planner requirements

In industrial applications, the configuration space changes frequently. For example, as soon as the robot changes tools, grasps or deforms an object, or when a new obstacle enters the workspace, the feasible part $\mathcal{C}_{\mathcal{F}}$ is affected. A planner useful in practice must be able to plan in new configuration spaces instantly, so long preprocessing must be avoided. Ideally, the time required for planning should relate to the difficulty of the planning task, i.e., a simple path in an uncluttered environment should be found quickly, while a more complicated path may require more time.

In a similar way, the planning time should relate to the desired quality of the solution path. The quality of a path is difficult to quantify (see further discussion in Section 3.2.1), but in general we prefer short paths in $\mathcal{C}$, with respect to some metric. Consider a case where the planner generates a path that will be executed continually. Then the quality of the path is more important than the planning time, and we need a parameter, tuned by the user, that intuitively adjusts the properties of the planner.

We would also like the planner to learn to some extent, i.e., to use information from previous queries in order to speed up subsequent queries. For example, if the algorithm finds a path through a narrow passage in $\mathcal{C}_{\mathcal{F}}$, it should be able to use that information when searching for a new path back through the passage.

1.2 Contributions and outline of this paper

In this paper we further develop probabilistic planning techniques in the direction of achieving general and practically useful single query planners. We present a new approach to the probabilistic roadmap method (PRM). The algorithm – called Lazy PRM – is based upon a general scheme for lazy evaluation of the feasibility of the roadmap. Lazy PRM is tailored for single planning queries and has the properties discussed in Section 1.1. We address standard industrial applications characterized by complex geometry and high-dimensional, relatively uncluttered configuration spaces. To handle the complex geometry, the main theme of the algorithm is to minimize the number of collision-checks performed during planning. Experiments in a typical industrial environment show that a very large percentage, in average 26%, of the total number of collision checks are actually performed on the collision-free solution paths, and are therefore inevitable.

The scheme we suggest for lazy evaluation of roadmaps is general and can be applied to any graph that needs to be explored. In addition to Lazy PRM, other related algorithms, and variations of PRM, can benefit from this scheme and significantly increase performance. We propose two simple variations of Lazy PRM for the cases of more cluttered configuration spaces and faster collision-checking than in typical industrial applications. These cases are handled to a certain extent, but neither these cases nor the narrow passage problem (see [2, 8, 18]) are our main objectives. This paper extends the results presented in [7] and [6]. Related ideas about lazy evaluation has been developed concurrently and independently in [37]. Lazy PRM is described in detail in Section 3. Its performance is theoretically analyzed in Section 4, and experimentally evaluated in Section 5 using a real industrial environment.

2 Path planning techniques

Path planning is becoming increasingly important in automated manufacturing industry and for mobile robots, but has also found applications in computer graphics animations, medical surgery, and molecular biology [14, 29]. The problem has been extensively studied in the last two decades, and there exist a large number of planners based on a variety of approaches. See [14], [21], and [28] for overviews.

The complexity of certain versions of the problem is proven to be very high. In the case of a robot consisting of polyhedral bodies among polyhedral obstacles, the problem is PSPACE-hard [40]. Hence, there is strong evidence that a solution requires time that grows exponentially with the number of dof of the robot.

An algorithm is called *complete* if it always will find a solution or determine that none exists. Most complete methods, however, are only applicable to problems in low-dimensional configuration spaces, say of dimension three, or less [15]. A complete algorithm, working for arbitrary dimension, was given in [11]. Although the algorithm is exponential in the number of dof, it has the lowest time complexity of all complete algorithms known so far. It is too slow to be useful in practice, and is mostly used in theoretical analysis as an upper bound on the complexity of the path planning problem, see [14, 28].

We distinguish between deterministic and probabilistic algorithms. Trading completeness for speed and simplicity, probabilistic techniques have recently gained considerable attention due to their capability of solving problems in high-dimensional configuration spaces. In this paper, we focus on probabilistic algorithms and, in particular, we describe the Probabilistic Roadmap method in detail since it is relevant to the rest of the paper and forms the base of our solution.

2.1 Probabilistic planners

Probabilistic algorithms have been successfully applied in a wide variety of environments, and are now the methods of choice for complex problems. The Randomized Path Planner (RPP) in [5] has successfully solved problems for robots with more than 60 dof [27]. The planner uses a potential field as a guidance towards the goal, and random walks to escape local minima.

Another interesting approach is presented in [36] – the Ariadne's clew algorithm. Considering the initial configuration as a landmark, the algorithm incrementally builds a tree of feasible paths as follows. Genetic optimization is used to search for a collision-free path from one of the landmarks to a point as far as possible from previous landmarks. A new landmark is then placed at this point, and a path to the goal configuration is searched. New landmarks are placed until the goal configuration can be connected to the tree.

2.2 Probabilistic roadmap method

The idea behind the basic Probabilistic Roadmap Method (PRM), described in [25, 26, 39], is to represent and capture the connectivity of $\mathcal{C}_{\mathcal{F}}$ by a random network, a *roadmap*, whose nodes and edges respectively correspond to randomly selected configurations, and path segments. In a preprocessing step, or a *learning phase*, a large number of points are distributed uniformly at random in $\mathcal{C}$, and those found to be in $\mathcal{C}_{\mathcal{F}}$ are retained as nodes in the roadmap. A local planner is then used to find paths between each pair of nodes that are sufficiently close together. If the planner succeeds in finding a path between two nodes, they are connected by an edge in the roadmap. In the *query phase*, the user specified start and goal configurations are connected to the roadmap by the local planner. Then the roadmap is searched for a shortest path between the given points.

Even though a powerful local planner will require few nodes to obtain a well connected roadmap, most implemented PRMs show that it is computationally more efficient to distribute nodes densely and use a relatively weak, but fast, local planner, see [26, 39]. The local planner may for instance only check the straight line between two nodes. Other local planners are discussed and evaluated in [1].

Often the learning phase of basic PRM has a *node enhancement* step in order to increase the connectivity of the roadmap by adding more nodes in difficult regions of $\mathcal{C}_{\mathcal{F}}$. Different techniques are used to identify these regions; one way is to distribute new points close to a number of *seeds* randomly selected among the existing nodes. In [25], the probability that a node is selected is proportional to $\frac{1}{1+b}$, where b is the number of edges connected to the node. An alternative selection can be based on a node's ratio of failed attempts by the local planner to find paths to other nodes [26]. Other techniques to increase the connectivity of the roadmap are described in [2] and [17].

Basic PRM has shown to work well in practice in high-dimensional configuration spaces, see [26]. Indeed, it is useful for multiple queries since once an adequate roadmap has been created, queries can be answered very quickly.

2.3 Variations of PRM and related algorithms

Some of the methods using probabilistic roadmaps do not divide the planning process into a learning phase and a query phase. Given an initial and a goal configuration, the planner in [38] builds a tree by inserting randomly distributed nodes in $\mathcal{C}_{\mathcal{F}}$, one at a time, and connecting them to the different components of the roadmap by a local planner. New nodes are inserted until the initial and goal configurations can be found in the same connected component of the roadmap. See also [12] and [22] for relevant work. The latter paper gives an adaptive scheme that could be used to adjust the power of the local planner.

Although the node enhancement step was developed to increase the connectivity of the roadmap, basic PRM still has weaknesses in finding paths through narrow passages in $\mathcal{C}_{\mathcal{F}}$. Several recent approaches are intended to improve basic PRM in this respect by using different sampling strategies. The underlying idea is to distribute nodes close to the boundary of $\mathcal{C}_{\mathcal{F}}$. The planner in [18] initially allows the robot to penetrate the obstacles to a certain extent. Small neighborhoods around the configurations just in collision are then re-sampled in order to place nodes close to the boundary of $\mathcal{C}_{\mathcal{F}}$. The Obstacle Based PRM (OBPRM) in [2] and [3], repeatedly determines a configuration in collision to be the origin of a number of rays. Binary search is then used along each ray to find points on the boundary of $\mathcal{C}_{\mathcal{F}}$, where roadmap nodes are placed. In [8], another idea is presented. The planner identifies the boundary of $\mathcal{C}_{\mathcal{F}}$ by distributing points in pairs. Each pair is generated by first picking one point uniformly at random in $\mathcal{C}$, and then picking another point close to the first one. One of the points is added to the roadmap only if it is in $\mathcal{C}_\mathcal{F}$ and the other point is not. Yet another technique to increase the number of nodes in narrow passages of $\mathcal{C}_\mathcal{F}$ is presented in [42]. Points are picked uniformly at random in $\mathcal{C}$ and then retracted onto the medial axis of $\mathcal{C}_\mathcal{F}$. A related technique is described in [9], where points are generated on the medial axis of the workspace and then transformed into $\mathcal{C}$.

The randomized methods described in [19, 20] and [31, 32], build two trees rooted at the initial and goal configurations respectively. As soon as the trees intersect, a feasible path can be extracted. The methods differ in the way of expanding the trees. In [19, 20], the trees are expanded by generating new nodes randomly in the vicinity of the two trees, and connecting them to the trees by a local planner. The planner in [31, 32] iteratively generates a configuration, an attractor, uniformly at random in $\mathcal{C}$. Then, for both trees, the node closest to the attractor is selected and a local planner searches for a path of a certain maximum length towards the attractor. A new node is placed at the end of both paths. A new attractor is selected until the two trees intersect.

The algorithm in [30] is a method to keep the number of nodes in the roadmap to a minimum. Candidate nodes are generated uniformly at random, one at a time. A node is inserted to the roadmap only if it can be connected to at least two components of the roadmap, or if it does not see any other node. In the former case the components are merged, and in the latter case a new component is created. Variations of PRM have also been used for manipulation planning and for robots with closed kinematic chains, see [16, 33, 37].

The general theme for roadmap algorithms is to construct a network of paths verified to be collision-free by a local planner. Unfortunately, it is difficult to find a global strategy that can use these local planners efficiently in order to avoid traps and dead ends. In industrial environments, with complex geometry and expensive collision-checks, this often means that too much time is spent on planning local paths that will not appear in the solution path.

Our solution is to avoid using local planners as much as possible, and instead keep a global view through the entire planning process. In the next section we present Lazy PRM – a path planning algorithm tailored for single queries in high-dimensional, relatively uncluttered configuration spaces. We address the problem of finding simple paths quickly in industrial environments with complex geometry. In these environments collision-checking is computationally expensive, so to make the planner fast, the main theme is to minimize the number of collision checks.

We have observed excellent performance in cases that do not involve extremely narrow passages. As in the case with all comparable PRMs and all published work, it is hard to evaluate when Lazy PRM stops being efficient. The only thing we can suggest is to apply Lazy PRM for a period of time and if a solution has not been found, employ one of the PRMs with heuristics explicitly for narrow passages [2, 8, 9, 18, 42]. However, these heuristics can certainly benefit from the scheme for lazy evaluation of the feasibility of the roadmap that we describe in next section.

3 Lazy PRM

This section describes a new algorithm for single and multiple query path planning. The algorithm is similar to the basic PRM in [26] in the sense that the aim is to find the shortest path in a roadmap generated by randomly distributed configurations. In contrast with existing PRMs, we do not build a roadmap of feasible paths, but rather a roadmap of paths *assumed* to be feasible. The idea is to lazily evaluate the feasibility of the roadmap as planning queries are processed.

In other words, let $\boldsymbol{q}_{init}$, $\boldsymbol{q}_{goal}$, and a number of uniformly distributed configurations form nodes in a roadmap. We connect by edges each pair of nodes being sufficiently close together. Lazy PRM finds a shortest feasible path in the roadmap by repeatedly searching for a shortest path, and then checking whether it is collision-free or not. Each time a collision occurs, the corresponding node or edge is removed from the roadmap, and then Lazy PRM searches for a new shortest path.

This procedure can terminate in either of two ways. If there exist feasible paths in the roadmap between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$, we will find a shortest one among them. Otherwise, if there is no feasible path, we will eventually find $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$ in two disjoint components of the roadmap. In the latter case, we can either report failure, or, if we still have time, add more nodes to the roadmap in a similar way to the node enhancement in [25, 26], and start searching again. A high-level description of the algorithm is given in Figure 1.

The point by using this scheme for lazy evaluation is that we only explore the part of the roadmap that is needed for the current query. The scheme is

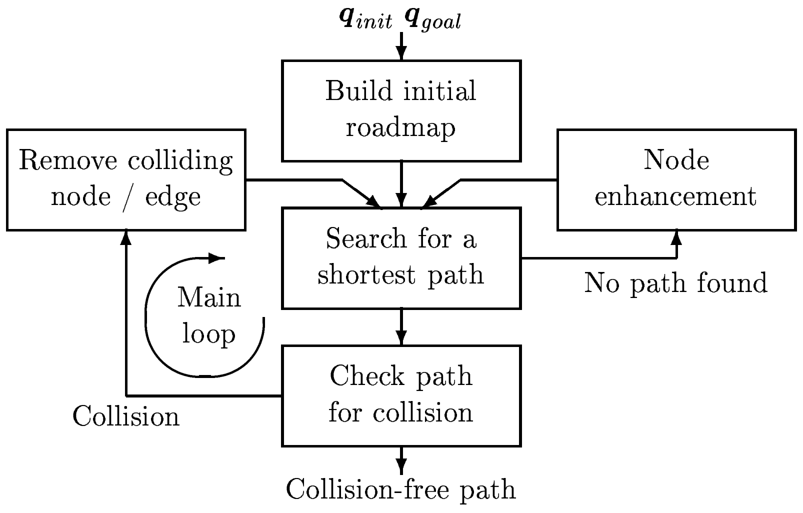


Figure 1: High-level description of Lazy PRM.

simple, general and can be applied also to other roadmap planners in order to increase performance. The strength is to either find a collision-free path or to conclude that none exists in the roadmap by using a small number of collision checks. It is always an advantage to use lazy evaluation since we can never do more work, in terms of collision checking, than basic PRM would do.

The rest of this section explains the different steps of the algorithm in more detail, Section 4 gives a proof of its probabilistic completeness, and Section 5 shows some experimental results.

3.1 Building the initial roadmap

The first step in the algorithm is to build a roadmap $\mathcal{G}$ in $\mathcal{C}$. There are two parameters that determine the size of $\mathcal{G}$; the number of nodes, N_{init}, and the expected number of neighbors, M_{neighb}, connected to each node.

3.1.1 Initial distribution of nodes

Initially, we distribute N_{init} points uniformly at random in $\mathcal{C}$. These points, together with $\boldsymbol{q}_{init} \in \mathcal{C}_{\mathcal{F}}$ and $\boldsymbol{q}_{goal} \in \mathcal{C}_{\mathcal{F}}$, form nodes in $\mathcal{G}$. An important issue is the choice of N_{init}. The initial density of nodes, determined by

N_{init}, is strongly correlated to the probability of finding a short path, if one exists. The correlation is hard to quantify, but the following example may give an illustration. Assume there exist only two ways to get to the goal configuration; either a short path through a rather narrow corridor, or a somewhat longer path through a wide corridor. If $\mathcal{G}$ is sufficiently dense, the algorithm will find a short path through the narrow passage. If $\mathcal{G}$ is sparse, the algorithm will find a longer path through the wide passage. In the worst case, if the roadmap is too sparse, there will be no feasible path at all in the roadmap, and the algorithm has to go to the enhancement step to generate more nodes. On the other hand, if N_{init} is too large, we will distribute more nodes than necessary. Although we may obtain better paths, this will lead to somewhat longer planning times.

However, the idea behind the algorithm is that only a small fraction of the nodes in the roadmap will be necessary to check for collision. This makes the algorithm relatively insensitive to high density of nodes, so we can choose N_{init} relatively large. (In our experiments we start with $N_{init} = 10000$ nodes and check on average 320 nodes in one of the most difficult planning tasks, see Task $I \rightarrow J$ in Table 1(a), Section 5.) The number of nodes required to find a path is further explored in Section 4.

3.1.2 Selecting neighbors

To build the roadmap we connect each node in $\mathcal{G}$ by edges to a set of neighbor nodes. An edge represents the straight line path in $\mathcal{C}$ between two nodes. Neither the nodes nor the edges are being checked for collision in the initial step, but we want, of course, to have edges which are likely to be feasible. Since it would require far too much memory to connect all pairs of nodes, and it is unlikely that the straight line path between two nodes far apart is feasible, it is natural to only consider nodes which are sufficiently close together.

In order to select appropriate neighbors, we need a metric $\rho_{coll} : \mathcal{C} \times \mathcal{C} \rightarrow [0, \infty)$ such that the distance between two configurations under this metric reflects the difficulty of connecting them by a collision-free straight line path. Then we connect each pair of nodes $(\boldsymbol{q}, \boldsymbol{q}')$ such that $\rho_{coll}(\boldsymbol{q}, \boldsymbol{q}') \leq R_{neighb}$. For any fixed radius R_{neighb}, the number of neighbors of a node is a random variable, so depending on the initial number of nodes N_{init}, we choose R_{neighb} such that the expected number of neighbors equals the parameter M_{neighb} introduced in the beginning of Section 3.1.

In many cases it is harder to make feasible connections in certain directions than in others. Consider for instance an articulated robot arm; then it is more likely that a collision occurs when the base joint is moving one unit, than if a joint close to the end-effector is moving one unit. With this in mind, we let ρ_{coll} be a weighted Euclidean metric,

$$\begin{aligned}\rho_{coll}(\boldsymbol{x},\boldsymbol{y}) &= \Big(\sum_{i=1}^{d} w_i^2(x_i-y_i)^2\Big)^{1/2} \\ &= \big((\boldsymbol{x}-\boldsymbol{y})^T W(\boldsymbol{x}-\boldsymbol{y})\big)^{1/2}, \qquad (1)\end{aligned}$$

where d is the dimension of $\mathcal{C}$, $\{w_i\}_{i=1}^d$ are positive weights, $W = \mathrm{diag}(w_1^2,\dots,w_d^2)$, and $\boldsymbol{x}^T$ is the transpose of $\boldsymbol{x}$. The weights are chosen in proportion to the maximum possible distance (Euclidean distance in the workspace) traveled by any point on the robot, when moving one unit in $\mathcal{C}$ along the corresponding axis. This metric is easy to use and has been shown to work well in our experiments presented in Section 5.

3.2 Searching the roadmap for a shortest path

The second step in the algorithm is to find a shortest path in $\mathcal{G}$ between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$, or determine that none exists. We use the A^* algorithm [35], and a metric $\rho_{path} : \mathcal{C}\times\mathcal{C} \to [0,\infty)$ to measure the length of a path and the remaining distance to $\boldsymbol{q}_{goal}$.

If the search procedure succeeds in finding a path, we need to check it for collision. Otherwise, if no path exists in the roadmap, we either report failure, or go to the node enhancement step to add more nodes to the roadmap and start searching again depending on the overall time allowed to solve the problem.

3.2.1 Choosing an appropriate metric for A^*

The tool available to give preference to certain paths and reject others is the metric ρ_{path}. Thus, by defining this metric we decide which paths are assumed to be of high quality and which paths are assumed to be of poor quality.

In this paper we focus on articulated robots and use the Euclidean configuration space $I_1 \times \cdots \times I_d$, where I_i is the range of joint i and d is the number of dof. Thus, we do not identify angles equal modulo 2π as being equal, although they define the same position in the workspace. This is because a real robot in general has supply wires, etc., which otherwise would be entangled. The metric ρ_{path} is a weighted Euclidean metric, similar to (1), where the weights are equal to $\frac{1}{v_i}$, $i = 1, ..., d$, where v_i is the maximum angular velocity of joint i. This tends to give preference to paths with short execution time, which in many applications is the most interesting response variable.

In the general case, however, there are a large number of other response variables to consider. Some of them are measurable such as energy consumption, dynamic forces on joints, etc. Others are more subjective; for example, the motion should look natural and smooth from the user's point of view. Under any Euclidean metric, the straight line path in $\mathcal{C}$ between two configurations is the shortest, but considering all of these response variables, the straight line path is not necessarily optimal. Thus, the choice of a configuration space parameterization and an appropriate metric is a very difficult task in itself. It is considered an open question that deserves further investigation and will impact Lazy PRM and all other PRM based planners.

3.3 Checking paths for collision

When the A^* algorithm has found a shortest path in the roadmap between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$, we need to check the nodes and edges along the path for collision. In most applications it is straightforward to perform a collision check for a given configuration, i.e. determine whether a point is in $\mathcal{C}_{\mathcal{F}}$ or not [41]. It is somewhat more expensive to calculate the minimum distance between the robot and the obstacles [10, 34], and it is considerably more complex to obtain more information, for instance to check whether a path segment is entirely in $\mathcal{C}_{\mathcal{F}}$ or not. Our algorithm only requires a collision checker for points in $\mathcal{C}$. Path segments, i.e. edges in the roadmap, are discretized and checked with a certain resolution.

The overall purpose of the Search, Check, and Remove steps of our algorithm (the main loop in Figure 1), is roughly to identify and remove colliding nodes and edges from the roadmap until the shortest path between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$ is feasible. Accordingly, when checking a path for collision, we are not primarily interested in verifying whether an individual node or edge is in $\mathcal{C}_{\mathcal{F}}$ or not, but rather to remove colliding nodes and edges as efficiently as possible. Since a removal of a node implies all its connected edges to be removed, it seems reasonable to check the feasibility of the nodes along the path before checking the edges.

3.3.1 Checking nodes

Starting respectively with the first and the last node on the examined path and working toward the center, we alternately check the nodes along the path. As soon as a collision is found, we remove the corresponding node and its connected edges from the roadmap, and search for a new shortest path.

The reason for checking the nodes in this order is that the probability of having the shortest feasible path via a particular node is higher if the node is close to either $\boldsymbol{q}_{init}$ or $\boldsymbol{q}_{goal}$. Consider, for instance, the nodes connected to $\boldsymbol{q}_{init}$; a shortest feasible path (if one exists) must pass through at least one of them. Since, in a cluttered space, we cannot give preference to certain directions, the probability of having the shortest feasible path via a particular neighbor of $\boldsymbol{q}_{init}$ is at least $1/b$, where b is the number of neighbors of $\boldsymbol{q}_{init}$. Nodes connected to $\boldsymbol{q}_{goal}$ have a similar probability, whereas nodes further away from both $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$ have a much lower probability of being in the shortest feasible path. Therefore, we check the nodes along a path starting from the end-nodes and working toward the center.

3.3.2 Checking edges

If all nodes along the path are in $\mathcal{C}_{\mathcal{F}}$, we start checking the edges in a similar fashion; working from the outside in. However, to minimize the risk of doing unnecessary collision checks, we first check all edges along the path with a coarse resolution, and then do stepwise refinements until the specified resolution is reached. As with the nodes, if a collision is found, we remove the corresponding edge, and search for a new shortest path. If no collision is found along the path, the algorithm terminates and returns the collision-free path. Figure 3.3.2 offers an illustration. To make the overall algorithm efficient, we record which nodes have been checked for collision, and to which resolution each edge has been checked, in order to avoid checking any point in $\mathcal{C}$ more than once.

The total number of collision checks depends on the resolution with which the edges along the path are checked. Again, since ρ_{coll} reflects the probability of collision, we determine the resolution with respect to this metric. The resolution is quantified by a step-size δ, but we prefer not to let the user specify the step-size by a certain number, because the resolution should depend on the scale of $\mathcal{C}$ and the weights defining the metric. A better way is to introduce a parameter M_{coll}, specifying the number of collision checks required to check the longest possible straight line path in $\mathcal{C}$. In other words, assuming that $\mathcal{C}$ is a d-dimensional rectangle and $\boldsymbol{q}$ and $\boldsymbol{q}'$ are two opposite corners, the step-size is related to the length of the diagonal of $\mathcal{C}$ according to

$$\delta = \frac{\rho_{coll}(\boldsymbol{q}, \boldsymbol{q}')}{M_{coll}}.$$

3.4 Node enhancement

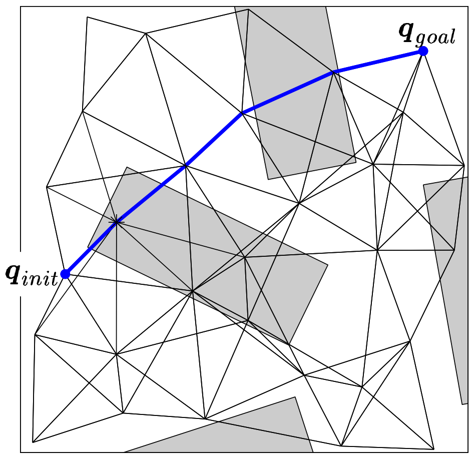
If the search procedure fails, no feasible path between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$ exists in the roadmap, and more nodes are necessary in order to find one. In the node enhancement step, we generate N_{enh} new nodes, add them to $\mathcal{G}$, and select neighbors in the same way as when $\mathcal{G}$ was initially built.

We may not only distribute the new nodes uniformly, but rather use the information available in the roadmap (or what is left of the roadmap), in order to distribute new nodes in difficult regions of $\mathcal{C}$. In a method similar to the node enhancement in [25, 26], we randomly select a number of points in $\mathcal{G}$, called *seeds*, and then distribute a new point close to each of them. Our experience is that it is better to select many seeds and distribute one new node around each of them, instead of selecting few seeds and distribute several nodes around each of them; the latter method is more dependent on the selection of seeds.

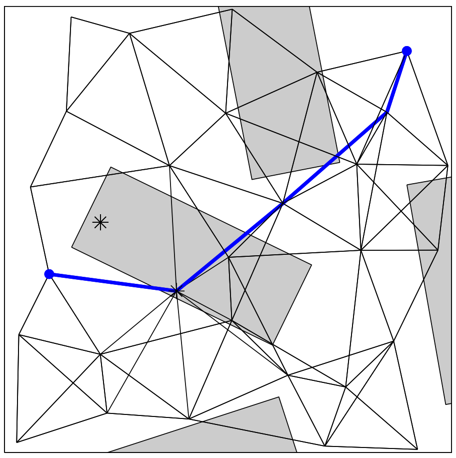
Although the seeds may help us identify difficult regions of $\mathcal{C}$, we still want to maintain a smooth distribution all over $\mathcal{C}$, because the knowledge about $\mathcal{C}$ is limited and we do not want to rely too much on the selection of seeds. To ensure probabilistic completeness, we also distribute new nodes uniformly at random in each step. In our algorithm, we let half of the enhancement nodes be uniformly distributed, and the rest distributed around seeds.

3.4.1 Selecting seeds

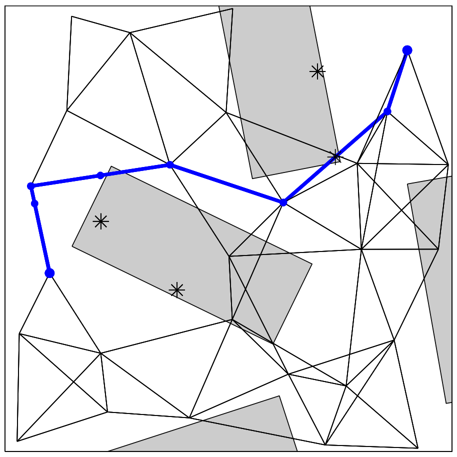
The set of edges which have been removed from the roadmap and have at least one end-point in $\mathcal{C}_{\mathcal{F}}$, will certainly intersect the boundary of $\mathcal{C}_{\mathcal{F}}$. Using the mid-points of these edges as seeds may help us distribute points close to the boundary of $\mathcal{C}_{\mathcal{F}}$, thus increase the probability of finding paths through narrow passages in $\mathcal{C}_{\mathcal{F}}$.



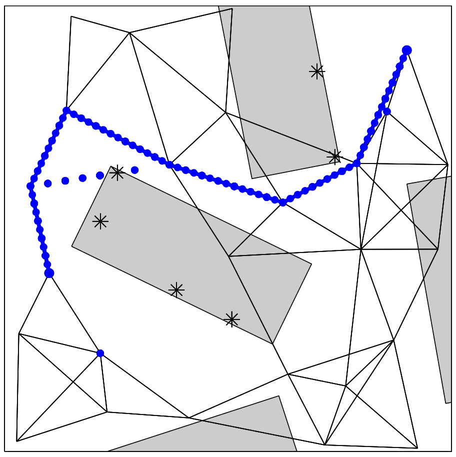
(a): Lazy PRM searches for a shortest path and checks the nodes. A collision is detected ($*$) and corresponding node is deleted.



(b): Then Lazy PRM searches for a new shortest path, detects a new collision ($*$) and deletes corresponding node.



(c): After a few iterations, a sequence of feasible nodes is found. When checking the edges with a coarse resolution a collision is found ($*$). The edge is deleted from the roadmap, and the planner searches for a new shortest path.



(d): Eventually, the planner finds a path whose nodes are collision-free, and whose edges are collision-free to a specified resolution.

Figure 2: Example of a planning query in a 2-dimensional configuration space with rectangular obstacles (grey). *All* collision checks performed are marked with $*$ (collision) or $\bullet$ (collision-free).

However, if the enhancement step is executed several times, this may cause problems with clustering of nodes. Assume that we add a new node $\boldsymbol{q}$. This node will give rise to a number of edges which in the next enhancement step may increase the probability of adding even more nodes close to $\boldsymbol{q}$. Thus, the distribution of new enhancement nodes depends on the preceding enhancement steps, and may eventually cause undesired clusters of nodes. To avoid this phenomenon, we only use edges whose end-nodes are generated uniformly at random when selecting seeds.

3.4.2 Distributing new nodes

When distributing a new point $\boldsymbol{q}$ around a seed $\boldsymbol{\eta}$, we use the multivariate normal distribution. This distribution is smooth, easy to use, and allows us to control the distribution of $\boldsymbol{q}$ in terms of the metric ρ_{coll}. Hence, we can stretch the distribution in directions where the probabilities of making feasible connections are higher.

Introducing two parameters $\alpha \in (0,1)$ and $\lambda > 0$, we can choose the distribution such that

$$\rho_{coll}(\boldsymbol{q}, \boldsymbol{\eta}) \leq \lambda R_{neighb} \tag{2}$$

is an event with probability $1 - \alpha$, see Figure 3. R_{neighb} is the maximum length of an edge defined in Section 3.1.2. To achieve this property, we define a covariance matrix Σ as follows:

$$\Sigma = \frac{\lambda^2 R_{neighb}^2}{\chi_d^2(\alpha)} W^{-1}. \tag{3}$$

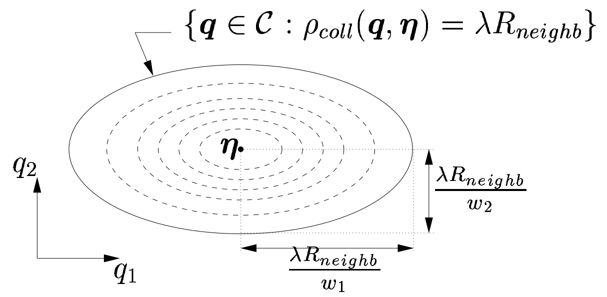
Here W is the same as in (1) and $\chi_d^2(\alpha)$ is the upper α percentile of a χ^2-distribution with d dof. Then we let the new point $\boldsymbol{q} \sim N_d(\boldsymbol{\eta}, \Sigma)$, i.e., $\boldsymbol{q}$ is multivariate normally distributed with d dof, mean $\boldsymbol{\eta}$, and covariance matrix Σ. Since Σ is diagonal, this simply means that each component $q_i, i = 1, \ldots, d$, of $\boldsymbol{q}$ is normally distributed with mean η_i and variance $\Sigma_{i,i}$.

To show (2), we use that $(\boldsymbol{q} - \boldsymbol{\eta})^T \Sigma^{-1} (\boldsymbol{q} - \boldsymbol{\eta})$ is χ^2-distributed with d dof [23]. Thus, the event

$$(\boldsymbol{q} - \boldsymbol{\eta})^T \Sigma^{-1} (\boldsymbol{q} - \boldsymbol{\eta}) \leq \chi_d^2(\alpha)$$

has probability $1 - \alpha$. Using (1) and (3) gives the confidence ellipsoid in (2).

We see in (3) that Σ depends on the the ratio $\lambda^2/\chi_d^2(\alpha)$. Since both λ^2, $\lambda > 0$, and $\chi_d^2(\alpha)$, $\alpha \in (0,1)$, are continuous functions whose ranges are



$\{\boldsymbol{q} \in \mathcal{C} : \rho_{coll}(\boldsymbol{q}, \boldsymbol{\eta}) = \lambda R_{neighb}\}$
$\boldsymbol{\eta}$
q_2
q_1
$\frac{\lambda R_{neighb}}{w_2}$
$\frac{\lambda R_{neighb}}{w_1}$

Figure 3: Example of a seed $\boldsymbol{\eta}$ in a 2-dimensional configuration space. If a new point $\boldsymbol{q}$ is distributed according to $N_d(\boldsymbol{\eta}, \Sigma)$, with Σ as in (3), then $\boldsymbol{q}$ is distributed within the confidence ellipse (solid line) with probability $1 - \alpha$. The dashed ellipses are contours of the distribution function. w_1 and w_2 are the weights defined in (1).

$(0, \infty)$, one of the two parameters α and λ is redundant, so we can without loss of generality choose $\alpha = 0.05$. Then, the parameter λ controls the size of the 95% confidence ellipsoid relative to R_{neighb} as shown in Figure 3. In our experiments we found that $\lambda = 1$ is a suitable choice.

Another possibility of distributing the new point $\boldsymbol{q}$, is to let it be uniformly distributed in a rectangular box centered at $\boldsymbol{\eta}$. If we let the sides of the box be of equal length under ρ_{coll}, we stretch the box in a similar way as the ellipsoids above. In our path planning algorithm, however, the normal distribution has a major advantage compared to the uniform distribution; the contours of the distribution function are ellipsoids around $\boldsymbol{\eta}$ (see Figure 3). Hence, under the metric ρ_{coll}, which reflects the difficulty of making connections, the distribution is symmetric around $\boldsymbol{\eta}$. In contrast, the uniform distribution favors the directions of the corners of the box, and nodes are more frequently distributed there than in other directions.

3.5 Multiple queries

When the planner has found a collision-free path, it terminates and returns the path. The information about which nodes and edges have been checked for collision is stored in the roadmap. As long as the configuration space remains the same, we use the same roadmap when processing subsequent queries. Thus, we benefit from the information obtained in each planning query. The new initial and goal configurations are simply added to the roadmap, and the same algorithm, except for the initial generation of nodes, is run again.

As several queries are processed, more and more of the roadmap will be explored, and the planner will eventually find paths via nodes and edges which have already been checked for collision. This makes the planner efficient for multiple queries.

Even in the long run, many nodes and edges may never be explored since they are located in odd regions of $\mathcal{C}$. Thus, given a fixed size of the roadmap, the number of collision checks performed by Lazy PRM will never exceed the number of collision checks performed by the basic PRM described in Section 2.2. Accordingly, there is no reason to entirely evaluate the roadmap unless we explicitly want it. The lazy evaluation scheme will find the shortest feasible path in the roadmap by using less collision checks.

4 Probabilistic completeness

In this section we give a proof of probabilistic completeness of Lazy PRM. First we need some notation. Let $\gamma : [0, L] \to \mathcal{C}_{\mathcal{F}}$ be a curve (also called path) parameterized by arc length and with continuous tangent. A *tube* τ of radius r around $\gamma(s)$ is the set of points at distance r from γ measured perpendicular to the tangent $\gamma'(s)$. Similarly, the corresponding *solid tube* is the set of points at distance $\leq r$ from γ. For simplicity, we usually omit the word solid.

A *regular tube* is a tube that does not intersect itself. If γ is enclosed by a regular tube of radius r, this particularly implies that its curvature, $\kappa(s) = |\gamma''(s)|$, is bounded from above by $1/r$. Otherwise the tube would be folded. The following lemma, proved in [13], states a useful property of regular tubes.

Lemma 1. *The volume enclosed by a regular tube around a curve in a d-dimensional Euclidean space is the product of the length of the curve and the $(d-1)$-dimensional area of a cross-section.*

In other words, if B_r^d is the ball of radius r in a d-dimensional space, and μ_d the Lebesgue measure, we can express the volume of a regular tube τ of radius r around γ as

$$\mu_d(\tau) = L\mu_{d-1}(B_r^{d-1}) = Lr^{d-1}\,\mu_{d-1}(B_1^{d-1}), \tag{4}$$

where L is the length of γ.

Assuming there exists a path between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$, enclosed by a regular tube in $\mathcal{C}_{\mathcal{F}}$, the following theorem gives an upper bound on the probability of failure to find a path between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$. The assumption of an enclosing tube in $\mathcal{C}_{\mathcal{F}}$ is relevant since $\mathcal{C}_{\mathcal{F}}$ is an open subset of $\mathcal{C}$. Moreover, the theorem says that the probability of failure decreases exponentially in the *total* number of uniformly distributed nodes N. Since N increases in each enhancement step (Figure 1 and Section 3.4), the probability of failure vanishes as time tends to infinity. This is equivalent to the definition of *probabilistic completeness* [21]. Thus, Lazy PRM is a probabilistically complete path planner. Since the configuration space is at least 2-dimensional (otherwise path planning is trivial), we assume that $d \geq 2$. Recalling the parameter R_{neighb} from Section 3.1.2 and the matrix W defined in (1) with norm $\|W\|$, we formulate the theorem as follows.

Theorem 1. *Let N be the total number of nodes generated uniformly at random in $\mathcal{C}$. If there exists a path γ between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$, enclosed by a regular tube τ of radius $R \leq \frac{1}{2\|W\|^{1/2}} R_{neighb}$ entirely in $\mathcal{C}_{\mathcal{F}}$, then Lazy PRM will fail to find a path with probability at most*

$$\frac{Ld}{R} e^{-\beta N},$$

where $\beta = \frac{R^d\,\mu_{d-1}(B_1^{d-1})}{3d\,\mu_d(\mathcal{C})}$ and L is the length of γ.

Proof. Let $u = R/d$, $r = R(1-1/d)$, and $k = \lfloor L/u \rfloor$. The idea of the proof is to take a tube of radius r, divide it into $k-1$ cells of length u, and calculate the probability of having at least one node in each cell. We will show that any two points in adjacent cells can be connected by a straight line, and that one node in each cell is enough for the planner to succeed. Assume first that $k \geq 2$. The case $k < 2$ is trivial and will be considered at the end of the proof.

Let $s_i = iu$, $i = 1, \ldots, k$, and let τ_i be the tube segment around $\gamma(s)$ for $s \in [s_i, s_{i+1})$, $i = 1, \ldots, k-1$, see Figure 4. The tube segments $\{\tau_i\}_{i=1}^{k-1}$ are

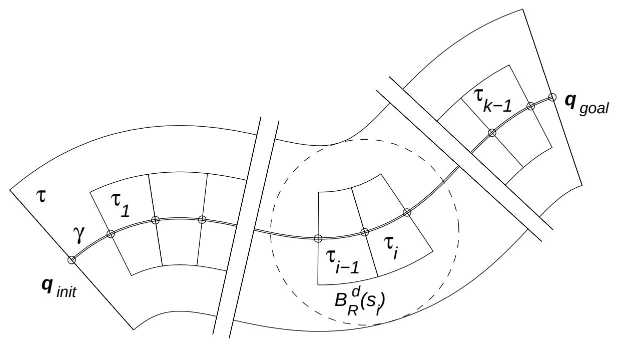


Figure 4: Illustration to the proof of Theorem 1.

pairwise disjoint and, by (4),

$$\frac{\mu_d(\tau_i)}{\mu_d(\mathcal{C})} = ur^{d-1}\,\frac{\mu_{d-1}(B_1^{d-1})}{\mu_d(\mathcal{C})}.$$

Now, for $d \geq 2$, $(1-1/d)^{d-1}$ is a decreasing function whose limit is e^{-1}, and since

$$ur^{d-1} = \frac{R^d}{d}\Big(1-\frac{1}{d}\Big)^{d-1} \geq \frac{R^d}{d}e^{-1} \geq \frac{R^d}{3d},$$

we get that

$$\frac{\mu_d(\tau_i)}{\mu_d(\mathcal{C})} \geq \frac{R^d\,\mu_{d-1}(B_1^{d-1})}{3d\,\mu_d(\mathcal{C})} = \beta. \tag{5}$$

The N points generated by the algorithm are uniformly and independently distributed in $\mathcal{C}$. Thus, the probability that τ_i is empty equals $\big(1-\frac{\mu_d(\tau_i)}{\mu_d(\mathcal{C})}\big)^N$, which, by (5), can be estimated:

$$\Big(1-\frac{\mu_d(\tau_i)}{\mu_d(\mathcal{C})}\Big)^N \leq (1-\beta)^N. \tag{6}$$

Let $B_R^d(s)$ be a ball of radius R centered at $\gamma(s)$, i.e., $B_R^d(s)$ has the same radius as τ. Unless $B_R^d(s)$ is close to the end-points of γ, it will be covered by τ, see Figure 4. If it is close to the end-points, however, it might intersect the circular discs at the ends of the tube. Nevertheless, the intersection between $B_R^d(s)$ and τ is still convex, a property we will need later.

Now, let $\boldsymbol{q}_{i-1} \in \tau_{i-1}$ and $\boldsymbol{q}_i \in \tau_i$. By the definition of a tube there exists an $\sigma_i \in [s_i, s_{i+1})$ such that $|\boldsymbol{q}_i - \gamma(\sigma_i)| \leq r$. Since γ is parameterized by arc length, it follows that $|\gamma(s) - \gamma(t)| \leq |s - t|$, and, by the triangle inequality,

$$\begin{aligned} |\boldsymbol{q}_i - \gamma(s_i)| &\leq |\boldsymbol{q}_i - \gamma(\sigma_i)| + |\gamma(\sigma_i) - \gamma(s_i)| \\ &\leq r + u = R. \end{aligned}$$

Hence, the ball $B^d_R(s_i)$ contains τ_i. Similarly, we can show that it also contains τ_{i-1}. Since both cells are covered by τ, they are contained in the convex set $B^d_R(s_i) \cap \tau$ which is entirely in $\mathcal{C}_{\mathcal{F}}$. Thus, $\boldsymbol{q}_{i-1}$ and $\boldsymbol{q}_i$ are at most $2R$ apart and the straight line between them lies entirely in $\mathcal{C}_{\mathcal{F}}$. From (1) we get that

$$\begin{aligned} \rho_{coll}(\boldsymbol{q}_{i-1}, \boldsymbol{q}_i) &\leq |\boldsymbol{q}_{i-1} - \boldsymbol{q}_i| \, \|W\|^{1/2} \\ &\leq 2R \, \|W\|^{1/2} \\ &\leq R_{neighb}, \end{aligned}$$

i.e., any node in τ_{i-1} is in the neighborhood of any node in τ_i and will therefore be interconnected by Lazy PRM. Moreover, since $\boldsymbol{q}_{init} \in B^d_R(s_1)$ and $\boldsymbol{q}_{goal} \in B^d_R(s_k)$, they will be connected to any node in τ_1 and τ_{k-1} respectively. Consequently, it is enough to have at least one node in each of the cells $\tau_1, \ldots, \tau_{k-1}$, in order for Lazy PRM to find a collision-free path between $\boldsymbol{q}_{init}$ or $\boldsymbol{q}_{goal}$.

The probability of failure for our algorithm, $P_{failure}$, can now be estimated:

$$\begin{aligned} P_{failure} &\leq P(\text{some } \tau_i \text{ is empty}) \\ &\leq \sum_{i=1}^{k-1} P(\tau_i \text{ is empty}) \\ &\leq (k-1)(1-\beta)^N, \end{aligned}$$

where we used Boole's inequality and (6) in the second and third step respectively. Using that $k - 1 \leq Ld/R$ and $(1-\beta)^N \leq e^{-\beta N}$ gives the desired estimation.

What remains is the case $k < 2$, i.e., $L < 2u = 2R/d \leq 2R$. Then both $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$ are contained in the convex set $B^d_R(L/2) \cap \tau$ which is entirely in $\mathcal{C}_{\mathcal{F}}$. This guarantees that Lazy PRM will find the straight line path between $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$, so the probability of failure is zero. □

Note that a related theorem regarding basic PRM can be found in [4] and [24]. Both theorems give a bound on the failure probability expressed in terms of, among other variables, the density of nodes. An important difference is that Lazy PRM has to reach a certain density of nodes in $\mathcal{C}$, while basic PRM has to reach approximately the same density in $\mathcal{C}_{\mathcal{F}}$. This seems like a weakness of our method, but looking at how the nodes in basic PRM are generated, we see that this is not the case. In order to reach the desired density in $\mathcal{C}_{\mathcal{F}}$, basic PRM has to distribute nodes uniformly all over $\mathcal{C}$ and exclude those in collision. Consequently, for both algorithms to reach the same density, the number of nodes checked for collision in the learning phase of basic PRM has to be the same as the number of uniformly distributed nodes in Lazy PRM. So whether the density is specified in $\mathcal{C}_{\mathcal{F}}$ or in $\mathcal{C}$ does not matter. The difference of practical significance is that Lazy PRM avoids checking *all* of the nodes for collision.

5 Experimental results

In this section we present performance tests of Lazy PRM when applied to a six dof robot in a realistic industrial environment. The planner has been implemented in C++ as a plug-in module to RobotStudio[1] – a simulation and off-line programming software running under Windows NT. The collision checks are handled internally in RobotStudio. The experiments have been run on a PC with a 400 MHz Pentium II processor and 512 MB RAM. In all tests we let $N_{init} = 10000$, $M_{neighb} = 60$, $M_{coll} = 200$, and $N_{enh} = 500$.

5.1 Path planning tasks

The test example is a part of a real manufacturing process in which an ABB 4400 robot is tending press breaking. Metal sheets are formed by the hydraulic press shown in Figure 5. In this particular example, plane sheets of metal are picked at a pallet, bent twice, and then placed at another pallet.

The process is divided into several steps, and our aim is to automatically plan the unconstrained paths of the robot. We let A to J denote ten different configurations shown in Figures 5,6 and 7. These are used as either initial or goal configurations in four planning tasks, denoted for example $A \to B$, where A is the initial configuration and B is the goal configuration.

[1]RobotStudio is developed by ABB Robotics, Göteborg, Sweden.

Figure 5: The workspace used in the experiments. The robot is in its home configuration denoted by A.

The scenario is as follows. Starting from the home configuration A, the robot picks a sheet of metal from the pallet at B (task $A \rightarrow B$), adjusts the grip at configuration C (task $B \rightarrow C$), and puts the sheet-metal at the press D (task $C \rightarrow D$). After the breaking, the robot grasps the sheet-metal at E, moves to the re-gripper, F (task $E \rightarrow F$), and places the sheet-metal. The robot grasps the sheet-metal from the other side at G (task $F \rightarrow G$) and moves to the press (task $G \rightarrow H$). After the second breaking, the sheet-metal is grasped at configuration I and placed at the pallet J (task $I \rightarrow J$). Then the robot returns to the home configuration A (task $J \rightarrow A$).

Thus, we have eight paths to plan. Note that during this series of steps, the configuration space changes several times. As soon as we grasp or place a sheet of metal, the collision-free part, $\mathcal{C}_{\mathcal{F}}$, is changing. Neglecting the small displacement of the sheet-metal caused by the centering operation at C, the tasks $B \rightarrow C$ and $C \rightarrow D$ can be planned in the same configuration space. Accordingly, we have seven different configuration spaces in which to plan, and we have have to build one roadmap in each of them.

The results include the number of collision checks, the number of enhancement steps, and the planning time. The minimum, average, and maximum values, based on 20 consecutive runs for each task, are shown in Table 1(a) - 1(c). The average number of collision checks performed on nodes and edges respectively are presented, as well as the average number of collision checks

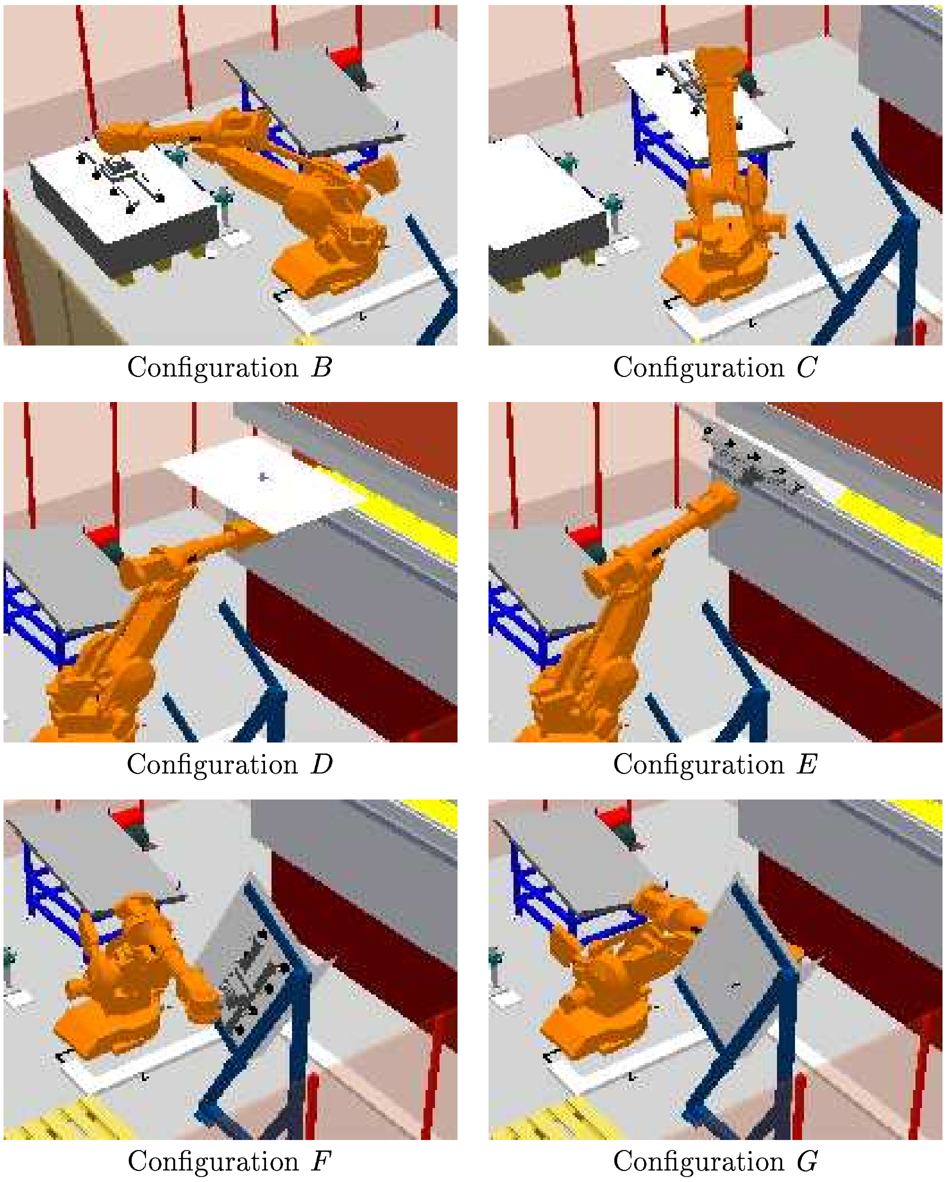
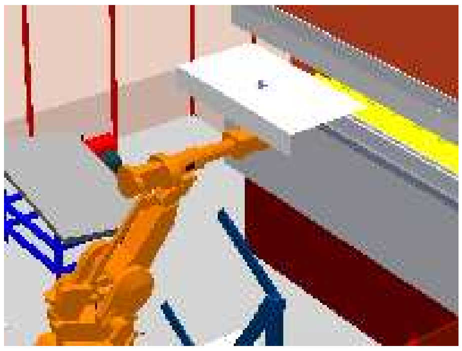
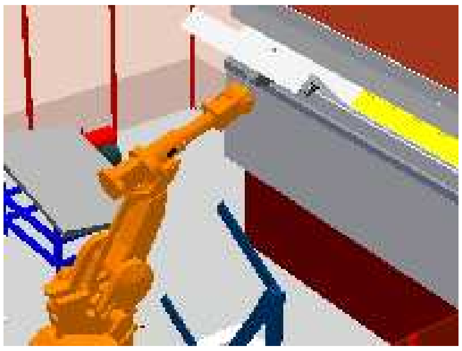


Figure 6: Configurations B to G used in the experiments.

performed on the collision-free paths that the planner returned. Since paths are checked for collision with a certain resolution (see Section 3.3.2), the latter figures correspond to the lengths of the collision-free paths.


Configuration H


Configuration I

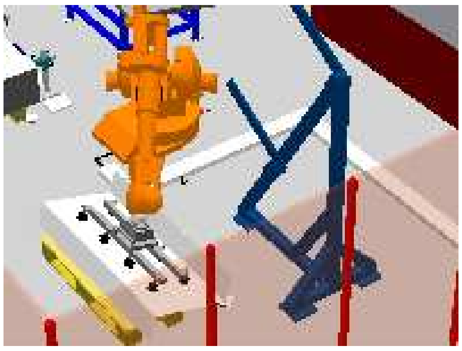

Configuration J

Figure 7: Configurations H to J used in the experiments.

The running times in Table 1(c) are divided into three parts. Firstly, graph building, which includes distance calculations between nodes in $\mathcal{C}$ as well as node and edge adding, secondly, graph searching, and finally collision checking.

In the last column of Table 1, the average values of the recorded data are summed up. Thus, the last column indicates the average number of collision checks and average planning times for the entire press breaking operation.

In Table 1(d), we have included some results corresponding to the learning phase without node enhancement of basic PRM. For each task, we generated a roadmap in exactly the same way as Lazy PRM generates roadmaps in the initial step. Then we checked *all* nodes for collision, deleted the colliding ones, and then checked *all* of the remaining edges as described in Section 3.3.2. In other words, we checked the *entire* roadmap for collision as efficiently as possible. Due to the long running times, only one full roadmap was explored for each task. The result gives an indication of how large fraction of the roadmap that really has to be explored, and the amount of work saved by our lazy approach, in this particular example. Note this is a conservative estimate since even with this long preprocessing, there is no guarantee that the remaining roadmap will contain a feasible path. Table 1(d) shows whether a collision-free path was found or not. We see in 1(b) that several enhancement steps are needed with Lazy PRM, thus indicating that node enhancement also is needed with basic PRM, and this will further increase running times and the number of collision checks.

5.2 Interpretation of results

We clearly see in Table 1(c) that collision checking represents the vast majority of the planning time (80%), but also that the graph building takes a lot of time (18%). Interestingly, the time spent on graph searching is negligible, about 2%. Even though we carefully select the points to check for collision by frequently searching the roadmap for a shortest path, the total time spent on that is very short.

The initial roadmaps consists of $N_{init} = 10,000$ nodes in all experiments. We see in Table 1(d) that the number of collision checks required to explore one entire roadmap is of order 500,000. Table 1(a) shows, on the other hand, that Lazy PRM in average solves the same planning tasks in 92 to 682 collision checks. Thus, Lazy PRM only explores a small fraction, less than 0.1%, of the roadmap. This is the strength of the algorithm; to either find a collision-free path or to conclude that none exists in the roadmap by using a small number of collision checks.

We also see in Table 1(a) that a large percentage, 26%, of the total number of collision checks are actually performed on the collision-free solution paths, and are therefore inevitable. This large percentage can be explained by two reasons. Firstly, the algorithm finds a sequence of collision-free nodes before edges are being checked. This prevents from planning local paths in dead ends and in regions from where no way out exists. Secondly, we check the edges along the path starting from both ends with increasing resolution, and stop as soon as a collision occurs. The colliding edge is removed from the roadmap, and a new shortest path is found. Thus, we avoid using a local planner and instead keep a global view throughout the planning process. As a consequence, very few edges – often only the edges along the final path – are checked with the finest resolution. This also makes the algorithm relatively insensitive to the resolution with which the paths are checked.

Since all of the nodes in the initial roadmap are uniformly distributed, the number of collision-free nodes found by basic PRM will give a good estimation of the relative size of $\mathcal{C}_{\mathcal{F}}$. We see in Table 1(d) that for the tasks $A \to B$, $F \to G$, and $J \to A$ approximately 40% of $\mathcal{C}$ is collision-free. For the other tasks approximately 30% of $\mathcal{C}$ is collision-free. As expected, the free part of $\mathcal{C}$ is reduced when the robot grasps a sheet of metal.

Furthermore, from the planner's point of view, the robot's tool includes both the gripper and possibly also a sheet of metal attached to it. If the tool is large and irregularly shaped, then its orientation becomes more important, whereas if the tool is small (e.g. the gripper only), the wrist motions of the robot, which basically determine the orientation, become less important. In this kind of environment, the planning problem is significantly easier if the tool is small. This explains why the tasks $A \to B$, $F \to G$, and $J \to A$ are successfully planned without any node enhancement, and reveals the strength of our method in adapting to the difficulty of the problem.

6 Discussion

The aim of Lazy PRM is essentially to minimize the number of collision checks while searching for the shortest feasible path in a roadmap in the context of a PRM planner. This is done on the expense of frequent graph search. For a complex robot working in a complex workspace, like our six dof example, collision checking is an expensive operation, and careful selection of the points being checked for collision reduces the planning time considerably.

However, if the robot and the obstacles have a very simple geometry, then collision checking is very fast. Frequent graph searching may, instead of speeding up the planning, become a bottleneck. Trading some collision checking for less graph searching may increase performance of Lazy PRM. So, instead of re-planning the entire path every time a collision is found, we can try to remove several nodes from the roadmap in each iteration of the main loop, see Figure 1. A simple way would be to always check *all* nodes along a path before searching for a new path.

Another modification of Lazy PRM is necessary when the configuration space is very cluttered. This is, for instance, the case with the ten dof robot in [25], where more than 99% of the configuration space is infeasible. If we run our algorithm, we would need a large number of nodes in the initial roadmap, and then remove from the roadmap approximately 99% of the nodes being checked, which would take a lot of time. Fortunately, we can easily modify Lazy PRM to check all nodes *before* we insert them into the roadmap. This would certainly cause unnecessary nodes to be checked for collision, but, on the other hand, we would save many inserting and removing operations in the roadmap. After that, we still have the efficient way of exploring the edges along paths. In this way, the lazy approach can be employed with most of the existing sampling schemes and variations of PRM discussed in Section 2.3.

Our primary interest in this project has been path planning in industrial environments, and the experimental results show that Lazy PRM works well in practice. By using either or both of the two modifications of the algorithm suggested above, we can tune the amount of graph search according to the application and the time required to perform a collision check, so that Lazy PRM becomes efficient for an even wider range of problems.

7 Future work

Lazy PRM has essentially one parameter that is critical for the performance – N_{init}, the initial number of nodes. As indicated in Theorem 1, N_{init} is strongly correlated to the probability of finding a feasible path without using the node enhancement step. The optimal choice depends on the dimension of $\mathcal{C}$, the workspace, the planning task, and the desired quality of the collision-free path. Our future work includes an investigation of the dependence between N_{init} and the planning time in different environments, as well as different distributions of the nodes.

Probabilistic techniques, like Lazy PRM, often give very fast planning. However, in Table 1(c), we can see that the maximum planning time is approximately twice as long as the average planning time. New improved enhancement techniques, in order to make the algorithms more robust in the sense that the worst case performance is improved, will also be a topic of our future research.

8 Summary and conclusions

In this paper we further develop probabilistic planning techniques in the direction of achieving general and practically useful single query planners. We address standard industrial applications characterized by complex geometry and high-dimensional, relatively uncluttered configuration spaces. The algorithm – called Lazy PRM – is based upon a general scheme for lazy evaluation of the feasibility of the roadmap. The scheme is simple and general and can be applied to any graph that needs to be explored. In addition to Lazy PRM, most other existing variations of PRM, and other related algorithms, can benefit from this scheme and significantly increase performance.

Acknowledgements

The authors would like to thank Bo Johansson and ABB Digital Plant Technologies AB for initiating the project and for providing suitable software. Parts of this work was performed during the visit of Robert Bohlin to the Robotics and Physical Computing Group at the Computer Science Department, Rice University. Robert Bohlin was supported by NUTEK, the Swedish National Board for Industrial and Technical Development, project P10499. Work on this paper by Lydia Kavraki has been supported in part by NSF CAREER Award IRI-970228, NSF CISE SA1728-21122N, and a Sloan Fellowship.

References

[1] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1998.

[2] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective*, pages 630–637. AK Peters, 1998.

[3] N.M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 113–120, 1996.

[4] J. Barraquand, L. E. Kavraki, J. C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for path planning. *Int. J. of Robotics Research*, 16(6):759–775, 1997.

[5] J. Barraquand and J.C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. of Rob. Research*, 10:628–649, 1991.

[6] R. Bohlin. *Motion Planning for Industrial Robots.* Licentiate thesis, Chalmers University of Technology, 1999.

[7] R. Bohlin and L.E. Kavraki. Path planning using Lazy PRM. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 2000.

[8] V. Boor, M.H. Overmars, and F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 1018–1023, 1999.

[9] L. Kavraki C. Holleman. A framework for using the workspace medial axis in PRM planners. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 2000.

[10] S. Cameron. Enhancing GJK: Computing minimum distance and penetration distanses between convex polyhedra. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 3112–3117, 1997.

[11] J.F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, MA, 1988.

[12] B. Glavina. Solving findpath by combination of goal-directed and randomized search. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 1718–1723, 1990.

[13] A. Gray. *Tubes.* Addison-Wesley, Redwood City, CA, 1990.

[14] K. Gupta and A.P. del Pobil. *Practical Motion Planning in Robotics.* John Wiley, West Sussex, England, 1998.

[15] D. Halperin and M. Sharir. A near-quadratic algorithm for planning the motion of a polygon in a polygonal environment. *Discrete Comput. Geom.*, 16:121–134, 1996.

[16] L. Han and N.M Amato. Kinematics-based probabilistic roadmap method for closed chain systems. In *Wokshop on the Algorithmic Foundations of Robotics*, 2000.

[17] T. Horsch, F. Schwarz, and H. Tolle. Motion planning for many degrees of freedom - random reflections at C-space obstacles. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1994.

[18] D. Hsu, L.E. Kavraki, J.C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In P. Agarwal, L. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective*, pages 141–154. A K Peters, 1998.

[19] D. Hsu, R. Kindel, J.C Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. In *Wokshop on the Algorithmic Foundations of Robotics*, 2000.

[20] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1997.

[21] Y.K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Comp. Surveys*, 24(3):219–291, 1992.

[22] P. Isto. A two-level search algorithm for motion planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 2025–2031, 1997.

[23] R.A. Johnson and D.W. Wichern. *Applied Multivariate Statistical Analysis.* Prentice Hall, New Jersey, 1998.

[24] L.E. Kavraki, M.N. Kolountzakis, and J.C. Latombe. Analysis of probabilistic roadmaps for path planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 3020–3025, 1996.

[25] L.E. Kavraki and J.C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1994.

[26] L.E. Kavraki, P. Švestka, J.C. Latombe, and M. Overmars. Probabilistic roadmaps for fast path planning in high dimensional configuration spaces. *IEEE Tr. on Rob. & Aut.*, 12:566–580, 1996.

[27] Y. Koga, K. Kondo, J. Kuffner, and J.C. Latombe. Planning motions with intentions. *Computer Graphics (SIGGRAPH'94)*, pages 395–408, 1994.

[28] J.C. Latombe. *Robot Motion Planning.* Kluwer, Boston, MA, 1991.

[29] J.C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *Int. J. of Rob. Research*, 18(11):1119–1128, 1999.

[30] J.P. Laumond and T. Siméon. Notes on visibility roadmaps and path planning. In *Wokshop on the Algorithmic Foundations of Robotics*, 2000.

[31] S.M. LaValle and J.J. Kuffner, Jr. Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1999.

[32] S.M. LaValle and J.J. Kuffner, Jr. Rapidly-exploring random trees: Progress and prospects. In *Wokshop on the Algorithmic Foundations of Robotics*, 2000.

[33] S.M. LaValle, J.H. Yakey, and L.E. Kavraki. A proababilistic roadmap approach for systems with closed kinematic chains. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 1671–1676, 1999.

[34] M.C. Lin and J.F. Canny. A fast algorithm for incremental distance computation. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 1008–1014, 1991.

[35] G.F. Luger and W.A. Stubblefield. *Artificial intelligence and the design of expert systems.* Benjamin/Cummings, Redwood City, CA, 1989.

[36] E. Mazer, J.M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. *J. of Art. Intelligence Research*, 9:295–316, 1998.

[37] C.L. Nielsen and L.E. Kavraki. A two level fuzzy PRM for manipulation planning. Technical Report TR2000-365, Rice University, 2000.

[38] M. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-32, Utrecht University, the Netherlands, 1992.

[39] M. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In K.Y. Goldberg, D. Halperin, J.C. Latombe, and R.H. Wilson, editors, *Algorithmic Foundations of Robotics*, pages 19–37. A K Peters, 1995.

[40] J. Reif. Complexity of the mover's problem and generalizations. In *Proc. 20th IEEE Symp. on Found. of Comp. Sci.*, pages 421–427, 1979.

[41] F. Thomas and C. Torras. Interference detection between non-convex polyhedra revisited with a practical aim. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1994.

[42] S.A. Wilmarth, N.M. Amato, and P.F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 1024–1031, 1999.

Table 1: Performance data for Lazy PRM based on 20 consecutive runs for each task. Table 1(d) shows data for PRM based on one run for each task. The initial number of nodes, N_{init}, is 10000 in all tests.

		Task			
		$A \to B$	$B \to C$	$C \to D$	$E \to F$
Lazy PRM					
Collision checks					
for nodes	ave	9	41	172	263
for edges	ave	83	125	273	235
for returned path	ave	78	60	86	82
	min	74	45	143	154
total	ave	92	166	445	499
	max	131	463	701	1010

Table 1(a).

		$A \to B$	$B \to C$	$C \to D$	$E \to F$
No. of enh. steps					
	min	0	0	0	0
	ave	0	0.3	0.8	1.9
	max	0	1	2	5

Table 1(b).

		$A \to B$	$B \to C$	$C \to D$	$E \to F$
Running time (sec.)					
graph building	ave	6.6	6.7	0.8	8.3
graph searching	ave	0	0.1	0.5	1.3
coll. checking	ave	6.1	13.3	35.4	42.3
	min	11.2	9.7	10.8	19.7
total	ave	12.7	20.2	36.8	52.0
	max	16.2	45.7	60.9	97.3

Table 1(c).

	$A \to B$	$B \to C$	$C \to D$	$E \to F$
PRM				
Collision checks				
for nodes	10000	10000	10000	10000
of which in $\mathcal{C}_{\mathcal{F}}$	4085	2942	2975	3047
for edges	763063	409561	423443	451254
total	773063	419561	433443	461254
Running time (sec.)				
total	56625	31428	32299	35840

Table 1(d).

Table 1: continued.

Task				
$F \to G$	$G \to H$	$I \to J$	$J \to A$	Total

129	134	320	18	1088 (40%)
283	158	361	124	1643 (60%)
121	80	114	82	704 (26%)
175 412 820	135 293 442	139 682 1290	81 142 299	2730

Table 1(a).

0 0 0	0 0.8 2	0 1.6 4	0 0 0	

Table 1(b).

6.5	7.3	8.2	6.6	51.0 (18%)
0.9	0.4	3.0	0	6.3 (2%)
38.3	24.7	59.8	11.6	231.6 (80%)
22.1 45.7 87.3	16.8 32.5 47.8	17.8 71.0 129.5	13.0 18.2 31.2	289.0

Table 1(c).

10000	10000	10000	10000	
3976	3038	3090	4121	
728012	447541	447000	787507	
738012	457541	457000	797507	

51774	35097	35200	56234	

Table 1(d).

Paper II

Path planning in practice; Lazy evaluation on a multi-resolution grid

Robert Bohlin
Department of Mathematics
Chalmers University of Technology
SE-412 96 Göteborg, Sweden

Abstract

We present a resolution complete path planner based on an implicit grid in the configuration space. The planner can be described as a two-level process in which a global planner restricts a local planner to certain subsets of the grid. The global planner starts by letting the local planner search in a coarse subset of the grid, and successively refines the grid until a solution is found. The local planner applies a scheme for lazy evaluation on each subgrid in order to minimize collision checking and thereby increase performance.

Experimental results in an industrial application show that lazy evaluation on a grid is very efficient in practice. The algorithm is particularly useful in high dimensional, relatively uncluttered configuration spaces, especially when collision checking is computationally expensive. Single queries are handled quickly since no preprocessing is required.

Keywords: Collision avoidance, motion planning, path planning, robotics.

1 Introduction

Path planning for robots has received much attention over the last decades. The general problem is to find collision-free paths for a robot in an environment containing obstacles. Algorithms for its solution are, however, rarely used in practice due to their computational complexity.

To be useful in practice the planner must be fast, in particular for easy problems, and generate paths that are short and smooth enough to be executed by a real robot. In industrial applications, the geometry of the robot and its environment is typically very complex, making collision checking computationally expensive. Since many existing planners rely on fast collision checking, their practical use in these situations is limited.

Our aim with this paper is to meet the requirements above and design a planner that is useful in industrial applications. By using an implicit multi-resolution grid combined with a lazy evaluation technique, we can reduce collision checking and thereby increase speed.

Before describing the algorithm, we introduce some notation and give a brief overview of existing planners.

1.1 Notation

We let $\mathcal{W}$ denote a subset of $\mathbf{R}^2$ or $\mathbf{R}^3$ in which a robot $\mathcal{A}$ is moving. The position of $\mathcal{A}$ is described by a *configuration* $\boldsymbol{q}$ such that the position of every point on $\mathcal{A}$ can be determined relative to a fixed frame in $\mathcal{W}$. The set of all configurations is called the *configuration space* and is denoted by $\mathcal{C}$. For a configuration $\boldsymbol{q} \in \mathcal{C}$, $\mathcal{A}(\boldsymbol{q})$ denotes the subset of $\mathcal{W}$ occupied by $\mathcal{A}$.

The cardinality of $\mathcal{C}$ is generally infinite since the robot is assumed to move continuously in $\mathcal{W}$. In what follows we assume that $\mathcal{C}$ can be identified with a subset of $\mathbf{R}^d$, where d is equal to the number of degrees of freedom (dof) of $\mathcal{A}$. For convenience we let $\mathcal{C}$ also denote this subset of $\mathbf{R}^d$, thus $\boldsymbol{q}$ also denotes a point in $\mathbf{R}^d$. For example if $\mathcal{A}$ is an articulated robot arm, we can let $\mathcal{C} = I_1 \times \cdots \times I_d$, where I_j is the range of joint j.

The aim of path planning is to avoid a set of obstacles $\mathcal{W}_\mathcal{O}$ in $\mathcal{W}$. If $\mathcal{A}$ intersects $\mathcal{W}_\mathcal{O}$ we say that $\mathcal{A}$ *collides* with the obstacles, and we define the mapping $\Phi : \mathcal{C} \to \{0,1\}$ as

$$\Phi(\boldsymbol{q}) = \begin{cases} 0 & \text{if} \quad \mathcal{A}(\boldsymbol{q}) \bigcap \mathcal{W}_\mathcal{O} \neq \emptyset \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

This mapping, which is called the *collision checker*, divides the configuration space into two disjoint sets $\mathcal{C}_\mathcal{O}$ and $\mathcal{C}_\mathcal{F}$ such that $\mathcal{C}_\mathcal{O} = \Phi^{-1}(0)$ is the set of colliding configurations and $\mathcal{C}_\mathcal{F} = \Phi^{-1}(1)$ is the set of collision-free (or *feasible*) configurations.

Given an initial configuration $\boldsymbol{q}_{init} \in \mathcal{C}_\mathcal{F}$ and a final configuration $\boldsymbol{q}_{goal} \in \mathcal{C}_\mathcal{F}$, we define the path planning problem as follows: Find a continuous path $\gamma : [0,1] \to \mathcal{C}$ such that $\gamma(0) = \boldsymbol{q}_{init}$, $\gamma(1) = \boldsymbol{q}_{goal}$ and $\gamma(t) \in \mathcal{C}_\mathcal{F}$ for all $t \in [0,1]$, or determine that no such path exists. An algorithm that solves this problem, or a variation of it, is called a *path planner* or simply a *planner*.

To measure distances in $\mathcal{C}$ we need a metric ρ_{path}. This metric is also used to measure lengths of paths. For simplicity paths are ranked by length, and we prefer short paths with respect to ρ_{path}. So, by defining this metric, we decide which paths are of high quality and which paths are of poor quality.

1.2 Previous work

The path planning problem has been extensively studied in the last decades, and a number of different approaches are proposed; see [7, 11, 15] for overviews. An algorithm is called *complete* if it always will find a solution or determine that none exists. However, due to the complexity of the path planning problem, complete planners are too slow to be useful in practice [5].

Another category of planners discretize the configuration space. If these planners are complete in the limit as the discretization approaches a continuum, they are called *resolution complete*. See [6, 8, 14] for resolution complete planners.

A general problem of discretizing the configuration space is that the memory requirement grows rapidly with the dimension. In [8], this problem is solved by an *implicit* representation of a grid. A version of the A*-algorithm [17] is then used to find a feasible path in the grid. They also show different ways of choosing the discretization of the configuration space.

Trading completeness for speed, randomized techniques have been successfully applied to many problems in high-dimensional configuration spaces. The Randomized Path Planner (RPP) in [2] uses a potential field as guidance towards the goal, and random walks to escape local minima.

The Ariadne's clew algorithm in [18] incrementally builds a tree of feasible paths using genetic optimization. Considering the initial configuration as a landmark, the planner finds a path from one of the landmarks to a point as far as possible from all previous landmarks. A new landmark is placed at this point. New landmarks are placed until the goal configuration can be connected to the tree.

The Probabilistic Roadmap Method (PRM) [12, 13, 20] is a method that has been shown to work well in practice in high-dimensional configuration spaces. The idea is to represent and capture the connectivity of $\mathcal{C}_{\mathcal{F}}$ by a random network, a *roadmap*, whose nodes correspond to randomly selected configurations. If a local planner finds a feasible path between two nodes, they are connected by an edge. See also [1, 9] for methods to increase the connectivity of the roadmap. If the start and goal configurations can be connected to the same component of the roadmap, then a solution has been found. PRM is particularly useful for multiple queries, since once an adequate roadmap has been created, queries can be answered very quickly.

Lazy PRM in [3] is a probabilistic roadmap planner well suited for single queries. The underlying idea is to minimize collision checking by introducing a scheme for lazy evaluation of the nodes and edges in the roadmap. The scheme is particularly useful when collision checking is expensive, for example in industrial applications with complex geometry. See also [19] for a related technique. A recent approach using quasi-random sampling is described in [4]

Other methods, described in [10] and [16], build two trees rooted at the initial and goal configurations respectively. In [10], the trees are expanded by generating new nodes randomly in the vicinity of the two trees, and connecting them to the trees by a local planner. The planner in [16] iteratively generates a configuration, an attractor, uniformly at random in $\mathcal{C}$. Then, for both trees, the node closest to the attractor is selected and a local planner searches for a path of a certain maximum length towards the attractor. A new node is placed at the end of both paths. The process stops when the two trees intersect.

1.3 Information collection

A path planner may obtain information about the configuration space, or rather the obstacles in the configuration space, in different ways. Canny [5] represents the boundary of $\mathcal{C}_{\mathcal{F}}$ by a set of algebraic equations giving complete information about the obstacles in $\mathcal{C}$. Unfortunately, they are very complex, and are difficult to use in practice.

A simpler, but much less informative way, is to sample at certain points in $\mathcal{C}$. To evaluate Φ at a configuration $\boldsymbol{q}$, we determine the position and orientation of all links of the robot and check whether or not they intersect the obstacles $\mathcal{W}_{\mathcal{O}}$. The minimum distance δ to the obstacles gives somewhat more information. If $\delta > 0$, then it is possible to determine a ball around $\boldsymbol{q}$ which is entirely in $\mathcal{C}_{\mathcal{F}}$, and if $\delta \leq 0$, then $\boldsymbol{q} \in \mathcal{C}_{\mathcal{O}}$.

Most of the resolution complete planners and roadmap planners mentioned in Section 1.2 use either of the two sampling methods above. For simple robots and obstacles they are relatively straightforward, but for robots and obstacles with complex geometric descriptions (e.g. thousands of polyhedra) they are computationally expensive. In particular the latter method, to find the minimum distance to the obstacles, become very complex when the geometric model contains curved surfaces.

The planner we describe in this paper can be used with either of the above sampling methods. Our aim, however, is to keep the planner as simple and general as possible. Therefore we only calculate intersections in $\mathcal{W}$ and not distances, i.e., we require that the planner only obtains information by evaluating Φ point-wise.

2 The algorithm

The path planner presented in this paper is based on a discretization of the configuration space. A dense graph (a grid) $\mathcal{G}$ is placed in $\mathcal{C}$, and only paths contained in the graph are considered. We assume that $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$ have been specified, and for simplicity we assume that they coincide with two nodes (also denoted $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$) in $\mathcal{G}$. Unless otherwise stated, a *path* will always refer to a path in $\mathcal{G}$ connecting $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$.

The planner is resolution complete in the sense that if a feasible path exists in the graph, it will be found. The essential is to find a feasible path in a minimum number of evaluations of Φ.

2.1 Algorithm overview

The approach to search in $\mathcal{G}$ for a feasible path can, as many other planners (e.g. [6, 13, 18]), be described as a two-level planning process. A global planner on the top level restricts the local planner on the lower level to a certain subset $\mathcal{G}'$ of $\mathcal{G}$. The subset $\mathcal{G}'$ is successively extended until a solution is found or $\mathcal{G}' = \mathcal{G}$. If the local planner still fails in the latter case, then no solution exists in $\mathcal{G}$ and the planner returns failure.

The motivation for using subsets of $\mathcal{G}$ is to let the local planner search for simple solutions first. A relatively coarse subset of $\mathcal{G}$ is often sufficient to find a solution in many practical situations, so the scheme of refining $\mathcal{G}'$ makes the planner efficient for simple as well as more difficult planning tasks. Before going into the details of the global and local planners, we need to describe the representation of $\mathcal{G}$ and $\mathcal{G}'$.

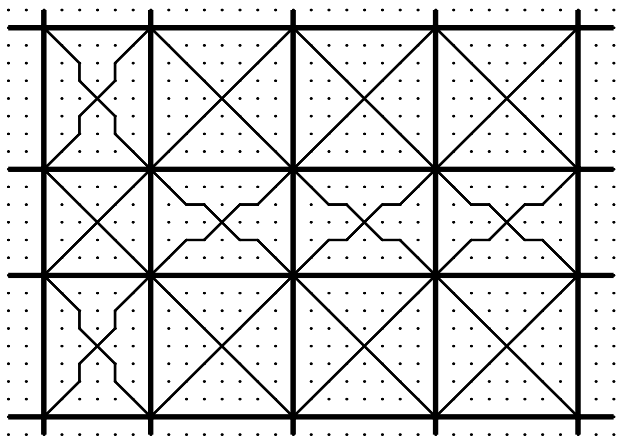


Figure 1: Example of a 35×25 grid defining $\mathcal{G}$ in a two-dimensional configuration space. Thick lines show enabled hyper planes and thin lines show diagonal edges of $\mathcal{G}'$. Edges of $\mathcal{G}$ are omitted.

2.2 Configuration space representation

To simplify the presentation, we assume that $\mathcal{C}$ is an axis aligned rectangle in $\mathbf{R}^d$. The nodes in $\mathcal{G}$ are defined by the points in a rectangular grid of size $n_1 \times \cdots \times n_d$. We would like to traverse $\mathcal{G}$ parallel with the coordinate axes as well as along diagonals, so we add the appropriate edges. That is, each interior node gets $3^d - 1$ neighbors.

The resolution specified by the parameters $\{n_i\}_{i=1}^d$ is the finest resolution that will be used in $\mathcal{C}$, and determines, for example, the step size with which path segments are checked; if adjacent nodes are in $\mathcal{C}_\mathcal{F}$, we consider the edge between them as being feasible. This means that Φ is only evaluated at nodes in $\mathcal{G}$.

The dimension d of $\mathcal{C}$ is often high and the number of grid points grows rapidly with the dimension, so there is no way to explicitly represent $\mathcal{G}$ in a computer. (In our experiments presented in the next section we let n_i be up to 255, giving more than 10^{14} nodes.) A convenient way of implicitly representing the underlying grid is to define each grid point as the intersection of d hyper planes. For each dimension i, place n_i planes equally spaced in $\mathcal{C}$ and perpendicular to the i:th coordinate axis. Then we get $\prod_{i=1}^d n_i$ intersections between d planes which define the grid points.

2.3 Subgraphs

We can get a sparser graph $\mathcal{G}'$ by disabling some of the planes in the underlying grid, and define the nodes in $\mathcal{G}'$ as the intersections of the enabled planes only, see Figure 1. Note that the only planes that *must* be enabled are the planes that contain $\boldsymbol{q}_{init}$ or $\boldsymbol{q}_{goal}$; if these planes are disabled, $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$ will not be nodes in $\mathcal{G}'$.

In $\mathcal{G}'$ we also introduce edges so that we can traverse the graph parallel with the coordinate axes and along diagonals. Note that since we introduced a number of new edges, $\mathcal{G}'$ is no longer a subgraph of $\mathcal{G}$, but if we instead associate each edge of $\mathcal{G}'$ with the sequence of nodes in G that is "covered" by the edge, we can consider $\mathcal{G}'$ as a subgraph of $\mathcal{G}$, see Figure 1.

2.4 Evaluated nodes

As the planning process proceeds Φ will be evaluated at more and more nodes, but since $\mathcal{G}$ is implicitly represented we cannot associate unique information to each node. However, only evaluated nodes need to be represented; one set of feasible nodes and one set of colliding nodes. These sets are kept updated at any time in order to avoid any node being evaluated more than once. The colliding nodes can be seen as nodes deleted from $\mathcal{G}$.

Since evaluating Φ is expensive (≈ 0.08 seconds on average in our test example in Section 3), the number of nodes that within reasonable running time can be evaluated is only a small fraction of the total number of nodes in $\mathcal{G}$. Thus, these sets of nodes will never be too large.

2.5 Global planning

The global planner controls to which subgraph $\mathcal{G}'$ the local planner is restricted. By starting with a sparse subgraph $\mathcal{G}'$ defined by a only a few enabled planes, the local planner quickly solves many easy planning tasks. If no solution exists in $\mathcal{G}'$, the global planner refines the grid by enabling a few more planes. If all planes are enabled and the local planner still cannot find a solution, then there is no feasible path in $\mathcal{G}$.

In our experiments presented in Section 3, we apply the planner to an articulated robot arm with six dof. The following simple strategy of enabling hyper planes show great performance. Initially, we enable the planes containing $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$, and in each of the dimensions 1,2 and 3, we enable another six planes as evenly as possible. (This generally gives $(2+6)^3 \cdot 2^3 = 4096$ nodes in $\mathcal{G}'$.) Each time the local planner fails, we enable the plane in $\mathcal{G}$ that is furthest from an enabled plane. Here, of course, the distances are measured between parallel planes only.

2.6 Local planning

The local planner is given a subgraph $\mathcal{G}' \subset \mathcal{G}$ from the global planner and starts an exhaustive search for a solution. The technique is similar to the scheme of lazy evaluation presented in [3], i.e., the aim is to find the shortest (with respect to the metric ρ_{path}) feasible path in $\mathcal{G}'$ in a minimum number of evaluations of Φ.

Based on the current status of $\mathcal{G}'$, in which some nodes have been evaluated and others have not, the local planner picks the shortest path $\mathcal{P}$, called a candidate path. Recall that all nodes that have been evaluated to collision are deleted from $\mathcal{G}$. The candidate path is then checked for collision according to a certain scheme described below. As soon as a collision is detected, we know that this is not the path we are looking for, so we delete the colliding node and pick a new candidate path. This procedure is repeated until a feasible path is found or no candidate path exists in $\mathcal{G}'$.

When checking the path, we first evaluate the nodes. Starting respectively with the first and the last node on $\mathcal{P}$ and working toward the center, we alternately evaluate the nodes along the path. If all nodes are feasible then we check the edges on $\mathcal{P}$, first with a coarse resolution and then we do stepwise refinements until all edges are feasible. As soon as a collision is found, we delete the corresponding node from $\mathcal{G}$ (recall that edges of $\mathcal{G}'$ refers to a sequence of nodes of $\mathcal{G}$, see Figure 1), and reject $\mathcal{P}$.

The hope when applying this scheme to a candidate path is of course that it is feasible. On the other hand, if it is not feasible, we would like to detect that as quickly as possible in order to avoid evaluating nodes on a path that will be rejected anyway. At the same time we want to reject as many *other* colliding candidate paths as possible. The following two observations motivate the procedure of evaluating nodes before edges, and checking nodes from outside in. First, a colliding node in $\mathcal{G}'$ will cut away more than a colliding edge in $\mathcal{G}'$. Second, a node close to $\boldsymbol{q}_{init}$ or $\boldsymbol{q}_{goal}$ is likely to cut away a larger portion of the colliding candidate paths than a node far away from $\boldsymbol{q}_{init}$ and $\boldsymbol{q}_{goal}$.

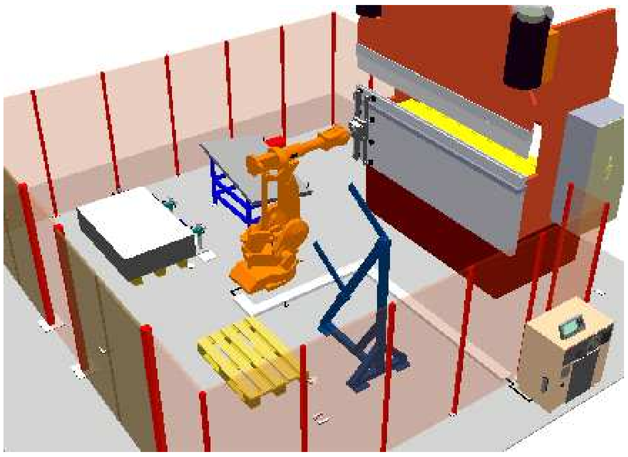


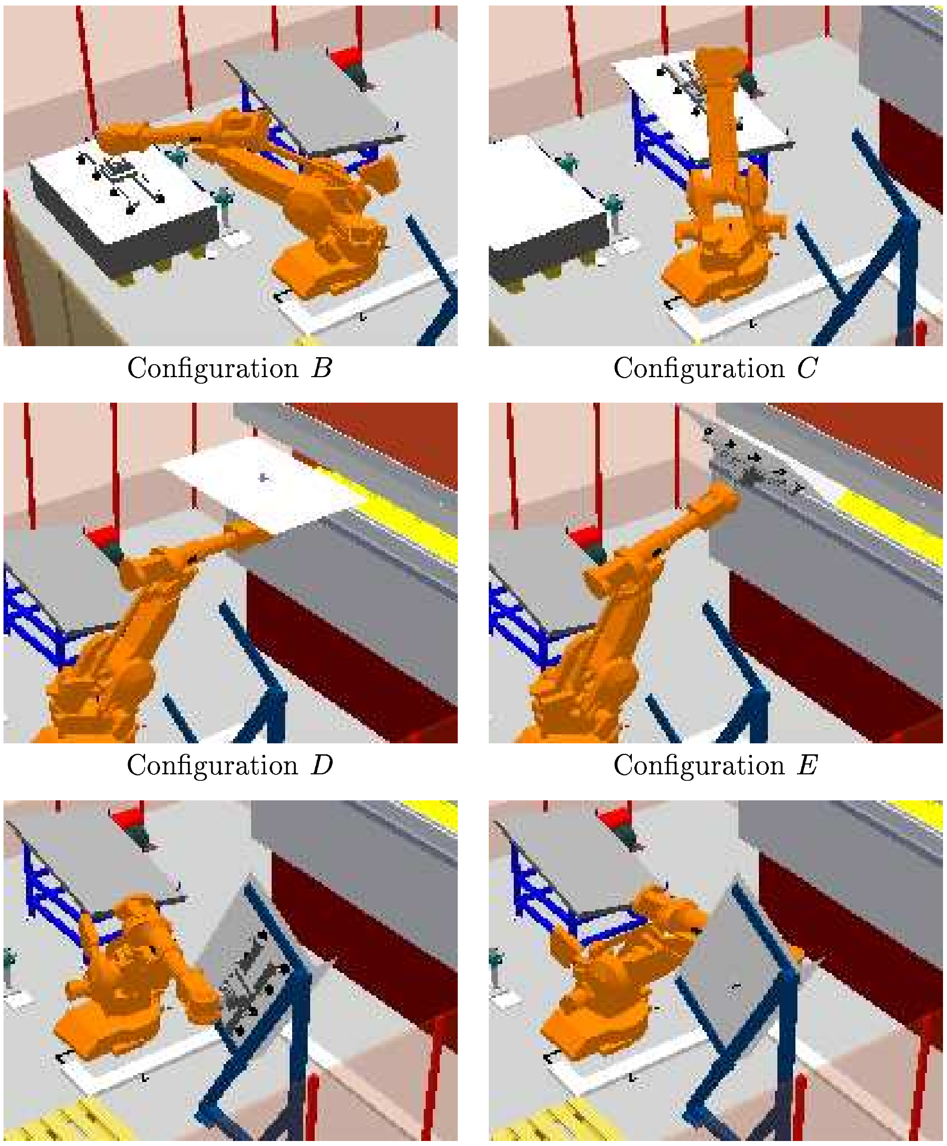
Figure 2: The workspace used in our experiments. The robot is in its home configuration denoted by A.

2.7 Tree of shortest paths

A question is how the local planner finds a candidate path in an implicitly represented graph. Common for all graph search algorithms like Dijkstra's and A* [17] is that they visit nodes one by one and insert them into a tree whose root is the start node. The tree contains the shortest path from each of the visited nodes to the root. New nodes are visited until the goal node is reached. Then the shortest path can be found by tracing the way back to the root.

Our local planner uses the A* algorithm. Unfortunately one cannot avoid an explicit representation of the tree, which in the worst case can contain every node in the graph. The iterative behavior of the local planner makes the candidate paths longer and longer, so the tree grows over larger and larger portion of $\mathcal{G}'$. However, what seems to be a problem is also a solution; in each iteration one node is deleted, so the growth will be impeded as well. In practice only a fraction of the nodes will be visited.

Each time a candidate path is rejected, the tree of shortest paths becomes invalid because a deleted node is contained in the tree. Instead of rebuilding the entire tree, we can update the part of the tree that is affected. Only the subtree whose root is the deleted node needs to be updated, which saves much time.



Configuration B Configuration C

Configuration D Configuration E

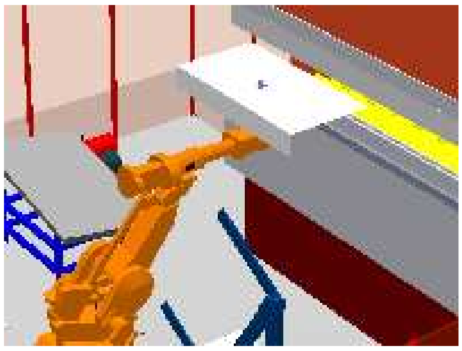
Configuration F Configuration G

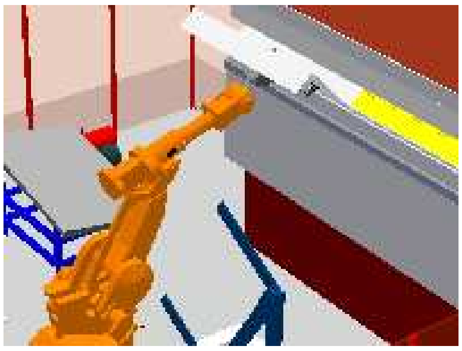
Figure 3: Configurations B to G used in the experiments.

3 Experimental results

The planner described in previous section has been implemented in C++ as a plug-in module to RobotStudio[1] – a simulation and off-line programming

[1]RobotStudio is developed by ABB Robotics, Göteborg, Sweden.


Configuration H


Configuration I

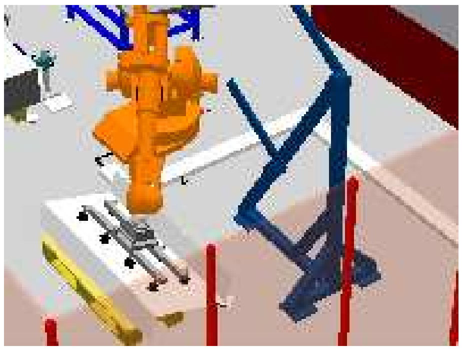

Configuration J

Figure 4: Configurations H to J used in the experiments.

software running under Windows NT. The collision checks are handled internally in RobotStudio. The experiments have been run on a PC with a 400 MHz Pentium II processor and 512 MB RAM.

Our test example is a part of a manufacturing process in which an ABB 4400 robot is tending press breaking. In this particular case, plane sheets of metal are picked from a pallet, bent twice by the hydraulic press shown in Figure 2, and then placed at another pallet.

Ten different configurations denoted A to J are shown in Figures 2 and **??**. These are used as either initial or goal configurations in eight planning tasks, denoted for example $A \rightarrow B$, where A is the initial configuration and B is the goal configuration. This setting is identical with the experiments reported in [3] and we have chosen the parameters $\{n_i\}_{i=1}^{d}$ to conform to the parameters in [3]. Our aim is to compare our planner with Lazy PRM.

	Collision checks				Planning time			
Task	Total		On returned path		Total		Collision checking	
$A \to B$	127	(92)	92	(78)	7.6	(12.7)	7.4	(6.1)
$B \to C$	63	(166)	59	(60)	5.0	(20.2)	4.9	(13.3)
$C \to D$	364	(445)	103	(86)	32.6	(36.8)	27.8	(35.4)
$E \to F$	282	(499)	80	(82)	24.3	(52.0)	23.5	(42.3)
$F \to G$	333	(412)	131	(121)	26.4	(45.7)	25.9	(38.3)
$G \to H$	67	(293)	54	(80)	5.0	(32.5)	4.9	(24.7)
$I \to J$	337	(682)	111	(114)	31.5	(71.0)	28.8	(59.8)
$J \to A$	98	(142)	87	(82)	6.8	(18.2)	6.4	(11.6)
Sum:	1671	(2730)	717	(704)	139.2	(289.0)	129.8	(231.6)
			43%	(26%)			93%	(80%)

Table 1: Performance data for our planner in a test environment. Within parenthesis are corresponding data for Lazy PRM based on 20 consecutive runs for each task.

Table 1 shows the total number of collision checks and the total planning time required to solve each of the tasks. The number of collision checks on the solution path and the time spent on collision checking are also shown. Within parenthesis are the corresponding results for Lazy PRM reported in [3]. Since Lazy PRM is a probabilistic planner, we use the average values over 20 consecutive runs.

In Table 1 we find a significant difference between the planners. The number of collision checks to solve all eight tasks is clearly reduced (1671 compared with 2730). This also reduces the planning time to the same extent (139 seconds compared with 289 seconds).

To be a randomized planner, Lazy PRM is very efficient since as much as 26% of the collision checks are on the solution path. Our new planner is even more efficient, 43% are on the solution path. Moreover, our planner spends less time building and searching in the graph. As much as 93% of the planning time is spent on collision checking, whereas Lazy PRM spends 80% on collision checking.

The underlying principle that makes the new planner better in many situations is that the nodes are distributed on a grid. Sparse regions and dense clusters certainly occur if the nodes are randomly distributed, but on a grid (or a subgrid) these can be avoided, or even controlled to improve planning.

4 Summary

We have presented a resolution complete path planner based on an implicit multi-resolution graph in $\mathcal{C}$. To minimize collision checking, and thereby increase performance, a scheme for lazy evaluation is applied to the graph.

The algorithm is particularly useful in high dimensional, relatively uncluttered configuration spaces, especially when collision checking is an expensive operation. Single queries are handled very quickly since no preprocessing is required. The planner uses only a boolean collision checker which makes the planner easy to apply even in situations with complex geometry, like in industry. A comparison with Lazy PRM shows significant improvements in terms of planning time.

Acknowledgements

The author would like to thank Bo Johansson for valuable supervision and support, Lydia Kavraki for discussions giving insights in the field of path planning, and ABB Robotics for initiating the project and for providing suitable software.

References

[1] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective*, pages 630–637. AK Peters, 1998.

[2] J. Barraquand and J.C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. of Rob. Research*, 10:628–649, 1991.

[3] R. Bohlin and L.E. Kavraki. Path planning using Lazy PRM. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 2000.

[4] M.S. Branicky, S.M LaValle, K. Olson, and L. Yang. Quasi-randomized path planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 2001.

[5] J.F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, MA, 1988.

[6] P. C. Chen and Y. K. Hwang. SANDROS:a dynamic graph search algorithm for motion planning. *IEEE Tr. on Rob. & Aut.*, 14(3):390–403, 1998.

[7] K. Gupta and A.P. del Pobil. *Practical Motion Planning in Robotics.* John Wiley, West Sussex, England, 1998.

[8] D. Henrich, C. Wurll, and H. Wörn. On-line path planning with optimal c-space discretization. In *Proc. IEEE/RSJ Int. Conf. on Int. Rob. and Syst.*, 1998.

[9] T. Horsch, F. Schwarz, and H. Tolle. Motion planning for many degrees of freedom - random reflections at C-space obstacles. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1994.

[10] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1997.

[11] Y.K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Comp. Surveys*, 24(3):219–291, 1992.

[12] L.E. Kavraki and J.C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1994.

[13] L.E. Kavraki, P. Švestka, J.C. Latombe, and M. Overmars. Probabilistic roadmaps for fast path planning in high dimensional configuration spaces. *IEEE Tr. on Rob. & Aut.*, 12:566–580, 1996.

[14] K. Kondo. Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration. *IEEE Tr. on Rob. & Aut.*, 7(3):267–277, 1991.

[15] J.C. Latombe. *Robot Motion Planning.* Kluwer, Boston, MA, 1991.

[16] S.M. LaValle and J.J. Kuffner, Jr. Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1999.

[17] G.F. Luger and W.A. Stubblefield. *Artificial intelligence and the design of expert systems.* Benjamin/Cummings, Redwood City, CA, 1989.

[18] E. Mazer, J.M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. *J. of Art. Intelligence Research*, 9:295–316, 1998.

[19] C.L. Nielsen and L.E. Kavraki. A two level fuzzy PRM for manipulation planning. In *Proc. IEEE/RSJ Int. Conf. on Int. Rob. and Syst.*, 2000.

[20] M. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In K.Y. Goldberg, D. Halperin, J.C. Latombe, and R.H. Wilson, editors, *Algorithmic Foundations of Robotics*, pages 19–37. A K Peters, 1995.

Paper III

Rigid body path planning using the Green kernel in SE(3)

Robert Bohlin
Department of Mathematics
Chalmers University of Technology
SE-412 96 Göteborg, Sweden

Abstract

In this paper we present a novel potential field method for rigid body path planning. The planning is performed directly in the group $SE(3)$ and is reinforced by a potential function in the workspace. Thus, the planner benefits from the explicit representation of the workspace obstacles at the same time as the planning takes place in the 6-dimensional configuration space. The potential function is harmonic and is composed of translates of the Green kernel in $SE(3)$. The planner collects information about the configuration space by sampling, and updates the potential function accordingly. Experimental results from challenging examples show that the method has great potential for future research. We provide the mathematical tools needed for further development in various directions.

Keywords: Collision avoidance, Green kernel, fundamental solution, harmonic functions, motion planning, path planning, potential fields, robotics.

Contents

1 Introduction

Path planning has been a central topic in robotics research for several decades. The path planning problem is to find a feasible path for a robot from a given initial configuration to a goal configuration. The increasing interest in automatically generated paths is mainly due to the wide variety of applications. Planners can be used for instance with off-line programming systems, in computer animations, by autonomous vehicles and space craft, and in pharmaceutical drug design, see [10, 17, 28, 34, 35, 38]. From these applications we see that a robot, from the planner's point of view, may be *any* object moving in an environment containing obstacles.

To focus on the core of the problem, we consider the basic form of path planning where we assume that a complete geometric model of the robot and its environment is available. Then we have full knowledge of the kinematics of the robot, no errors are present, and no sensing is needed. We also assume that all constraints on the robot are holonomic. Such constraints can somehow be eliminated by re-parameterizing the robot, whereas non-holonomic constraints cannot be eliminated. Non-holonomic constraints typically appear for vehicles with limited turning radius, see [37].

One family of path planning problems is called the *generalized mover's problem.* This family includes, for example, planning for one or more rigid bodies moving freely in space, and for robot arms. Complete algorithms do exist, but they are rarely used in practice due to their computational complexity, see [7, 49]. Reif showed in [46] that the generalized mover's problem belongs to the class of PSPACE-hard problems. It is believed that any complete algorithm requires time exponential in the number of degrees of freedom of the robot. This complexity bound is obtained by the complete algorithm by Canny in [7] .

The complexity issue, the broad definition of robot, and the large variation of environments and applications, motivate the development of planners customized for specific conditions. To circumvent the computational complexity in high-dimensional configuration spaces, randomized methods have recently gained much interest. Trading completeness for speed and applicability, these methods have successfully solved many difficult planning tasks, see [3, 30].

In this paper we develop the theoretical tools that are needed for a novel potential field planner. The planner is tailored for rigid bodies whose configuration space is a subset of the Special Euclidean group $SE(3)$. Our aim is to create a harmonic potential and conduct path planning directly in the group $SE(3)$. The planner collects information by sampling. The sampling is directed by a harmonic potential in the workspace. Before describing the theoretical tools and the algorithm, we introduce some notation and give a brief overview of existing potential field planners.

1.1 Notation

Let $\mathcal{W}$ denote a workspace in which a robot $\mathcal{A}$ is moving. We assume that $\mathcal{W}$ is a subset of $\mathbf{R}^2$ or $\mathbf{R}^3$ and that there is a fixed coordinate system $F_{\mathcal{W}}$ in $\mathcal{W}$. The subset of $\mathcal{W}$ which is occupied by obstacles is denoted by $\mathcal{W}_{\mathcal{O}}$. Its complement, $\mathcal{W}_{\mathcal{F}} = \mathcal{W} \setminus \mathcal{W}_{\mathcal{O}}$, is the set of free points in the workspace.

The position of $\mathcal{A}$ is described by a *configuration* $\boldsymbol{q}$ such that the position of every point on $\mathcal{A}$ can be determined relative to the fixed frame $F_{\mathcal{W}}$. The set of all configurations is called the *configuration space* and is denoted by $\mathcal{C}$. In general, the cardinality of $\mathcal{C}$ is infinite since the robot is assumed to move continuously in $\mathcal{W}$. For a configuration $\boldsymbol{q} \in \mathcal{C}$, $\mathcal{A}(\boldsymbol{q})$ denotes the subset of $\mathcal{W}$ occupied by $\mathcal{A}$.

A configuration is said to be *feasible* if the robot does not violate any constraint that may be imposed. The most obvious constraint arises from the obstacles $\mathcal{W}_{\mathcal{O}}$, but other constraints such as joint limits and self-intersection must also be considered. These constraints divide the configuration space into two disjoint sets; the set of feasible configurations is denoted by $\mathcal{C}_{\mathcal{F}}$ and its complement, $\mathcal{C}_{\mathcal{O}} = \mathcal{C} \setminus \mathcal{C}_{\mathcal{F}}$, is called the configuration space obstacle. Although all constraints not necessarily relate to collisions in the workspace, we say that a configuration is *colliding* if it belongs to $\mathcal{C}_{\mathcal{O}}$.

The aim of path planning is to generate feasible paths between pairs of configurations. In the configuration space the problem is formulated as follows. Given an initial configuration $\boldsymbol{q}_{init} \in \mathcal{C}_{\mathcal{F}}$ and a final configuration $\boldsymbol{q}_{goal} \in \mathcal{C}_{\mathcal{F}}$, find a continuous curve in $\mathcal{C}_{\mathcal{F}}$ from $\boldsymbol{q}_{init}$ to $\boldsymbol{q}_{goal}$, or determine that no such path exists. An algorithm that solves this problem, or a variation of it, is called a *path planner* or simply a *planner*.

1.2 Previous work

Many path planners have been proposed over the years, see [21, 24, 37] for overviews. The planners can be divided into categories based on the ideas involved. In this paper we further develop potential field planners, and restrict our overview to this category.

1.2.1 Potential field methods

The idea of potential field planners is to define a potential function over the configuration space. The negative gradient of the potential function induces a vector field that we interpret as a force field. Recall that the robot is a point in the configuration space. Then we can let the force field influence the robot and let the robot follow the flow of the field by simply iteratively taking a small step in the steepest descent direction of the potential.

The crucial in potential field planning is to construct a potential field that helps us solve the problem. In low dimensions, the space can be discretized by a grid and each node can be assigned a potential in several ways. Starting at the goal node, a breadth first search scheme (c.f. Dijkstra's graph search and wave front propagation) can be applied and assign a potential to each grid point in proportion to the distance in $\mathcal{C}_{\mathcal{F}}$ to the goal, see [3, 37] for related techniques.

Another way to create a potential function in $\mathcal{C}$ is to connect neighbors in the grid by edges of unit resistance, see [42, 52, 54]. By assigning 0 to the grid points in $\mathcal{C}_{\mathcal{O}}$ and -1 to the goal point and then calculating the resulting potential in the rest of the grid, we get a potential function without local minima except at the goal. In fact, the discretized function is harmonic. This method is known as the resistive grid method and is basically the same as the method described in [9, 12] and in Section 4.1.1. A similar technique is used in [55].

The method of discretizing the configuration space with high resolution and then numerically calculating a potential restricts the grid method to low dimensional configuration spaces. In higher dimension, there are other ways to construct the potential field. A frequently used method is to let the potential be composed by one force field that pulls the robot towards the goal and one force field that pushes the robot away from the obstacles. The difficulty, however, lies in constructing a potential function without local minima in which the robot gets stuck, see [32]. Several techniques have been developed in order to escape from local minima, see [8]. The planner in [3], for instance, uses random walks.

Other related techniques are considered in [29] and [45]. The former method uses neural networks for developing potential fields. The latter method is called the virtual springs method and considers a robot manipulator as a dynamical system in which the links of the robot are somewhat flexible springs. While the end-effector is attracted to a hare following a prescribed trajectory in the workspace, the rest of the arm is repelled from the obstacles by a force field. As with other potential field methods, the robot sometimes get stuck in a local minimum without reaching the target.

If the configuration space is extended by a time dimension, similar techniques can be used to coordinate multiple robots by adding more fields repelling the robots from each other, see [32]. Also moving obstacles can be treated in this way.

To create an ideal potential field in $\mathcal{C}$, the obstacles in $\mathcal{W}$ must be mapped into $\mathcal{C}$. This is easy in cases that only involve translations. Then the configuration space obstacle is the set $\mathcal{W}_{\mathcal{O}} \ominus \mathcal{A}$, see Section 4.1.2. In other cases, the configuration space obstacle can be represented by a semi-algebraic set if the robot is parameterized in a certain way, see [7]. However, the complexity of these equations make them difficult to use for robots with many degrees of freedom, say more than three. Moreover, as soon as rotations are involved the configuration space obstacles become very complex. For practical robots, for example rigid bodies and articulated robot arms in $\mathbf{R}^3$, this method is difficult to apply.

To the author's knowledge, potential fields arising from harmonic functions that have been used for path planning have either been calculated numerically, see [11, 12, 40, 42, 52], or have only been applicable to special cases in 2-dimensional configuration spaces, see [16, 26, 33, 52]. Similar techniques for time-varying workspaces can be found in [16, 48]. In this paper we introduce harmonic functions in the six-dimensional group $SE(3)$. Then we combine the potential field in $SE(3)$ with a field in the workspace. The latter field is computed using the well established method in [12].

2 Rigid body motions

For a rigid body robot $\mathcal{A}$ moving freely in space, a configuration specifies the position and orientation of $\mathcal{A}$. If we attach a coordinate frame to $\mathcal{A}$, we can identify each configuration with a unique rigid body transformation of this frame with respect to a fixed frame in $\mathcal{W}$. That is, we interpret the configuration space as the space of rigid body transformations.

The most general displacement of a rigid body is a translation plus a rotation, see [19]. Hence, a rigid body transformation is a pair (R, t) where R belongs to the rotation group $SO(n)$ of $\mathbf{R}^n$ and t is a translation. The set of vectors in $\mathbf{R}^n$ is a natural parameterization of translations, so in the following sections we will first pay attention to the rotation group. Robotics are mainly concerned with rotations in $\mathbf{R}^2$ and $\mathbf{R}^3$, so from now on we restrict our attention to $SO(3)$, and to some extent $SO(2)$.

Section 2.1 describes different ways of parameterizing $SO(3)$. In Section 2.2 we return to general rigid body transformations and look into the structure of the set of pairs (R, t). Most of the results in this section are more or less well known and references for further reading are given. Some results will be explicitly referred to later in this article, whereas other are used in a more latent way. In particular, most tools and formulas are needed for the implementation described in Section 5.2.

2.1 The rotation group $SO(3)$

The group $SO(3)$ is the *special orthogonal group*, here represented by the space of orthogonal matrices with determinant $+1$. The conditions for pairwise orthogonality and unit length of columns give six equations that must be satisfied. Thus, in principle, we can choose three suitable elements in the matrix and then let the six equations determine the rest of the matrix. This parameterization, however, is not very intuitive and useful in applications. In what follows we will describe more common ways to parameterize $SO(3)$.

Before going into the details of $SO(3)$ we look at $SO(2)$. The group $SO(2)$ is a subgroup of $SO(3)$ and can be seen as rotations about the z-axis. The group $SO(2)$ can be represented by matrices of the form

$$R = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

and can be parameterized by an angle $\theta \in [0, 2\pi)$ together with modulo 2π arithmetic. This parameterization, however, is not a global chart, i.e. a coordinate system, on $SO(2)$, because the group manifold is homeomorphic to a circle.

2.1.1 Euler angles

A widely used parameterization of rotations is Euler angles. A rotation is specified by applying three successive elementary rotations around the x-, y-

or z-axis. The elementary rotations matrices $R_x(\gamma)$, $R_y(\beta)$, and $R_z(\alpha)$ are explicitly given in Appendix B. The classical roll-pitch-roll Euler angles give the rotation

$$R_{\text{Euler}}(\alpha, \beta, \gamma) = R_z(\alpha)R_y(\beta)R_z(\gamma),$$

and is interpreted as a rotation about the z-axis by an angle α, followed by a rotation about the new y-axis by an angle β, and finally a rotation about the new z-axis by an angle γ. Another common set of elementary rotations is the roll-pitch-yaw angles, which are interpreted in a similar fashion, and give the rotation

$$R_{\text{RPY}}(\alpha, \beta, \gamma) = R_z(\alpha)R_y(\beta)R_x(\gamma).$$

A drawback of Euler angles, or any similar combination of elementary rotations, is that the parameterization is singular. Although the mapping from the set of Euler angles onto $SO(3)$ is continuous, the inverse is not continuous. In particular

$$R_{\text{Euler}}(\alpha, 0, \gamma) = R_{\text{Euler}}(\gamma, 0, \alpha)$$

and

$$R_{\text{RPY}}(\alpha, -\pi/2, \gamma) = R_{\text{RPY}}(\gamma, -\pi/2, \alpha)$$

for any pair (α, γ). Singularities can never be eliminated in any 3-dimensional parameterization of $SO(3)$ for a similar reason that there is no global coordinate chart on a sphere.

2.1.2 Equivalent axis

Euler's theorem says that each rotation $R \in SO(3)$ is a rotation about some axis, see [19]. This axis is called the axis of rotation, and vectors parallel to this axis are unaffected by the rotation R. Without loss of generality, let the rotation axis be specified by a unit vector a and let the angle of rotation, θ, be in the range $[0, 2\pi)$. Then the scaled vector $\theta a \in B(0, 2\pi) \subset \mathbf{R}^3$, where $B(0, 2\pi)$ is the ball of radius 2π centered at the origin. The vector θa is called the equivalent axis, or exponential coordinates, of the rotation. For convenience, we will also call the pair (θ, a) exponential coordinates.

Let $x(t)$ be a point moving with unit angular velocity about a unit vector a. The velocity $\dot{x}(t)$ is perpendicular to both a and $x(t)$ and can be expressed by the vector product

$$\dot{x}(t) = a \times x(t).$$

If we write the vector product $a \times x(t)$ as a matrix multiplication, $\hat{a}x(t)$, where $\hat{a}$ is obtained by the isomorphism

$$a = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \longleftrightarrow \hat{a} = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix}, \tag{1}$$

we get a first order linear differential equation whose solution is $x(t) = e^{\hat{a}t}x(0)$. Thus, a rotation of an angle θ about a is given by

$$R(a, \theta) = e^{\hat{a}\theta}. \tag{2}$$

The matrix exponential can easily be calculated using Rodrigues' formula, see [25, 41]:

$$e^{\hat{a}\theta} = I + \hat{a}\sin\theta + \hat{a}^2(1 - \cos\theta).$$

That the mapping is many-to-one is easily seen. Even if the angle θ is restricted to the interval $[0, \pi]$, the mapping is singular. It is one-to-one except for $\theta = \pi$ where a and $-a$ give the same rotation. As a consequence, and because $\bar{B}(0, \pi)$ is not homeomorphic to $SO(3)$, the inverse is not continuous, see Appendix B.

The exponential map (2) is a mapping from the Lie algebra $so(3)$ onto the Lie group $SO(3)$. The Lie algebra $so(3)$ is the tangent space at the identity and consists of all skew-symmetric 3×3 matrices $\vartheta\hat{a}$. (Here ϑ can be identified with θ. The difference is that we have restricted θ to $[0, 2\pi)$ whereas ϑ can be any real number.) By the isomorphism (1), $so(3)$ can be identified with the Euclidean space $\mathbf{R}^3$. We will use both representations of $so(3)$ and switch back and forth by (1). The Lie algebra product – or the Lie bracket – $so(3) \times so(3) \to so(3)$ is defined by

$$[\hat{\boldsymbol{v}}, \hat{\boldsymbol{w}}] = \hat{\boldsymbol{v}}\hat{\boldsymbol{w}} - \hat{\boldsymbol{w}}\hat{\boldsymbol{v}} = (\boldsymbol{v} \times \boldsymbol{w})\hat{}$$

or

$$[\boldsymbol{v}, \boldsymbol{w}] = \boldsymbol{v} \times \boldsymbol{w}.$$

2.1.3 Quaternions

A common way of representing rotations is by using quaternions. Just as unit complex numbers can be used to represent a rotation in the plane, unit quaternions can be used to represent rotations in space. Quaternions generalize complex numbers into four dimensions and, unlike Euler angles and equivalent axis, give a global parameterization of $SO(3)$.

Quaternions are numbers of the form

$$q = x_1\boldsymbol{i} + x_2\boldsymbol{j} + x_3\boldsymbol{k} + x_4$$

where the coefficients x_1, x_2, x_3 and x_4 are real, and $\boldsymbol{i}, \boldsymbol{j}$ and $\boldsymbol{k}$ are the quaternionic units defined by

$$\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1.$$

Note that for example $\boldsymbol{ij} = -\boldsymbol{ji} = \boldsymbol{k}$, e.g. multiplication is not commutative. It is convenient to treat q as having a scalar part $w = x_4$ and a vector part $\boldsymbol{v} = (x_1, x_2, x_3)$ and use the notation

$$q = [(x_1, x_2, x_3), x_4] = [\boldsymbol{v}, w].$$

The set of quaternions, denoted by H, forms a group with respect to quaternion multiplication. With the above notation we can write the product as

$$[\boldsymbol{v}_1, w_1][\boldsymbol{v}_2, w_2] = [w_1\boldsymbol{v}_2 + w_2\boldsymbol{v}_1 + \boldsymbol{v}_1 \times \boldsymbol{v}_2, w_1w_2 - \boldsymbol{v}_1 \cdot \boldsymbol{v}_2].$$

where $\cdot$ denotes the scalar product on $\mathbf{R}^3$. The unit element is the quaternion 1 and the inverse is given by

$$q^{-1} = q^*/|q|^2,$$

where the conjugate q^* and the modulus $|q|$ are defined by

$$\begin{aligned} q^* &= [\boldsymbol{v}, w]^* = [-\boldsymbol{v}, w] \\ |q|^2 &= q^*q = x_1^2 + x_2^2 + x_3^2 + x_4^2. \end{aligned}$$

In what follows we will sometimes identify a real number w with the quaternion $[0, w]$, and a vector $\boldsymbol{v} \in \mathbf{R}^3$ with the quaternion $[\boldsymbol{v}, 0]$. The converse relations will also be used. By this convention we can multiply vectors in $\mathbf{R}^3$ with quaternions and scalars with quaternions. The product with scalars coincide with the usual product for vectors, i.e. $w_1q = [0, w_1]q = [w_1\boldsymbol{v}, w_1w]$. Together with the standard addition

$$q_1 + q_2 = [\boldsymbol{v}_1 + \boldsymbol{v}_2, w_1 + w_2],$$

H is a real vector space.

The group manifold of $SO(3)$ is homeomorphic to the real projective space $\mathbf{PR}^3$, see [50]. The space $\mathbf{PR}^3$ can be regarded as the space of all lines in $\mathbf{R}^4$ passing through the origin, or as the unit sphere $S^3 \subset \mathbf{R}^4$ with antipodal points identified. Hence we can create a mapping from S^3 to $SO(3)$ whose pre-image of each element consists of two elements of S^3. The following map,

$$(x_1, x_2, x_3, x_4)^T \mapsto \begin{pmatrix} 1-2(x_2^2+x_3^2) & 2(x_1x_2 - x_3x_4) & 2(x_1x_3+x_2x_4) \\ 2(x_1x_2+x_3x_4) & 1-2(x_1^2+x_3^2) & 2(x_2x_3-x_1x_4) \\ 2(x_1x_3-x_2x_4) & 2(x_2x_3+x_1x_4) & 1-2(x_1^2+x_2^2) \end{pmatrix}, \tag{3}$$

is a two-to-one map from S^3 to $SO(3)$. See Appendix B for further details.

Under the mapping (3), the point

$$(a_1 \sin(\theta/2), a_2 \sin(\theta/2), a_3 \sin(\theta/2), \cos(\theta/2))^T$$

on S^3 representing the rotation (θ, a) is mapped onto the same rotation matrix as if (θ, a) was mapped to $SO(3)$ by Rodrigues' formula.

The set of unit quaternions is a subgroup of H. From now on we identify the unit quaternions with S^3 in the natural way.

$$[(x_1, x_2, x_3), x_4] \longleftrightarrow (x_1, x_2, x_3, x_4)^T.$$

With this identification, S^3 is a Lie group under quaternion multiplication. The Lie algebra of S^3 is the tangent space at the identity element $[0, 1]$. Hence, the Lie algebra is the set of quaternions of the form

$$[v, 0].$$

Here v is an arbitrary vector in $\mathbf{R}^3$.

The unit quaternion

$$q(\theta, a) = [a \sin(\theta/2), \cos(\theta/2)] \tag{4}$$

is a representation of the rotation (θ, a). Notice that the quaternions q and $-q$ corresponds to the same rotation. This is natural since rotating by an angle θ about a is equivalent to rotating by $2\pi - \theta$ about $-a$; the latter rotation corresponding to $-q$.

For a vector $\boldsymbol{v} \in \mathbf{R}^3$, we can calculate the rotated vector $\boldsymbol{v}' = R(\theta, a)\boldsymbol{v}$ by the formula

$$\boldsymbol{v}' = q\boldsymbol{v}q^*,$$

where $q = q(\theta, a)$ as in (4). The expression may need some explanation. The vector $\boldsymbol{v}$ on the right hand side is interpreted as a quaternion $[\boldsymbol{v}, 0]$. A simple calculation shows that the quaternion product $q\boldsymbol{v}q^*$ has zero scalar part and can therefore be interpreted as the vector $\boldsymbol{v}'$.

The mapping (3) from S^3 to $SO(3)$ is C^∞ and preserves the group structure, i.e. it is a *Lie group homomorphism.* The property of preserving the group structure is important in order to compose rotations. Clearly, the identity is mapped to the identity. Moreover, let q_1 and q_2 be two quaternions mapped onto the matrices R_1 and R_2, respectively. Then the image of the composed rotation q_1q_2 coincides with R_1R_2. Since (3) is two-to-one, it is called a double covering.

The major advantages of using unit quaternions for representing rotations are the existence of a homomorphism onto $SO(3)$, the compact way of storage, and that there is an easy way of interpolating between quaternions. The latter is important especially when animating motions, see [51]. As an example of interesting techniques, see [14] where they use Bezier curves on S^3 in order to interpolate between a sequence of rotations. The resulting motion is smooth and looks natural to the observer, and would have been difficult to achieve using other representations such as Euler angles, equivalent axis, or matrix representation.

In this paper we are primarily interested in simple interpolation between two rotations represented by two unit quaternions q_1 and q_2. For $s \in [0, 1]$, the shortest great arc on S^3 between q_1 and q_2 is the curve

$$s \mapsto q_1(q_1^* q_2)^s. \tag{5}$$

Exponentiation of a quaternion $q = [a \sin(\theta/2), \cos(\theta/2)]$ is defined by the exponential function and the logarithm function;

$$\begin{aligned} q &= e^{[a\,\theta/2, 0]}, \\ \log(q) &= [a\,\theta/2, 0], \\ q^s &= e^{s \log q} = [a \sin(s\theta/2), \cos(s\theta/2)], \end{aligned}$$

see [14]. The Lie algebra of S^3 consists of quaternions of the form $[a\,\vartheta/2, 0]$. The exponential function above maps Lie algebra elements onto S^3. Comparing with exponential coordinates, we see that the curve $s \mapsto q^s$ on S^3 corresponds to the curve

$$s \mapsto e^{s\theta\hat{a}}$$

in $SO(3)$, and we immediately see that the curve has constant velocity. In [14], it is shown that the great arc (5) between q_1 and q_2 also can be calculated by

$$s \mapsto \frac{q_1 \sin((1-s)\alpha) + q_2 \sin(s\alpha)}{\sin \alpha},$$

where α is the angle between q_1 and q_2 defined by $\cos\alpha = q_1 \cdot q_2$. (Here q_1 and q_2 are interpreted as vectors in $\mathbf{R}^4$.) The expression is not defined for $q_1 = \pm q_2$, but then q_1 and q_2 correspond to the same rotation and no interpolation is needed.

2.2 The special Euclidean group $SE(3)$

A rigid body transformation (R, t) acts on a vector $x \in \mathbf{R}^n$ as

$$(R, t)x = Rx + t. \tag{6}$$

The set of pairs (R, t) has the structure of a group which is denoted $SE(n)$ - the special Euclidean group, or the group of proper rigid body transformations in $\mathbf{R}^n$. This group is the semi-direct product of the special orthogonal group with $\mathbf{R}^n$,

$$SE(n) = SO(n) \rtimes \mathbf{R}^n.$$

The effect of two successive transformations (R_1, t_1) and (R_2, t_2) is given by the group operation

$$(R_2, t_2)(R_1, t_1) = (R_2R_1, R_2t_1 + t_2). \tag{7}$$

It is common to identify a vector x in $\mathbf{R}^n$ by the vector $\tilde{x}^T = (x^T, 1)$ in $\mathbf{R}^{n+1}$ and use the representation

$$(R, t) \mapsto \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$

of $SE(n)$. Then we have

$$\begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} Rx + t \\ 1 \end{pmatrix}$$

and the product

$$\begin{pmatrix} R_2 & t_2 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_1 & t_1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R_2R_1 & R_2t_1 + t_2 \\ 0 & 1 \end{pmatrix} \tag{8}$$

which exactly match the action (6) and the group operation (7). In the case of $n = 3$ this representation is called the homogeneous representation and is often used in robotics and computer graphics, see [22, 44]. In the homogeneous representation, the identity element in $SE(3)$ is the identity matrix and the inverse of (R, t) is

$$\begin{pmatrix} R^T & -R^T t \\ 0 & 1 \end{pmatrix}.$$

The structure of $SE(n)$ is non-Euclidean, but locally $SE(n)$ is a smooth manifold diffeomorphic to $\mathbf{R}^{n(n+1)/2}$, see [37]. In fact, $SE(n)$ is a Lie group, see [50]. Elements of the Lie algebra $se(3)$ are velocities of rigid bodies and, in the homogeneous representation, are of the form

$$\begin{pmatrix} \vartheta\hat{a} & \boldsymbol{v} \\ 0 & 0 \end{pmatrix}$$

where $\vartheta\hat{a} \in so(3)$ for a unit vector a, and $\boldsymbol{v} \in \mathbf{R}^3$. For $\vartheta \neq 0$, the above matrix can be written as ϑM, where

$$M = \begin{pmatrix} \hat{a} & \boldsymbol{v}/\vartheta \\ 0 & 0 \end{pmatrix}.$$

Since $M^4 = -M^2$, the exponential map from $se(3)$ onto $SE(3)$ can be written as

$$\begin{aligned} e^{\vartheta M} &= I + \vartheta M + (1 - \cos\vartheta)M^2 + (\vartheta - \sin\vartheta)M^3 \\ &= \begin{pmatrix} e^{\vartheta\hat{a}} & \boldsymbol{v} + (1-\cos\vartheta)\hat{a}\boldsymbol{v}/\vartheta + (\vartheta - \sin\vartheta)\hat{a}^2\boldsymbol{v}/\vartheta \\ 0 & 1 \end{pmatrix}. \end{aligned} \tag{9}$$

For pure translations $\vartheta = 0$ and the exponential map from $se(3)$ onto $SE(3)$ gives the matrix

$$\begin{pmatrix} I & \boldsymbol{v} \\ 0 & 1 \end{pmatrix}.$$

2.3 $\mathcal{M} = S^3 \times \mathbf{R}^3$ representing SE(3)

Because of the advantages of representing rotations by quaternions, we use quaternions also when representing rigid transformations in $SE(3)$. We know that there is a double covering from S^3 onto $SO(3)$, so by introducing the appropriate product on $\mathcal{M} = S^3 \times \mathbf{R}^3$, the manifold $\mathcal{M}$ will be a Lie Group. Then we can find a double covering from $\mathcal{M}$ onto $SE(3) = SO(3) \rtimes \mathbf{R}^3$.

Let (q_1, t_1) and (q_2, t_2), where $q_i \in S^3$ and $t_i \in \mathbf{R}^3$ for $i = 1, 2$, be points in $\mathcal{M}$ and let the group operation be the product $\mathcal{M} \times \mathcal{M} \to \mathcal{M}$ defined by

$$(q_2, t_2)(q_1, t_1) = (q_2 q_1, q_2 t_1 q_2^* + t_2). \tag{10}$$

This product is the effect of the two successive transformations, (q_1, t_1) followed by (q_2, t_2), and corresponds exactly to the product in $SE(3)$ defined by (7). The identity is $(1, 0)$ (here 1 is interpreted as the quaternion $[(0,0,0),1]$) and the inverse of (q, t) is $(q^*, -q^* t q)$. The latter follows since

$$\begin{aligned} (q,t)(q^*, -q^*tq) &= (qq^*, q(-q^*tq)q^* + t) = (1,0) \\ (q^*, -q^*tq)(q,t) &= (q^*q, q^*tq - q^*tq) = (1,0). \end{aligned}$$

The mapping $h : \mathcal{M} \to SE(3)$ defined by

$$h : (q,t) \mapsto \begin{pmatrix} R(q) & t \\ 0 & 1 \end{pmatrix}, \tag{11}$$

where $R(q)$ is obtained by (3), is a two-to-one Lie group homomorphism. The group structure is preserved because the identity is mapped to the identity and the image of the product $(q_2, t_2)(q_1, t_1)$ is

$$\begin{pmatrix} R(q_2)R(q_1) & R(q_2)t_1 + t_2 \\ 0 & 1 \end{pmatrix},$$

which, by (8), is easily seen to be equivalent to the product of the images of (q_2, t_2) and (q_1, t_1).

The Lie algebra $\mathfrak{m}$ corresponding to the Lie group $\mathcal{M}$ contains elements of the form

$$([a\vartheta/2, 0], \boldsymbol{v}),$$

were $\boldsymbol{v}$ is an arbitrary vector in $\mathbf{R}^3$. Because of the homomorphism, the exponential map from $\mathfrak{m}$ onto $\mathcal{M}$ matches the exponential map from $se(3)$ onto $SE(3)$ given by (9). Hence, letting $t = \boldsymbol{v} + (1 - \cos\vartheta)\hat{a}\boldsymbol{v}/\vartheta + (\vartheta - \sin\vartheta)\hat{a}^2\boldsymbol{v}/\vartheta$, the exponential map $\mathfrak{m} \mapsto \mathcal{M}$ is

$$e^{([a\vartheta/2,0],\boldsymbol{v})} = \left(e^{[a\vartheta/2,0]}, t\right)$$

The exponential map takes the Lie algebra element $\mu = ([a\theta/2, 0], \boldsymbol{v}) \in \mathfrak{m}$ to the element $(q, t) \in \mathcal{M}$, where $q = [a\sin(\theta/2), \cos(\theta/2)]$. By the double covering (3), the quaternion $\pm q$ corresponds to the rotation matrix $e^{\hat{a}\theta} \in SO(3)$. Hence, $(\pm q, t)$ represents the rigid body transformation

$$\begin{pmatrix} e^{\theta\hat{a}} & t \\ 0 & 1 \end{pmatrix}$$

in $SE(3)$. Taking the logarithm of this matrix, i.e. inverting the exponential mapping (9), we get the Lie algebra element

$$\begin{pmatrix} \theta\hat{a} & \boldsymbol{v} \\ 0 & 0 \end{pmatrix}$$

in $se(3)$. This element can be identified with the Lie algebra element μ that we started with. In fact, the homomorphism h in (11) induces a Lie algebra homomorphism $\mathrm{d}h : \mathfrak{m} \to se(3)$ defined by

$$\mathrm{d}h : ([a\vartheta/2, 0], \boldsymbol{v}) \mapsto \begin{pmatrix} \vartheta\hat{a} & \boldsymbol{v} \\ 0 & 0 \end{pmatrix}. \tag{12}$$

Thus, we have confirmed that

$$h(e^{\mu}) = e^{\mathrm{d}h(\mu)}$$

for any $\mu \in \mathfrak{m}$, which is a useful property of Lie group homomorphisms.

A Lie algebra element can easily be translated to the tangent space at an arbitrary point in the Lie group. In the case of a Lie algebra element $([a\vartheta/2, 0], \boldsymbol{v}) \in \mathfrak{m}$, it can be translated to the tangent space at the point $(q, t) \in \mathcal{M}$ by the product $(q, t)([a\vartheta/2, 0], \boldsymbol{v})$. This property will be used at the end of next section.

Interpolation in the product space $\mathcal{M} = S^3 \times \mathbf{R}^3$ is simply achieved by interpolating in the two spaces separately. Interpolation along great arcs on S^3 is given in (5). Thus, a path on $\mathcal{M}$ from (q_1, t_1) to (q_2, t_2) of constant velocity is given by

$$s \mapsto (q_1(q_1^* q_2)^s, (1-s)t_1 + st_2), \quad s \in [0, 1].$$

Notice that a consequence of the double covering of $SE(3)$ is that a path on $\mathcal{M}$ from the identity $([0, 1], 0)$ to $([0, -1], 0)$ is mapped onto a closed loop in $SE(3)$ starting and ending at the identity. This may seem like a drawback, but as soon as one is aware of it, the double covering will cause no problem at all.

The ideal situation when manipulating rigid body transformations would be to use $SE(3)$ directly. However, seen as a subset of $\mathbf{R}^7$, the manifold $\mathcal{M}$ is in many aspects easier to represent and work with. One particular advantage of $\mathcal{M}$ will be apparent in the next section, where we are able to derive the Green kernel on the manifold. Using that result, we will also, thanks to the double covering, be able to determine the Green kernel on $SE(3)$. After that, the Green kernel on $SE(3)$ is used to create a potential field in the configuration space of a free-flying rigid body robot. The potential field will then be an essential component in a path planning algorithm.

3 Green kernels

In this section we introduce the Green kernel on a Riemannian manifold and state some properties of harmonic functions. The main result is a derivation of the Green kernels on the manifolds $\mathcal{M} = S^3 \times \mathbf{R}^3$ and $SE(3)$ respectively. The group $SE(3)$ and its Green kernel will be important in the rest of this paper and will be explicitly used in Section 5.

3.1 Harmonic functions

Poisson's equation

$$-\Delta u = f \tag{13}$$

models many physical phenomena, for example stationary temperature, electrostatic potential, and chemical concentration in equilibrium. The equation (13) is understood either point-wise or in the sense of distributions. On $\mathbf{R}^d$ the Laplacian Δ in Cartesian coordinates is given by $\Delta = \frac{\partial^2}{\partial x_1^2} + \cdots + \frac{\partial^2}{\partial x_d^2}$, but the operator can be generalized to an arbitrary Riemannian manifold $\mathcal{R}$, see [47]. The function $f : \mathcal{R} \to \mathbf{R}$ to the right in (13) is called a source term. In what follows we will sometimes use the analogy of stationary temperature distribution. Then f is the heat source distribution on $\mathcal{R}$ and the solution u is the stationary temperature.

On open subsets U of $\mathcal{R}$ where f vanishes, u satisfies Laplace's equation

$$-\Delta u = 0, \tag{14}$$

and u is said to be harmonic on U. Harmonic functions have a large number of nice properties, see [2, 15]. The properties we want to emphasize here and use later on are the following:

1. u is smooth, in fact $u \in C^\infty(U)$.

2. If u is constant in any open subset of U, then u is constant in all of U.

3. u satisfies the mean value formulas:
$$u(x) = \frac{1}{|\partial B(x,r)|} \int_{\partial B(x,r)} u \, \mathrm{d}S = \frac{1}{|B(x,r)|} \int_{B(x,r)} u \, \mathrm{d}y$$
for each ball $B(x,r) \subset U$. Here $\mathrm{d}y$ is the Lebesgue measure on $B(x,r)$ and $\mathrm{d}S$ is the surface measure on $\partial B(x,r)$. By $|B(x,r)|$ and $|\partial B(x,r)|$ we mean the Lebesgue measure of $B(x,r)$ and the surface measure of $\partial B(x,r)$, respectively.

4. u has no local maxima or minima in U.

Moreover, the Laplacian commutes with orthogonal transformations, i.e.
$$\Delta(u \circ T) = (\Delta u) \circ T,$$
for any orthogonal transformation T. In particular the property that there are no local minima makes harmonic functions suitable for path planning, as we shall see later in Section 4.2.

A consequence of property 4 above is that if u is not constant, then there is a descent direction from each point in U. The negative gradient of u points in the *steepest descent* direction and induces a tangent vector field $V = -\mathrm{grad}\ u$ in U. Since $\mathcal{R}$ is smooth and $u \in C^\infty(U)$, the vector field V varies smoothly over U. The only points where grad u vanishes are saddle points. At a saddle point x, the Hessian matrix $Hu(x)$ is indefinite so the smallest eigenvalue λ is negative. We define the steepest descent direction at such a point x to be along any eigenvector of $Hu(x)$ corresponding to λ.

A *field line* is a curve which has at each of its points the direction of the field at that point. Where the field vanishes, we adopt the convention that the field lines continue in a steepest descent direction. The field lines of V are smooth curves except at saddle points where the tangent is discontinuous. By the definition of V follows that u is strictly decreasing along field lines, and by the convention follows that field lines cannot terminate in the interior of U (c.f [31] p. 41). We summarize in the following lemma.

Lemma 1. *Let u be harmonic in an open connected set $U \subset \mathcal{R}$. Then either u is constant, or through each point in U there is a field line of $V = -grad\ u$ terminating at ∂U or at infinity.*

3.2 Fundamental solutions and Green kernels

A function $\widetilde{K} : \mathcal{R} \times \mathcal{R} \to \mathbf{R}$ satisfying

$$-\Delta \widetilde{K}(\cdot, x) = \delta_x, \tag{15}$$

where δ_x is the Dirac measure at x, is called a *fundamental solution* to the Laplacian. The *smallest positive* fundamental solution, K, is unique and is called the Green kernel, see [20]. The solution to Poisson's equation (13) is obtained by the inverse Laplacian operator,

$$u(x) = -\Delta^{-1} f(x) = \int_{\mathcal{R}} K(x, y) f(y) \mathrm{d}y, \tag{16}$$

i.e. integration against the kernel, see [20].

From equation (15) we see that the fundamental solution $\widetilde{K}(\cdot, x)$ is harmonic except at the point x, hence so is K. From the definition of Green kernel follows that $K(x, x') > 0$ and that $\inf_x K(x, x') = 0$. The following well known properties of the Green kernel K will also be used in this paper.

1. $K(x, x')$ is singular along the diagonal in $\mathcal{R} \times \mathcal{R}$.
2. $K(x, x') = K(x', x)$ for $x \neq x'$.

In some cases the Green kernel can be expressed in a simple form. In $\mathbf{R}^d$, $d \geq 3$, for example, the kernel is

$$K(x, x') = \frac{1}{(d-2)|\partial B(0,1)|\, |x - x'|^{d-2}} ,$$

where $B(0,1)$ is the unit ball in $\mathbf{R}^d$. In $\mathbf{R}^2$ there exist no Green kernel. In fact, any positive harmonic function in $\mathbf{R}^2 \setminus \{x\}$ is constant, see [2]. There is, however, a fundamental solution to the Laplacian in $\mathbf{R}^2$,

$$\widetilde{K}(x, x') = \frac{-1}{2\pi} \log |x - x'|.$$

Associated with Laplace's equation is the heat equation

$$\frac{\partial}{\partial t} u - \Delta u = 0, \; t > 0. \tag{17}$$

A function $\widetilde{H}(t,\cdot,x) : (0,\infty)\times\mathcal{R}\times\mathcal{R}\to\mathbf{R}$ satisfying (17) and for which

$$\lim_{t\to 0^+}\widetilde{H}(t,\cdot,x)=\delta_x \tag{18}$$

is called a fundamental solution to the heat equation. In the same way as for the Laplacian, the smallest positive fundamental solution, H, is unique and is called the *heat kernel*, see [4]. The heat kernel is smooth in (t,x,x') and is symmetric on $\mathcal{R}\times\mathcal{R}$, that is $H(t,x,x')=H(t,x',x)$ for all $t>0$. Moreover, if F and G are two heat kernels on two spaces X and Y respectively, then the product FG is the heat kernel on the product space $X\times Y$, see [4]. Indeed, a simple calculation yields

$$\begin{aligned}\frac{\partial}{\partial t}FG-\Delta_{X\times Y}FG &= F_tG+G_tF-G\Delta_XF-F\Delta_YG\\ &= F(G_t-\Delta_YG)+G(F_t-\Delta_XF)\\ &= 0,\ \text{ for } t>0\end{aligned}$$

and

$$\lim_{t\to 0^+}F(t,\cdot,x)G(t,\cdot,y)=\delta_x\delta_y=\delta_{(x,y)},$$

that is, (17) and (18) are satisfied.

It is generally not as easy as a multiplication to obtain the Green kernel on a product space. However, if we know the heat kernels on the two spaces we can multiply them and compute the Green kernel on the product space by the following observation. Above we defined the Green kernel to be the the smallest positive fundamental solution to the Laplacian, but the Green kernel can equivalently be defined using the heat kernel H,

$$K(x,x')=\int_0^\infty H(t,x,x')\mathrm{d}t, \tag{19}$$

whenever K is finite, see [20]. The Green kernel $K(x,x')$ is known to be finite for $x\neq x'$, so when the integral diverges there is no Green kernel. One can show that a geodesically complete non-compact Riemannian manifold $\mathcal{R}$ has a Green kernel if and only if

$$\int^\infty\frac{\mathrm{d}r}{S(r)}\neq\infty,$$

where $S(r)$ is the boundary area of the geodesic sphere $\partial B(0,r)$ in $\mathcal{R}$, see [20]. By $\int^\infty f(r)\mathrm{d}r$ we mean the limit at infinity of the primitive function of f. This explains why there is a Green kernel on $\mathbf{R}^3$ but not in $\mathbf{R}^2$. In general, if the global dimension of the manifold is at least three, then we can integrate the heat kernel in time from 0 to infinity to obtain the Green kernel, see [4, 18].

The result (19) can easily be seen if we use another notation. We can see (17) as a first order differential equation in t whose solution is the operator $e^{t\Delta}$. Integrating in time from 0 to ∞ yields $-\Delta^{-1}$. The operator Δ^{-1} is the inverse of the Laplacian, which can be represented by the Green kernel as in (16). The integral is convergent because the Laplacian is negative definite, so $e^{t\Delta}$ vanishes as t goes to infinity.

3.3 The Green kernel on $\mathcal{M} = S^3 \times \mathbf{R}^3$

So far we have seen explicit formulas for Green kernels and heat kernels on $\mathbf{R}^d$. In other cases the calculations become more complex and in general we cannot find explicit expressions. In Section 2 we used the Riemannian manifold $\mathcal{M} = S^3 \times \mathbf{R}^3$ to represent rigid body transformations. Our aim in this section is to derive an expression for the Green kernel on $\mathcal{M}$.

We take $\mathcal{M}$ with its topology as a subspace of $\mathbf{R}^7$ and with the Riemannian metric induced by the standard inner product in $\mathbf{R}^7$. The metric on $\mathcal{M}$ is then given by

$$d_{\mathcal{M}}(z, z')^2 = d_{S^3}(q, q')^2 + |y - y'|^2,$$

where $z = (q, y) \in S^3 \times \mathbf{R}^3$ and d_{S^3} is the geodesic distance on S^3.

On S^3 we introduce spherical coordinates (η, ϕ, ψ), so on $\mathcal{M}$ we have the cylindrical coordinates according to

$$\begin{aligned} z_1 &= x_1 = \sin\eta \sin\phi \sin\psi \\ z_2 &= x_2 = \sin\eta \sin\phi \cos\psi \\ z_3 &= x_3 = \sin\eta \cos\phi \\ z_4 &= x_4 = \cos\eta \\ z_5 &= y_1 \\ z_6 &= y_2 \\ z_7 &= y_3 \end{aligned} \tag{20}$$

for $z = (q, y) \in S^3 \times \mathbf{R}^3$, and $0 \le \eta \le \pi,\ \ 0 \le \phi \le \pi,\ \ 0 \le \psi < 2\pi$. Let the origin, denoted by 0, be the point on $\mathcal{M}$ for which $\eta = |y| = 0$, i.e. the north pole on S^3 and the origin of $\mathbf{R}^3$. The distance from a point $z = (q, y)$ on $\mathcal{M}$ to the origin is then given by

$$d_{\mathcal{M}}(z, 0)^2 = d_{S^3}(q, 0)^2 + |y|^2 = \eta^2 + r^2$$

where $r = |y|$.

The idea now is to use the heat kernels on S^3 and $\mathbf{R}^3$ respectively, multiply them to get the heat kernel on $\mathcal{M}$, and then integrate in time to get the Green kernel on $\mathcal{M}$. To simplify the notation we will derive the Green kernel on $\mathcal{M}$ with a singularity only at the origin, that is we will find the Green kernel $K : \mathcal{M} \to \mathbf{R}$ that solves

$$-\Delta K = \delta.$$

In $\mathbf{R}^n$, the heat kernel is

$$H(t, y, y') = \frac{1}{(4\pi t)^{n/2}} e^{-\frac{|y-y'|^2}{4t}}, \ t > 0.$$

Since the kernel depends only on $|y - y'|$ we can express the heat kernel in $\mathbf{R}^3$ with singularity as $t \to 0$ at the origin as a function of $r = |y|$;

$$H_{\mathbf{R}^3}(t, r) = \frac{1}{(4\pi t)^{3/2}} e^{-\frac{r^2}{4t}}, \ t > 0.$$

Similarly we define the heat kernel on S^3. The kernel depends only on the distance η to the north pole on S^3 and will be denoted by $H_{S^3}(t, \eta)$. Accordingly, the heat kernel $H_{S^3}(t, \eta) H_{\mathbf{R}^3}(t, r)$ on $\mathcal{M}$, and thereby also the Green kernel on $\mathcal{M}$, depends only on η and r. The Green kernel on $\mathcal{M}$ with a singularity at the origin will be denoted by $K(\eta, r)$ and is obtained by the integral

$$K(\eta, r) = \int_0^\infty H_{S^3}(t, \eta) H_{\mathbf{R}^3}(t, r) \mathrm{d}t,$$

(c.f. (19)).

3.3.1 The heat kernel on S^3

In order to find the Green kernel on $\mathcal{M}$, we first need to find the heat kernel on the sphere S^{n-1} in $\mathbf{R}^n$ with a singularity at the north pole. In [53] we get a solution to

$$\frac{\partial}{\partial t} u = -\nu^2 u \ \text{ on } S^{n-1},$$

where $-\nu^2 = \Delta_{S^{n-1}} - \frac{(n-2)^2}{4}$. The kernel is

$$e^{-t\nu^2} = \Big(\frac{-1}{2\pi \sin\eta}\frac{\partial}{\partial\eta}\Big)^{\frac{n-2}{2}} \vartheta(\eta, t),$$

with

$$\vartheta(\eta, t) = \frac{1}{(4\pi t)^{1/2}} \sum_{k=-\infty}^{\infty} e^{\frac{-1}{4t}(2\pi k+\eta)^2}.$$

We want to find the heat kernel $H_{S^3}(t,\eta)$ on S^3, i.e., $n = 4$, so

$$\begin{aligned} H_{S^3}(t,\eta) &= e^{t\Delta_{S^3}} \\ &= e^{t(1-\nu^2)} \\ &= e^t \frac{-1}{2\pi \sin\eta} \quad \frac{\partial}{\partial\eta}\vartheta(\eta, t). \end{aligned}$$

In order to simplify later calculations we rewrite the sum in the expression for $\vartheta(\eta,t)$. Let

$$\varphi(k) = \frac{1}{2\pi} e^{-(tk^2+ik\eta)}.$$

Then, for $t > 0$, φ is a Schwartz function on $\mathbf{R}$, and its Fourier transform $\hat\varphi$ is easily seen to be

$$\hat\varphi(k) = \frac{1}{(4\pi t)^{1/2}} e^{\frac{-1}{4t}(k+\eta)^2}.$$

Using the Poisson summation formula we get

$$\begin{aligned} \sum_k \varphi(k) &= \sum_k \hat\varphi(2\pi k) \\ &= \sum_k \frac{1}{(4\pi t)^{1/2}} e^{\frac{-1}{4t}(2\pi k+\eta)^2} \\ &= \vartheta(\eta, t). \end{aligned}$$

Hence, we rewrite the heat kernel on S^3 as

$$\begin{aligned} H_{S^3}(t,\eta) &= e^t \frac{-1}{2\pi \sin\eta} \frac{\partial}{\partial\eta} \sum_k \frac{1}{2\pi} e^{-(tk^2+ik\eta)} \\ &= e^t \frac{i}{4\pi^2 \sin\eta} \sum_{k\neq 0} k e^{-(tk^2+ik\eta)} \\ &= \frac{i}{4\pi^2 \sin\eta} \sum_{k\neq 0} k e^{t-(tk^2+ik\eta)} \ . \end{aligned} \tag{21}$$

3.3.2 The Green kernel on $\mathcal{M}$

Now we are ready to apply (19) in order to calculate the Green kernel $K(\eta, r)$ on the product space $\mathcal{M} = S^3 \times \mathbf{R}^3$. That is, we multiply the heat kernels $H_{\mathbf{R}^3}(t,r)$ and $H_{S^3}(t,\eta)$ and integrate in time from 0 to ∞:

$$\begin{aligned} K(\eta,r) &= \int_0^\infty H_{\mathbf{R}^3}(t,r) H_{S^3}(t,\eta) \mathrm{d}t \\ &= \int_0^\infty \frac{1}{(4\pi t)^{3/2}} e^{-\frac{r^2}{4t}} \frac{i}{4\pi^2 \sin\eta} \sum_{k\neq 0} k e^{t-(tk^2+ik\eta)} \mathrm{d}t \\ &= \frac{i}{32\pi^{7/2} \sin\eta} \int_0^\infty \frac{1}{t^{3/2}} e^{-\frac{r^2}{4t}} \sum_{k\neq 0} k e^{t-(tk^2+ik\eta)} \mathrm{d}t \\ &= \frac{i}{32\pi^{7/2} \sin\eta} \sum_{k\neq 0} \int_0^\infty \frac{k}{t^{3/2}} e^{t-(tk^2+ik\eta+\frac{r^2}{4t})} \mathrm{d}t \\ &= \frac{i}{16\pi^3 r \sin\eta} \sum_{k\neq 0} k e^{-r(k^2-1)^{1/2}-ik\eta} \\ &= \frac{i}{16\pi^3 r \sin\eta} \sum_{k\neq 0} k e^{-r(k^2-1)^{1/2}} (\cos k\eta - i \sin k\eta) \\ &= \frac{i}{16\pi^3 r \sin\eta} \sum_{k=1}^{\infty} k e^{-r(k^2-1)^{1/2}} (-2i \sin k\eta) \\ &= \frac{1}{8\pi^3 r \sin\eta} \sum_{k=1}^{\infty} k e^{-r(k^2-1)^{1/2}} \sin k\eta. \end{aligned} \tag{22}$$

The Green kernel $K(\eta, r)$ is harmonic in $\mathcal{M}$ except at the singularity at the origin where $r = \eta = 0$. A tedious calculation shows that K satisfies

Laplace's equation on $\mathcal{M}$ given in Appendix A. For the moment we will consider $K(\eta, r)$ for $r > 0, 0 < \eta < \pi$. It is obvious that the series in (22) is convergent, but the rate of convergence can be improved dramatically, especially for small r, by the aid of Kummer's transformation: Let $\sum_{k=1}^{\infty} a_k$ and $\sum_{k=1}^{\infty} b_k$ be two convergent series such that

$$\lim_{k\to\infty} \frac{a_k}{b_k} = c \neq 0,$$

then

$$\sum_{k=1}^{\infty} a_k = c \sum_{k=1}^{\infty} b_k + \sum_{k=1}^{\infty} \left(1 - c\,\frac{b_k}{a_k}\right) a_k.$$

If $\sum_{k=1}^{\infty} b_k$ can be expressed in closed form and $1 - c\ b_k/a_k$ tends to zero faster than a_k, then the convergence rate is improved.

In (22), we have the series $\sum_{k=1}^{\infty} a_k$ with

$$a_k = a_k(\eta, r) = \frac{k}{r}\,\frac{e^{-r(k^2-1)^{1/2}} \sin k\eta}{8\pi^3 \sin \eta},\ \ k = 1, 2, \ldots.$$

The series $\sum_{k=1}^{\infty} b_k$, with

$$b_k = b_k(\eta, r) = \left(\frac{k}{r} + \frac{1}{2}\right) \frac{e^{-rk} \sin k\eta}{8\pi^3 \sin \eta},\ \ k = 1, 2, \ldots,$$

is a good approximation to $\sum_{k=1}^{\infty} a_k$, especially for small r. The estimates in the following lemma guarantees that the series $\sum_{k=1}^{\infty} \left(1 - c\ \frac{b_k}{a_k}\right) a_k$ converges much faster than $\sum_{k=1}^{\infty} a_k$.

Lemma 2. *Let* $0 < \eta < \pi$ *and* $0 < r < \infty$, *and define* a_k, b_k *and* c *as above. Then*

$$0 \leq 1 - c\,\frac{b_k}{a_k} \leq 1 - e^{-r/k}.$$

Proof. Let $c_k = k - (k^2 - 1)^{1/2}$. By the observation that $t \leq t^{1/2}$ for $0 \leq t \leq 1$ we see that

$$1 - \frac{1}{k^2} \leq \left(1 - \frac{1}{k^2}\right)^{1/2} \iff k - \frac{1}{k} \leq (k^2 - 1)^{1/2} \iff c_k \leq \frac{1}{k}$$

for all $k \geq 1$. Moreover, by series expansion of $(1 - 1/k^2)^{1/2}$ we get

$$\begin{aligned} c_k &= k - k\left(1 - \frac{1}{2k^2} - \frac{1}{8k^4} - \frac{1}{16k^6} - \cdots + \binom{1/2}{n}\frac{1}{(-k^2)^n} - \cdots\right) \\ &= \frac{1}{2k} + \frac{1}{8k^3} + \frac{1}{16k^5} + \cdots, \end{aligned}$$

and we conclude that

$$\frac{1}{2k} < c_k \leq \frac{1}{k}$$

for all $k \geq 1$. Now we can estimate b_k/a_k:

$$\begin{aligned} \frac{b_k}{a_k} &= \left(1 + \frac{r}{2k}\right) e^{-rc_k} \\ &\geq \left(1 + \frac{r}{2k}\right) e^{-r/k} \\ &\geq e^{-r/k} \end{aligned}$$

and

$$\begin{aligned} \frac{b_k}{a_k} &= \left(1 + \frac{r}{2k}\right) e^{-rc_k} \\ &< \left(1 + \frac{r}{2k}\right) e^{-\frac{r}{2k}} \\ &\leq 1. \end{aligned}$$

The final inequality follows because $(1+t)e^{-t} \leq 1$ for all t. Letting $k \to \infty$ in the estimates of b_k/a_k gives $c = 1$, and the lemma follows. □

The series $\sum_{k=1}^{\infty} b_k$ can be expressed in closed form as follows

$$
\begin{aligned}
\sum_{k=1}^{\infty} b_k &= \frac{1}{8\pi^3 \sin\eta} \sum_{k=1}^{\infty} \left(\frac{k}{r} + \frac{1}{2}\right) e^{-rk} \sin k\eta \\
&= \frac{1}{8\pi^3 \sin\eta} \left(\frac{-1}{r}\frac{\partial}{\partial r} + \frac{1}{2}\right) \sum_{k=1}^{\infty} e^{-rk} \sin k\eta \\
&= \frac{1}{8\pi^3 \sin\eta} \left(\frac{-1}{r}\frac{\partial}{\partial r} + \frac{1}{2}\right) \frac{e^{-r}\sin\eta}{1 - 2e^{-r}\cos\eta + e^{-2r}} \\
&= \frac{1}{16\pi^3} \left(\frac{-1}{r}\frac{\partial}{\partial r} + \frac{1}{2}\right) \frac{1}{\cosh r - \cos\eta} \\
&= \frac{2r^{-1}\sinh r + \cosh r - \cos\eta}{32\pi^3(\cosh r - \cos\eta)^2}. \qquad (23)
\end{aligned}
$$

Thus, using Lemma 2 and (23), the Green kernel in (22) can now be expressed as

$$
\begin{aligned}
K(\eta, r) &= \frac{2r^{-1}\sinh r + \cosh r - \cos\eta}{32\pi^3(\cosh r - \cos\eta)^2} + \sum_{k=1}^{\infty}(a_k(\eta, r) - b_k(\eta, r)) \\
&= \frac{2r^{-1}\sinh r + \cosh r - \cos\eta}{32\pi^3(\cosh r - \cos\eta)^2} + \\
&+ \frac{1}{8\pi^3 r \sin\eta} \sum_{k=1}^{\infty} \left(ke^{rc_k} - k - \frac{r}{2}\right) e^{-rk} \sin k\eta, \qquad (24)
\end{aligned}
$$

where $c_k = k - (k^2 - 1)^{1/2}$ as defined in the proof of Lemma 2.

The kernel (24) is smooth on $\mathcal{M}$ except at the singularity at $r = \eta = 0$. However, there are numerical problems along the curves $(\eta, 0)$ for $0 < \eta \leq \pi$, along $(0, r)$ and (π, r) for $r > 0$. We will derive these limit functions separately. The first part to the right in (24) causes no difficulty, so we focus on the series in the second part.

First we consider the limit function as $r \to 0$ for $0 < \eta < \pi$. The series in (24) is multiplied by $1/r$ and since the limit is finite for all $\eta \in (0, \pi]$, the series must vanish as $r \to 0$. The limit of the series is

$$\begin{aligned}
&\lim_{r\to 0} \frac{1}{8\pi^3 \sin\eta} \sum_{k=1}^{\infty} \frac{k}{r}\left(-\frac{r}{2k} + e^{rc_k} - 1\right) e^{-rk} \sin k\eta \\
=&\lim_{r\to 0} \frac{1}{8\pi^3 \sin\eta} \sum_{k=1}^{\infty} \frac{k}{r}\left(-\frac{r}{2k} + rc_k + O\big((rc_k)^2\big)\right) e^{-rk} \sin k\eta \\
=&\lim_{r\to 0} \frac{1}{8\pi^3 \sin\eta} \sum_{k=1}^{\infty} k\left(-\frac{1}{2k} + c_k + O(r)\right) e^{-rk} \sin k\eta \\
=&\frac{1}{8\pi^3 \sin\eta} \sum_{k=1}^{\infty} \left(kc_k - \frac{1}{2}\right) \sin k\eta. \qquad (25)
\end{aligned}$$

Here we have used the series expansion of e^{rc_k} and that $0 < c_k \leq 1$. Now, as $r^{-1}\sinh r \to 1$ as $r \to 0$, we can express the limit of (24) as

$$K(\eta,0) = \frac{3-\cos\eta}{32\pi^3(1-\cos\eta)^2} + \frac{1}{8\pi^3\sin\eta} \sum_{k=1}^{\infty} \underbrace{\left(kc_k - \frac{1}{2}\right)}_{O(1/k^2)} \sin k\eta. \qquad (26)$$

To see that the rate of convergence is of order $1/k^2$, we use the series expansion of c_k given in the proof of lemma 2,

$$kc_k - \frac{1}{2} = \frac{1}{8k^2} + \frac{1}{16k^4} + \cdots .$$

The limit function on the curve $(0,r)$ for $r > 0$, is obtained by letting $\eta \to 0$ in (24),

$$K(0,r) = \frac{2r^{-1}\sinh r + \cosh r - 1}{32\pi^3(\cosh r - 1)^2} + \frac{1}{8\pi^3 r} \sum_{k=1}^{\infty} k^2 \left(e^{rc_k} - 1 - \frac{r}{2k}\right) e^{-rk}.$$

Similarly, the limit of (24) as $\eta \to \pi$ gives the kernel on the curve (π, r) for $r > 0$;

$$K(\pi,r) = \frac{2r^{-1}\sinh r + \cosh r + 1}{32\pi^3(\cosh r + 1)^2} - \frac{1}{8\pi^3 r} \sum_{k=1}^{\infty} (-1)^k k^2 \left(e^{rc_k} - 1 - \frac{r}{2k}\right) e^{-rk}.$$

Finally, letting $\eta \to \pi$ in (26) gives the solution at the point $\eta = \pi, r = 0$,

$$K(\pi,0) = \frac{1}{32\pi^3} - \frac{1}{8\pi^3} \sum_{k=1}^{\infty} (-1)^k k \left(kc_k - \frac{1}{2}\right).$$

This expression can be rewritten by using Kummer's transformation once again. Notice that

$$\sum_{k=1}^{\infty}(-1)^k\frac{1}{k} = -\ln 2 \tag{27}$$

and that

$$\begin{aligned}K(\pi,0) &= \frac{1}{32\pi^3} - \frac{1}{8\pi^3}\sum_{k=1}^{\infty}(-1)^k k\left(kc_k - \frac{1}{2} - \frac{1}{8k^2} + \frac{1}{8k^2}\right)\\ &= \frac{2+\ln 2}{64\pi^3} - \frac{1}{8\pi^3}\sum_{k=1}^{\infty}(-1)^k \underbrace{k\left(kc_k - \frac{1}{2} - \frac{1}{8k^2}\right)}_{O(1/k^3)}\end{aligned}$$

The order of convergence is easily seen from the series expansion of $kc_k - 1/2$ above.

We have now proved the following theorem.

Theorem 1. *Let $q \in S^3$ with spherical coordinates (η, ϕ, ψ) according to* (20) *and $y \in \mathbf{R}^3$ with $r = |y|$. Then*

$$\begin{aligned}K(\eta,r) &= \frac{2r^{-1}\sinh r + \cosh r - \cos\eta}{32\pi^3(\cosh r - \cos\eta)^2} + \\ &+ \frac{1}{8\pi^3 r\sin\eta}\sum_{k=1}^{\infty} k\left(e^{rc_k} - 1 - \frac{r}{2k}\right)e^{-rk}\sin k\eta, \quad \textit{for } r>0, 0<\eta<\pi,\\ K(\eta,0) &= \frac{3-\cos\eta}{32\pi^3(1-\cos\eta)^2} + \frac{1}{8\pi^3\sin\eta}\sum_{k=1}^{\infty}\left(kc_k - \frac{1}{2}\right)\sin k\eta, \quad \textit{for } 0<\eta<\pi,\\ K(0,r) &= \frac{2r^{-1}\sinh r + \cosh r - 1}{32\pi^3(\cosh r - 1)^2} + \frac{1}{8\pi^3 r}\sum_{k=1}^{\infty}k^2\left(e^{rc_k} - 1 - \frac{r}{2k}\right)e^{-rk},\\ &\qquad \textit{for } r>0,\\ K(\pi,r) &= \frac{2r^{-1}\sinh r + \cosh r + 1}{32\pi^3(\cosh r + 1)^2} - \frac{1}{8\pi^3 r}\sum_{k=1}^{\infty}(-1)^k k^2\left(e^{rc_k} - 1 - \frac{r}{2k}\right)e^{-rk},\\ &\qquad \textit{for } r>0,\\ K(\pi,0) &= \frac{2+\ln 2}{64\pi^3} - \frac{1}{8\pi^3}\sum_{k=1}^{\infty}(-1)^k k\left(kc_k - \frac{1}{2} - \frac{1}{8k^2}\right),\end{aligned}$$

is the Green kernel on $\mathcal{M} = S^3 \times \mathbf{R}^3$ with a singularity at the origin. The coefficients c_k, $k = 1, 2, \dots,$ are given in the proof of Lemma 2.

In the theorem, the Green kernel is expressed in the coordinates (η, r). To simplify the notation later and to be consistent with the notation in Section 2.3, we go back to the notation introduced in Section 3.3. That is, a point on $\mathcal{M}$ is denoted by z and the Green kernel K is

$$K(z) = K(\theta/2, |t|), \tag{28}$$

where $z = (q, t) = ([a \sin(\theta/2), \cos(\theta/2)], t)$.

3.3.3 Negative gradient field on $\mathcal{M}$

This far we have only considered the Green kernel on $\mathcal{M}$ with a singularity at the origin. Since $\mathcal{M}$ is a Lie group it is easy to translate K along the manifold. Let $K^w(z) = K(w^{-1}z)$. Then K^w is the Green kernel with singularity at $w \in \mathcal{M}$.

Let V^w denote the negative gradient field of the Green kernel K^w and let i denote the identity element. Since K^w is smooth except at w, the vector field V^w is a smooth mapping that to each point $z \in \mathcal{M} \setminus \{w\}$ assigns a vector V^w_z in the tangent space at z. In terms of temperature, the Green kernel K^w models the stationary temperature on $\mathcal{M}$ due to a unit heat source at w. The heat flow starts at w and follows the field lines of V^w towards infinity. We define the positive direction of the field lines to be along the heat flow.

At the identity, the tangent vector V^w_i of $V^w$is an element in the Lie algebra $\mathfrak{m}$. To explicitly express the tangent vectors V^w_z for an arbitrary $z \neq w$, it is convenient to first translate z to the origin, find the appropriate element of the Lie algebra, and then translate the Lie algebra element to z.

Let $w = (q, t)$, where $q = [a \sin(\theta/2), \cos(\theta/2)] \in S^3$. On S^3, the kernel K^w only depends on the geodesic distance from the identity to w. Hence, the S^3-components of V^w_i are tangents to great arcs through the identity and q, and are therefore a multiple of $[a, 0]$. Similarly, in $\mathbf{R}^3$ the kernel K^w only depends on the distance from the origin. Hence, the $\mathbf{R}^3$-components of V^w_i are a multiple of t.

Now, let $K^w_1 = \frac{\partial}{\partial \eta} K(\eta, r)|_w$ and $K^w_2 = \frac{\partial}{\partial r} K(\eta, r)|_w$ be the partial derivatives of the Green kernel K evaluated at w (that is $\eta = \theta/2$ and $r = |t|$). Then $K^w_1 < 0$, $K^w_2 < 0$, and the Lie algebra element V^w_i is

$$V^w_i = (q[a, 0]\ K^w_1,\ t/|t|\ K^w_2).$$

By multiplying a Lie algebra element from the left by a group element z, we get a vector in the tangent space at z. Thus, the negative gradient of K^w at z is the tangent vector $V_z^w = zV_i^{z^{-1}w}$, where $V_i^{z^{-1}w}$ is the Lie algebra element owing to a singularity at $z^{-1}w$. The tangent vector V_z^w at z points away from the singularity at w, that is, V_z^w is parallel with the field lines of V^w at z.

3.4 Green kernel on SE(3)

Our aim in this section is to calculate the Green kernel, denoted by G, on the Lie group $SE(3)$. Recall that $\mathcal{M}$ double covers $SE(3)$ by the Lie group homomorphism h given by (11), and that $\mathrm{d}h$ given by (12) is an homomorphism from $\mathfrak{m}$ to $se(3)$. Using the double covering h, we can easily derive the Green kernel on $SE(3)$ from the Green kernel K on $\mathcal{M}$.

The Laplacian on a Lie group depends on the inner product in the tangent space at the identity. Thus, if we choose the inner product on $se(3)$ such that h becomes an isometry, then the inner product is preserved, hence so is the Laplacian. Let

$$l_1 = ([a_1\vartheta_1/2, 0], \boldsymbol{v}_1) \quad \text{and} \quad l_2 = ([a_2\vartheta_2/2, 0], \boldsymbol{v}_2)$$

be two elements in the Lie algebra m. On $\mathfrak{m}$, the inner product $< \, , \, >_{\mathfrak{m}}$ is inherited from $\mathbf{R}^7$.

$$< l_1, l_2 >_{\mathfrak{m}} = a_1 \cdot a_2 \ \vartheta_1\vartheta_2/4 \ + \ \boldsymbol{v}_1 \cdot \boldsymbol{v}_2,$$

where $\cdot$ is the dot product on $\mathbf{R}^3$. The Lie algebra homomorphism $\mathrm{d}h$ maps l_1 and l_2 to the elements

$$\mathrm{d}h(l_1) = \begin{pmatrix} \vartheta_1\hat{a}_1 & \boldsymbol{v}_1 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad \mathrm{d}h(l_2) = \begin{pmatrix} \vartheta_2\hat{a}_2 & \boldsymbol{v}_2 \\ 0 & 0 \end{pmatrix}$$

in $se(3)$. Thus, by defining the inner product $< \, , \, >_{se(3)}$ on $se(3)$ by

$$< \mathrm{d}h(l_1), \mathrm{d}h(l_2) >_{se(3)} = a_1 \cdot a_2 \ \vartheta_1\vartheta_2/4 \ + \ \boldsymbol{v}_1 \cdot \boldsymbol{v}_2,$$

and defining the Laplacian $\Delta_{SE(3)}$ on $SE(3)$ accordingly, the homomorphism h becomes an isometry that also preserves the Laplacian. Consequently,

$$\Delta_{\mathcal{M}}(F \circ h) = (\Delta_{SE(3)}F) \circ h.$$

In particular, $F \circ h$ is harmonic on $\mathcal{M}$ whenever F is harmonic on $SE(3)$.

For a point $z = (q, t) \in \mathcal{M}$, let $\tilde{z}$ denote the point $\tilde{z} = (-q, t)$. From Section 2.3 we know that z and $\tilde{z}$ correspond to the same rigid body transformation and that h maps them to the same transformation $\tau = h(z) = h(\tilde{z})$.

Theorem 2. *For $\tau \in SE(3)$, let z_τ and $\tilde{z}_\tau$ denote the distinct points in $\mathcal{M}$ for which $h(z_\tau) = h(\tilde{z}_\tau) = \tau$. Then*

$$G(\tau) = K(z_\tau) + K(\tilde{z}_\tau)$$

is the Green kernel on $SE(3)$ with singularity at the identity.

Proof. Assume without loss of generality that z_τ is closer to the identity $\iota \in \mathcal{M}$ than $\tilde{z}_\tau$. Then z_τ is the identity if and only if $\tau = \iota$.

Notice that

$$\begin{aligned}
-\Delta_{SE(3)}G(\tau) =& (-\Delta_{SE(3)}G) \circ h(z_\tau) \\
=& -\Delta_{\mathcal{M}}G(h(z_\tau)) \\
=& -\Delta_{\mathcal{M}}K(z_\tau) - \Delta_{\mathcal{M}}K(\tilde{z}_\tau) \\
=& -\Delta_{\mathcal{M}}K(z_\tau) \\
=& \delta_{\mathcal{M}}(z_\tau) \\
=& \delta_{SE(3)}(\tau),
\end{aligned}$$

where we in the final equality used that h preserves the inner product. Thus, G is a fundamental solution to the Laplacian with singularity at the identity. Since $G > 0$ and $G \to 0$ at infinity, the theorem follows. □

As with the Green kernel K on $\mathcal{M}$, we can translate G along $SE(3)$. Let G^ω denote the Green kernel with singularity at ω. Then

$$G^\omega(\tau) = G(\omega^{-1}\tau). \tag{29}$$

3.4.1 Negative gradient field on $SE(3)$

Let W^ω denote the negative gradient field of G^ω. Then $W^\omega_\tau = \tau W^{\tau^{-1}\omega}_\iota$ for each $\tau \neq \omega$. Thus, to express the tangent vector W^ω_τ it is sufficient to find the Lie algebra element $W^{\tau^{-1}\omega}_\iota$ in $se(3)$.

Lie algebra elements in $\mathfrak{m}$ can be mapped to $se(3)$ by the isometry $\mathrm{d}h$. Since

$$\begin{aligned} G^{\omega}(\tau) =& G(\omega^{-1}\tau) \\ =& K(z_{\omega^{-1}\tau}) + K(\tilde{z}_{\omega^{-1}\tau}) \end{aligned}$$

$W_{\iota}^{\omega} = \mathrm{d}h(V^{z_\tau}) + \mathrm{d}h(V^{\tilde{z}_\tau})$

The Green kernels G^{w} and its gradient field will be used in the following sections as essential ingredients in a path planning algorithm for rigid bodies.

4 Harmonic functions in robot path planning

That harmonic functions have no local minima is a well known property that many potential field planners have taken advantage of. The lack of local minima guarantees that there exists a steepest descent direction at each point, c.f. Lemma 1. In low-dimensional configuration spaces, say of dimension 2 or 3, it is possible to explicitly represent the configuration space obstacles, discretize the space, and numerically calculate a harmonic function with a unique minimum at the goal. Since discretization is needed, these methods are restricted to low-dimensional configuration spaces.

There is a trade-off between obstacle complexity and robot complexity when considering the planning problem in the workspace versus the configuration space. In $\mathcal{C}$, the obstacles are generally very complex and difficult to represent, whereas the robot is a simple point. On the other hand, in $\mathcal{W}$, the obstacles are known and explicitly described, whereas the robot typically is more complex than a point. Thus, due to the complexity of the obstacles $\mathcal{C}_{\mathcal{O}}$ in high-dimensional configuration spaces, we are restrained to partial information obtained by sampling.

Our aim in this paper is to develop a planner that uses harmonic potential functions also in high-dimensional configuration spaces. To avoid sampling in all of $\mathcal{C}$, we use information about the obstacles in $\mathcal{W}$ in order to direct the sampling. By this method we can exclude large portions of $\mathcal{C}$ and instead sample in regions that are promising.

The new planner is described in Section 5. The planner combines the potential fields from two separate planners, so before describing the final planner we need to introduce these two methods. The first one is a well known numerical potential field method that was originally developed in

[12]. This method is described in detail in Section 4.1. The second method, described in Section 4.2, is new and creates an exact potential function in $\mathcal{C}$ by combining translates of the Green kernel. Then, in Section 5, we combine these two techniques into a potential field planner specially designed for rigid bodies moving in $\mathbf{R}^3$.

4.1 Planning using numerically computed fields

Fisrt, consider the problem of planning a path for a point robot moving in $\mathbf{R}^d$ for $d = 2$ or 3. If no obstacles are present, that is $\mathcal{C}_\mathcal{F} = \mathbf{R}^d$, the problem is trivial, but a few observations are still worth considering. Assume that the goal configuration is the origin, and let u be the solution to

$$\begin{aligned} \Delta u &= 0 \text{ in } \mathbf{R}^3 \setminus \{0\} \\ u(0) &= -1 \\ u &= 0 \text{ at infinity.} \end{aligned} \tag{30}$$

Clearly, the potential function u is harmonic in $\mathbf{R}^3 \setminus \{0\}$ and is bounded. Our intuition may say that the negative gradient field will lead the particle towards the origin. However, the solution to (30) is the function

$$\begin{aligned} u &= 0 \text{ in } \mathbf{R}^3 \setminus \{0\} \\ u(0) &= -1. \end{aligned}$$

That is, the gradient field is 0 wherever it is defined. In fact, there is no meaning in specifying finite values at isolated points. There is a theorem saying that isolated singularities of a bounded harmonic function are removable, see [2]. This means that the function has a harmonic extension to all of $\mathbf{R}^3$. In the example above, the constant function $u = 0$ is such an extension. In terms of heat flow, the finite temperature $u = -1$ at the origin is not capable of absorbing any heat from the surroundings at all.

Consequently, the potential at isolated points in $\mathcal{C}_\mathcal{F}$ acting like attractors cannot be finite. Instead, near isolated singularities that influence the entire space, the function u must behave like a fundamental solution to the Laplacian. This may seem like a complication when solving Laplace's equation numerically, but it is not. When discretizing the space, we simply consider each grid point being a representative for a ball around it rather than an isolated point. Likewise, the value of u at a grid point is the average value over the ball instead of the value at the grid point itself. (However, the mean value formulas in Section 3.1 say that these two values coincide if u is harmonic in the entire ball.) Thus, by this interpretation it is possible to specify a finite value at a single grid point and still get influence all over the space.

4.1.1 Potential fields in $\mathbf{R}^2$ and $\mathbf{R}^3$

Now, consider the problem of finding a path for a point robot $\mathcal{A}$ in a bounded workspace $\mathcal{W}$. The position of $\mathcal{A}$ by can be specified by its coordinates with respect to the fixed frame $F_{\mathcal{W}}$ in $\mathcal{W}$. In this simple case, the workspace and the configuration space coincide, thus $\mathcal{C} = \mathcal{W}$, $\mathcal{C}_{\mathcal{F}} = \mathcal{W}_{\mathcal{F}}$, and $\mathcal{C}_{\mathcal{O}} = \mathcal{W}_{\mathcal{O}}$.

Instead of considering the problem (30), we look at the following very similar problem. Let $\epsilon > 0$ and

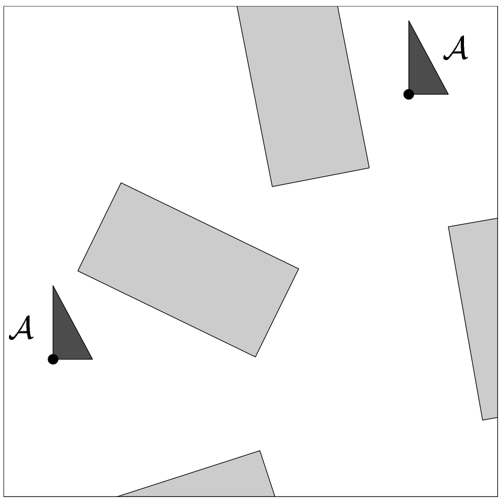
$$\begin{aligned} \Delta u &= 0 \quad \text{in } \mathcal{C}_{\mathcal{F}} \setminus B(\boldsymbol{q}_{goal}, \epsilon) \\ u &= 0 \quad \text{in } \mathcal{C}_{\mathcal{O}} \\ u &= -1 \quad \text{on } \partial B(\boldsymbol{q}_{goal}, \epsilon), \end{aligned} \tag{31}$$

Then the solution is a harmonic potential function u in $\mathcal{C}_{\mathcal{F}} \setminus B(\boldsymbol{q}_{goal}, \epsilon)$ that will guide $\mathcal{A}$ to the goal $B(\boldsymbol{q}_{goal}, \epsilon)$. A numerical solution to this problem can be found by several methods. The simplest is a finite difference scheme on an equally spaced grid in $\mathcal{C}$, see [9, 12]; then the grid is equivalent to the resistive grids used in [42, 52, 54]. An alternative is a finite element mesh, either structured or unstructured, in $\mathcal{C}_{\mathcal{F}}$. In either case, a solution is obtained by solving a sparse system of linear equations. For large systems, iterative methods such as Gauss-Seidel iteration may be used. The rate of convergence can be increased by successive overrelaxation, see [5, 12].

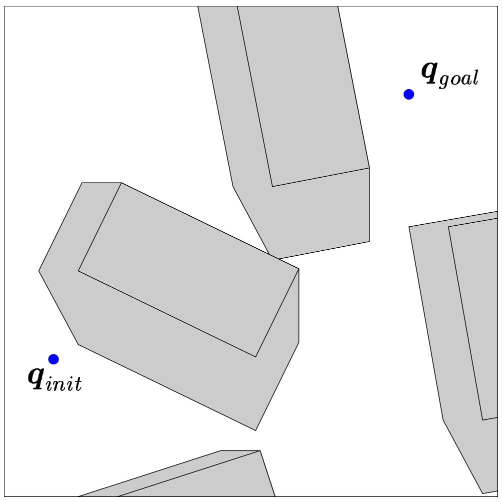
By interpolation, the potential u and its partial derivatives can be approximated at any point in $\mathcal{C}$. The potential u is strictly between -1 and 0 in the interior of $\mathcal{C}_{\mathcal{F}}$, and from any point in $\mathcal{C}_{\mathcal{F}}$ the steepest descent path leads to the goal. Since u is smooth, simple methods such as Euler forward or low order Runge-Kutta methods will be sufficient in order to numerically integrate the negative gradient.

4.1.2 Translating robots

The method above can easily be applied to other simple robots. Consider for instance a spherical robot $\mathcal{A}$ of radius r. If we let the center of $\mathcal{A}$ be its



(a): A 2-dimensional workspace $\mathcal{W}$ with $\mathcal{A}$ in the initial (left) and final configurations. The lower left corner of $\mathcal{A}$ is the origin of $F_{\mathcal{A}}$. The obstacles $\mathcal{W}_{\mathcal{O}}$ are light grey.



(b): Corresponding 2-dimensional configuration space. The configuration space obstacle is the set $\mathcal{C}_{\mathcal{O}} = \mathcal{W}_{\mathcal{O}} \ominus \mathcal{A}$.

Figure 1: Example of a polygonal robot $\mathcal{A}$ among polygonal obstacles. The robot is only allowed to translate.

reference point, then rotations can be ignored, and a configuration is feasible if and only if the reference point is at distance grater than r from any obstacle. Thus, the configuration space obstacle $\mathcal{C}_{\mathcal{O}}$ is obtained by expanding $\mathcal{W}_{\mathcal{O}}$ by r in all directions. The configuration space obstacle can then be calculated by the Minkowski difference $\mathcal{W}_{\mathcal{O}} \ominus \mathcal{A} = \mathcal{W}_{\mathcal{O}} \ominus B(0,r)$. The Minkowski difference is defined as $X \ominus Y = \{x - y : x \in X,\ y \in Y\}$. Once $\mathcal{C}_{\mathcal{O}}$ is calculated, the method for point robots described above in Section 4.1.1 applies.

More generally, if $\mathcal{A}$ is a robot which is only allowed to translate in $\mathcal{W}$, then we can expand the obstacles as follows. Let $\mathcal{A}$ be placed in its home configuration, that is, the frames $F_{\mathcal{A}}$ and $F_{\mathcal{W}}$ coincide. The configuration space obstacle can then be defined as the set $\mathcal{C}_{\mathcal{O}} = \mathcal{W}_{\mathcal{O}} \ominus \mathcal{A}$, see Figure 1 and [37]. Again, we have transformed the robot into a point in the configuration space with known obstacles. In low dimension, the configuration space obstacle can be explicitly constructed and we can use the method in Section 4.1.1 to solve path planning problem.

4.2 Path planning using Green kernels

The method above requires complete knowledge of $\mathcal{C}_{\mathcal{F}}$ and that each grid point (or vertex in a finite element mesh) can be categorized as either feasible or colliding. Analytic solutions to (31) are unattainable, so some kind of discretization is needed. In low-dimensional configuration spaces with moderately sized grids this is not a big problem, but the complexity grows rapidly with increasing dimension. Already in dimension four, the grid needed to achieve reasonable resolution is very large. On the other hand, the categorization is only performed once, and can be seen as a pre-processing step that all subsequent planning queries benefit from.

Instead of requiring complete knowledge of the configuration space obstacles, our aim in this section is to develop a planner that tries to solve the planning problem with partial information obtained from sampling in $\mathcal{C}$. Each sample is costly, so in order to reduce planning time, we wish to sample in an iterative manner. In each iteration, the planner creates a potential function based on the information obtained so far and tries to solve the problem. In contrast with the technique discussed above, the potential function is exact. No discretization or numerical approximation is needed.

The planner presented in this section is by no means complete; it is easy to construct a problem for which the planner fails. However, in Section 5, this planner we will be reinforced with a numerically calculated potential function according to the method in previous section. The combination gives a powerful planner tailored for rigid bodies with six degrees of freedom.

The numerical method in Section 4.1 can be described as keeping the obstacles $\mathcal{C}_{\mathcal{O}}$ at the fixed temperature 0 and a sphere around the goal at the fixed temperature -1 and then calculating the resulting temperature in the interior of $\mathcal{C}_{\mathcal{F}}$. The same potential function can be obtained by a continuous distribution of heat sources on $\partial\mathcal{C}_{\mathcal{F}}$ and a distribution of heat sinks on the sphere around the goal. The latter observation inspired the construction of the potential function used by planner to be presented. That is, heat sources are placed at the colliding sample points and a heat sink is placed at the goal configuration. A particle following the heat flow will then be repelled from the heat sources and attracted to the goal.

In most planning task, the desired goal is a single configuration $\boldsymbol{q}_{goal} \in \mathcal{C}_{\mathcal{F}}$. However, to simplify later extensions of the algorithm, we prefer to have a set $A = \{a_1, \ldots, a_m\}$, $m \geq 1$, of goal configurations, all of which are in the interior of $\mathcal{C}_{\mathcal{F}}$. Moreover, let $B = \{b_1, \ldots, b_n\}$ denote the set of colliding sample points at a certain state of the planning process. Initially B is empty, but as soon as a colliding configuration is found it is added to B. From the sampling scheme described later, it is clear that all points in B belongs to the interior of $\mathcal{C}$.

4.2.1 Overall description

The objective of the planner is to successively build a tree[1] T of feasible configurations. The tree T is initialized with the point $\boldsymbol{q}_{init}$, and is expanded as the planner proceeds. The planner uses a potential function u composed of two parts; $u = U_a + U_r$. The first part, U_a, attracts the robot towards the set of goal points in A, while U_r repels the robot from the set of colliding sample points in B. As soon as a new colliding point is found, the set B is updated and the potential u is recalculated.

In the beginning of each step, one configuration, $\boldsymbol{q}_s \in T$, is selected as start point. Then the potential function u is chosen such that the steepest descent path from $\boldsymbol{q}_s$ is guaranteed to terminate at one of the goal configurations in A. How this is done is described in Section 4.2.3. The potential function u is harmonic in $\mathcal{C} \setminus (A \cup B)$, so there are no local minima, except at the goal, that can trap the robot.

When the potential u is determined, the robot starts at $\boldsymbol{q}_s$ and follows the steepest descent path with a pre-specified step length and collects samples along the path. As long as no collision is found, the tree T is updated with the feasible configurations. If a collision is detected, the set B is updated and the planner starts all over again by selecting a new starting point $\boldsymbol{q}_s$. The planner terminates when it reaches a point in the set A. Then T contains a feasible path that can be found by tracing the way back to the root $\boldsymbol{q}_{init}$.

Now, two choices remain; the potential function u, and the starting point $\boldsymbol{q}_s$ in the tree T. We begin with the potential function.

4.2.2 Potential function

The idea is to let the attracting and the repelling potentials, U_a and U_r respectively, be composed of translates of the Green kernel. Let G^y denote the Green kernel in $\mathcal{C}$ with singularity at y, and define

$$U_a(x) = -\sum \alpha_i G^{a_i}(x) \quad \text{for} \quad \alpha_i > 0. \tag{32}$$

[1]The procedure of building a tree of feasible paths is a well established technique that several planners use, see [23, 27, 30, 39, 43].

From the definition of a Green kernel, see Section 3.2, follows that $U_a < 0$ in the interior of $\mathcal{C}$, $U_a \to 0$ on the boundary or at infinity, and $U_a(a_i) = -\infty$.

Each point $b_j \in B$ is a configuration in $\mathcal{C}_{\mathcal{O}}$, and it is likely that configurations near b_j also are colliding. That is, we wish to keep the robot away from the points in B. Similar to the attracting potential, we compose the repelling potential of Green kernels;

$$U_r(x) = \sum \beta_j G^{b_j}(x) \quad \text{for} \quad \beta_j > 0.$$

Then $U_r > 0$ in the interior of $\mathcal{C}$, $U_r \to 0$ on the boundary or at infinity, and $U_r(b_j) = \infty$.

With these choices of U_a and U_r, the potential u is the solution to the following partial differential equation:

$$\begin{aligned} -\Delta u &= -\sum \alpha_i \delta_{a_i} + \sum \beta_j \delta_{b_j} \\ u &= 0 \quad \text{on } \partial\mathcal{C} \text{ or at infinity.} \end{aligned} \tag{33}$$

Here δ_a is the Dirac measure at the point a in $\mathcal{C}$. The solution u can be interpreted as a temperature distribution in $\mathcal{C}$. Assume that we place heat sinks of intensity $\alpha_1, \ldots, \alpha_m$ at the goal points $a_1, \ldots, a_m$, and heat sources of intensity $\beta_1, \ldots, \beta_n$ at the colliding points $b_1, \ldots, b_n$. Then u is the stationary temperature distribution in $\mathcal{C}$. The heat flows from the heat sources into the sinks. It is possible that there is a flow from or towards the boundary of $\mathcal{C}$ (or infinity if $\mathcal{C}$ is unbounded).

4.2.3 Selecting weights

One suitable choice of $\{\alpha_i\}$, $i = 1, \ldots, m$, is to let all of them have the same weight, for example $\alpha_i = 1$. Then all sinks in A are equally strong. To our help when selecting the weights $\{\beta_j\}$, we have the following lemma:

Lemma 3. *Let u be a solution to* (33). *Then, $u(\boldsymbol{q}) \leq 0$ for a point $\boldsymbol{q}$ in the interior of $\mathcal{C}_{\mathcal{F}}$ implies that the steepest descent path from $\boldsymbol{q}$ leads to a point in the goal set A.*

Proof. Since there is at least one sink, the potential u is non-constant and harmonic in $\mathcal{C} \setminus (A \cup B)$. From Lemma 1 in Section 1 follows that the steepest descent path ends either at A, B or at the boundary of $\mathcal{C}$. Since $\boldsymbol{q}$ is in the interior of $\mathcal{C}_{\mathcal{F}}$, the potential u is strictly decreasing along the steepest descent path, excluding A, $\partial\mathcal{C}$, and infinity as terminating points. $\square$

The repelling potential U_r shall prevent the robot from getting too close to the colliding points in B. Clearly, the repelling force must be strong where the attracting force is strong. A good choice is to let β_j be proportional to $U_a(b_j)$ for $j = 1, \ldots, n$. These intensities are fast to calculate and works well in our examples. They are given in the following theorem.

Theorem 3. *Let $\boldsymbol{q}_s$ be in the interior of $\mathcal{C}_\mathcal{F}$ and let $\beta_j = -w_n U_a(b_j)$, $j = 1, \ldots, n$, where*

$$w_n = \frac{U_a(\boldsymbol{q}_s)}{\sum_{k=1}^n U_a(b_k) G^{b_k}(\boldsymbol{q}_s)},$$

with U_a given in (32). *Then $\beta_j > 0$, $j = 1, \ldots, n$. Moreover, the steepest descent path starting at $\boldsymbol{q}_s$ of the potential u, as defined above, terminates at a point in A.*

Proof. Since $U_a < 0$ in the interior of $\mathcal{C}$ and $\boldsymbol{q}_s$ and b_j are interior points, the weight w_n is positive. Hence, $\beta_j > 0$.

The potential at $\boldsymbol{q}_s$ is

$$\begin{aligned} u(\boldsymbol{q}_s) &= U_a(\boldsymbol{q}_s) + U_r(\boldsymbol{q}_s) \\ &= U_a(\boldsymbol{q}_s) + \sum_{j=1}^n \beta_j G^{b_j}(\boldsymbol{q}_s) \\ &= U_a(\boldsymbol{q}_s) - \frac{\sum_{j=1}^n U_a(\boldsymbol{q}_s) U_a(b_j) G^{b_j}(\boldsymbol{q}_s)}{\sum_{k=1}^n U_a(b_k) G^{b_k}(\boldsymbol{q}_s)} \\ &= 0 \end{aligned}$$

Clearly $u = U_a = U_r = 0$ on $\partial\mathcal{C}$, so the theorem follows from Lemma 3. □

4.2.4 Selecting start point

The simplest choice of starting point $\boldsymbol{q}_s$ is to always select the root in T, that is, restart from $\boldsymbol{q}_{init}$ each time. This method, however, does benefit from the information obtained about the feasible points in T; only information about colliding points is used. An alternative method is to pick a point in T at random. A suggestion is to pick a point $\boldsymbol{q} \in T$ with probability in proportion to $-U_a(\boldsymbol{q})$, c.f. the method in [23].

The method we have used here has been to pick a point $\boldsymbol{q} \in T$ for which $u(\boldsymbol{q})$ is small. Recall that a starting point is selected *before* the potential u

is updated. That is, when the planner detects a collision, the previous point on the path is likely to be the point in T for which u is smallest. This point, however, is very close to $\mathcal{C}_{\mathcal{O}}$ and may not be a good choice. To include the new colliding point b_{n+1} when selecting $\boldsymbol{q}_s$, we pick the point $\boldsymbol{q} \in T$ for which

$$u(\boldsymbol{q}) - w_n U_a(b_j) G^{b_{n+1}}(\boldsymbol{q})$$

is minimal. Here w_n is the weight in Theorem 3 that was used when the weights were calculated at previous time.

4.2.5 Variations

Inspired by the bi-directional planners in [23, 36], we can build two trees of feasible points; T_{init} rooted at $\boldsymbol{q}_{init}$ and T_{goal} rooted at $\boldsymbol{q}_{goal}$. By alternately searching a path from a point in T_{init} to T_{goal} and vice versa, the trees are simultaneously attracted towards each other. When T_{init} and T_{goal} grow larger, subsets of the trees can be used in order to keep the complexity down. This method may be more powerful than growing only one tree, because it takes more advantage of the feasible points that have been found. However, in some cases it is better to use only one tree, c.f. the discussion of uni-directional versus bi-directional expansion in [23].

4.2.6 Potential fields in $\mathbf{R}^d$

In some simple, but yet important, configuration spaces the Green kernels can be expressed in a simple form. The most important is probably $\mathbf{R}^d$ since many configuration spaces are isomorphic to $\mathbf{R}^d$ or can be embedded in $\mathbf{R}^d$. If $\mathcal{C}$ is a proper subset of $\mathbf{R}^d$ we can define all points outside of $\mathcal{C}$ as being colliding configurations. That is, we extend $\mathcal{C}$ to cover all of $\mathbf{R}^d$.

In $\mathbf{R}^d$, for $d \geq 3$, we have already seen the Green kernel in Section 3.2. In $\mathbf{R}^2$ there is no Green kernel. However, for any compact disk D in the plane, there is one. The radius r of the disk can be arbitrary large, so in practice this is no restriction. Recall from Section 3.2 that a fundamental solution to the Laplacian in $\mathbf{R}^2$ is $\widetilde{K}(x) = \frac{-1}{2\pi} \log |x|$. Assume for simplicity that D is the unit disk and that we want a singularity at y in D. By reflecting y in ∂D we get the dual point $\tilde{y} = y/|y|^2$, and we can express the Green kernel on D as

$$G^y(x) = \widetilde{K}(x-y) - \widetilde{K}(|y|(x-\tilde{y})), \tag{34}$$

see [15]. Appropriate scaling gives the Green kernel on a disk of arbitrary radius.

5 Rigid body planning using the Green kernel in $SE(3)$

In this section we turn our attention to path planning for rigid bodies moving in a 3-dimensional workspace $\mathcal{W}$. Our aim is to reinforce the planner in 4.2 by using a potential function in $\mathcal{W}$ similar to the one described in Section 4.1. As before, the planner collects information about $\mathcal{C}$ by sampling. The aim of the potential in $\mathcal{W}$ is to direct the sampling into regions that are promising and thereby reduce the number of samples. Thus, the planner benefits from the explicit representation of $\mathcal{W}_{\mathcal{O}}$ at the same time as the planning takes place in the 6-dimensional configuration space.

In what follows we assume that $\mathcal{W}$ is bounded and that the reference point of $\mathcal{A}$ (that is, the origin of $F_{\mathcal{A}}$) is contained in $\mathcal{A}$. To simplify the presentation, we also assume that the set of goal configurations is a single point $\boldsymbol{q}_{init}$. Moreover, we will identify a point in $\mathcal{W}$ with the corresponding vector in the coordinate system $F_{\mathcal{W}}$. That is, the reference point of $\mathcal{A}$ at configuration $\boldsymbol{q} = (R, t)$ is identified with the vector t. Then we can define the configuration space as the set $\mathcal{C} = SO(3) \times \mathcal{W} \subset SE(3)$.

Let $\boldsymbol{q} = (R, t)$, where $R \in SO(3)$ and $t \in \mathcal{W}$, denote a configuration in $\mathcal{C}$ and let $\boldsymbol{q}_{init} = (R_{init}, t_{init})$ and $\boldsymbol{q}_{goal} = (R_{goal}, t_{goal})$. Recall the definition of the Green kernel $G^{\boldsymbol{q}_0}(\boldsymbol{q})$ from Section 3.4. Without confusion, we will also use $G^{\boldsymbol{q}_0}(R, t)$ to denote the same kernel.

5.1 Attracting potential

The essential difference between this planner and the sampling planner in Section 4.2 is the attracting potential U_a. We will modify U_a such that it has two parts u_0 and u_1 calculated in $\mathcal{W}$ and one part $G^{\boldsymbol{q}_{goal}}$ calculated in $\mathcal{C}$. Define the set

$$\widetilde{\mathcal{C}}_{\mathcal{F}} = SO(3) \times \mathcal{W}_{\mathcal{F}}.$$

Then $\mathcal{C}_{\mathcal{F}} \subset \widetilde{\mathcal{C}}_{\mathcal{F}}$. Let u_0 be the potential in $\mathcal{W}$ according to the method in Section 4.1.1 with a minimum at the point t_{init} and let u_1 be the harmonic function for which

$$\begin{aligned} \Delta u_1 &= 0 \text{ in } \mathcal{W}_{\mathcal{F}} \\ u_1(t) &= G^{\boldsymbol{q}_{goal}}(R_{goal}, t) \text{ on } \partial\mathcal{W}_{\mathcal{F}}. \end{aligned}$$

Then, this theorem gives the attracting potential.

Theorem 4. *Let $\delta = u_1(t_{init}) - G^{\boldsymbol{q}_{goal}}(\boldsymbol{q}_{init})$. If $\delta > 0$, pick α such that*

$$0 < \alpha < -\frac{u_0(t_{init})}{\delta},$$

otherwise, pick any $\alpha > 0$. For $\boldsymbol{q} = (R, t)$, define

$$U_a(\boldsymbol{q}) = u_0(t) + \alpha(u_1(t) - G^{\boldsymbol{q}_{goal}}(\boldsymbol{q})).$$

Then,

a) $U_a \geq 0$ on $\partial\widetilde{\mathcal{C}}_{\mathcal{F}}$,

b) U_a is harmonic in $\widetilde{\mathcal{C}}_{\mathcal{F}} \setminus \{\boldsymbol{q} : t = t_{goal}\}$,

c) U_a has a singularity at $\boldsymbol{q}_{goal}$, and $U_a \to -\infty$ as $\boldsymbol{q} \to \boldsymbol{q}_{goal}$,

d) U_a has no local minima in $\widetilde{\mathcal{C}}_{\mathcal{F}} \setminus \{\boldsymbol{q}_{goal}\}$, and

e) $U_a(\boldsymbol{q}_{init}) < 0$.

Proof. Let G be the Green kernel in $SE(3)$ with singularity at the identity $(I, 0)$, where I denotes the identity in $SO(3)$. Using (29) and (8), a simple calculation gives

$$\begin{aligned} G^{\boldsymbol{q}_{goal}}(\boldsymbol{q}) &= G(\boldsymbol{q}_{goal}^{-1}\boldsymbol{q}) \\ &= G(R_{goal}^{-1}R, R_{goal}^{-1}(t - t_{goal})) \end{aligned}$$

and

$$\begin{aligned} G^{\boldsymbol{q}_{goal}}(R_{goal}, t) &= G(\boldsymbol{q}_{goal}^{-1}(R_{goal}, t)) \\ &= G(I, R_{goal}^{-1}(t - t_{goal})). \end{aligned}$$

Since $G(R,t) \leq G(I,t)$ for all (R,t), we see that $G^{\boldsymbol{q}_{goal}}(\boldsymbol{q}) \leq G^{\boldsymbol{q}_{goal}}(R_{goal},t)$. Hence, for $\boldsymbol{q} \in \partial\widetilde{\mathcal{C}}_{\mathcal{F}}$ we have

$$\begin{aligned} U_a(\boldsymbol{q}) &= u_0(t) + \alpha u_1(t) - \alpha G^{\boldsymbol{q}_{goal}}(\boldsymbol{q}) \\ &\geq 0 + \alpha u_1(t) - \alpha G^{\boldsymbol{q}_{goal}}(R_{goal},t) \\ &= 0, \end{aligned}$$

and a) follows.

Since $u_0(t) + \alpha u_1(t)$ is harmonic in $\mathcal{W}_{\mathcal{F}} \setminus \{t_{goal}\}$, the extension to $\mathcal{C}$ is harmonic for $t \neq t_{goal}$. Moreover, the extension has its minimum (and is constant) in the set $\{\boldsymbol{q} : t = t_{goal}\}$. Since $-\alpha G^{\boldsymbol{q}_{goal}}(\boldsymbol{q})$ decreases monotonically towards $-\infty$ as $\boldsymbol{q}$ approaches $\boldsymbol{q}_{goal}$ along a geodesic, there can be no local minima in $\widetilde{\mathcal{C}}_{\mathcal{F}} \setminus \{\boldsymbol{q}_{goal}\}$.

To finally show e), assume first that $\delta > 0$. Since t_{init} is in the interior of $\mathcal{W}_{\mathcal{F}}$, the potential $u_0(t_{init})$ is negative, so the interval $(0, -u_0(t_{init})/\delta)$ in non-empty. Thus,

$$\begin{aligned} U_a(\boldsymbol{q}_{init}) &= u_0(t_{init}) + \alpha(u_1(t_{init}) - G^{\boldsymbol{q}_{goal}}(\boldsymbol{q}_{init})) \\ &= u_0(t_{init}) + \alpha\delta \\ &< u_0(t_{init}) - \frac{u_0(t_{init})}{\delta}\delta \\ &= 0. \end{aligned}$$

If $\delta \leq 0$, the statement e) follows immediately, and the proof is complete. □

Now, we can use the potential U_a as the attracting potential in the sampling algorithm presented in Section 4.2. The idea is that U_a shall direct the robot to configurations that are promising. In particular, since $U_a \geq 0$ on $\partial\widetilde{\mathcal{C}}_{\mathcal{F}}$ the robot will never leave $\widetilde{\mathcal{C}}_{\mathcal{F}}$. That is, the reference point of $\mathcal{A}$ will never intersect the obstacles.

Remark 1: The set $\widetilde{\mathcal{C}}_{\mathcal{F}}$ to which the planner is restricted, covers the set $\mathcal{C}_{\mathcal{F}}$. It is therefore an advantage to find a set $\widetilde{\mathcal{C}}_{\mathcal{F}}$ that covers $\mathcal{C}_{\mathcal{F}}$ as tightly as possible. If $\mathcal{A}$ completely covers a ball of radius r with center at the reference point of $\mathcal{A}$, then the method in Section 4.1.2 applies; by expanding the obstacles in $\mathcal{W}$ by r and defining $\widetilde{\mathcal{C}}_{\mathcal{F}}$ accordingly, the planner has a more accurate estimate of $\mathcal{C}_{\mathcal{F}}$.

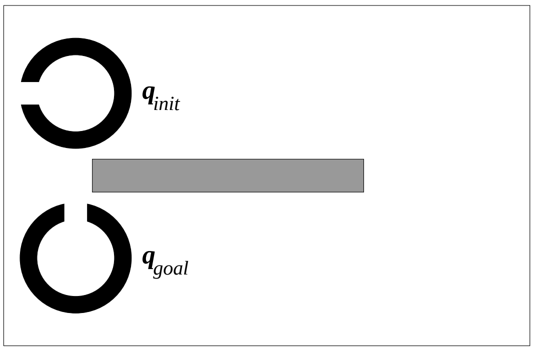


Figure 2: An example of a narrow passage where the robot cannot pass through.

Remark 2: A large value of α in Theorem 4 can make $U_a > 0$ on parts of $\partial\widetilde{\mathcal{C}}_{\mathcal{F}}$, and therefore prevent the robot to reach certain regions in $\mathcal{C}_{\mathcal{F}}$ that may be essential to solve the planning task. A solution to this problem is to successively decrease α towards 0 if the planner does not find a solution. The region that the robot can reach increases to $\widetilde{\mathcal{C}}_{\mathcal{F}}$ as α tends to zero.

On the other hand, a very small value of α makes the sink at $\boldsymbol{q}_{goal}$ weak. However, the robot will still be pulled towards the set $\{\boldsymbol{q} : t = t_{goal}\}$ where u_0 is minimal. On this set, U_a is not harmonic, so when the repelling potential is added to U_a, local minima may occur. Consequently, the robot may get stuck at the right position t_{goal}, but with wrong orientation. This problem needs further analysis, but a possible solution may be to switch off the potential u_0 as the robot comes very close to t_{goal}.

Remark 3: The sampling planner in Section 4.2 is not complete. The potentials u_0 and u_1 reinforce the planning capability a lot for rigid body robots. However, there are still situations in which the planner takes very long time to find a solution, or even fails. Imagine a situation like in Figure 2. The robot is close to the goal, but cannot pass the narrow passage. Because of the geometry of the robot, the obstacles in $\mathcal{W}$ cannot be expanded very much (c.f. Remark 1). The flow due to u_0 will be strong through the narrow passage, and it will take long time to "fill up" the passage with singularities before the flow from $\boldsymbol{q}_{init}$ takes the way through the large passage. In the example, the obvious solution would be to approximate the robot with a circular disk and expand the obstacles accordingly. Then the narrow passage

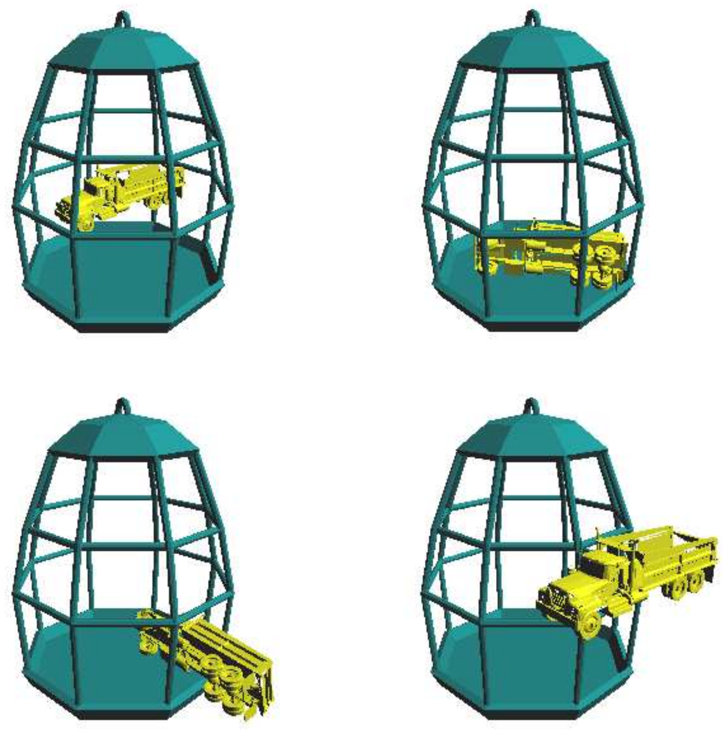


Figure 3: A truck (22284 triangles) escaping from a cage (1032 triangles).

would be closed. This is not a general solution, however.

Another solution, that has not yet been further investigated, would be to modify u_0 gradually during the planning process. Assume that a collision is found at $\boldsymbol{q} = (R, t)$. In addition to updating the repelling potential, we slightly increase the potential u_0 at t. This will destroy the harmonicity at t, but since we increase u_0, only local maxima will occur. One suggestion is to let the new value of $u_0(t)$ be $\gamma u_0(t)$ for some $\gamma \in (0, 1)$ close to 1. (Recall that u_0 is negative in the interior of $\widetilde{\mathcal{C}}_{\mathcal{F}}$.) Of course u_0 must be updated in the remaining part of $\widetilde{\mathcal{C}}_{\mathcal{F}}$.

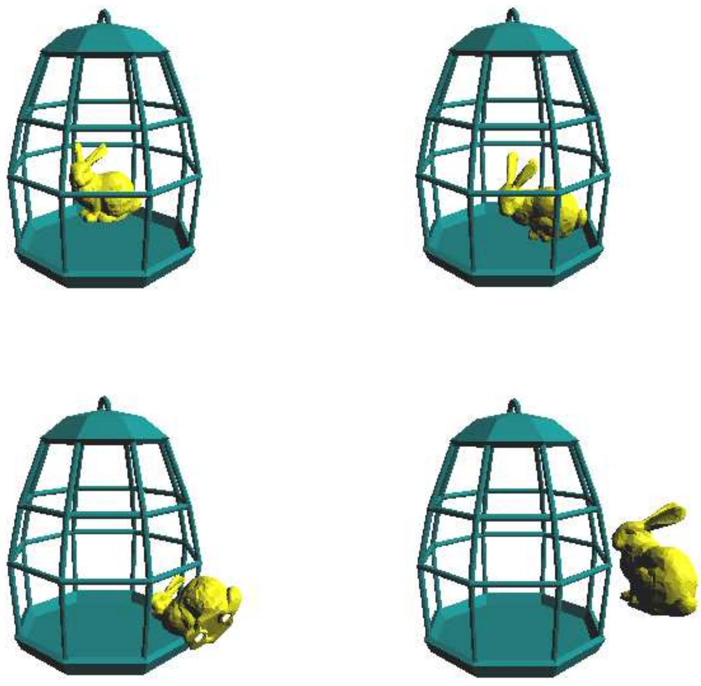


Figure 4: A bunny (2204 triangles) escaping from a cage (1032 triangles).

5.2 Experimental results

In this section we present some preliminary performance tests for the planner. The algorithm was implemented in C++ on a 1GHz PC and the collision checks were handled by the Proximity Query Package (PQP) from the University of North Carolina.

In the two examples shown in Figure 3 and 4, we performed several experiments. In each case, the start configuration was located inside the cage[2] and the goal configuration was located outside the cage. The figures show snapshots along paths that were generated by the planner. The upper left pictures shows the initial configurations and the bottom right pictures shows the goal configurations.

[2]The cage is delivered with the Motion Strategy Library (MSL) from the University of Illinois.

	Pre-proc	Calc. u_0 and u_1	Min	Average	Max
Truck	3.00	0.51	0.47	7.35	31.23
Bunny	0.73	0.09	2.04	10.70	20.87

Table 1: Running times for the examples with the truck and the bunny based on 20 consecutive runs each.

The workspace was bounded by a box and the potentials u_0 and u_1 were calculated on a grid with 40000 points in the truck[3] example and 10000 points in the bunny[4] example. The reference points of the truck and the bunny were enclosed by balls, and the obstacles $\mathcal{W}_\mathcal{O}$ were expanded accordingly, see Remark 1 on page 45. The first column of Table 1 shows the pre-processing time required to cathegorize the grid points as inside or outside the expanded obstacles.

A small change of the start or the goal configuration may affect the potential field so that the planning process takes another course. Therefore the running times vary somewhat for different start and goal configurations. We performed 20 trials on each of the two examples. The second column of Table 1 shows the average computation time used to calculate u_0 and u_1. The last three columns show the minimum, maximum, and average time used for the iterative construction of the tree T until a solution is found.

Our experiment involving the truck is similar to one of the experiments conducted in [6]. The planner in [6] uses a lattice in a cube in $\mathbf{R}^6$ to represent $\mathcal{C}$ and the A* algorithm for searching on the lattice. A comparison of the running times shows that the new potential field method presented in this paper is significantly faster in the truck example than the lattice-based Lazy PRM in [6].

6 Summary and discussion

In this paper we have introduced a novel approach to rigid body path planning. The fundamental tool used by the planner is the Green kernel in the configuration space. A harmonic potential function composed of translates

[3]The truck was kindly provided by Steven LaValle, University of Illinois.

[4]The bunny is delivered with the Proximity Query Package (PQP) from the University of North Carolina

of the Green kernel is used to guide the robot towards the goal. The planning takes place directly in the group $SE(3)$ and is reinforced by a potential function in the workspace.

The experiments are challenging and the results so far are promising, although the planner is not complete. Having in mind that this is a novel approach that is not as developed and mature as other planning techniques, it shows great potential for future research. Several variations of the planner are possible, and a few of them are proposed in Section 5 and in the remarks on page 46. Some of them have yet to be investigated in theory as well as in practice.

The applications for rigid body path planners are many. In addition to free-flying robots, articulated robot arms may take advantage of a rigid body planner. In industry, robots are typically articulated and their configuration spaces are usually parameterized by a subset of $\mathbf{R}^n$. However, the robot may also be programmed by specifying the position and orientation of the end-effector. The robot configuration (given by the joint values) is then calculated using inverse kinematics. So, if the trajectory of the tool is of great importance, or if the payload is large in comparison to the robot, it may be an advantage to use the tool position and orientation in the planning process instead of the joint angles. In such situations, the rigid body planner presented here can work as a complement to an ordinary planner.

Acknowledgments

The author would like to thank Bo Johansson for valuable supervision, support and comments on the manuscript, and Peter Sjögren, Grigori Rozenblioum and Bo Berndtsson for their help on Green kernels.

A The Laplacian on $S^3 \times \mathbf{R}^3$

Spherical coordinates (r, ϕ, ψ) in $\mathbf{R}^3$ are related to Cartesian coordinates (x, y, z) by

$$\begin{aligned} x &= r \sin\phi \sin\psi \\ y &= r \sin\phi \cos\psi \\ z &= r \cos\phi, \end{aligned}$$

and the Laplacian is given by

$$\Delta u = u_{rr} + \frac{2}{r} u_r + \frac{1}{r^2 \sin\phi} (u_\phi \sin\phi)_\phi + \frac{1}{r^2 \sin^2\phi} u_{\psi\psi}, \tag{35}$$

see e.g. [1, 13]. Introducing cylindrical coordinates in $\mathbf{R}^4$ as follows

$$\begin{aligned} x &= r \sin\phi \sin\psi \\ y &= r \sin\phi \cos\psi \\ z &= r \cos\phi \\ w &= w, \end{aligned}$$

we immediately obtain the Laplacian in cylindrical coordinates;

$$\Delta u = u_{rr} + \frac{2}{r} u_r + \frac{1}{r^2 \sin\phi} (u_\phi \sin\phi)_\phi + \frac{1}{r^2 \sin^2\phi} u_{\psi\psi} + u_{ww}. \tag{36}$$

In $\mathbf{R}^4$ we have the following spherical coordinates

$$\begin{aligned} x &= \rho \sin\eta \sin\phi \sin\psi \\ y &= \rho \sin\eta \sin\phi \cos\psi \\ z &= \rho \sin\eta \cos\phi \\ w &= \rho \cos\eta. \end{aligned} \tag{37}$$

We see that the cylindrical variables (w, r) are related to the spherical variables (ρ, η) by the equations

$$\begin{aligned} w &= \rho \cos\eta \\ r &= \rho \sin\eta. \end{aligned}$$

These are exactly the equations relating polar and Cartesian coordinates in the plane. Therefore, using the Laplacian in polar coordinates, we have

$$u_{ww} + u_{rr} = u_{\rho\rho} + \frac{1}{\rho}u_\rho + \frac{1}{\rho^2}u_{\eta\eta}. \tag{38}$$

By the chain rule,

$$\begin{aligned} u_\rho &= \frac{\partial u}{\partial w}\frac{\partial w}{\partial \rho} + \frac{\partial u}{\partial r}\frac{\partial r}{\partial \rho} = u_w \cos\eta + u_r \sin\eta \\ u_\eta &= \frac{\partial u}{\partial w}\frac{\partial w}{\partial \eta} + \frac{\partial u}{\partial r}\frac{\partial r}{\partial \eta} = -u_w \rho \sin\eta + u_r \rho\cos\eta. \end{aligned}$$

Multiplying these equalities by $\sin\eta$ and $\rho^{-1}\cos\eta$ respectively, and adding them, gives

$$u_r = u_\rho \sin\eta + \rho^{-1}u_\eta \cos\eta. \tag{39}$$

Substitution of (38) and (39) into (36) gives

$$\begin{aligned} \Delta u &= u_{\rho\rho} + \frac{1}{\rho}u_\rho + \frac{1}{\rho^2}u_{\eta\eta} + \frac{2}{r}u_r + \frac{1}{r^2\sin\phi}(u_\phi \sin\phi)_\phi + \frac{1}{r^2\sin^2\phi}u_{\psi\psi} \\ &= u_{\rho\rho} + \frac{3}{\rho}u_\rho + \frac{1}{\rho^2}u_{\eta\eta} + \frac{2\cos\eta}{\rho^2\sin\eta}u_\eta + \frac{1}{\rho^2\sin^2\eta\sin\phi}\Big((u_\phi\sin\phi)_\phi + \frac{1}{\sin\phi}u_{\psi\psi}\Big) \\ &= u_{\rho\rho} + \frac{3}{\rho}u_\rho + \frac{1}{\rho^2\sin^2\eta}(u_\eta \sin^2\eta)_\eta + \frac{1}{\rho^2\sin^2\eta\sin\phi}\Big((u_\phi\sin\phi)_\phi + \frac{1}{\sin\phi}u_{\psi\psi}\Big). \end{aligned}$$

On the unit sphere S^3 in $\mathbf{R}^4$ we have $\rho = 1$, and if $u = u(\eta)$ the formula simplifies to

$$\Delta u = u_{\eta\eta} + \frac{2\cos\eta}{\sin\eta}u_\eta = \frac{1}{\sin^2\eta}(u_\eta \sin^2\eta)_\eta \tag{40}$$

Let $x \in S^3$, $y \in \mathbf{R}^3$ and $z = (x, y) \in \mathcal{M} = S^3 \times \mathbf{R}^3$. By (40) and (35) we see that the Laplacian for a function $u : \mathcal{M} \to \mathbf{R}$ such that $u(z) = \tilde{u}(\eta, r)$, where η is the angle from the north pole on S^3 (c.f. the spherical coordinates (37)) and $r = |y|$, is

$$\Delta u = \frac{1}{\sin^2\eta}(\tilde{u}_\eta \sin^2\eta)_\eta + \tilde{u}_{rr} + \frac{2}{r}\tilde{u}_r.$$

B Conversions

In this appendix we show the conversions between the different representations of rotations used in this paper. We use right-handed coordinate systems and the direction of rotation about an axis is determined by the right-hand rule: If the vector is held with the right hand and the thumb is pointing along the vector, then the fingers point in the positive direction.

B.1 Euler angles to $SO(3)$

The elementary rotations about the coordinate axis are given by the matrices

$$R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix}$$
$$R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}$$
$$R_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

For the Euler angles (α, β, γ) we have the rotation matrix

$$R_{\text{Euler}}(\alpha,\beta,\gamma) = R_z(\alpha)R_y(\beta)R_z(\gamma) = \begin{pmatrix} c_\alpha c_\beta c_\gamma - s_\alpha s_\gamma & -c_\alpha c_\beta s_\gamma - s_\alpha c_\gamma & c_\alpha s_\beta \\ s_\alpha c_\beta c_\gamma + c_\alpha s_\gamma & -s_\alpha c_\beta s_\gamma + c_\alpha c_\gamma & s_\alpha s_\beta \\ -s_\beta c_\gamma & s_\beta s_\gamma & c_\beta \end{pmatrix}.$$

Here $c_\alpha = \cos\alpha$, $s_\alpha = \sin\alpha$ and similarly for the other angles.

Conversely, if we are given a rotation matrix R, then we can find a set of Euler angles that matches R as follows. Looking at R_{Euler} above, we see that for $-1 < r_{33} < 1 \Rightarrow \beta \neq 0$,

$$\begin{aligned} \alpha &= \operatorname{atan2}(r_{23}, r_{13}) \\ \beta &= \operatorname{atan2}(\sqrt{r_{31}^2 + r_{32}^2}, r_{33}) \\ \gamma &= \operatorname{atan2}(r_{32}, -r_{31}), \end{aligned}$$

where r_{ij} are the elements of R and $\operatorname{atan2}(b, a)$ gives the argument in the range $[0, 2\pi)$ for the complex number $a + ib$. For $r_{33} = 1 \Rightarrow \beta = 0$, we cannot distinguish α from γ, so we arbitrarily define $\gamma = 0$ and get the solution

$$\begin{aligned} \alpha &= \operatorname{atan2}(r_{21}, r_{11}) \\ \beta &= 0 \\ \gamma &= 0 \end{aligned}$$

Similarly, for $r_{33} = -1 \Rightarrow \beta = \pi$, we get

$$\begin{aligned} \alpha &= \operatorname{atan2}(-r_{21}, -r_{11}) \\ \beta &= \pi \\ \gamma &= 0 \end{aligned}$$

The above formulas gives an invers solution for which α and γ are restricted to $[0, 2\pi)$ and β is restricted to $[0, \pi]$.

B.2 Equivalent axis to $SO(3)$

Each rotation in $\mathbf{R}^3$ can be seen as a rotation by an angle θ about a vector $a = (a_1, a_2, a_3)^T$. Without loss of generality, we restrict θ to be in the interval $[0, 2\pi)$. For $\theta \neq 0$, we let a be a unit vector and for $\theta = 0$ we define $a = 0$. The vector θa is the equivalent axis of the rotation. Clearly, $\theta a \in B(0, 2\pi)$, where $B(0, 2\pi)$ is the ball of radius 2π centered at the origin. The equivalent axis, θa, is also called exponential coordinates for the rotation. We will also refer to the pair (θ, a) as exponential coordinates, which will cause no confusion.

A rotation by an angle θ about a is given by the matrix

$$R = e^{\theta \hat{a}} = I + \hat{a} \sin\theta + \hat{a}^2 (1 - \cos\theta), \tag{41}$$

where I is the 3×3 identity matrix and

$$\hat{a} = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix}.$$

The formula (41) is sometimes called Rodrigues' formula [25, 41]. The mapping from $B(0, 2\pi)$ to $SO(3)$ is two-to-one except that the only pre-image of the identity in $SO(3)$ is $\theta\hat{a} = 0$. For $\theta \neq 0$, the exponential coordinates $(2\pi - \theta, -a)$ give the same rotation.

To convert from the rotation matrix R to equivalent axis representation, we notice that

$$\hat{a}^2 = \begin{pmatrix} a_1^2 - 1 & a_1 a_2 & a_1 a_3 \\ a_1 a_2 & a_2^2 - 1 & a_2 a_3 \\ a_1 a_3 & a_2 a_3 & a_3^2 - 1 \end{pmatrix} = aa^T - I$$

is symmetric, hence

$$R - R^T = 2\hat{a}\sin\theta,$$

and that

$$\operatorname{trace}(R) = 3 - 2(1 - \cos\theta).$$

Let $\boldsymbol{v}$ be the vector for which $\hat{\boldsymbol{v}} = (R - R^T)/2$. Then $\boldsymbol{v} = a\sin\theta$ and one solution is

$$\theta = \operatorname{atan2}(|v|, (\operatorname{trace}(R) - 1)/2). \tag{42}$$

For $\theta = 0$, we let $a = 0$ and for $\theta \notin \{0, \pi\}$, a follows from the relation $\boldsymbol{v} = a\sin\theta$. The case $\theta = \pi$ requires some extra care. Let $M = (R + I)/2$. Then, for $\theta = \pi$, $M = aa^T$ and we can take a as the i:th column of $M/\sqrt{M_{ii}}$, where i is chosen such that M_{ii} is the largest diagonal element of M. We summarize: Let

$$a = \begin{cases} 0 & \text{if } \theta = 0 \\ M_{\cdot i}/\sqrt{M_{ii}} & \text{if } \theta = \pi \\ \frac{1}{\sin\theta}\boldsymbol{v} & \text{otherwise,} \end{cases}$$

where θ is given by (42) and $M_{\cdot i}$ is the i:th column of M. Then the pre-image of $R \in SO(3)$ in Rodrigues' formula are the exponential coordinates (θ, a) and, if $\theta \neq 0$, $(2\pi - \theta, -a)$,

B.3 Equivalent axis to quaternions

The mapping

$$\theta a \mapsto q = [\boldsymbol{v}, w] = [a^T \sin(\theta/2), \cos(\theta/2)] \tag{43}$$

from $B(0, 2\pi) \to H^1$ is a bijection onto its range. The inverse is given by

$$\theta = 2\arccos w, \qquad a = \begin{cases} \frac{1}{\sin(\theta/2)}\boldsymbol{v} & \text{if } \sin(\theta/2) \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

The only point in H^1 that is not in the range of the mapping (43) is the quaternion $[0,-1]$. This point corresponds to a rotation by 2π about some axis, and we define its inverse to be the exponential coordinates $(\theta, a) = (0, 0)$. Thus, we can use the formula for the inverse on all of H^1 having in mind that the preimage of $(0,0)$ are the quaternions $[0,1]$ and $[0,-1]$.

B.4 Quaternions to $SO(3)$

The set of unit quaternions H^1 is a Lie group that we identify with S^3 by

$$q = [(x_1, x_2, x_3), x_4] \longleftrightarrow x = (x_1, x_2, x_3, x_4)^T.$$

Let θa be the equivalent axis that (43) maps to the quaternion

$$q = [(a_1, a_2, a_3)\sin(\theta/2), \cos(\theta/2)].$$

Then Rodrigues' formula, the substitution $[(x_1, x_2, x_3), x_4]$ for $[(a_1, a_2, a_3)\sin(\theta/2), \cos(\theta/2)]$, and an elementary calculation gives

$$\begin{aligned} R(q) &= I + 2\hat{a}\sin(\theta/2)\cos(\theta/2) + 2\hat{a}^2\sin^2(\theta/2) \\ &= \begin{pmatrix} 1-2(x_2^2+x_3^2) & 2(x_1x_2 - x_3x_4) & 2(x_1x_3 + x_2x_4) \\ 2(x_1x_2 + x_3x_4) & 1-2(x_1^2+x_3^2) & 2(x_2x_3 - x_1x_4) \\ 2(x_1x_3 - x_2x_4) & 2(x_2x_3 + x_1x_4) & 1-2(x_1^2+x_2^2) \end{pmatrix} \end{aligned} \tag{44}$$

A map between Lie groups is a (Lie group) homomorphism if it is differentiable and preserves the group structure. The map (44) is a surjective homomorphism from H^1 onto $SO(3)$. A tedious calculation shows that $R(q)R(q') = R(qq')$ for two quaternions q and q'.

To find the pre-image of R, we see that we can use the off-diagonal elements of R to determine x_1, x_2, x_3 and x_4. The trouble is that we must divide with either of x_1, x_2, x_3 or x_4. Each of them can be zero, so to make sure to avoid dividing with zero, and also to increase the numerical precision, we wish to divide with the element with the largest modulus. Let

$$v = \begin{pmatrix} 1 & -1 & -1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} r_{11} \\ r_{22} \\ r_{33} \\ 1 \end{pmatrix}$$

and

$$M = \frac{1}{2}\begin{pmatrix} v_1 & r_{12}+r_{21} & r_{13}+r_{31} & r_{32}-r_{23} \\ r_{12}+r_{21} & v_2 & r_{23}+r_{32} & r_{13}-r_{31} \\ r_{13}+r_{31} & r_{23}+r_{32} & v_3 & r_{21}-r_{12} \\ r_{32}-r_{23} & r_{13}-r_{31} & r_{21}-r_{12} & v_4 \end{pmatrix}$$

Then $v = 4(x_1^2, x_2^2, x_3^2, x_4^2)^T$ and $M = 2(x_1x, x_2x, x_3x, x_4x)$. Thus, if i is chosen such that v_i is the largest element in v, then the quaternion q corresponding to the i:th column of $\pm M/\sqrt{v_i}$ is the pre-image of R. The sign of q cannot be determined, because q and $-q$ gives the same matrix R. Hence, (44) is a two-to-one homomorphism.

References

[1] G.B. Arfken and H.J. Weber. *Mathematical methods for physicists.* Academic Press, 1995.

[2] S. Axler, P. Bourdon, and W. Ramey. *Harmonic Function Theory.* Springer, 1992.

[3] J. Barraquand and J.C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. of Rob. Research*, 10:628–649, 1991.

[4] N. Berline, E. Gretzler, and M. Vergne. *Heat Kernels and Dirac Operators.* Springer, 1992.

[5] Å. Björck. *Numerical Methods for Least Squares Problems.* SIAM, Philadelphia, 1996.

[6] M.S. Branicky, S.M LaValle, K. Olson, and L. Yang. Quasi-randomized path planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 2001.

[7] J.F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, MA, 1988.

[8] H. Chang. A new technique to handle local minimum for imperfect potential field based motion planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1996.

[9] H. Choset and J. Burdick. Sensor based planning and nonsmooth analysis. In *Proc. of IEEE Int. Conf. Robotics and Automation*, pages 3034–3041, San Diego, CA, 1994.

[10] H. Choset and D. Kortenkamp. Path planning and control for free-flying inspection robot in space. *Journal of Aerospace Engineering*, 12(2):74–81, 1999.

[11] C.I. Connolly and J.B. Burns. Path planning using Laplace's equation. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 2102 – 2106, 1990.

[12] C.I. Connolly and R.A. Grupen. The applications of harmonic functions to robotics. *Journal of Robotic Systems*, 1993.

[13] R. Courant and D. Hilbert. *Methods of mathematical physics Vol. 1.* Interscience Publishers, 1953.

[14] E.B. Dam, M. Koch, and M. Lillholm. Quaternions, interpolation and animation. Technical Report DIKU-TR98/5, Dept. of Computer Science, University of Copenhagen, Denmark, 1998.

[15] L.C. Evans. *Partial Differential Equations.* American Mathematical Society, 1998.

[16] H.J.S. Feder and J.J.E. Slotine. Real-time path planning using harmonic potentials in dynamic environments. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, pages 874–881, 1997.

[17] P. Finn and L. Kavraki. Computational approaches to drug design. *Algorithmica*, 25:347–371, 1999.

[18] G.B. Folland. *Introduction to Partial Differential Equations.* Princeton University Press, 1976.

[19] H. Goldstein, C. Pole, and J. Safko. *Classical Mechanics.* Addison Wesley, 2002.

[20] A. Grigor'yan. Analytic and geometric background of recurrence and non-explosion of the brownian motion on riemannian manifolds. *Bulletin of Amer. Math. Soc.*, 36:135–249, 1999.

[21] K. Gupta and A.P. del Pobil. *Practical Motion Planning in Robotics.* John Wiley, West Sussex, England, 1998.

[22] D. Hearn and M.P. Baker. *Computer Graphics.* Prentice Hall, 1994.

[23] D. Hsu, J.C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Int. Journal of Computational Geometry and Applications*, 9(4-5):495–512, 1999.

[24] Y.K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Comp. Surveys*, 24(3):219–291, 1992.

[25] A. Iserles, H.Z. Munthe-Kaas, S.P. Nørsett, and A. Zanna. Lie-group methods. *Acta Numerica*, 9:215–365, 2000.

[26] V.I. Utkin J. Guldner. Sliding mode control for an obstacle avoidance strategy based on an harmonic potential field. In *Conf. on Decision and Control*, 1993.

[27] J.J. Kuffner, Jr. and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 2000.

[28] M. Kalisiak and M. van de Panne. A grasp-based motion planning algorithm for character animation. *Journal of Visualization and Computer Animation*, 12(3):117–129, 2001.

[29] A.A. Kassim and B.V.K.V. Kumar. Path planners based on the wave expansion neural network. *Robotics and Autonomous Systems*, 26(1):1–22, 1999.

[30] L.E. Kavraki, P. Švestka, J.C. Latombe, and M. Overmars. Probabilistic roadmaps for fast path planning in high dimensional configuration spaces. *IEEE Tr. on Rob. & Aut.*, 12:566–580, 1996.

[31] O.D. Kellogg. *Foundations of Potential Theory.* Frederick Ungar Publishing Company, 1929.

[32] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. of Robotics Research*, 5:90–98, 1986.

[33] J.O. Kim and P. Kosla. Real-time obstacle avoidance using harmonic potential functions. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1991.

[34] Y. Koga, K. Kondo, J. Kuffner, and J.C. Latombe. Planning motions with intentions. *Computer Graphics (SIGGRAPH'94)*, pages 395–408, 1994.

[35] J.J. Kuffner, Jr. *Autonomous Agents for Real-time Animation.* PhD thesis, Stanford University, Stanford, CA, 1999.

[36] F. Lamiraux and L.E. Kavraki. Planning paths for elastic objects under manipulation constraints. *Int. J. of Rob. Research*, 20(3), 2001.

[37] J.C. Latombe. *Robot Motion Planning.* Kluwer, Boston, MA, 1991.

[38] J.C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *Int. J. of Rob. Research*, 18(11):1119–1128, 1999.

[39] S.M. LaValle and J.J. Kuffner, Jr. Randomized kinodynamic planning. *Int. J. of Rob. Research*, 20(5):378–400, 2001.

[40] Z.X. Li and T.D. Bui. Robot path planning using fluid model. *Journal of Intelligent and Robotic Systems*, 21:29–50, 1998.

[41] J.E. Marsden and T.S. Ratiu. *Introduction to Mechanics and Symmetry.* Springer, 1999.

[42] G.F. Marshall and L. Tarassenko. Robot path planning using resistive grids. In *Int. Conf. on Artificial Neural Networks*, 1991.

[43] E. Mazer, J.M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. *J. of Art. Intelligence Research*, 9:295–316, 1998.

[44] Phillip J. McKerrow. *Introduction to Robotics.* Addison-Wesley, 1991.

[45] A. McLean and S. Cameron. Path planning and collision avoidance for redundant manipulators in three dimensions. In *Int. Conf. Intelligent Autonomous Systems*, 1995.

[46] J. Reif. Complexity of the mover's problem and generalizations. In *Proc. 20th IEEE Symp. on Found. of Comp. Sci.*, pages 421–427, 1979.

[47] S. Rosenberg. *The Laplacian on a Riemannian Manifold.* Cambridge University Press, 1997.

[48] G.K. Schmidt and K. Azarm. Mobile robot path planning and execution based on a diffusion equation strategy. *Advanced Robotics*, 7(5):479–490, 1993.

[49] J.T. Schwartz and M. Sharir. On the 'piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4:298–351, 1983.

[50] J.M. Selig. *Geometrical Methods in Robotics.* Springer, 1996.

[51] K. Shoemake. Animating rotation with quaternion curves. *Computer Graphics*, 19(3), 1985.

[52] L. Tarassenko and A. Blake. Analogue computation of collision-free paths. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 1991.

[53] M.E. Taylor. *Noncommutative harmonic analysis.* American Mathematical Society, 1986.

[54] N.C. Tsourveloudis, K.P. Valavanis, and T. Hebert. Autonomous vehicle navigation utilizing electrostatic potential fields and fuzzy logic. *IEEE Tr. on Rob. & Aut.*, 17(4), 2001.

[55] Y. Wang and G.S. Chirikjian. A new potential fiel method for robot path planning. In *Proc. IEEE Int. Conf. on Rob. & Aut.*, 2000.