

# RIGID BODY DYNAMICS SIMULATION FOR ROBOT MOTION PLANNING

THÈSE N° 3663 (2006)

PRÉSENTÉE LE 17 NOVEMBRE 2006

À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR

Laboratoire de systèmes robotiques 1

PROGRAMME DOCTORAL EN SYSTÈMES DE PRODUCTION ET ROBOTIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

**Alan ETTLIN**

ingénieur informaticien diplômé EPF  
de nationalité suisse et originaire de Kerns (OW)

acceptée sur proposition du jury:

Prof. A. Billard, présidente du jury  
Prof. H. Bleuler, directeur de thèse  
Prof. G. A. de Paula Caurin, rapporteur  
Prof. D. Thalmann, rapporteur  
Dr R. Vuillemin, rapporteur



ÉCOLE POLYTECHNIQUE  
FÉDÉRALE DE LAUSANNE

Suisse  
2006



## ABSTRACT

The development of robot motion planning algorithms is inherently a challenging task. This is more than ever true when the latest trends in motion planning are considered. Some motion planners can deal with kinematic and dynamic constraints induced by the mechanical structure of the robot. Another class of motion planners fulfill various types of optimality conditions, yet others include means of dealing with uncertainty about the robot and its environment. Sensor-based motion planners gather information typically afflicted with errors about a partially known environment in order to plan a trajectory therein. In another research area it is investigated how multiple robots best cooperate to solve a common task.

In order to deal with the complexity of developing motion planning algorithms, it is proposed in this document to resort to a simulation environment. The advantages of doing so are outlined and a system named Ibex presented which is well suited to support motion planner development. The developed framework makes use of rigid body dynamics algorithms as simulation kernel. Further, various components are included which integrate the simulation into existing engineering environments. Simulation content can be conveniently developed through extensions of well-established 3D modelling tools. The co-simulation with components from other domains of physics is provided by the integration into a leading dynamic modelling environment. Robotic actuator models can be combined with a rigid body dynamics simulation using this mechanism. The same configuration also allows to conveniently develop control algorithms for a rigid body dynamics setup and offers powerful tools for handling and analysing simulation data. The developed simulation framework also offers physics-based models for simulating various sensors, most prominently a model for sensor types based on wave propagation, such as laser range finding devices.

Application examples of the simulation framework are presented from the mobile robotics rough-terrain motion planning domain. Three novel rough-terrain planning algorithms are presented which are extensions of known approaches. To quantify the navigational difficulty on rough terrain, a new generic measure named “obstacleness” is proposed which forms the basis of the proposed algorithms.

The first algorithm is based on Randomised Potential Field Planners (RPP) and consequently is a local algorithm. The second proposed planner extends

RRT<sub>connect</sub>, a bi-directional Rapidly Exploring Random Tree (RRT) algorithm and biases exploration of the search space towards easily traversable regions. The third planner is an extension of the second approach and uses the same heuristic to grow a series of additional local RRTs. This allows it to plan trajectories through complex distributions of navigational difficulty benefitting from easy regions throughout the motion plan.

A complete example is shown in which the proposed algorithms form the basis for sensor-based dynamic re-planning simulated in the presented framework. In the scenario, a simulated planetary rover navigates a long distance over rough terrain while gathering sensor data about the terrain topography. Where obstacles are sensed which interfere with the original motion plan, dynamic re-planning routines are applied to circumnavigate the hindrances.

In the course of this document a complete simulation environment is presented by means of a theoretical background and application examples which can significantly support the development of robot motion planning algorithms. The framework is capable of simulating setups which fulfil the requirements posed by state-of-the-art motion planning algorithm development.

## KEYWORDS

**rigid body dynamics, robot motion planning, rough-terrain navigation,  
sensor simulation, simulation content tool-chain**

## ZUSAMMENFASSUNG

Die Entwicklung von Pfadplanungsalgorithmen für Roboter ist an sich schon eine schwierige Aufgabe. Dies ist erst recht wahr, wenn die neuesten Entwicklungen in der Pfadplanung berücksichtigt werden. Einige Pfadplaner können mit kinematischen und dynamische Einschränkungen umgehen, die von der mechanischen Struktur des Roboters erzeugt werden. Eine andere Kategorie von Pfadplanern erfüllt diverse Optimalitätskriterien, wieder andere Planer beinhalten Methoden um mit Ungewissheiten über den Roboter sowie dessen Umgebung umgehen zu können. Sensorbasierte Pfadplaner erfassen typischerweise fehlerbehaftete Daten über eine teilweise bekannte Umgebung um darin eine Trajektorie zu errechnen. In einem anderen Forschungsfeld wird untersucht wie mehrere Roboter am Besten miteinander kooperieren können um eine gemeinsame Aufgabe zu erfüllen.

Um die Komplexität in der Entwicklung von Pfadplanungsalgorithmen besser handhaben zu können wird in diesem Dokument vorgeschlagen auf eine Simulationsumgebung zurückzugreifen. Die Vorteile eines solchen Vorgehens werden erläutert und ein System namens Ibex vorgestellt, welches gut geeignet ist um die Pfadplanungsalgorithmenentwicklung zu unterstützen. Die entwickelte Simulationsumgebung verwendet Starrkörperdynamikalgorithmen als Simulationskern. Ferner sind weitere Komponenten entwickelt worden, welche die Simulation in bestehende Entwicklungsumgebungen integrieren. Das Erstellen von Simulationen wird mittels Erweiterungen von etablierten 3D Modellierungswerkzeugen ermöglicht. Die Ko-simulation mit Komponenten aus anderen Gebieten der Physik wird durch die Integration in eine führende Entwicklungsumgebung zum erstellen dynamischer Modelle sichergestellt. Modelle von Robotik-Aktoren können auf diese weise einfach mit einer Starrkörperdynamiksimulation kombiniert werden. Dieselbe Konfiguration bietet auch eine mächtige Möglichkeit Regelalgorithmen für einen Starrkörperdynamikaufbau zu entwickeln sowie die anfallenden Simulationsdaten zu handhaben und auszuwerten. Die entwickelte Simulationsumgebung bietet ebenfalls physikbasierte Modelle zur Simulation verschiedener Sensortypen an, z.B. ein Modell für Sensoren welche auf Wellenausbreitung basieren, wie etwa ein Laser-Distanzmessgerät.

Es werden Anwendungsbeispiele der Simulationsumgebung aus dem Gebiet der mobilen Roboter-Pfadplanung auf unebenem Gelände präsentiert. Drei neuarti-

ge Pfadplanungsalgorithmen zur Navigation auf unebenem Gelände werden vorgestellt, die Erweiterungen von existierenden Ansätzen sind. Zur Quantifizierung der Navigationsschwierigkeit auf unebenem Gelände wird ein neues generisches Mass namens "Obstacleness" vorgestellt welches die Basis für die vorgeschlagenen Algorithmen bildet.

Der erste Algorithmus basiert auf dem "Randomised Potential Field Planner" (RPP) Ansatz und ist daher ein lokaler Algorithmus. Der zweite vorgeschlagene Pfadplaner erweitert  $RRT_{connect}$ , ein bidirektionaler "Rapidly Exploring Random Tree" (RRT) Algorithmus, durch eine gewichtete Erkundung des Suchraums zugunsten von einfach traversierbaren Regionen. Der dritte Planer ist eine Erweiterung des Zweiten und verwendet dieselbe Heuristik um eine Anzahl zusätzlicher, lokaler RRT-Bäume zu erstellen. Dies ermöglicht es ihm, Trajektorien durch komplexe Verteilungen der Navigationsschwierigkeit zu errechnen und dabei Vorteile aus einfachen Regionen entlang des gesamten Pfades zu ziehen.

Es wird ein komplettes Beispiel gezeigt, in welchem die vorgeschlagenen Algorithmen die Basis für sensor-basierte dynamische Neuplanung von Trajektorien in der Simulationsumgebung bilden. Im gezeigten Szenario legt ein simulierter geländegängiger Roboter eine lange Strecke auf einer unebenen planetarischen Oberfläche zurück und sammelt fortwährend Sensordaten über die Geländetopographie. Detektierte Hindernisse welche das Verfolgen der ursprünglichen Trajektorie verunmöglichen werden mittels dynamischer Neuplanung umfahren.

Im Verlauf dieses Dokuments wird anhand einer theoretischen Einführung sowie von Anwendungsbeispielen eine komplette Simulationsumgebung vorgestellt welche die Entwicklung von Pfadplanungsalgorithmen für Roboter bedeutend unterstützen kann. Die Simulationsumgebung kann Szenen simulieren welche den Anforderungen der Entwicklung aktueller Pfadplanungsalgorithmen genügen.

## SCHLÜSSELWÖRTER

**Starrkörperdynamik, Roboter-Pfadplanung, Navigation auf unebenem Gelände, Sensorsimulation, Entwicklungsumgebung für Simulationsinhalt**

## ACKNOWLEDGEMENTS

I would like to thank my advisor, Hannes Bleuler for having supervised this thesis and for having hosted me as an external PhD student at the Robotics Systems Laboratory 1. Without him, all of this would not have been possible.

Further, I would like to express my gratitude to Ronald Vuillemin, my boss and mentor at the University of Applied Sciences of Central Switzerland. He has always given me an exceptionally high degree of freedom and responsibility with all the projects during this time while giving me the occasional crucial nudge in the right direction. This liberty has proven to be an immensely valuable experience for me; I believe I have learnt a lot more this way than would ever have been possible in a conventional setting.

During my thesis I have enjoyed the hospitality of Glauco Caurin at the Mechatronics Laboratory of the Escola de Engenharia de São Carlos in Brazil. I would like to thank him for this most memorable and instructive time as well as for accepting to be a member of my thesis committee.

The remaining member of the committee is Daniel Thalmann and I would like to thank him together with the other examiners for carefully reading my thesis and providing me with constructive feedback.

Thanks also to all my colleagues in Horw. In particular I would like to thank Soufiane Aiterrami for supporting this work with his great CAD models. I would also like to thank Tom Graf who has been like an additional advisor to me. He has always been ready to help me with some physics-related problems and above all his experienced advice beyond purely scientific work.

Patrick Büchler who started this journey together with me has always been a good friend. I would like to thank him for the great time we have spent together and the helpful ideas on this work. Thanks go as well to my good friend Marco Lüthi with whom I've enjoyed some valuable discussions on this work, apart from many other great moments, of course.

My special thanks go to Mum and Dad for always having lovingly encouraged and supported me all the time. I would like to take this opportunity to also thank them for having given me the great gift of languages.

Finally, my thanks go to my love Dani. Thank you for making my life so wonderful - may it forever be this way!



# CONTENTS

1. <i>Introduction</i> . . . . .	1
2. <i>Introduction to Motion Planning</i> . . . . .	7
2.1 Previous Work . . . . .	8
2.1.1 Kinematic Constraints . . . . .	10
2.1.2 Kinodynamic Motion Planning . . . . .	12
2.1.3 Inclusion of Sensor Information and Uncertainty . . . . .	13
2.1.4 Multiple Robots . . . . .	15
2.2 System Level Interactions . . . . .	17
2.3 Motivation of Simulation for Motion Planning Development . . . . .	20
2.3.1 Environment Modelling . . . . .	20
2.3.2 Control of Information . . . . .	21
2.3.3 Management of Experiments . . . . .	21
3. <i>Simulation Environment for Motion Planning Algorithm Development</i> . . . . .	23
3.1 Desired Functionality . . . . .	23
3.2 Related Work . . . . .	26
3.3 Simulation Framework for Algorithm Prototyping . . . . .	29
3.3.1 Overview . . . . .	29
3.3.2 Rigid Body Dynamics . . . . .	33
3.3.3 Terrain Model . . . . .	54
3.3.4 Sensor Simulation . . . . .	60
3.3.5 Simulation Content Tool-Chain . . . . .	67
3.3.6 Integration into Industry-Standard Dynamic Modelling Environment . . . . .	73
3.4 Validations and Performance . . . . .	77
3.4.1 Validation of Simulations . . . . .	77
3.4.2 Performance Analysis . . . . .	91
3.4.3 Limitations of Simulation Approach . . . . .	98

4. <i>Robot Motion Planning</i> . . . . .	101
4.1 All-Terrain Motion Planning . . . . .	102
4.1.1 The “Degree of Obstacleness” . . . . .	104
4.1.2 Randomised Potential Field (RPP) Approach . . . . .	113
4.1.3 Rapidly-Exploring Random Trees (RRT) Approach . . . . .	124
4.1.4 Performance Comparison . . . . .	138
4.1.5 Trajectory Following . . . . .	143
4.2 Dynamic Environments . . . . .	147
4.2.1 Scenario Description . . . . .	147
4.2.2 Integration of Sensor Information . . . . .	148
4.2.3 Dynamic Re-Planning . . . . .	151
4.3 Results . . . . .	155
5. <i>Conclusion</i> . . . . .	159
6. <i>Outlook</i> . . . . .	165
 <i>Appendix</i> . . . . .	 169
A. <i>Definitions</i> . . . . .	171
A.1 List of Symbols . . . . .	171
A.2 List of Abbreviations . . . . .	173
 <i>Bibliography</i> . . . . .	 175

## LIST OF TABLES

3.1	Unconstrained rigid-body dynamics algorithm outline . . . . .	38
4.1	Performance of motion planning algorithms: experiment 1 . . . . .	139
4.2	Performance of motion planning algorithms: experiment 2 . . . . .	140
4.3	Performance of motion planning algorithms: experiment 3 . . . . .	141
4.4	Performance of motion planning algorithms: experiment 4 . . . . .	142
A.1	List of symbols . . . . .	172
A.2	List of abbreviations . . . . .	173



## LIST OF FIGURES

1.1	Mars Exploration Rover - MER . . . . .	2
1.2	ExoMars rover . . . . .	3
2.1	System-level interactions . . . . .	18
3.1	High-level abstract architecture of the simulation framework. . . . .	30
3.2	Content-based simulation framework architecture . . . . .	33
3.3	Relationship between local and global coordinate systems . . . . .	34
3.4	Illustration of angular velocity . . . . .	36
3.5	Types of rigid body contact . . . . .	40
3.6	Bounding volume approximations . . . . .	41
3.7	Four common joint types . . . . .	49
3.8	Joint simplification . . . . .	50
3.9	Geometry approximation example . . . . .	51
3.10	Visual vs. physical representations . . . . .	52
3.11	MicroDelta: real robot and visualisation . . . . .	53
3.12	Mechanical details of MicroDelta robot . . . . .	53
3.13	Terrain triangulation . . . . .	57
3.14	Terrain rigid body visualisation . . . . .	57
3.15	Rock-fall application gallery mesh . . . . .	58
3.16	Rock-fall application Matterhorn example . . . . .	59
3.17	Discrete distance sensor simulation principle . . . . .	61
3.18	Discrete distance sensor on a mobile robot . . . . .	62
3.19	Ray-casting principle . . . . .	63
3.20	Physics-based ray-casting sensor model . . . . .	65
3.21	Range-finding sensor test environment . . . . .	66
3.22	Range-finding sensor map building example . . . . .	66
3.23	Range-finding sensor 3D geometry reconstruction . . . . .	67
3.24	Tool-chain data flow paths . . . . .	68
3.25	Autodesk <sup>®</sup> Maya <sup>®</sup> to Ibex exporter . . . . .	70
3.26	SolidWorks <sup>®</sup> to Ibex exporter . . . . .	72
3.27	Ibex-Simulink model of map drawing example . . . . .	75
3.28	Plot of rigid body penetration on collision. . . . .	79
3.29	Plots of rigid body collision validation. . . . .	80
3.30	Plots of kinetic friction validation. . . . .	82

---

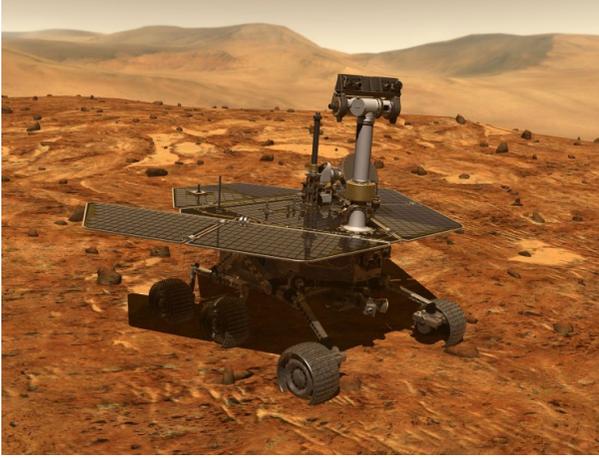
3.31	Kinetic friction interpolation and correction. . . . .	83
3.32	Plots of static friction validation. . . . .	85
3.33	Plots of rolling validation: sphere . . . . .	88
3.34	Plots of rolling validation: cylinder . . . . .	89
3.35	Plots of rolling validation: capsule . . . . .	90
3.36	Rolling validation: rolling boundary . . . . .	91
3.37	Performance plots: dependency on time-step size . . . . .	92
3.38	Performance plots: dependency on visualisation resolution . . . . .	94
3.39	Performance plots: dependency on object count . . . . .	95
3.40	Performance plots: dependency on object complexity . . . . .	96
3.41	Complete setup of medieval clockwork . . . . .	97
3.42	Setup illustrating chaotic behaviour . . . . .	100
4.1	Sample terrain - visualisation . . . . .	107
4.2	Sample terrain - inclination obstacleness component . . . . .	107
4.3	Tracked vehicle . . . . .	108
4.4	Tracked vehicle - track detail . . . . .	108
4.5	Lateral obstacleness component coefficient . . . . .	109
4.6	Sample terrain - lateral inclination obstacleness component . . . . .	110
4.7	Illustration of lateral inclination angle . . . . .	118
4.8	Obstacleness computation example . . . . .	119
4.9	RPP <sub>obst</sub> algorithm in operation . . . . .	121
4.10	RPP planner local minima problem . . . . .	122
4.11	RPP <sub>obst</sub> planner multiple runs . . . . .	123
4.12	The RRT algorithm . . . . .	125
4.13	Extend function of RRT algorithm . . . . .	126
4.14	The RRT <sub>obst</sub> algorithm . . . . .	128
4.15	RRT <sub>obst</sub> algorithm in operation . . . . .	129
4.16	RRT <sub>obst</sub> planner operating on “RPP local minimum” scenario . . . . .	131
4.17	The RRT <sub>obst way</sub> algorithm . . . . .	133
4.18	RRT <sub>obst way</sub> algorithm in operation . . . . .	134
4.19	RRT <sub>obst way</sub> planner operating on “RPP local minimum” scenario . . . . .	136
4.20	RRT <sub>obst way</sub> trajectory post-processing . . . . .	137
4.21	Planetary rover design used for rough-terrain simulations . . . . .	143
4.22	Path smoothing scheme . . . . .	144
4.23	Trajectory tracking scheme 1 . . . . .	145
4.24	Trajectory tracking scheme 2 . . . . .	145
4.25	Trajectory tracking example . . . . .	146
4.26	Physical model of rover and sensor array . . . . .	148
4.27	Rover sensor values and terrain reconstruction . . . . .	149
4.28	Detection of deviations from expected terrain . . . . .	151
4.29	Dynamic re-planning example . . . . .	153
4.30	Dynamic re-planning example - continued . . . . .	154
6.1	A long way to go. . . . .	168

# 1. INTRODUCTION

This thesis is concerned with robot motion planning. Simply put, robot motion planning deals with how to tell a robot how it should move from point  $A$  to point  $B$ . This seemingly simple task (after all, humans deal with it as a matter of routine in their everyday lives) turns out to be surprisingly hard to solve for a robot. Motion planning has been studied for more than 30 years and still remains an active field of research.

In the beginnings, motion planning was a purely geometrical problem. This is illustrated by the name given to the task in some early literature: “the piano movers’ problem”. The task consisted of finding a way of moving a piano from one room in a house to another without hitting anything. While moving the piano, it was possible to turn it however required to fit past obstacles. This simplifying assumption ignores the dynamics of the piano and, more generally speaking, any differential constraints present in the system. Further, time is only implicitly considered in the motion plan by specifying the sequence of positions and orientations the piano assumes during its progress. No information is given as to how fast the progress is to be along the computed trajectory.

The simplifications assumed in the piano movers’ problem have been increasingly dropped in more recent planning approaches. Some of the extensions cover systems with *kinematic constraints*, e.g. occurring in car-like steering mechanisms. While the piano could be translated and rotated freely, a car can only move forwards and backwards as well as turn following an arc of lower-bounded radius. The *dynamic behaviour* of a system is explicitly considered in kinodynamic planning algorithms. A wide variety of *optimality measures* has been suggested which are fulfilled by pertinent planners: minimisation of traversal time or energy consumption, the adhesion to some safety measure or to ensure line-of-sight contact with some landmark just to name a few. Other research studies the *inclusion of uncertainty* into motion planners. Uncertainty can arise either from unknown characteristics of the robot and its environment or in sensor-based motion planning. *Sensor-based* planners gather data about the only partially known environment they are navigating in and plan a trajectory with incomplete information. Yet other research studies the interactions of *multiple robots* which are to be coordinated for some common task.



*Fig. 1.1:* Artist's impression of a Mars Exploration Rover operating autonomously on rough terrain. Image courtesy NASA/JPL-Caltech.

Motion planning algorithms have also long crossed the border to other disciplines. Planning techniques originally devised for robotics have found successful applications in fields as diverse as the study of molecular folding, mechanical assembly tasks, computer graphics animation and the planning of minimally-invasive surgery.

With such a diversity of application domains and technical issues to keep track of it becomes apparent that the development of motion planning algorithms will benefit from any amount of support available. A classical supporting technique in engineering is the use of simulations. This is precisely what is studied in the present work: how the design of motion planners can be supported by rigid body dynamics simulations. In rigid body dynamics the interactions of non-deformable geometries are studied. Such a model of reality is well-suited to simulate many environments encountered in robot motion planning.

A paramount aspect of this work is that the simulation program is not studied in isolation but in the context of the requirements posed by the development process. This has led to the development of a complete framework centered on the rigid body dynamics simulation which is embedded in the established engineering work-flow.

The simulation framework is well-suited for a wide range of settings encountered in robot motion planning. Nevertheless, the examples used throughout this document are drawn from the mobile robotics domain and more precisely from rough-terrain motion planning. In rough-terrain planning, the task consists of computing a trajectory for a mobile robot navigating on a non-planar terrain, such as



*Fig. 1.2:* Artist's impression of the European Space Agency's ExoMars rover. Image courtesy ESA.

typically encountered in outdoor off-road settings or in planetary missions.

Planetary missions are the application example used in most of the simulations presented in this document to illustrate the simulation framework supporting motion planning algorithm development. The success of recent missions to Mars has illustrated the progress made in autonomous robotics and motion planning to the broader public. Figure 1.1 shows an artist's impression of the Mars Exploration Rover developed by NASA. At the time of writing, two such rovers, "Spirit" and "Opportunity" are successfully operating on the red planet long after their intended mission duration.

Rough-terrain motion planning is poised to become an increasingly important technology for autonomous planetary navigation and a whole variety of Earth-bound tasks. In planetary missions, the communication latency even increases the need for autonomous navigation. For the near future a series of further missions to Mars is planned by NASA. The European Space Agency ESA plans to launch their own mission including the ExoMars rover depicted in figure 1.2.

The rest of this document is structured as follows: in chapter 2, a general introduction to robot motion planning is given. In section 2.1, an outline of historic development in the domain illustrates the progress made and highlights issues which are of particular interest in current research. An overview of system-level inter-

actions given in section 2.2 shows how a motion planner is embedded within an overall robotic system and which other components it interacts with. In section 2.3, it is motivated how a simulation environment can support the development of motion planning algorithms.

Chapter 3 deals with the simulation framework which has been developed as part of the present research. First, general requirements for a simulation environment designed to support motion planning algorithm development are established in section 3.1 based on the findings of chapter 2. Other simulation programs with similar characteristics are put into relation with the proposed solution in section 3.2.

The design of the developed solution is presented in section 3.3. First, a high-level overview of the system is given, followed by an introduction to rigid body dynamics algorithms. This serves to give a better understanding of how the simulation operates and what kind of virtual experiments can be conducted with it. In rough-terrain motion planning, the interaction between the robot and the terrain assumes great importance. The terrain model used for this research is introduced in section 3.3.3. One important advantage of using a simulation to support motion planner development is the amount of control over data which can be exerted when using simulated sensors. The sensor simulations implemented in the proposed solution are presented in section 3.3.4. A major bottleneck when developing simulations is created by the lack of adequate tools to generate simulation content. Section 3.3.5 describes the tool-chain established to tackle the problem. Section 3.3.6 concludes the description of the simulation framework by describing a particularly powerful configuration thereof in which the rigid body dynamics routines operate as co-simulation within a well-established engineering environment, MATLAB<sup>®</sup> Simulink<sup>®</sup>.

In section 3.4, a series of tests is executed with the proposed simulation. A validation of the used rigid body dynamics library is given in section 3.4.1. Performance measurements are presented in section 3.4.2. The chapter is completed in section 3.4.3 by the discussion of some general limitations encountered when using simulations and rigid body dynamics simulations in particular.

Chapter 4 deals with the development of motion planning algorithms. In section 4.1, all-terrain motion planning algorithms are discussed as category including both planar and rough-terrain planners. The focus of this work lies on rough-terrain planners since a flat terrain can be considered a special case of a rugged one.

The “degree of obstacleness” is introduced in section 4.1.1 as generic measure for the navigational difficulty of traversing rough terrain based on its topographical and physical (material) characteristics. Using this definition, three novel rough-terrain motion planning algorithms are presented which extend known motion planners from other domains to become applicable to rough-terrain settings.

The first extension is performed with a Randomised Potential Field Planner (RPP) in section 4.1.2. The resulting RPP<sub>obst</sub> algorithm uses the obstacleness def-

initiation as basis for computing the repulsive potential used in potential-field motion planning. This allows it to deviate the locally computed trajectory to regions of low navigational difficulty. Like all RPP planners, the algorithm displays a tendency to get trapped in local minima of the potential function due to its local “greedy” behaviour.

In order to obtain more global planners, Rapidly-Exploring Random Trees (RRT) are extended in section 4.1.3. The first proposed algorithm called  $\text{RRT}_{\text{obst}}$  extends  $\text{RRT}_{\text{connect}}$ , a bi-directional variant of RRT. The algorithm biases the exploration of configuration space towards easily traversable regions. The approach produces good results in obstacleness topologies with one local maximum along the computed trajectory. If more maxima are present, regions of lower obstacleness in between are not optimally exploited.

The third proposed algorithm,  $\text{RRT}_{\text{obst way}}$ , is an extension of  $\text{RRT}_{\text{obst}}$  which grows additional local RRTs using the same heuristic. This allows to benefit from local obstacleness minima along the entire motion plan in complex distributions of navigational difficulty. Therefore, of the discussed algorithms it produces the best results when navigating on complex terrains. A comparison of the proposed algorithms with existing solutions on scenarios of varying complexity is presented in section 4.1.4.

To link the motion planning algorithms with the simulation environment, a trajectory tracking module needs to be added. The simple solution implemented in this work is introduced in section 4.1.5. The proposed tracking scheme allows e.g. a fictional planetary rover to approximately follow the computed motion plans on the terrain in the simulation environment.

Section 4.2 studies how dynamic environments can be simulated using the developed simulation. Dynamic environments in this context refers to settings which include obstacles not known to the motion planner, as described in section 4.2.1. For the motion planner to deal with the presence of uncharted obstacles, it must have a means of gathering information about the surroundings of the robot. How sensor data can be integrated into a motion planner in the simulation environment is illustrated in section 4.2.2. Given raw sensor data, the control logic of the robot must generate a representation of the environment based on the currently available information. Such a representation allows the motion planner to take evasive action if an obstacle is detected which affects the previously planned trajectory. An example of this process implemented in the simulator is given in section 4.2.3. The results of the motion planning related work presented throughout chapter 4 are briefly summarised in section 4.3.

A conclusion of the work described in this document is drawn up in chapter 5. The research described includes aspects from a wide range of different fields, resulting in a large number of possible future extensions. Some promising directions for future research are discussed as conclusion of this thesis in chapter 6.



## 2. INTRODUCTION TO MOTION PLANNING

Robot motion planning is described as “A Journey of Robots, Molecules, Digital Actors, and Other Artifacts” in [Latombe 99], a survey paper by J.-C. Latombe, a researcher who has significantly influenced the field during the last decades. This description already summarises the wide applicability of motion planning algorithms and the progress which has been made in the field.

In its simplest form, robot motion planning attempts to solve the task of finding a collision-free path for a rigid robot amidst clearly defined static rigid objects. This *generalised piano movers’ problem* was described in the seminal paper [Schwartz 83a].

Already this basic task is computationally hard to solve as soon as the number of degrees of freedom increases above a few. Nowadays, motion planners are available which solve complex practical problems for robots with many degrees of freedom operating in cluttered dynamic environments. The inherent complexity of the task has been attacked using powerful heuristics and specifically randomised algorithms. Also, the research field has diversified to include a number of sub-domains. Extensions of the basic problem include algorithms which consider moving obstacles and operate under kinematic and dynamic constraints limiting robot motions. Some planners deal with complex structures of the robot including articulated mechanisms or deformable geometries, or robots which are capable of reconfiguring their modular structure during the planning process. Other planners emphasise on computing optimal trajectories given some performance measure. Multiple robots can be coordinated by dedicated algorithms or motion plans computed based on incomplete and error-afflicted sensor information.

Some motion planners specifically deal with navigation on rough terrain, as encountered in planetary missions or off-road settings outdoors. Such environments pose a number of challenges not encountered in classical motion planning. The examples of motion planning algorithms given in chapter 4 are drawn from the rough-terrain planning domain.

Increasingly, planners are emerging which also take into account multiple of the above mentioned issues. Planners are also becoming increasingly practical in the sense that they are successfully applied to real-world systems for non-trivial tasks where robust behaviour is crucial.

Robot motion planning is by no means restricted to applications in classical robotics and has found its application in domains as diverse as surgery [Tombropoulos 99], mechanical assembly [Chang 95], computer graphics animation [Kallmann 03a] or computational biology [Singh 99], just to name a few.

The survey presented in the following section is not intended to be a complete summary of all robot motion planning research but gives an outline of the development in the field from the beginnings to present-day topics of interest. The category of rough-terrain motion planning algorithms is studied in more detail in section 4.1 and is not explicitly included in this summary. Nevertheless various algorithms presented here in another context are drawn from rough-terrain planning.

Some ambiguity exists in literature about the terms *path planning* and *motion planning*. In some terminologies, motion planning refers to the extensions of the purely geometric path planning task. The extensions include the computation of collision-free paths under dynamic constraints, using sensor information, cooperative path planning etc.. In other usage, the term motion planning refers to the basic task whereas “trajectory planning” describes the extended variants. In this document “motion planning” and “path planning” are used synonymously and the distinction made explicit where relevant. Similarly, the terms “path” and “trajectory” are used interchangeably where not explicitly stated otherwise.

## 2.1 Previous Work

First robot motion planning algorithms emerged in the late 1960s, e.g. the *visibility graph* technique [Nilsson 69] which uses a graph search to find the shortest path past polygonal obstacles for a robot represented by a point.

In 1979, the concept of *reducing* a (polygonal) robot to a point was formalised [Lozano-Pérez 79]. This notion led to the development of the key concept of *configuration space*, [Lozano-Pérez 83]. The configuration space (typically denoted by  $\mathcal{C}$ ) is a parameter space which represents all degrees of freedom of a robot. In such a space, the robot can be represented as single point; its coordinates define the configuration of the robot. Obstacles in the workspace of the robot are translated into obstacles in configuration space. The motion planning task is hence reduced to finding a path through the subspace of  $\mathcal{C}$  not occupied by configuration space obstacles (called free space).

A complete path planner is an algorithm which always finds a solution to the passed path planning problem if one exists or indicates that it does not exist otherwise. Such path planners have proven to be computationally forbidding for non-trivial tasks, leading to the development of heuristic path planners.

Cell decomposition algorithms rely on decomposing the free space into “cells”, such that a path between any two configurations in a cell can easily be generated. A non-directed graph (connectivity graph) between neighbouring cells is constructed

and searched. In *approximate cell decomposition* [Brooks 83], cells of some pre-defined shape (e.g. rectangloids) are used such that the union of all cells is strictly included in free space. *Exact cell decomposition* algorithms subdivide the entire free space into cells of different geometries, e.g. trapezoids for polygonal obstacles. The connectivity graph links cells and not individual configurations. An advantage thereof is that some freedom remains in the found *channel* (sequence of cells to traverse) to compute a path which e.g. accommodates dynamic constraints. Approximate cell decomposition is resolution-complete<sup>1</sup> while exact cell decomposition is a complete algorithm.

Another heuristic approach to path planning makes use of artificial potential fields [Khatib 86]. Although potential field methods were originally designed for on-line collision avoidance they can be extended to perform global motion planning tasks by combining them with graph searching techniques [Barraquand 91]. In potential field methods, the obstacles generate an artificial repulsive potential and the goal configuration an attractive potential. The superposition of both potentials is used to compute artificial forces acting on the robot. At every configuration the resulting force is considered the most promising direction of motion.

Potential field algorithms are local methods since only the neighbourhood of the current configuration is considered when computing the forces. This makes the potential field approach well-suited to operate in unknown or uncertain environments using sensor data. Potential field approaches can be very efficient compared to other methods but since they use fastest descent optimisation, they can get trapped in local minima. Either potential functions without local minima (*navigation functions*) have to be designed or mechanisms to escape from local minima used.

One successful approach to escaping local minima was introduced with *randomised potential field planners* (RPP) [Barraquand 91]. RPP algorithms perform a series of random motions when a local minima is detected before proceeding with the steepest descent. An RPP planner forms the basis of one of the algorithms discussed and extended in section 4.1.2.

Another randomised path planning approach consists of building a *probabilistic roadmap* (PRM) [Kavraki 96]. The PRM algorithm connects random samples in the configuration space (*milestones*) using a local planner to build the roadmap. Every sample and local path needs to be checked for collisions with obstacles before being introduced into the data structure. Using a collision checker has the advantage of avoiding the costly explicit representation of free space. PRM planners have been successfully applied to a wide range of problems, also including high numbers of degrees of freedom.

Both RPP and PRM belong to the category of *sampling-based* motion planners. Sampling-based algorithms do not explicitly represent the obstacles but rather rely on being able to determine for any desired configuration whether it lies in an obsta-

---

<sup>1</sup> Depending on the discretisation resolution, the algorithm is complete.

cle or in free space. Sometimes instead of this binary criterion, the distance to the nearest obstacle is computed<sup>2</sup>.

Sampling-based planners inherently are susceptible to the so-called *narrow-passage* problem [Hsu 98], [Hsu 99]. Difficulties arise with uniform sampling techniques, when different subsets of free space are separated by narrow passages. When connecting samples, e.g. with a local planner, connections between the separated regions are hard to achieve. Basic algorithms require a prohibitive number of random samples to connect free space components separated by narrow passages. Various measures for the difficulty of computing a good connectivity graph for a given space have been developed. Also, heuristic techniques have been proposed to tackle the problem, e.g. the addition of extra samples in difficult regions [Kavraki 96] or using a sampling strategy biased towards areas near the boundary of free space [Amato 96].

### 2.1.1 Kinematic Constraints

Various kinematic constraints can apply to the motion planning scenario which influence the way in which a motion plan can be computed. In the following a short categorisation of kinematic constraints is given together with the required modifications of motion planning techniques.

*Holonomic* constraints are constraints on the parameters composing the configuration space of the robot. Holonomic equality constraints are equality relations among the parameters which can be solved for one of them. By doing so, that parameter can be eliminated and henceforth a subspace of  $\mathcal{C}$  used which has one dimension less. Multiple independent holonomic equality constraints can apply which reduce the dimensionality of the resulting configuration space accordingly. Additionally, the constraints can be time-dependent. Motion planning under holonomic equality constraints is fundamentally the same problem as without, albeit in a different configuration space. Holonomic equality constraints arise for example from articulations (joints) in the robot structure.

Holonomic inequality constraints are inequality relations between the parameters of the configuration space (and possibly time). These constraints define a subspace of  $\mathcal{C}$  which has the original dimensionality. Holonomic inequality constraints typically correspond to obstacles or e.g. mechanical stops.

*Non-holonomic* constraints are constraints on the parameters composing the configuration space which include both the parameters and their time derivatives in a way that the time derivatives cannot be eliminated. Non-holonomic constraints do not reduce the dimensionality of the configuration space but rather the dimension of the space of possible differential motions (i.e. the space of achievable ve-

---

<sup>2</sup> Such distance computations represent a common algorithmic foundation of motion planning and rigid body dynamics, the two primary domains of this text.

locity directions) at any configuration. A typical case for non-holonomic equality constraints arises from car-like steering mechanisms (without sliding effects). The configuration space of such a vehicle on a plane is three-dimensional: a two-dimensional Cartesian position and the orientation. The velocity of the midpoint between the (non-steering) rear wheels is always tangent to the vehicle orientation. This eliminates one degree of freedom from the space of achievable velocities.

Non-holonomic inequality constraints usually restrict the set of achievable velocities of the robot without reducing its dimensionality. In the car-like steering example, mechanical stops limit the steering angle of the front wheels. Consequently, the space of achievable velocities at any configuration is further restricted to a two-dimensional cone around the neutral position.

Non-holonomic path planning considers systems where non-holonomic constraints apply to the robot. The solution path must both be collision free and consist of motions which are executable by the robot. Such paths are called *feasible* paths.

For *locally controllable*<sup>3</sup> robots, e.g. [Barraquand 93], the existence of a free path is equivalent to the existence of a feasible path since the free path can be approximated by an arbitrarily close feasible path. This property forms the basis of a family of algorithms which decompose the computation of a feasible path into two phases. First a free path is computed for a holonomic robot geometrically equivalent to the non-holonomic one studied. Next, a feasible path is derived considering the non-holonomic constraints, knowing that such a path exists, e.g. [Laumond 94].

A heuristic brute-force approach to non-holonomic motion planning is presented in [Barraquand 93]. The approach consists of concurrently building and searching a tree whose nodes represent axis-aligned cells in the configuration space. Starting with the initial configuration, new nodes are added by computing new configurations based on discrete control values applied to the configuration of the node being expanded. The approach does not require local controllability but in the worst case an exhaustive search of the discretised configuration space is performed. In [Ferbach 98], the system is extended to handle higher-dimensional configuration spaces by first computing a free path and then progressively introducing the non-holonomic constraints. The non-holonomic constraints are iteratively enforced by searching for a path in the neighbourhood of the existing one which satisfies a more complete set of constraints.

Probabilistic roadmaps have also been applied to motion planning with non-holonomic constraints, e.g. [Svestka 95]. The focus of attention lies on the local planner which must enforce the non-holonomic constraints. The computational complexity of the local planner is also of importance since it is invoked not only for computing a single-shot path but a graph covering the entire configuration space.

---

<sup>3</sup> A non-holonomic robot is said to be *locally controllable at a configuration*  $\mathbf{q} \in \mathcal{C}$  if and only if, given non-zero manoeuvring space, some motion in all directions can be achieved. The robot is said to be *locally controllable* if the above condition holds true for every  $\mathbf{q} \in \mathcal{C}$

On the other hand, once this roadmap has been computed, multiple queries can be efficiently retrieved. The two-level approach discussed above has also been applied to non-holonomic PRM planners [Sekhavat 98] and extended to comprise multiple levels of refinement.

### 2.1.2 Kinodynamic Motion Planning

Kinodynamic motion planning [Donald 93] deals with situations where not only kinematic constraints but also dynamic constraints (such as bounds on velocities, accelerations, forces and torques) are considered.

Traditionally, path planning has often been decoupled from the control engineering task of following the computed trajectory satisfying system dynamics. In the first phase of such *two-stage approaches*, a purely kinematic free path is computed which is converted to a trajectory in the second stage. The trajectory in this context is a time-parametrised path which can be executed by the dynamic system, ideally in a time-optimal manner.

A two-stage approach for robot manipulators is presented in [Shin 84] which can be applied to all robotic systems with known dynamic equations. A similar approach is presented in [Bobrow 85], where the resulting multi-dimensional optimal control problem is transformed into a one-dimensional one by parametrising the progress along the path with a single variable, e.g. the distance. Further developments of the basic idea avoid linearising the equations of motion : [Shiller 90b], [Shiller 91a]. Difficulties with this two-stage planning approach are that the results of the kinematic planner may not be executable by the robot due to limitations of the forces and torques available from the actuators. Also, global optimality cannot be guaranteed since only the optimal solution following the original kinematic path (or its homotopic<sup>4</sup> class) is computed.

A second broad class of kinodynamic planning algorithms uses a state space<sup>5</sup> formulation instead of the classical configuration space. State spaces include the time derivatives of the configuration parameters. A  $n$ -dimensional kinodynamic planning problem is hence transformed into a  $2n$ -dimensional non-holonomic problem. Generally, finding a truly optimal solution is computationally hard and can lead to solutions which pass dangerously close to obstacles. This is undesirable in practice due to control imprecisions when executing the motion plan.

In [Donald 93], a graph is computed which represents the discretised state space of the robot and its connectivity considering the given dynamic constraints. An approximately optimal solution is presented for a mass-point in Euclidean space. A graph search is performed to determine the sequence of controls which moves

<sup>4</sup> Informally, two paths are homotopic, if they can be deformed onto one another while keeping the end-points fixed and not “leaping over any obstacles”.

<sup>5</sup> State space is also called *tangent bundle* or *phase space* in literature.

the robot from start to goal, optimising a cost function based on total time and a velocity-dependent obstacle-avoidance penalty. In [Donald 95], the algorithm has been adapted to robot manipulators. The run-time performance is optimised by non-uniform sampling of the state space in [Reif 97]. The incremental approach of [Ferbach 98] described above is also an example of this category of planner.

In [LaValle 99], *Rapidly Exploring Random Trees* (RRTs) operating in state space are presented as planning method particularly tailored for solving kinodynamic planning (While also being successfully applicable to standard, e.g. non-holonomic planning). The strength of the RRT approach lies in the good exploration abilities of the tree data structures being grown in configuration/state space during the execution. The expansion of the trees is strongly biased towards unexplored regions. This is achieved by selecting a node for expansion with a probability proportional to the area of its Voronoi region in the search space. RRT algorithms and in particular their bi-directional extensions are discussed and adapted to rough-terrain planning in section 4.1.3.

A variant of the PRM approach which includes kinodynamic constraints is presented in [Hsu 02]. The approach explicitly includes moving obstacles and operates in the  $\{state \times time\}$ -space. New milestones are generated by integrating randomly selected admissible controls over a short time from previous milestones. This procedure which replaces the local planner ensures kinodynamic constraints are satisfied.

### 2.1.3 Inclusion of Sensor Information and Uncertainty

*Sensor-based motion planning* and *planning under uncertainty* refer to a class of motion planning algorithms which operate on incomplete information of the environment. Often some rudimentary knowledge of the environment is known beforehand which is supplemented by sensor readings gathered while executing the motion plan. Many different forms of including uncertainty into the motion planning task exist, some of which are listed in the following.

Similarly to kinodynamic planning, a two-phase approach has also been suggested for the inclusion of uncertainty. First a plan is computed without taking into consideration uncertainty. This plan is adapted in a second phase to yield a robust motion plan, e.g. taking into account the propagation of uncertainty during the motion plan execution. One variant consists of conceptually applying the second stage while following the path. In such a case, the motion plan is complemented by additional actions targeted at reducing uncertainty where required. Examples of two-phase planning under uncertainty are [Lozano-Pérez 76] and [Taylor 88]. A two-phase approach for rough-terrain motion planning which includes uncertainty about the robot and terrain models, range-sensor data and path-following errors is presented in [Iagnemma 99].

*Pre-image back-chaining* algorithms [Lozano-Perez 84] explicitly take uncertainty into account during the initial planning process. A pre-image is a subset of free configuration space from which a goal region can be guaranteed to be reached. The algorithm consists of finding a series of such pre-images (and the associated commands) from the goal region backwards to a pre-image including the initial configuration.

In [Takeda 94], a “sensory uncertainty field” is defined to quantify the quality of the “sensed configuration” (e.g. the position estimate for mobile robots) at each configuration. Contrary to dead-reckoning techniques (such as odometry), the error in the sensed configuration is not accumulative and does not depend on the acquisition history. This fact motivated the definition of the static sensory uncertainty field. When planning a trajectory, the quality of expected sensor readings is considered together with obstacle avoidance.

Uncertainty in sensor information as well as control inaccuracies are considered in [Bouilly 95] for a circular mobile robot operating in a polygonal environment. Uncertainty is accumulated over the set of executed motion primitives. The future motions are planned considering both strategies to reach the goal and also to reduce the uncertainty through sensor measurements.

In [Haït 96], the inclusion of uncertainty in rough-terrain motion planning is described. Uncertainty in the terrain model is included by considering an error interval of the terrain when computing the validity of discrete “placements” (configurations)<sup>6</sup>. Control errors are considered by imposing the validity not only of the path but also of its surroundings.

An integration of sensor-based motion planning with kinodynamic planning is presented in [Lumelsky 95] and [Shkel 97]. Perfect sensors are assumed with a limited range. The algorithm is intended to be executed on-line in real-time. The examples given are for a point-mass navigating in a 2D environment. The key idea is to select the velocity at every time step, such that an emergency stop can be safely executed within the known, sensed area. The direction of motion is selected using the “VisBug” algorithm [Lumelsky 90] which essentially follows obstacle boundaries while circumnavigating them.

A different sensor-based motion planning aspect is focussed on in the *active sensing* domain. The underlying problem is to compute a motion strategy for optimal sensor data acquisition in some specific task. The *next best view problem* [Maver 93] studies how to maximise the amount of information added to a partially built model with the next sensor reading. The definition of *viewpoint entropy* [Vazquez 02] allows to quantify the amount of information visible in a 2D rendering of a 3D scene, an overview of alternative measures is also included.

In [Gonzalez-Baños 98] *autonomous observers* are defined as mobile robots which autonomously perform vision tasks. Example applications include building a

---

<sup>6</sup> Validity includes static stability with wheel-terrain contact and body-terrain collision avoidance.

3D model of an unknown environment or localising and tracking objects of interest. Motion strategies are discussed in which both motion obstructions and visibility obstructions are considered.

#### 2.1.4 Multiple Robots

Another extension of the basic path planning problem considers multiple robots operating on the same task. Many problems can only be solved by coordinating multiple robots, e.g. some of the vision tasks discussed in [Gonzalez-Baños 98] explicitly require multiple robots. Motion planning in the presence of moving obstacles (e.g. [Latombe 91]) can be considered a simpler case of multi-robot planning in which the trajectories of the “other robots” are predefined.

Motion planning for multiple robots can be categorised in *centralised* and *decoupled* approaches. Centralised multi-robot planners consider the separate robots as a single system and typically perform the planning in a combined configuration space. Decoupled planning algorithms generate paths for each robot with a high degree of independence and then consider interactions of the robots following their individual paths.

In theory, complete algorithms can be achieved with centralised planners. The main problem with the approach is the “curse of dimensionality” [Bellman 61] which makes the time-complexity of a complete planner exponential in the dimension of the combined configuration space [Svestka 98].

An exact cell decomposition algorithm for several disks in an environment with polygonal obstacles is described in [Schwartz 83b]. Randomised potential field planners (RPP) have been applied to multi-robot planning in [Barraquand 91], where an example of two 3-DOF robots in a 2D workspace with narrow passages is given. In [Barraquand 92], a variant of the RPP planner which solves tasks for multiple disk-shaped robots in narrow-passage environments is shown. One example operates in a 20-dimensional configuration space for 10 disk robots and includes many narrow passages. Potential-field methods are good examples of *reactive style planners*: real-time capable algorithms which can be used on-line, reacting to sensor inputs of the environment.

Decoupled planners on the other hand typically have difficulties with the robots obstructing one another. In the worst case this can lead to deadlocks, completely blocking the progress of the robots. In *prioritised planning* approaches [Erdmann 87], each robot is assigned a priority. Individual motion plans are computed for each robot in order of priority, taking into account static obstacles and previously planned robots.

*Path coordination* techniques [O’Donnell 89] first compute the paths of the two robots to be coordinated independently. Next, the trajectory coordination problem is considered a scheduling problem. For scheduling, a two-dimensional “coordi-

nation diagram” is build in which each Cartesian position represents the positions of the two robots along their parametrised paths. If along the paths the two robots would collide, those positions are forbidden in the coordination diagram. Scheduling now is reduced to finding a path in the coordination diagram which connects the points representing both robots at their initial and goal configurations, without passing through forbidden areas. The result is a velocity coordination of the two robots. A similar technique has been applied to robot manipulators e.g. in [Chang 94]. A path-velocity decomposition is also performed in [Guo 02], where the  $D^*$  algorithm [Stentz 94] is used to find a path in the coordination diagram. Motion plans for more than 100 robots are computed using bounding boxes for obstacles in the coordination diagram in [Siméon 02].

In [Liu 89], collisions are detected between independently planned paths of mobile robots. Where intersections occur, the paths are either locally or globally replanned with a graph searching algorithm.

A large number of mobile robots can be coordinated using *traffic rules*, such as the *plan merging paradigm* [Alami 95]. A number of robots operates independently in the same workspace and asynchronously receives navigation tasks to solve. Each robot computes its own motions such that they are compatible with the motion plans of all other robots, which are forbidden to be altered. No central global plan is maintained but robots exchange information about their present state and future actions. Transitive deadlock detection is also included, in which case a centralised multi-robot plan would be invoked.

Some more recent approaches attempt to achieve a compromise between decoupled and centralised planning. In decoupled planning, strong constraints are placed on the motions of individual robots before any interactions are considered. Centralised planning on the other hand imposes no such constraints, as interactions are considered integrally with generating the motion plans. This can also be seen as a tradeoff between high computational complexity (centralised planning) and the loss of completeness (decoupled planning).

A number of “simple robots” are conceptually combined to form a “composite robot” in [Svestka 98]. Each robot is a non-holonomic car-like vehicle for which a probabilistic roadmap (PRM) is designed in the (static) environment. Combining all such “simple roadmaps” results in a high-dimensional “super-graph” for the composite robot. For a given static environment such a super-graph only needs to be computed once. Thereafter multiple queries can be solved effectively. The solution is shown to be probabilistically complete for the composite robot. The constraints imposed on each robot restrict it not to a path but to an entire roadmap. One benefit of such an approach is that only self-collisions need to be avoided in the composite graph while all harder (e.g. non-holonomic) constraints are considered in the simple roadmaps.

In [LaValle 98b] a similar approach is presented with the additional inclusion

of a performance measure to be optimised. Importantly, a performance measure for each robot is considered since a joint measure might lead to bad results for some individual robots. It is shown how including configurations in the “simple roadmaps” of each robot where it cannot collide with any other robot (i.e. “private garages”) leads to completeness for the overall problem.

“Dynamic robot networks” are described in [Clark 03]. Such networks are dynamically established between robots whenever communication and sensing capabilities allow to do so. It is possible for the robots to share global environment information within a network. Also, the motions of the robots in a network are coordinated using a distributed planning approach. Each robot in a network computes a *centralised* motion plan for all robots in the network. These plans are broadcast and the best plan according to a predefined performance measure executed by all robots. By using a centralised planner, some degree of completeness can be achieved. But importantly, the costly algorithm only needs to be applied to the subset of the entire problem defined by the local network.

## 2.2 System Level Interactions

In any real-world setup, robot motion planning algorithms are integrated within a larger system. In this section a high-level overview of typical components in such a system is given. The main purpose of doing so is to illustrate the context in which motion planning algorithms operate as well as to introduce some terminology used in the remainder of this text.

In a fully autonomous robot, the motion planning algorithm is necessarily embedded on the robot. Alternatively, it is possible to externalise the complete control logic and use a communication link to continuously transmit individual motion commands to the robot. Various intermediate forms of autonomy (and hence control logic locations) can also be implemented. Figure 2.1 shows an interaction diagram of an overall robotic system including the environment of the robot. Solid arrows represent a flow of information between the involved entities. The focus of the diagram lies on the control logic, which is depicted outwith the robot entity - without implying such a “physical” location.

The *motion planner* itself takes an intermediate position between the *task logic* and the *trajectory tracker*. In this context, task logic refers to high-level decision-taking algorithms. For motion planning (in the simplest case) task logic defines the goal configuration to be reached. Based on the current task specification, the motion planner computes a trajectory to be followed. This may either be a global plan leading the whole way to the goal or a local path segment. Based on the trajectory which is valid at a given point in time, the trajectory tracker (also referred to as *path following* module) generates commands for the actuators of the robot to execute. In reality, the trajectory tracker might well be structured as a hierarchy of

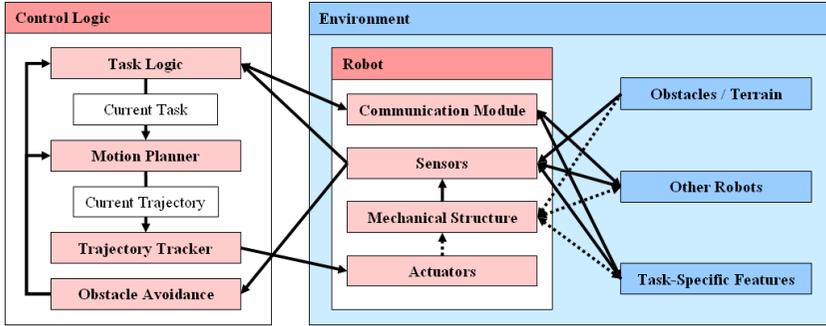


Fig. 2.1: Interactions of the motion planner embedded in the overall robot system, including the surrounding environment.

controllers, but is treated as entity in this high-level overview.

The control logic is completed by *obstacle avoidance* routines which notify the motion planner of obstacles that might influence the current motion plan. High-level task logic uses obstacle information within its decision-taking routines, e.g. if a goal is not attainable due to unexpected obstacles. Obstacle avoidance is treated as separate entity here to reflect the fact that such modules are only present in reactive sensor-based environments where incomplete knowledge is available at the time of initial path-planning.

Depending on the implemented system architecture, some of the interactions may vary, e.g. it is conceivable that the obstacle avoidance routines directly notify the trajectory tracker of a required emergency stop. Depending on the requirements, some control logic entities can also be merged or omitted altogether.

The robot itself is decomposed into four modules for this description. The *mechanical structure* of a robot determines the possible behaviour of the robot and represents the current configuration of the robot. *Actuators* are used to apply forces and torques to the mechanical structure, hence modifying the configuration of the robot. This “physical” type of interaction (as opposed to information flow) is represented by a dotted arrow in figure 2.1. Actuators are regarded as including the relevant controllers in this context.

During the operation of the robot, *Sensors* are used to acquire information about the robot as well as its environment. Within the robot, sensing is performed to determine the configuration, e.g. the position of some joint. Odometry sensors are an example of sensing performed on the robot structure to infer “external” configuration parameters, e.g. the position in the environment. Raw environmental sensor information is passed on to the obstacle avoidance module, which computes an abstraction of the detected obstacles and interacts with the rest of the control logic as mentioned above. The task logic can build on sensor information for task-

specific routines other than evading obstacles, e.g. taking a tactical decision in a robot football setting based on sensor data about the ball.

The *communication module* is depicted here to emphasise the cooperative nature of some motion planners. In particular multi-robot algorithms often require communication between the robots. Some task-specific features of the environment may also be entities the robot communicates with, e.g. a relay station which transmits new tasks to the robot. As the example illustrates, information received using a communication link is required at the task logic level, while other control logic entities are only indirectly influenced thereby. In the remainder of this text communications issues are not explicitly dealt with since the induced issues can be treated in isolation and do not directly influence the discussed setups.

The environment of the robot is represented in figure 2.1 by three explicit entities. *Obstacles/terrain* refers to the characteristics of the environment which determine the spatial distribution of navigation difficulty. In a setting where only binary obstacles are considered, the subset of the workspace occupied by those denotes a forbidden region. When navigating on rough terrain (or more generally with continuous obstacles), the distinction is less clear. A possible representation of navigational difficulty in rough-terrain motion planning is discussed in section 4.1.1. For pure motion planning tasks, sensor data gathered about the environment is primarily intended to detect (and characterise) obstacles.

In multi-robot planning, the *other robots* form a part of the environment from the point of view of an individual robot. These robots are often of identical design but this is no requirement. Sensor data is gathered about other robots, e.g. for collision avoidance purposes. In cooperative settings, communication between the robots is maintained. It is possible to cooperate simply by means of sensing other robots in the environment but more sophisticated interactions require explicit, active communication links.

The third entity depicted as part of the environment are *task-specific features*. This category represents all parts of the environment which hold a special semantic meaning for the task at hand, e.g. a configuration where a mobile robot can recharge its batteries or reflectors mounted for aiding orientation. Similarly to the interactions with other robots, task-specific features can be sensed and explicitly communicated with.

Implicitly, the environment includes all conditions which influence the motions of the robot. In particular, this includes the laws of physics which govern the behaviour of the robot as consequence of a commanded action. A physical interaction is depicted between the obstacles/terrain and the mechanical structure to represent this interdependency. Likewise, other robots can influence the mechanical structure, e.g. by colliding with the robot. Depending on their characteristics, task-specific features can also physically interact with the robot, e.g. a lift which transports the robot to another floor in a building.

### 2.3 Motivation of Simulation for Motion Planning Development

In the previous sections, an outline of the progress made in robot motion planning has been given. The role of motion planners within an overall robotic system has also been illustrated. Both have served to highlight some of the complexity involved in the development of motion planning algorithms.

Increasingly complex algorithms generally require more extensive testing since the level to which they can be theoretically verified is reduced. This is particularly true for heuristic algorithms and approaches which rely on randomisation. The testing phase is precisely where the development process can benefit from the use of simulations. Importantly, testing must not be considered a separate task carried out in isolation. It is much rather an interleaved process executed concurrently with the other development tasks. This is both true for the motion planning algorithm development described in this document and within the larger picture of mechatronics development.

In this section, it is motivated how the use of a software simulation environment can support the development of motion planners. At the same time, the foundation is laid for the elaboration of requirements established when designing the simulation environment. The requirements are presented in section 3.1.

The advantages of using a simulation environment for robot motion planning algorithm development have been divided into three categories: *environment modelling*, *control of information* and *management of experiments*. Some of the arguments listed below implicitly consider a simulation which can be interacted with on-line, others are of greater generality.

#### 2.3.1 Environment Modelling

*Environment modelling* refers to advantages gained through the higher flexibility when designing a simulation experiment compared to its real-world counterpart.

- Arbitrary environments can be simulated which might not be feasible to operate in otherwise (e.g. planetary exploration scenarios). Furthermore, it is easy to explore hypothetical settings which do not exist in reality. Simulation environments can easily offer a high level of freedom when modelling the physical behaviour. As a simple example, the gravity applied to a simulation can be typically adjusted by simply modifying a single parameter. The influence of each parameter in an experimental setup can be inspected independently of side-effects which arise in the real world.
- The virtual experiments are not restricted to simulating robots which are readily available in the real world. All that is required is an adequate model of the robot. Hence, the cost of simulating some specific robot can be incomparably lower in relation to operating it in reality. Also, given the model of

a robot for a software simulation, further copies thereof can be added to the experiment with minimal effort.

- The cost of failure can practically be neglected when using simulations: if the robot or other valuable equipment in the environment get damaged in a virtual experiment, no physical damage has been suffered. This allows to explore situations which are too risky if expensive equipment is at stake.

### 2.3.2 Control of Information

*Control of information* deals with advantages related to the better control of information which can typically be exerted during the execution of simulated experiments.

- A well-designed simulation environment gives the user full access to all information contained in the experiment. This can often not be achieved in reality or only with a high effort.
- The amount of information made accessible to the motion planner can be precisely controlled. This allows to explore the influence of data availability on algorithm performance. In sensor-based motion planning, data can be passed on to the motion planner through a simulated sensor model. A genuinely perfect sensor can be created, thus eliminating a potential source of errors during development.
- Randomisation can be introduced to model uncertainties in a controlled manner. Uncertainties can be applied to both the environment model and the information passed on to the motion planner. In the model of the environment, known error bounds can be used to determine worst-case scenarios of approximately known situations. Controlled errors introduced into the simulated sensor data can be used to perform more realistic sensor simulations.

### 2.3.3 Management of Experiments

*Management of experiments* includes advantages which do not fall into either of the previous two categories. These advantages are not directly related to the content of the experiment being conducted but rather to the way in which experiments are handled.

- Virtual experiments are typically far more effective (in terms of both time and costs) when creating and maintaining a test setup. This effect is even more pronounced when modifying existing experiments. The reduced effort to create a single experiment allows to perform more thorough testing involving a higher number of different scenarios for a given amount of resources.

- Deterministic simulations provide perfect experiment repeatability independently of the environmental complexity involved. As shown in section 2.1, the simple geometric environment models used in early motion planners have been replaced by sophisticated representations of the robot and its surroundings. Often relevant effects arise from the interactions between individual entities in the tested environment. In the real world, this has caused it to be increasingly complicated to achieve precisely repeatable experiments. From another point of view, when comparing different algorithms a deterministic simulation environment can be relied on to present each of them with an identical environment. As a result, all differences in behaviour can be attributed to the algorithms, instead of some variation in the test setup.
- Using a software simulation has the additional benefit of easily creating backup copies of experimental setups. This allows to set up an archive of readily available test environments which can be reused with little effort. At a lower level of granularity, the reusability of individual simulation components is also higher than could be achieved with their real-world equivalents. Components in this context refers to subsets of the environment, e.g. a sensor array or a specific topographic feature of a terrain.
- Simulation environments allow to take control of the execution time. Often, they have the potential to be run faster than real-time. This is in itself an advantage by shortening experiment durations. Running faster than real-time becomes particularly important when developing algorithms which require a long learning phase. This is typically the case with artificial intelligence approaches which require many iterations to tune their parameter set. On the other hand, it is possible to deliberately slow down (or pause) the progress of the simulation to achieve close on-line inspection of some phenomenon. Depending on the simulation environment it may also be possible to “rewind” the progress of the simulation, thus having full control over the simulation time.
- Software simulations can easily be run in parallel on multiple computers. As many experiments can be run simultaneously, as there are instances of the required host hardware available. This crude method of parallelisation is a simple way of increasing the number of experiment runs in given time.

The advantages listed above motivate the use of an on-line software simulation for supporting the development of robot motion planning algorithms. They are transformed into concrete requirements for the simulation framework developed as part of this work. A list of the requirements derived from the above arguments is given in section 3.1 together with other desired features of the simulation framework.

### 3. SIMULATION ENVIRONMENT FOR MOTION PLANNING ALGORITHM DEVELOPMENT

The advantages of using an on-line simulation for supporting robot motion planning algorithm development have been outlined in section 2.3. As part of the research presented in this document, a simulation framework named Ibex has been developed. The development goal has been to create an environment which allows to benefit from the potential advantages and hence offer valuable support for the hard task of developing motion planning algorithms.

A list of requirements for the Ibex simulation environment is presented in section 3.1 before a brief review of related simulations used in robotics is given in section 3.2. In section 3.3, the Ibex framework itself is discussed in some detail. The simulation kernel of Ibex is formed by the Ageia™ PhysX™ rigid body dynamics libraries. To give a better understanding of the involved algorithms, a general description of rigid body dynamics simulation techniques is included. A validation of the PhysX libraries within Ibex as well as a performance analysis are presented in section 3.4. The chapter is concluded by highlighting some issues related to employing simulations in general and rigid body dynamics simulations in particular for robot motion planning.

#### *3.1 Desired Functionality*

A primary design goal for a simulation framework intended to support the development of motion planning algorithms is necessarily that a large part of current motion planning algorithms can be tested with the program. An overview of historic development and the current state in the motion planning domain is given in section 2.1. Based thereon, a first set of desired features in Ibex is derived:

- The simulation of three-dimensional workspaces is a strict requirement.
- It must be possible to simulate robot dynamics as well as kinematics to support kinodynamic motion planning. Likewise, kinematic motion constraints of mechanical structures must be representable in the simulation.

- Possibilities must be provided to simulate sensor abstractions which give the motion planner access to data about the environment. It should be possible to implement perfect sensors which provide error-free measurements. This allows to support the development of sensor-based motion planning algorithms with the simulation.
- The simulation must be scalable to include multiple robots to allow the testing of multi-robot motion planners.
- It must be possible to include models of all entities described as part of an overall robotic system and its environment in section 2.2. Primarily, possibilities must be provided to simulate actuators as well as to include task logic into the setup apart from the components listed above.

The motivation for designing a simulation environment to support motion planning algorithm development is directly driven by the potential advantages listed in section 2.3. Naturally, the simulation should provide those benefits to the highest achievable degree.

- It must be possible to interact with the simulation on-line, i.e. while it is running. To enhance the interactivity, the simulation must provide maximal feedback to the user about the simulation state, including options for “intuitive” understanding of the simulation state, e.g. based on computer graphics.
- A simulation content generation tool-chain needs to be established which optimally supports the development of simulation setups. A design goal is the ease of creating and modifying experimental scenarios. Following the above listed argument, the design of the on-line visualisation must also be conveniently supported.
- Detailed access to information about the simulation state and progress must be possible. The data should be accessible in a way which easily allows its further usage.
- It must be possible to precisely control the amount of data made available to the motion planner operating in the simulation. Of particular interest is the possibility to include abstractions of sensors which give the planner access to simulation data through a similar interface to the one used for real sensors.
- The simulation must be deterministic to ensure experiment repeatability. Increases in complexity of the simulated environment must not compromise the determinism.
- Creating back-up copies of experiments consisting of complex environments and robots must be possible in a convenient manner.

- The user must have the possibility to control the progress of simulation time. In particular, a possibility to interrupt the simulation for close inspection is desirable.

Finally, some general considerations motivate further requirements which add to the usefulness of the simulation framework. They are not directly related to simulation features required to develop current motion planners or to the advantages of using a simulation for testing such algorithms:

- For many of the tasks involved in the creation and execution of simulations highly developed solutions exist. Where possible, the simulation and its associated components (e.g. in the tool-chain) should blend into existing development environments. Doing so significantly reduces the learning effort required to use the simulation for users familiar with the standard solutions. For a better integration with existing environments, standardised file formats should also be used where feasible.
- Ideally a range of options should be offered as to how motion planning algorithms are implemented. At an early stage, fast progress can be achieved by using scripting or graphical programming languages. Later in the development process, an optimal solution should allow to test a motion planner using the same program code used to compile the module for the real robot.
- The hardware requirements posed by the simulation should be as modest and generic as possible. In particular, no dedicated simulation hardware should be required if it can be avoided.

To accommodate the design goals listed above, the choice has been made to centre the simulation environment on a rigid body dynamics library as simulation kernel. Recent progress in rigid body dynamics simulation algorithms and PC host hardware allow to fulfill most of the directly simulation-related requirements above with such a solution. One point which is explicitly not fulfilled is support for deformable robots, but the bulk of current motion planning algorithms is developed for robots which either are rigid or can be satisfyingly approximated with a rigid model. Therefore this restriction has been deemed acceptable. Furthermore, progress in deformable-body simulation algorithms included in “rigid-body dynamics” libraries suggests that in the near future those algorithms will have reached a level of maturity which make it feasible to include them as well.

For the implementation, the Ageia™ PhysX™ libraries have been selected which offer state-of-the-art performance in the field. The libraries include unconstrained rigid body dynamics, joint definitions as well as collision detection and collision resolution algorithms (among other features not used in Ibex). The underlying functional principles of rigid body dynamics algorithms are described in section 3.3.2.

Rigid body dynamics simulation allows to model the mechanical structure of the robot as well as the environment. Apart from these purely mechanical entities, possibilities must exist to include the other components of the overall robotic system and its environment discussed in section 2.2.

Some concrete sensor simulations have been implemented in Ibex, as shown in section 3.3.4. Ray-casting techniques are used to simulate sensors which are based on wave propagation, such as laser time-of-flight sensors or ultrasonic devices. As a complementary approach, geometries can be specified which register intersections with objects in the simulated scene. This mechanism serves to simulate e.g. light barriers or other sensors with rigid detection geometries. The general structure of Ibex allows the relatively easy inclusion of further sensors simulations.

Actuators are simulated in Ibex by either directly applying forces and torques to rigid bodies or doing so at the level of the joint abstractions. Ibex has been integrated into an industry-standard dynamic modelling and control engineering software, MathWorks Simulink<sup>®</sup> (cp. section 3.3.6). Indeed, this represents the most versatile configuration of the simulation framework. As an important benefit, the dynamic behaviour of actuators can be faithfully modelled in Simulink and the computed forces applied to the dynamic model of the mechanical structure co-simulated in Ibex.

Control logic can be integrated into an Ibex simulation in two ways. The Ibex API (application programming interface) can be used to include Ibex as a library into an application program which encompasses the control logic. Alternatively, the versatile MATLAB/Simulink environment can be used to develop such algorithms and execute them in parallel with an Ibex simulation.

The *task-specific features* and *communication module* entities shown in figure 2.1 are not further dealt with in this text since they are typically strongly dependent on a concrete application. If the inclusion of these or further entities is required, the same mechanisms described above can be used to interact them with Ibex.

### 3.2 Related Work

Many different types of simulation environment used for robot motion planning can be found in literature and on the Internet. In this section a few examples are shown in an attempt to put existing solutions into relation with Ibex. The focus lies on simulations which include system dynamics and which are not limited to some specific type of mechanism, e.g. legged or industrial robots. Likewise solutions have been preferentially listed which encompass collision detection and collision resolution routines.

An object-oriented rigid body dynamics framework named *I-GMS* is described in [Son 00]. Earlier dynamics simulation environments mostly focussed on one specific type of rigid body setup and can therefore not be considered general sim-

ulation frameworks. An overview of earlier environments is given in [Son 00]. An important design goal of I-GMS was the general applicability to various types of rigid body setups, with an emphasis on the simulation of linkages. Like in Ibox, functionality abstractions are used to encapsulate functionality in an object-oriented framework. One remarkable feature of I-GMS is the on-line interaction which can be performed by means of a haptic device. Rigid body dynamics as well as collision detection and resolution are performed by native routines (as opposed to using an external library).

*Darwin2K* [Leger 00] is an open-source toolkit focussed on supporting evolutionary robotics design. Kinematics and dynamics simulations with collision detection are included as well as an evolutionary algorithm which iteratively develops the structure of a robot. Among other options, the OpenGL library is used for visualisations. The software is available for Linux and Irix. The last update was made to the project in 2003 and its usage appears to be declining.

*Gazebo* [Koenig 04] is an open-source simulation environment which forms part of the larger Player/Stage project [Vaughan 03], [Collett 05]. Player is a network server for controlling multiple robots. It provides an abstract interface to the sensors and actuators on a real robot which is connected to the server using the TCP/IP protocol. Stage is a simple 2D multi-robot simulation environment focussed at computationally cheap simulation of numerous robots. It can be replaced by the Gazebo simulation which offers 3D rigid body dynamics simulations. An important feature of the framework is that Player is designed to work transparently both with real robots or either of the simulation environments. Modelling of robots and the environment is done using XML files with predefined environment elements and robots. Apart from the available robot models, new robots can be defined using the C programming language. Gazebo is based on the Open Dynamics Engine (ODE, [ODE 06]) which is an open-source rigid body dynamics library. Scenes are visualised in 3D by means of the OpenGL library. The software runs on Linux, Solaris and BSD.

Targeted at the robot football “RoboCup” competitions but of wider applicability is *SimRobot* described in [Laue 06]. Robot models are defined using a specific XML format. As simulation kernel ODE is used, visualisations are performed using OpenGL. Objects are represented as combinations of primitive shapes (such as boxes, spheres, cylinders) but not polygonal mesh shapes. Generic actuator and sensor models are provided. As sensors, a synthetic vision [Renault 90] “camera” and a similar depth sensor have been implemented. Touch sensors and “actuator sensors” returning joint angles and velocities are also included.

*OpenSim* [Ope 06] is a further open-source simulation using ODE. The focus lies on supporting the development of autonomous mobile robots. Development on inclusion and enhancement of dynamics simulation appears to be progressing slowly since (inverse) kinematics are frequently preferred by the users. Visualisa-

tions are supported with OpenGL. The framework is designed for Linux.

*Yobotics! Simulation Construction Set* [Yob 06] is a commercial software package mainly focussing on legged robots but readily applicable to other structures. Extensions of simulations are possible using the Java programming language, in which the simulation is programmed. Controllers are also included using Java. The simulation offers a GUI in which the robot is visualised and parameters can be inspected. The user interface allows to navigate through time and record video clips or pictures of the simulation. Contact detection and resolution appears to be only between specified points of the robot and a terrain. The terrain can be defined very generally as arbitrary Java method which implements a specific interface.

Another commercial simulation package somewhat similar to Ibex is *Webots<sup>TM</sup>*, produced by Cyberbotics Ltd., a spin-off company of Swiss Federal Institute of Technology in Lausanne (EPFL) [Michel 04]. The development of Webots was commenced a number of years ago (the spin-off occurred in 1998) and has reached a high level of maturity in the meanwhile. This is underlined by the large number of pre-configured models of real-world components (e.g. sensors and actuators) which can be included in simulations. Webots allows to either use a purely kinematic or a dynamic simulation. The simulation kernel is based on the ODE engine. The physical properties and visual appearances of rigid bodies are edited textually in the scene tree within the proprietary Webots GUI. It is also possible to import VRML geometry and appearance definitions which can be generated in most 3D modelling software. A strong point of Webots is that controller code (written in the C, C++ or Java languages) can be included in a simulation. The same source code can be tested in the simulation and transferred to the real robot.

A number of MathWorks Simulink toolboxes exist which simulate robot kinematics and dynamics but typically lack support for collision detection. These packages usually directly implement the equations of motion of rigid bodies. Simulations within this category include e.g. [Corke 96]. MathWorks itself markets a library named SimMechanics which provides rigid body kinematics and dynamics but no collision detection or collision resolution. Also, visualisations similar to those in Ibex are only possible using a further extension of MATLAB/Simulink, the Virtual Reality Toolbox.

Rigid body dynamics are computed in Ibex using the Ageia<sup>TM</sup> PhysX<sup>TM</sup> library. Many of the simulation environments listed above include ODE as simulation kernel. The two packages differ somewhat in their simulation capabilities. Most importantly for the present research, in ODE no support is given for per-triangle material properties in polygonal mesh shapes. Among other situations, this is a crucial property of sophisticated terrain models as described in section 3.3.3. Also, during informal testing, ODE has been found to require higher computation times than PhysX for comparable simulation complexities. Similar findings have been reported in [Seugling 06].

Other rigid body dynamics libraries include Tokamak Game Physics SDK [Tok 06], Havok Physics<sup>TM</sup> [Hav 06] and Newton<sup>TM</sup> Game Dynamics [New 06]. Newton and Tokamak are free for commercial and non-commercial use (closed source) while Havok requires licensing for all use. In [Seugling 06], ODE and Newton receive worse ratings than PhysX in a series of tests. Tokamak is not directly compared for performance but receives significantly worse marks for offered features, documentation and usability. Further, no recent development activities can be recognised, with the latest release and web-site updates being over one year old at the time of writing. The same is true for a number of other physics engines listed in [Seugling 06].

### 3.3 Simulation Framework for Algorithm Prototyping

The simulation framework Ibex [Ettlin 05b] has been developed as an effort to fulfill the requirements listed in section 3.1. The simulation software itself as well as a number of additional components which form a complete simulation framework for robot motion planning are discussed in this section. An overview of these modules and their interactions is given in section 3.3.1. The Ibex framework makes use of the Ageia<sup>TM</sup> PhysX<sup>TM</sup> rigid body dynamics libraries [Phy 06] (previously known as NovodeX SDK) as simulation kernel. To give a better understanding of how the simulation operates, rigid body dynamics algorithms are introduced in section 3.3.2 together with a number of related techniques employed in the Ibex framework. Modelling of the interaction between a robot and the terrain it operates on is of paramount importance in physics-based mobile robotics motion planning. The model used in the present work is introduced in section 3.3.3. As motivated in section 3.1, a requirement for realistic robotics simulation environments is the ability to obtain simulated sensor data on which the decision-taking algorithms can base their actions. The physics-based approach chosen in Ibex is presented in section 3.3.4. A further requirement discussed above is the need for a well-established flow of simulation content from model creation to the simulation proper. The tool-chain which has been developed to this end in Ibex is introduced in section 3.3.5. In order to integrate well with existing software solutions and significantly increase its scope, the Ibex simulation has been integrated into an industry-standard software suite, MathWorks Simulink<sup>®</sup>. Section 3.3.6 details on this integration which has resulted in a co-simulation environment for Ibex rigid body dynamics and the complete MATLAB/Simulink functionality.

#### 3.3.1 Overview

This section gives a high-level overview of the Ibex simulation environment (cp. [Ettlin 04], [Ettlin 05d]) before subsequent sections deal with individual aspects of

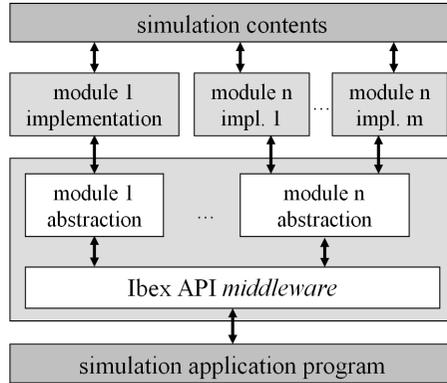


Fig. 3.1: High-level abstract view of the architecture employed in the Ibex simulation program.

the framework in more detail. At the most general level, Ibex is a generic discrete-time simulation framework. Following good software engineering practice (e.g. [Pressman 00]), extensive use of the concept of abstraction has been made during the development. Such an approach allows to consider a piece of software in terms of functionality rather than concrete implementations. Self-contained modules encompassing well-defined functionality can be designed at a conceptual level. Such a procedure leads to maximising inter-modular cohesion while minimising intra-modular coupling, which is again considered good practice when developing software.

The abstract architecture of a stand-alone Ibex-based simulation is depicted in figure 3.1. The Ibex simulation consists of the two middle tiers in the figure. From this abstract point of view, Ibex is generic in the sense that it is not bound to any specific type of simulation or application domain. The type of simulation is specified by selecting specific implementations of the generic simulation functionality. The only application-specific functionality (and hence program code) is contained in the bottom tier labelled “simulation application program”. Each concrete simulation experiment requires simulation content to operate on. Simulation content ranges from physical data about the involved entities to artwork used to design more sophisticated visualisations. In figure 3.1, simulation content is represented by the top tier.

The abstract framework described above is highly customisable and allows implementations from a wide range of domains as long as they fulfill basic discrete-time simulation properties. To date, this general framework has been specialised for rigid body dynamics simulations. The resulting rigid body dynamics abstractions have in turn been implemented using the Ageia™ PhysX™ libraries [Phy 06]

as simulation kernel.

A similar process of specialisation and implementation has been performed for modules which give users access to simulation data, termed “observers”. The implementations developed so far again deal with rigid body dynamics and thus implement an interface which allows to access data generated by such simulations. The scope of implemented observers ranges from a configurable log file generator over on-line spreadsheet-style textual displays of object properties to 3D graphics visualisations. The implementations of the latter are based on three different graphics engines: the open-source project Irrlicht [Irr 06], a graphics engine derived from Ageia™ PhysX™ preview utility Rocket™ and the Nebula Device 2 graphics engine [Neb 06].

Supervisory modules exist which coordinate the execution of the simulation at a generic level, independently of the simulation domain [Büchler 05]. These modules also manage the resources allotted to multiple simulation and observer modules running simultaneously. An important function of the supervisory modules is to control the execution rate of the simulation. Four modes have been implemented:

- *CPU speed*: Runs the simulation as fast as possible, using all available resources. Execution speed may vary in the course of the simulation due to fluctuations in resource availability or simulation complexity.
- *Step-based real-time*: Attempts to make the execution time of every discrete time-step of the simulation correspond precisely to its equivalent in the real-world. Committed errors are not corrected, which can lead to accumulating divergencies. Only possible for setups which can run equal or faster than real-time.
- *Accumulated real-time*: Attempts to match the accumulated simulation time to the corresponding time in the real-world. Errors committed in earlier time-steps are compensated as soon as allowed by resource availability. Only possible for setups which can run equal or faster than real-time.
- *Step/pause*: Advances the simulation a single time-step and waits for further execution commands. 3D graphics observers operate independently of simulation tasks, allowing a detailed inspection of the simulated setup.

In the stand-alone configuration of Ibex, a module for designing graphical user interfaces (GUIs) can be optionally included. By default an Ibex GUI contains a number of dialogues which give a user control of generic Ibex functionality (such as execution speed and observer controls). For each application, the generic GUI can be adapted and extended to meet the application-specific needs. The Ibex GUI abstraction has been implemented making use of the open-source wxWidgets widget library [wxW 06].

The entirety of Ibex has been programmed in the C++ language and has been developed under Microsoft<sup>®</sup> Windows<sup>®</sup>. Wherever possible third-party libraries have been selected which are platform-independent.

### *Simulink Integration*

Alternatively to the described stand-alone configuration, Ibex can be used embedded in MathWorks Simulink<sup>®</sup>, a dynamic modelling and control engineering software. This integration is described in more detail in section 3.3.6. The Ibex rigid body dynamics functionality is contained in a library of so-called blocks, the functional units used in the graphical Simulink user interface. All application-specific programming is also performed within the Simulink GUI. As an important benefit, this allows to use the wide variety of resources offered by MATLAB/Simulink to develop application functionality.

When using Ibex within Simulink, simulation control is yielded to the Simulink user interface. This eliminates the need for both the supervision modules and the Ibex GUI. The GUI functionality has been replaced by interfaces attached to the individual Simulink blocks composing the Ibex simulation. The execution speed is controlled by Simulink, which does not support real-time execution without dedicated hardware. Importantly, the execution of the Simulink model and the co-simulated Ibex functionality are coupled but independent. For each simulation, individual time-steps can be specified and it is even possible to operate the Simulink model with variable time-steps while the Ibex simulation uses fixed time-step sizes. The Ibex-Simulink integration only supports one single rigid body dynamics module - which has been found to cover most practical requirements. Also, only the Nebula 2 graphics engine has been ported to the Simulink mode.

### *Simulation Content Tool-Chain*

A content-based view of the overall Ibex framework architecture encompassing both the stand-alone configuration and the Simulink integration is shown in figure 3.2. Third-party software is shaded blue while Ibex components are coloured red. The source of simulation content is typically a piece of 3D modelling software (bottom left-hand box). At present the most convenient options are SolidWorks<sup>®</sup> and Autodesk<sup>®</sup> Maya<sup>®</sup>, for which Ibex plug-ins have been developed. As described in section 3.3.5, both programs require some adaptations to be compliant with Ibex rigid body dynamics definitions. The physics-enhanced computer-aided design (CAD) is exported by the Ibex plug-ins. The exportation consists of an XML file describing the modelled setup as well as a number of resource files containing object geometries and texture files for visualisation. These files represent all simulation content required to run an Ibex simulation. They can be used equally

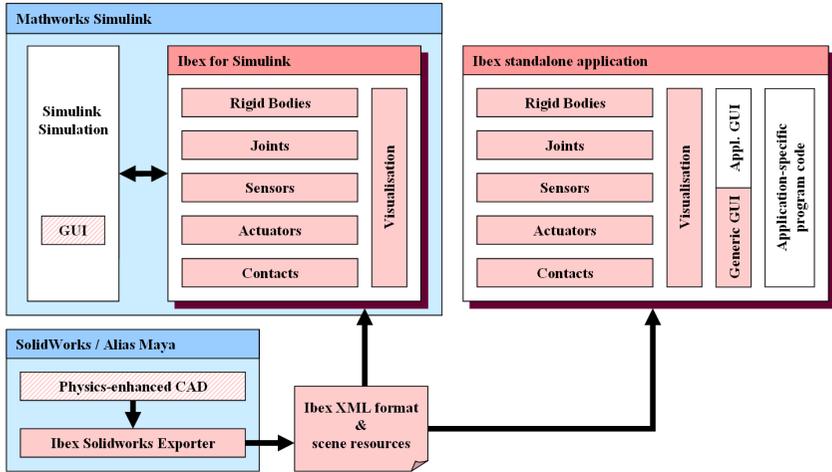


Fig. 3.2: Content-based architecture of the complete Ibex framework, including the stand-alone application configuration as well as Ibex embedded in MathWorks Simulink<sup>®</sup>.

by stand-alone applications linked to Ibex and by Ibex simulations running within Simulink. Figure 3.2 also shows the conceptual differences of entities such as application-specific program code or the GUI in both configurations.

Following this high-level overview of the complete Ibex framework, individual modules are introduced in more detail. The rigid body dynamics simulation module is presented in section 3.3.2, followed by a discussion of the implemented terrain model for rough-terrain robot motion planning in section 3.3.3 which represents a special case of rigid body. Details of the content-generation tool-chain are given in section 3.3.5, section 3.3.6 deals with the creation of a co-simulation environment by integrating Ibex into Simulink.

### 3.3.2 Rigid Body Dynamics

Rigid body dynamics simulation deals with computing the motion of non-deformable geometries in space. Considering a single rigid body as a black box, its state can be specified for a single point in time by determining its position  $x(t)$  and orientation  $\Theta(t)$  (also called the *spatial variables* of the rigid body). On the other hand, the rigid body can be influenced in a physically consistent manner by applying forces and torques to it. As will be shown later, this simple interface definition forms the basis of the interface provided for users of Ibex to interact with

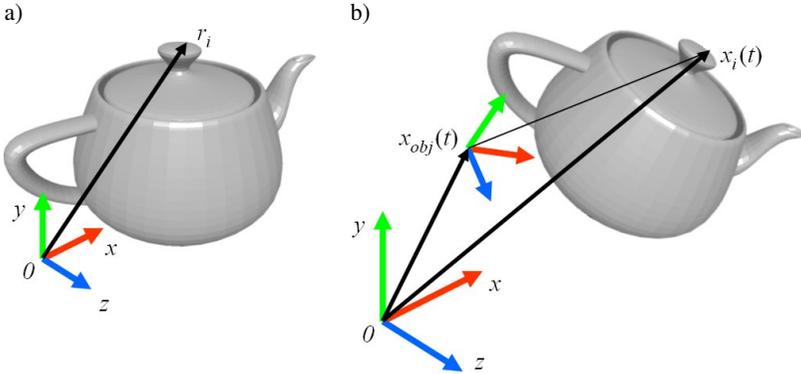


Fig. 3.3: Relation between a) local and b) global coordinate systems.

rigid bodies.

A rigid body has an unchanging geometry which is defined in terms of a coordinate system relative to the origin of the rigid body (called the local coordinate system). By applying the position and orientation of the body to the local coordinate system, the volume of space occupied by the rigid body in world space (relative to a global coordinate system) can be determined.

Figure 3.3 a) shows a teapot (modelled as a single rigid body) as 3-dimensional geometry relative to a local coordinate system. The coordinates  $r_i$  of each point  $i$  of the geometry relative to the local origin can be pre-computed. In figure 3.3 b), the teapot and its local coordinate system have been translated and rotated to some position and orientation relative to the global coordinate system. The global coordinates of the individual points can be expressed as

$$x_i(t) = x_{obj}(t) + r_i(t) = \underbrace{x_{obj}(t)}_{\text{translation}} + \underbrace{\Theta(r_i, t)}_{\text{rotation}} \quad (3.1)$$

where  $\Theta$  represents the rotation about the local origin of the rigid body. In IbeX, the most general form of specifying the geometry of a rigid body is by means of a triangulated polygonal mesh. Such a representation consists of a list of vertices as well as information about their inter-connectedness such that triangular polygons are defined which delimit the contained volume. The resulting closed, piecewise planar surfaces can be used to approximate arbitrary shapes at any desired precision. In particular, it is possible to model not only convex but also concave 3D geometries. The enclosed volume may also contain “holes” (cp. the handle of the teapot depicted in figure 3.3). During the simulation, the position of each vertex on a mesh is conceptually computed using equation 3.1. Given the constant relative locations of the mesh vertices this causes the entire volume of the rigid object to

be translated and rotated accordingly (without any unwanted transformations, e.g. scaling or skewing).

A common assumption in literature is that the local coordinate system is anchored at the centre of mass of a rigid body. In Ibex (and in figure 3.3), this assumption has been dropped. This allows more flexibility in the simulation content tool-chain by decoupling the geometric representation from the physical characteristics of a rigid body. On the downside, the physical meaning of a translation induced by the position vector of a rigid body gets somewhat diluted. Therefore, in the algorithm descriptions in this document it is assumed that the position of a rigid body does coincide with its centre of mass.

In Ibex, positions are represented as Cartesian coordinates as has been implicitly introduced above. The orientation is represented as quaternion internally but due to its non-intuitive nature, Euler angles are used at the user interface level. The main reason for not directly employing Euler angles in the computation is that they are subjected to a phenomenon called Gimbal lock [Shoemake 85]. Quaternions are an extension of complex numbers with three imaginary parts instead of one. Normalised (unit) quaternions  $q$  can be used to represent arbitrary rotations effectively:

$$q = \left( \cos\left(\frac{\varphi}{2}\right), \sin\left(\frac{\varphi}{2}\right) \cdot (a_x, a_y, a_z) \right) \quad (3.2)$$

Since every rotation can be expressed as a rotation about an axis, rotation quaternions are defined as in equation 3.2 with  $(a_x, a_y, a_z)$  the rotation axis and  $\varphi$  the angle of rotation. The position of a point after a rotation can be efficiently computed as

$$\Theta(x) = q \cdot x \cdot q^{-1} \quad (3.3)$$

with  $x = (0, x_1, x_2, x_3)$  holding the coordinates of the original (unrotated) point.

The linear (translational) velocity of a rigid body is given by the time derivative of its position  $v(t) = \dot{x}(t)$ . Likewise, the angular (rotational) velocity is  $\omega(t) = \dot{\Theta}(t)$ . Depending on the concrete representation of rotations, the computation of  $\omega(t)$  is not straightforward. Independently of the representation in use,  $\omega(t)$  can be viewed as the rotation of the body about an axis passing through its origin (the centre of mass under the simplifying assumption introduced above). Similarly to the axis-angle definition of rotations alluded to above,  $\omega(t)$  then is composed of a unit vector which gives the direction of the axis of rotation and its magnitude  $|\omega(t)|$  representing the velocity of rotation.

Given such a definition of angular velocity, the motion of a point on the rigid body can be studied. Let  $r_i(t) = x_i(t) - x_{obj}(t)$  be the position (in global coordinates) of a point on the rigid body in relation to the local origin of the body; as

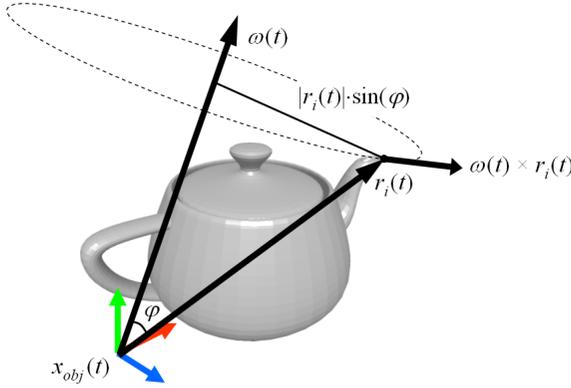


Fig. 3.4: Illustration of angular velocity. The tip of  $r_i(t)$  follows a circle of diameter  $|r_i(t)| \cdot \sin(\varphi)$  around  $\omega(t)$  with velocity  $|\omega(t)| |r_i(t)| \cdot \sin(\varphi)$ .

shown in figure 3.4 for the tip of the teapot's spout. Since this vector represents a direction, it is independent of translations and in particular,  $\dot{r}_i(t)$  is independent of the linear velocity  $v(t)$  of the origin  $x_{obj}(t)$  of the local frame.

For a constant angular velocity  $\omega(t)$ , the tip of  $r_i(t)$  follows a circular trajectory around  $\omega(t)$  at a distance  $|r_i(t)| \cdot \sin(\varphi)$ . At any point in time, the change of  $r_i(t)$  can be expressed as  $\dot{r}_i(t) = \omega(t) \times r_i(t)$ .

The total velocity of a point  $i$  on the rigid body can be written in global coordinates as:

$$\dot{r}_i(t) = \omega(t) \times \underbrace{(x_i(t) - x_{obj}(t))}_{r_i(t)} + v(t) \quad (3.4)$$

After these preliminary definitions, a general description of an algorithm for unconstrained rigid body dynamics simulation is outlined. In terms of the black-box model introduced at the beginning of this section, it is shown how forces and torques applied to the rigid body lead to changes of its linear and angular velocities. In order to simplify the following equations, a discretised model of the rigid body is assumed where the total mass  $M$  of the body is composed by a discrete number of point masses  $m_i$  at locations  $r_i$ . Also, as numerical integration scheme, Euler integrations are used which serve for illustration purposes but are typically replaced by more powerful higher-order integration methods in practical simulations.

Forces applied to a rigid body are applied to a single point on the object. Let  $F_i(t)$  denote the total external force acting on point  $i$  of the rigid body. The external torque acting on point  $i$  is defined as:

$$\tau_i(t) = r_i(t) \times F_i(t) \quad (3.5)$$

Again assuming the local coordinate system is rooted at the centre of mass of the body, the acceleration (the change in linear velocity) is computed as

$$a(t) = \ddot{x}(t) = \frac{\sum_i F_i}{\sum_i m_i} = \frac{F}{M} \quad (3.6)$$

where  $F$  denotes the total force applied to the object and  $M$  its total mass. Without discussing the issues related to numerical integration methods which would exceed the scope of this general introduction, let it suffice to state that two integration steps lead to updated values for the velocity  $v(t) = \dot{x}(t)$  and the position  $x(t)$ .

The linear momentum of a rigid body is  $P(t) = Mv(t)$  which relates to the total applied force as  $\dot{P}(t) = F(t)$ . On the other hand, the angular momentum for a rigid body is not equally intuitive. Nevertheless it is a helpful concept due to angular momentum conservation (contrary to angular velocity which is not conserved). In analogy to the linear momentum, the angular momentum  $L(t)$  relates to the total torque as  $\dot{L}(t) = \tau(t)$ . Angular momentum is related to angular velocity through

$$L(t) = I(t)\omega(t) \quad (3.7)$$

with the inertia tensor  $I(t)$  having taken the position of the total mass in the linear case. The inertia tensor is a description of the mass distribution of a rigid body relative to the centre of mass. In the discrete case used for illustration in this section, it is computed as:

$$I(t) = \sum_i m_i \begin{pmatrix} r_{iy}^2 + r_{iz}^2 & -r_{ix}r_{iy} & -r_{ix}r_{iz} \\ r_{iy}r_{ix} & r_{ix}^2 + r_{iz}^2 & -r_{iy}r_{iz} \\ r_{iz}r_{ix} & -r_{iz}r_{iy} & r_{ix}^2 + r_{iy}^2 \end{pmatrix} \quad (3.8)$$

where  $r_i = (r_{ix}, r_{iy}, r_{iz})^T$  is shorthand for  $r_i(t)$  defined above. Since the (relative) coordinates of  $r_i(t)$  refer to the global coordinate system,  $I(t)$  is orientation-dependent and hence generally time-dependent.

Fortunately, there is no need to perform the costly operation of recomputing the inertia tensor at every time step (Especially in the continuous case which requires an integration over the volume of the body).  $I(t)$  can be decomposed into a constant inertia tensor  $I_{body}$  which can be pre-computed and from which  $I(t)$  can be efficiently derived using the orientation  $\Theta(t)$  at every time step. In the Ibex framework, the inertia tensors  $I_{body}$  can be explicitly specified for a body or computed by the Ageia™ PhysX™ simulation library for a volume of (piecewise) constant density.

Having computed the angular velocity  $\omega(t)$  of a rigid object at a given time, one integration step leads to its orientation  $\Theta(t)$ . When using numerical integration methods, errors accumulate over time. For translations, this leads to some drift of the simulated position with relation to the analytical solution. For the angular

Pre-computation	$M \leftarrow \sum m_i$ $r_{cm} \leftarrow (\sum r_i m_i) / M$ $I_{body}^{-1} \leftarrow (\sum m_i ((r_i^T r_i) \mathbf{1} - r_i r_i^T))^{-1}$
Initialisation	$x(0), \Theta(0), v(0), \omega(0)$
Simulation loop (Euler integration)	$\tau \leftarrow \sum r_i \times F_i$ $F \leftarrow \sum F_i$ $x \leftarrow x + \Delta t \cdot v$ $v \leftarrow v + \Delta t \cdot F / M$ $\Theta \leftarrow f(\Theta, \Delta t, \omega)$ $L \leftarrow L + \Delta t \cdot \tau$ $I^{-1} \leftarrow g(\Theta, I_{body}^{-1})$ $\omega \leftarrow I^{-1} L$

Tab. 3.1: Outline of the algorithm for unconstrained rigid body dynamics simulation.  $\mathbf{1}$  denotes the  $3 \times 3$  identity matrix.  $f()$  and  $g()$  are functions dependent of the orientation representation.

terms, the accumulation of numerical errors does not only lead to drift but can cause invalid rotation values (caused by redundant representations, e.g. quaternions and matrices, but not Euler angles). In the case of transformation matrices, unwanted skew can be introduced which needs to be removed by re-orthogonalising the matrix. For orientation quaternions, it must be ensured they remain unit quaternions by re-normalising them each time step.

The overall algorithm for unconstrained rigid body dynamics is outlined in table 3.1. The two update functions (for  $\Theta$  and  $I^{-1}$ ) which are dependent on the representation of the orientation  $\Theta$  have been omitted to avoid a commitment in this general overview. For quaternions, the update rule for the orientation  $\Theta(t)$  is

$$q \leftarrow q + \Delta t \frac{1}{2} (0, \omega) q \quad (3.9)$$

To compute the inverse inertia tensor  $I^{-1}(t)$ , first the quaternions  $q(t)$  are transformed to rotation matrix representation  $R(t)$  (e.g. [Shoemake 85]) and then the update rule

$$I^{-1} \leftarrow R I_{body}^{-1} R^T \quad (3.10)$$

is applied. In table 3.1, the dependency on time of the terms updated in the simulation loop has been omitted for improved readability. For the same reason, an Euler integration scheme has been assumed which typically would be replaced by a more accurate higher-order numerical integration method in a real simulation.

The algorithm outlined in table 3.1 specifies the behaviour of rigid bodies in an unconstrained environment. For most practical purposes and all cases of mobile

robot motion planning a number of additional constraints must be fulfilled. Firstly, the robot operates on the terrain and interacts with other rigid objects. This leads to non-penetration constraints having to be specified. Furthermore, any mechanical construction of interest relies on joints between individual rigid objects which constrain the relative motion of the connected bodies. These two aspects are highlighted in the following.

### *Non-Penetration Constraints*

In the previous section it has been described how rigid bodies occupy a volume of space defined by their geometry. The dynamic behaviour of an individual rigid body has also been outlined. For multiple rigid bodies to interact with each other in a physically meaningful way their respective volumes must not be allowed to intersect. Furthermore, rigid objects in contact must influence the motion of each other following a plausible physical model. The fundamental issues related with implementing non-penetration constraints for a rigid body dynamics simulation are described in this section.

### *Collision Detection*

The first step when designing non-penetrating behaviour of rigid bodies is to detect a collision between the geometries. In the context of the present document, time-discrete solvers are considered. Due to the discrete time steps such solvers operate with, the exact time of collision is missed and some penetration occurs. Let  $t_i$  be some time during the simulation at which the two bodies do not touch and  $\Delta t$  the time step performed at  $t_i$  such that  $t_c$ , the time of collision, lies between  $t_i$  and  $t_{i+1} = t_i + \Delta t$ . Clearly, at  $t_{i+1}$  the bodies have moved to configurations at which they intersect - violating the initially stated condition.

Fundamentally, there are two approaches to deal with this situation. Firstly, it is possible to exactly compute the collision time  $t_c$ , e.g. by performing a binary search between  $t_i$  and  $t_{i+1}$  [Moore 88]. The search iterates in time until  $t_c$  has been determined with some accuracy  $\epsilon$ . When the contact points have been precisely computed, a constraint force is applied at each contact point such that all external accelerations which would lead to an interpenetration are precisely cancelled e.g. [Baraff 93]. Such constraint-based methods can prevent any interpenetration (within numerical tolerances) but especially the exact calculation of contact points is a computationally expensive operation. Alternatively, continuous collision detection e.g. [Redon 02] techniques can be applied to determine the precise collision time instead of the time step at which some intersection has occurred. Unfortunately, continuous methods are usually slower than discrete methods and hence have often been discarded for real-time applications, e.g. [Snyder 95].

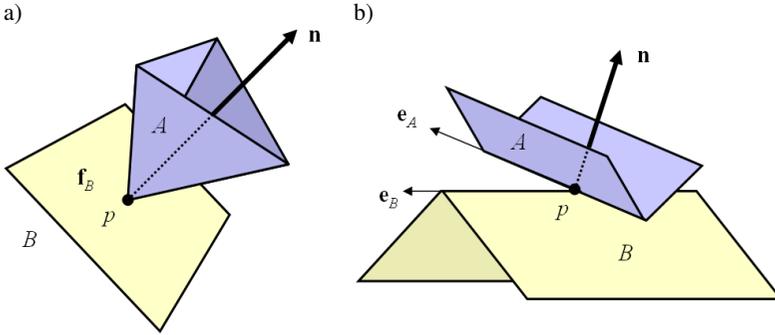


Fig. 3.5: Non-degenerate contact types between polygonal meshes: a) vertex-face contact b) edge-edge contact.  $p$  denotes the contact point,  $\mathbf{n}$  the contact normal.

For the remainder of this section, time-discrete approximative methods are considered. They avoid the exact computation of collision points and thus have the potential for significantly faster computations. Instead of applying constraint forces at precisely determined contact points, penalty forces are applied when the collision detection algorithms report an interpenetration. Examples of penalty method implementations are e.g. [McKenna 90] for articulated rigid bodies, [Gourret 89] for deformable bodies and [Lafleur 91] for cloth simulation - the wide range of applications illustrates the flexibility of the approach.

The penalty method is somewhat similar to a numerical solution method for constrained optimisation problems. The key idea is to convert a constrained problem into an unconstrained one in which deviations from the constraint are penalised. As a consequence, violations of the constraint (e.g. some penetration) are possible but discouraged. For non-penetration constraints, interpenetrations are penalised by applying a force which acts against the penetration. The magnitude of the penalty force is often computed modelling a linear spring force for an elongation corresponding to the penetration depth. Importantly, this force is independent of the physically motivated forces applied during contact resolution, it merely serves to prevent the interpenetration of the bodies.

Collisions between polygonal mesh geometries (including as subclass - in this context - the shape primitives introduced later in this section) occur at the level of their constituting primitives: faces, edges and vertices. The two non-degenerate cases are vertex-face and edge-edge collisions, cp. figure 3.5. Considering that individual objects may contain many such primitives (tens of thousands of vertices not being exceptional), it becomes obvious that a brute-force approach is not practicable. For two objects containing  $m$  and  $n$  primitives,  $O(mn)$  tests would need to be performed. On the other hand, most primitives on each object are far away

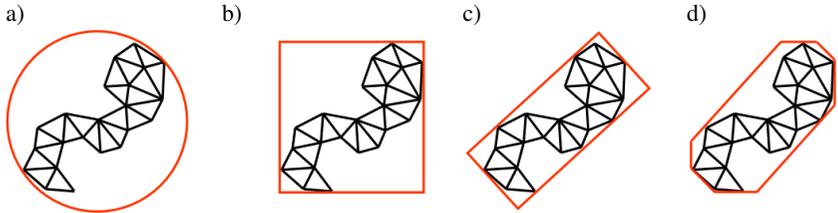


Fig. 3.6: Bounding volume approximations of a polygonal mesh: a) bounding sphere b) axis-aligned bounding box (AABB) c) object-oriented bounding box (OBB) d) discrete-orientation polytope “ $k$ -DOP”, the example is an 8-DOP with directions  $\varphi = n \cdot \frac{\pi}{4}$ . Along a horizontal axis, methods on the left are less complex in terms of computation time and allow faster overlap tests, while methods on the right offer a better approximation of the geometries.

from most primitives on the other object. This fact has led to numerous heuristic schemes having been designed, some of which are introduced in the following.

The definition of conservative bounding volumes allows fast approximative collision testing. Instead of checking all primitives of an object pair, first the bounding volumes are tested for intersection to determine whether a collision has potentially occurred.

As simplest approximation, bounding spheres have been proposed, e.g. [Quinlan 94], [Hubbard 96]. A collision test between spheres is trivial and as an additional benefit, spheres are orientation-independent, which makes this approach computationally effective. On the downside, the level of approximation achieved by a bounding sphere is far from optimal for many objects, e.g. those with elongated shapes.

Axis-aligned bounding boxes (AABBs) approximate rigid bodies by a minimal bounding cuboid with faces parallel to the global axes, e.g. [Beckmann 90]. AABBs allow a fast collision test by checking whether the extents of the AABBs overlap in all dimensions. In contrast to bounding spheres, the size of the AABBs is updated as the approximated object rotates in space. This leads to the interesting fact, that the quality of the object approximation is orientation-dependent and hence time-dependent.

Object-oriented bounding boxes (OBBs) to the contrary are rooted in object-space and rotate with the rigid body, e.g. [Barequet 96], [Gottschalk 96]. Since they can be oriented arbitrarily with respect to the rigid body, heuristics can be used to obtain good approximations of the object, e.g. by aligning it with the principal axes of the set of enclosed vertices. An overlap test for OBBs consists of finding some axis on which the projections of the boxes form intervals. If an axis can be found for which the intervals are disjoint, the objects do not intersect [Gottschalk 96].

The approximation with the highest precision discussed in this summary is achieved by using “discrete orientation polytopes” ( $k$ -DOPs) [Klosowski 98].  $k$ -DOPs are convex polytopes whose facets are determined by half-spaces with outward normals coming from a small *fixed* set of  $k$  orientations. In two dimensional space, a  $k$ -DOP is a polygon whose edges all have one of  $k$  orientations. In three dimensions, AABBs are a special case of 6-DOPs where all polytopes are aligned with the axes. Two  $k$ -DOPs do not overlap if the intervals of the projections onto axes of the  $k$  orientations do not overlap in at least one case.

The introduced bounding volumes have been presented in order of approximation quality. Figure 3.6 shows illustrations of the discussed approximations. When implementing collision detection algorithms, a tradeoff must be made between having better approximations of the geometries (which lead to fewer false positive potential collision detections) and faster approximation updates and intersection tests with lower quality approximations.

A widespread extension of these geometric approximations is the creation of bounding volume hierarchies. In fact, all the above mentioned publications consider some form of hierarchical decomposition of the object. The common idea is to first perform a collision test for the bounding volume enclosing the entire object. If that test returns a potential collision, the next level of the hierarchy is considered where the same type of approximation is applied to some partition of the object. Depending on the algorithm, this procedure is repeated multiple times before the lowest level of the hierarchy is reached and primitive-primitive tests need to be performed to precisely determine collision points.

So far, only two potentially colliding objects have been discussed. In more realistic scenes often numerous objects interact and are expected to adhere to non-penetration constraints. This requires the so-called  $n$ -body problem to be solved. A brute-force approach results in prohibitive  $\frac{n \cdot (n-1)}{2}$  collision tests being performed. To avoid testing object pairs which are sure to be disjoint, the “sweep and prune” culling procedure has been introduced [Cohen 95]. The underlying idea is an extension of the collision test for axis-aligned bounding boxes: the AABBs of all rigid bodies are projected onto the global axes and the resulting intervals sorted. All object pairs with disjoint intervals in at least one dimension can be pruned, a pairwise test needs to be performed on the remaining pairs only.

A second approach to collision detection does not strive to approximate the involved geometries but relies on partitioning the entire space. After doing so, only partitions need to be considered which are occupied by multiple rigid bodies. Furthermore, only those parts of the objects need to be tested which are contained in the conflicting partition.

The most straightforward spatial partitioning algorithms perform uniform grid partitioning. After each object transformation, a counter associated with each cell is incremented for each rigid body which is at least partly located in that cell. Cells

whose counter has registered more than one object occupancy contain potential collisions. Efficient implementations of this basic idea exist which make use of dedicated computer graphics hardware to accelerate the process [Baciu 03].

Extensions of this uninformed spatial partitioning exist, where the object locations are also considered when subdividing the space. Some of these hybrid approaches include the use of oct-trees (e.g. [Moore 88], [Noborio 89], [Bandi 95]), binary space partitioning trees (BSP-trees) (e.g. [Naylor 90]) and  $k$ -d trees ([Held 95]) which are a special case of BSP-trees where the subdivisions are performed orthogonally to the coordinate axes.

With all time-discrete collision detection algorithms, special consideration is required in situations where the dimensions of the involved objects are in the same scale as the relative distances travelled by the objects. Whenever this occurs, either due to fine collision geometries or high velocities, the risk is induced to miss a collision because at time  $t_i + \Delta t$  the objects do not intersect - since they have travelled through one another (this is also referred to as *tunnelling effect*). The PhysX libraries, which are time-discrete, avoid the issue by also including continuous collision detection algorithms in a hybrid approach.

So far, an overview of discrete time collision detection algorithms has been given. The outcome of these algorithms is a set of contact points at which rigid bodies touch or intersect. Returning to the original formulation of non-penetration constraints, two fundamentally different types thereof can be distinguished which deal with *colliding contacts* and *resting contacts* respectively. When two bodies collide (i.e. they are in contact at some point and have a velocity component towards each other), a velocity discontinuity occurs which needs to be resolved with a dedicated mechanism outwith the dynamic model presented at the beginning of this section. In resting contacts, the bodies in contact do not display any velocity towards one another and hence no discontinuity of velocities needs to be resolved. In this case, a contact force needs to be computed which prevents the two bodies from inter-penetrating. Such forces can be seamlessly added to the known dynamic equations.

### *Contact Resolution*

The forces generated by non-penetration constraints act along the normal to the contact surface and are conservative since they induce no loss of energy and hence are reversible. The contrary is the case if the physics-based response of the contact is modelled including friction. Friction forces act tangentially to the contact surface and are opposed to the relative motion of sliding objects or prevent such motion altogether. In rigid body dynamics, friction forces are applied at the contact points. They are typically computed following the Coulomb friction model which states a relationship between the normal force and friction forces. For computing this

relationship a distinction is made between the case where the bodies are in relative sliding motion or at rest in relation to one another.

For polygonal shapes, numerous situations occur where conceptually infinitely many contact points exist. This is the case whenever coplanar surfaces or collinear edges are in contact, either as consequence of resting contact or more rarely, a collision. In penalty-force based methods, some intersection is allowed and hence collisions frequently generate a large number of contact points in situations other than the pathological cases described. For the following computations it is a requirement to obtain a finite set of contact points. Furthermore, it is desirable to reduce the number of contacts to the smallest possible representative set and then apply the required forces at those points. Generally, for polygonal contacts, this is achieved by considering only the bounding vertices of colliding surfaces and edges [Baraff 89].

In the following, the moment in time  $t_c$  when two bodies are in contact but do not interpenetrate is considered. The procedures described are independent of how such a state is reached: either by applying continuous methods, bisection or ignoring some amount of interpenetration using a penalty method. Further it is assumed that the correct set of contact points has been determined, consisting of vertex-face and edge-edge contacts (without considering degenerate cases).

The first step to resolve an individual contact is to determine whether a colliding contact or a resting contact is being dealt with. Let  $p$  denote a contact point between two rigid bodies  $A$  and  $B$  such that  $p_A(t_c) = p_B(t_c) = p$  specify the contact point on the surfaces of the two bodies. Without loss of generality let  $A$  contain the vertex  $\mathbf{v}_A$  and  $B$  the face  $\mathbf{f}_B$  for vertex-face contacts. The contact normal  $\mathbf{n}$  can then be defined as either the normalised outward pointing face normal of the collision face or the normalised cross product of the involved collision edges  $\mathbf{e}_A$  and  $\mathbf{e}_B$ , by convention also pointing outwards from  $B$  (cp. figure 3.5):

$$\mathbf{n} = \begin{cases} \frac{\overline{\mathbf{f}}_B}{\mathbf{e}_A \times \mathbf{e}_B} & \text{vertex - face contact} \\ \mathbf{e}_A \times \mathbf{e}_B & \text{edge - edge contact} \end{cases} \quad (3.11)$$

Using equation 3.4, the global velocities  $\dot{p}_A(t_c)$  and  $\dot{p}_B(t_c)$  of the contacting point on the two bodies can be computed. To determine the nature (colliding or resting) of a contact, the component of the relative velocity in direction of  $\mathbf{n}$  is examined:

$$v_{rel} = \mathbf{n}(t_c) \cdot (\dot{p}_A(t_c) - \dot{p}_B(t_c)) \begin{cases} < 0 & \text{colliding contact} \\ = 0 & \text{resting contact} \\ > 0 & \text{vanishing contact} \end{cases} \quad (3.12)$$

If  $v_{rel}$  is negative, the bodies are moving towards one another and hence colliding at  $p$ .  $v_{rel} = 0$  indicates a resting contact between  $A$  and  $B$ . A positive relative

velocity means the bodies are moving apart and the contact will cease to exist at the next time step - no further consideration is required in this case.

First, the case of colliding contacts is described. From equations 3.11 and 3.12 it becomes apparent, that if no immediate action is taken the bodies will interpenetrate at time  $t_{i+1}$ . The intervention requires an immediate change in velocities of  $A$  and  $B$  which contradicts the unconstrained rigid body dynamics model developed earlier in this section. Conceptually, an instantaneous change in velocity is achieved by applying an impulse  $J$  to the bodies. An impulse is an infinite force applied for an infinitely short time period. The induced change in linear velocity is

$$\Delta v = \frac{J}{M} \tag{3.13}$$

Applying the impulse to a point  $x_i(t)$  on a body results in an impulsive torque

$$\tau_{impulse} = (x_i(t) - x_{obj}(t)) \times J \tag{3.14}$$

where  $x_{obj}(t)$  denotes the position of the centre of mass of the body. Such a torque in turn affects the angular velocity of the body:

$$\Delta\omega = I^{-1}(t) \cdot \tau_{impulse} \tag{3.15}$$

To compute the frictionless collision behaviour of two rigid bodies in the situation described above, following relation is applied:

$$v'_{rel} = -c_r \cdot v_{rel} \tag{3.16}$$

where  $v'_{rel}$  denotes the relative velocity of the rigid bodies immediately after the collision and  $c_r$  is the coefficient of restitution which is defined such that  $0 \leq c_r \leq 1$ . A value of  $c_r = 1$  results in an inversion of the relative velocity which means the bodies perform an “elastic” collision (despite their rigid geometries) and no kinetic energy is dissipated in the process. For  $c_r = 0$ , the new relative velocity becomes  $v'_{rel} = 0$  meaning that all relative kinetic energy has been lost and a resting contact results between  $A$  and  $B$ . In the real world, the kinetic energy is converted into internal energy in at least one of the bodies, leading to plastic deformations and an increase in temperature. Both these effects are typically not simulated in rigid body dynamics, where the kinetic energy is simply made to vanish.

The effect stated in equation 3.16 is achieved by applying an impulse

$$J = \pm j \cdot \mathbf{n}(t_c) \tag{3.17}$$

to the two bodies where the scalar  $j$  is computed based on equations 3.4 and 3.13 - 3.16. The computation is simple (e.g. [Goldstein 02], [Baraff 97]) but too extensive for the purposes of this overview. Special attention must be given to situations

where multiple contact points exist between two colliding rigid bodies. If they are resolved sequentially, the order may influence the “global” behaviour of the collision.

Resting contacts are generally harder to resolve than colliding contacts for a number of reasons. For one, it is far more common that multiple contact points between two rigid bodies occur. For resting contacts, they must be resolved simultaneously to ensure a stable and smooth behaviour of the involved bodies. A similar relation to equation 3.17 can be defined for the resting case but an unknown force  $F_{i\perp} = f_i \cdot \mathbf{n}(t)$  perpendicular to the surface needs to be computed at each contact point  $i$  instead of the impulse to achieve smooth behaviour. The goal of applying  $F_{i\perp}$  depends not only on one condition (for colliding contacts, equation 3.16 relates the impulse to the relative velocity and coefficient of restitution) but three:

- i  $F_{i\perp}$  must prevent interpenetration
- ii  $F_{i\perp}$  must be repulsive (adhesion effects are disallowed)
- iii  $F_{i\perp}$  must disappear when the bodies separate (conservative forces)

The first condition is tackled by defining a function  $d_i$  for contact point  $i$  which indicates the distance between the bodies at that point. Enforcing non-penetration corresponds to  $d_i(t) \geq 0$  for  $t > t_0$ . By using the convention that  $\mathbf{n}$  points outwards from  $B$ ,  $d_i$  can be written for both vertex-face and edge-edge contacts as

$$d_i(t) = \mathbf{n}_i(t) \cdot (p_A(t) - p_B(t)) \quad (3.18)$$

Given that  $d_i(t_0) = 0$  and  $\dot{d}_i(t_0) = v_{rel} = 0$  (due to the fact that a resting contact is being considered), it is sufficient to ensure

$$\ddot{d}_i(t) \geq 0 \quad (3.19)$$

The normal acceleration  $\ddot{d}_i(t)$  depends directly on the contact forces, thus a constraint for fulfilling the first condition has been found.

The second condition can be reformulated as all contact forces  $F_{i\perp}$  having to act outwards. Given the convention that  $\mathbf{n}_i$  points outwards, the second condition can be written as a constraint on the scalars  $f_i$ :

$$f_i(t) \geq 0 \quad (3.20)$$

The third condition states that  $f_i$  must be zero when the contact is breaking. This can be expressed as

$$f_i \ddot{d}_i(t) = 0 \quad (3.21)$$

since  $\ddot{d}_i(t)$  only becomes non-zero when contact is breaking,  $f_i = 0$  must hold true then. When remaining in contact,  $\ddot{d}_i(t)$  is zero and consequently  $f_i$  can assume any value for equation 3.21 to be fulfilled.

In order to fulfill the three conditions (based on expressions 3.19 to 3.21) for computing contact forces in resting contact situations, a system of  $i$  equations for the  $i$  unknown scalars  $f_i$  needs to be solved. This can be done efficiently by quadratic programming algorithms as described in [Baraff 94].

So far, the conditions for non-penetration constraints in frictionless systems have been outlined. A friction model can be added to the simulation by defining a friction force at contact point  $i$  which acts tangentially to the contact surface. Let  $f_{i\parallel}$  be the magnitude of that force as well as  $v_{i\parallel}$  and  $a_{i\parallel}$  the magnitudes of the relative velocity and acceleration of the rigid bodies  $A$  and  $B$  in the tangent plane.

If the relative tangential velocity  $v_{i\parallel}$  is zero, a static friction model is applied. In cases where the sum of all forces on  $A$  and  $B$  is such that  $a_{i\parallel} = 0$ , the velocity  $v_{i\parallel}$  remains zero and the only condition for the friction forces is

$$-\mu_s f_{i\perp} \leq f_{i\parallel} \leq \mu_s f_{i\perp} \tag{3.22}$$

where  $\mu_s$  is the static friction coefficient. If the effect of all other forces induces  $a_{i\parallel} \neq 0$ , a static friction force results which is opposed to that acceleration. This can be formulated by two additional conditions:

$$a_{i\parallel} f_{i\parallel} \leq 0 \tag{3.23}$$

which ensures that the direction of the friction force is opposed to the impending acceleration and

$$a_{i\parallel} (\mu_s f_{i\perp} - f_{i\parallel}) = 0 \tag{3.24}$$

which forces  $f_{i\parallel}$  to have magnitude  $\mu_s f_{i\perp}$  for non-zero tangential accelerations.

In analogy to the conditions formulated in equations 3.19 to 3.21, following conditions on the normal force and acceleration can be stipulated:

$$f_{i\perp} \geq 0, \quad a_{i\perp} \geq 0, \quad f_{i\perp} a_{i\perp} = 0 \tag{3.25}$$

Compared to equations 3.19 to 3.21, equations 3.25 have a slightly different focus: instead of preventing interpenetration, the conditions are stated to avoid *inducing* a penetration of the two bodies. Equations 3.22 - 3.25 can be solved as optimisation problem or more practically for fast simulations using the algorithms described in [Baraff 94].

A kinetic friction model is applied instead of static friction when the relative velocity  $v_{i\parallel}$  of the two bodies at a contact point  $i$  is non-zero. The magnitude of the kinetic friction force is

$$|f_{i\parallel}| = \mu_k f_{i\perp} \quad (3.26)$$

with  $\mu_k$  the kinetic friction coefficient. The direction of the kinetic friction force is opposed to the direction of relative tangential velocity. Contrary to the case of static friction, now  $f_{i\parallel}$  is directly dependent of  $f_{i\perp}$  which can cause the resulting system of equations to have no solution. As a result, it may become necessary to apply impulsive forces to ensure correct behaviour. This issue is detailed in [Baraff 94], where also the outline of a solution algorithm is presented which includes both static and kinetic friction.

### *Joint Constraints*

Mechanical constructions typically encountered in robotics consist not of individual rigid bodies but contain multiple such objects connected by some joint mechanism. An unconstrained rigid body has six degrees of freedom (DOF); three translational and three rotational which fully define the set of possible motions. Likewise, the relative motion between two unconstrained rigid bodies can be described by the same six DOF. At a conceptual level, a joint restricts the relative motion of two rigid bodies in one or more degrees of freedom.

In terms of the simulation techniques described above, a joint can be seen as a (resting) contact point which is never allowed to break. This can be expressed in the framework of non-penetration constraints by reformulating expression 3.19 to be an equality:

$$\ddot{d}_i(t) = 0 \quad (3.27)$$

Equation 3.27 states that the normal acceleration of two rigid bodies  $A$  and  $B$  which are in contact at some point  $p_i$  is zero. Since the normal velocity  $\dot{d}_i(t)$  is also zero, the contact never breaks if the required normal forces are applied. In order to do so, constraint 3.20 must be dropped for being able to apply normal forces which not only prevent interpenetration but also separation. These modifications allow to use the same solver for simulation of non-penetration and joint constraints in a single algorithm execution.

Given the ability to enforce motion constraints as described above, a number of different joint mechanisms can be constructed, cp. [Kalra 95] for an overview. Figure 3.7 shows illustrations of four common joint types. A spherical joint (ball-and-socket joint) depicted in figure 3.7 a) allows a rotation about all three axes and has no translational degree of freedom. These constraints can be formulated exactly as in equation 3.27 which states that the translative motion of both bodies is identical and no restriction is imposed on relative rotational motion.

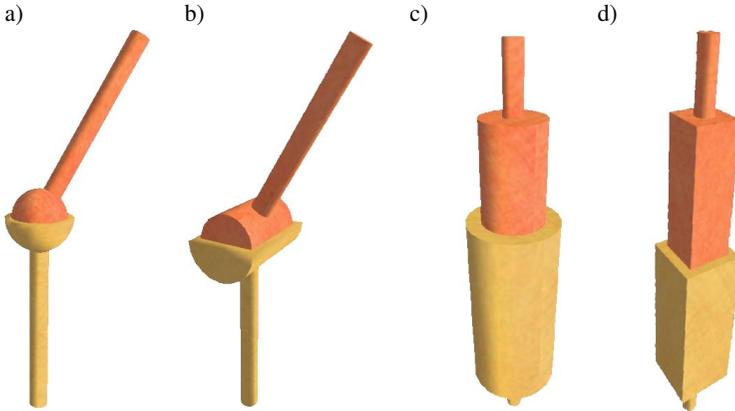


Fig. 3.7: Four common joint types used in rigid body dynamics simulations: a) spherical joint (3 rotational DOFs), b) revolute joint (1 rotational DOF), c) cylindrical joint (1 rotational + 1 translational DOF), d) prismatic joint (1 translational DOF).

By specifying two disjoint “contact points” as in equation 3.27 between two bodies, a single degree of relative rotational freedom remains, about the axis defined by the two points. This results in a revolute (hinge) joint (figure 3.7 b)), which is ubiquitous in mobile robotics, e.g. linking wheels to the body of a robot.

The constraint that two points should coincide can be extended by defining a general deviation function based on relative positions and orientations of the two bodies [Barzel 88]. Using two “point-on-line” constraints (itself a joint with three rotational DOF and one translational DOF) for two disjoint points and a single line, a cylindrical joint is created, cp. figure 3.7 c). Cylinder joints have one translational DOF and one rotational DOF about the same axis.

Combining a point-on-line constraint with an orientation constraint (which eliminates all relative rotational degrees of freedom) yields a prismatic joint as illustrated in figure 3.7 d). Prismatic joints allow motions only along a single translational degree of freedom.

Importantly, for designing joints it is not required that the “contact point” at which the above technique is applied (called the joint anchor) is located on the surface of both bodies. The same constraints can be enforced for arbitrary points which are defined at a constant position in the local coordinate system of the rigid bodies. In fact, doing so often allows the simplification of complex mechanisms by specifying the simulated joint at the location of the resulting virtual joint position of the mechanism.

An example is depicted in figure 3.8 where the two revolute joints used in figure 3.8 a) are simulated by a single joint in figure 3.8 b). Additional constraints

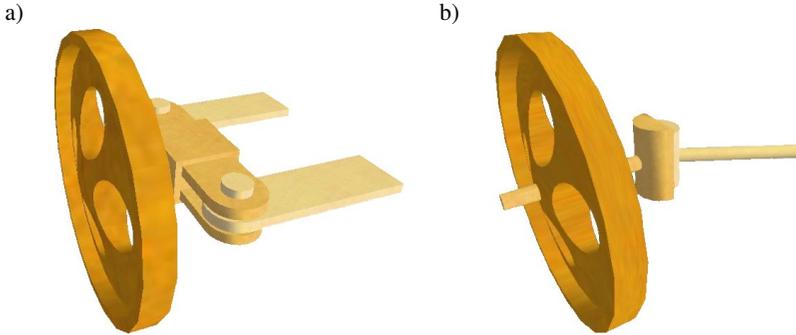


Fig. 3.8: Simplifications of joint mechanisms are often possible in rigid body dynamics simulations. The two revolute joints depicted in figure a) can be simulated by a single joint as shown in figure b).

on the joint motion allow to model the motion limits induced by the construction in figure 3.8 a). Without such limits, the wheel in figure 3.8 b) can describe a full circle around the axis represented by the joint visualisation. Often ingenuity is asked for from the model designer to obtain a faithful representation of reality in the simulation while striving to achieve maximal simplification to benefit computation speed.

At the beginning of this section a point of view was advocated where rigid bodies are considered as “black boxes” which provide the spatial variables position and orientation as time-dependent output and which can be influenced by applying forces and torques. Joints provide a further, comparable mechanism of interacting with the simulation. Depending on the type of joint, forces and torques can be exerted at joints which influence the relative motion of the joined bodies. Naturally, it is also possible to query the state of a joint (e.g. the position angle of a revolute joint). These features allow to implement realistic simulations of actuators. In particular, using the integration of Ibex into Simulink (cp. section 3.3.6), dynamic models of actuators can be conveniently developed and connected to joints in order to interact with the rigid body dynamics simulation.

### *Geometry Approximation*

Up to now, it has been assumed that rigid bodies are specified as polygonal mesh shapes consisting of vertices and edges connecting them to polygonal surfaces which form a closed shape. This is a general representation with which convex and concave solids can be approximated to any desired degree of accuracy. In many situations it is acceptable in terms of accuracy and beneficial in terms of computational costs to approximate the geometry of rigid bodies with primitive shapes or

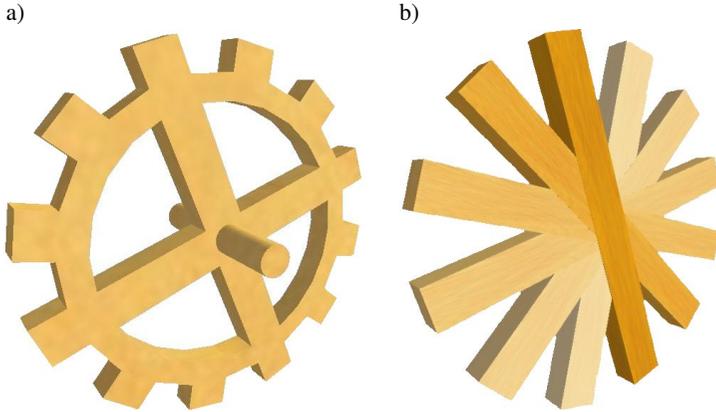


Fig. 3.9: Approximation of the desired geometry a) by a set of boxes (cuboids) b) which can be acceptable in many situations and leads to faster computational velocity.

collections of primitive shapes instead.

This possibility is offered by Ibex and the PhysX rigid body dynamics libraries contained. The available geometric primitives include cuboidal shapes (boxes), spheres and capsules (equidistant surfaces of a line, i.e. a cylinder with two hemispheres of equal radius at both ends). Figure 3.9 shows an example of a geometry approximation. The cogwheel shown in figure 3.9 a) is represented in the simulation environment as set of cuboids, cp. figure 3.9 b).

It is possible to combine all types of primitive shapes and even polygonal meshes for a single rigid body. The individual primitives can intersect (as in figure 3.9) or also be disjoint but nevertheless fixed with relation to one another.

This approximation technique together with the virtual axes introduced above often allows to significantly simplify a scene to be simulated. Depending on the model and the available resources, such techniques can help achieve some goal in terms of computational velocity (e.g. achieving real-time behaviour) which otherwise might not have been possible.

While these simplifications are beneficial for the physics simulation, it is often desirable to display more sophisticated visualisations than a rendering of the physical model. This has led to the implementation of a dual representation for rigid objects in Ibex. Each rigid object is composed of a single polygonal mesh file for visualisation and a set of “shapes” which define the physical characteristics. The same mesh used for visualisations can of course also be specified as physical shape. Additionally, texture files can be applied to enhance the visualisation. During the simulation it can be selected which view (“physical” or “visualised”) should be displayed.

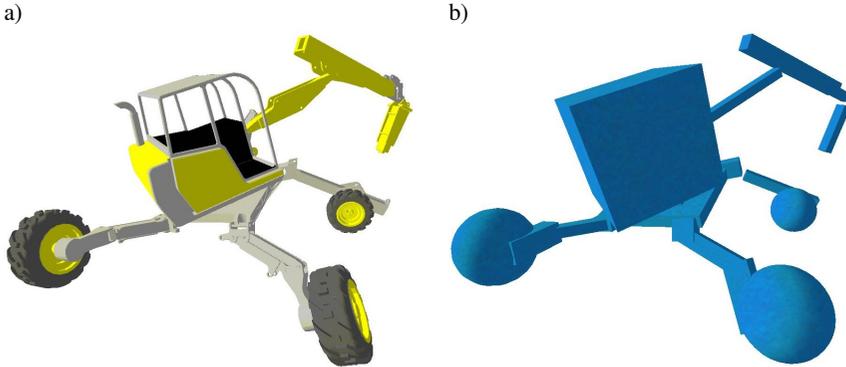


Fig. 3.10: Visualisation model of a construction machine a) and a corresponding significantly simplified physical model b).

Figure 3.10 a) shows the visual model of a construction machine simulated in Ibex. A highly simplified physical model thereof is shown in figure 3.10 b). Noteworthy aspects of the physical model are that the wheels (which are approximated as sphere shapes) are allowed to intersect with the supporting struts by permanently suppressing collisions between the involved bodies. Also, an example of joint anchors separated from the body surfaces can be seen at the connection between the tool-tip and the telescopic arm. The telescopic arm itself is an example of a prismatic joint.

### *Delta Robot Application*

To illustrate the concepts introduced so far in this section as well as to demonstrate the power of the implemented simulation framework a concrete application of Ibex is presented in the following. In cooperation with the Swiss Centre for Electronics and Microtechnology (CSEM) a simulation program for the MicroDelta [Codourey 04] robot of that institution has been developed. The MicroDelta is a small and fast high-precision Delta robot [Clavel 88] for micro-manipulation tasks. It has a parallel-kinematic structure with three resulting translational degrees of freedom for the tool-head. The structure consists of a closed loop mechanism with three chains of two links each. The chains are attached to a static frame parallel to the working surface. Each of the chains has a revolute joint at the “shoulder” position, actuated by an electric motor fixed to the static frame. The ends (“hands”) of the arms are joined at a travelling plate where tools like a gripper can be attached. A structure consisting of two parallel rods connects the “elbow” and “hand” of each arm. At each end, the parallel rods are attached to an axis parallel to the “shoulder”

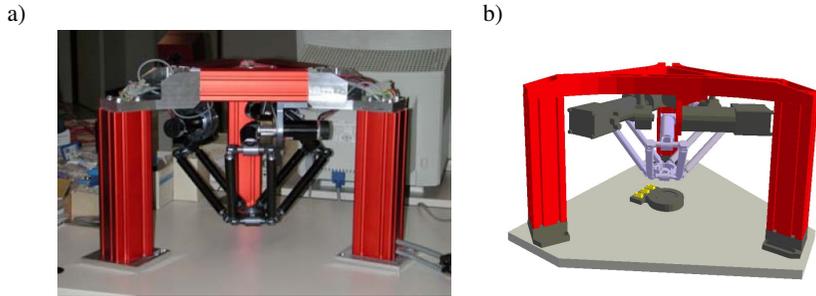


Fig. 3.11: View of the real MicroDelta robot a) and Ibx visualisation captured during a simulation b).

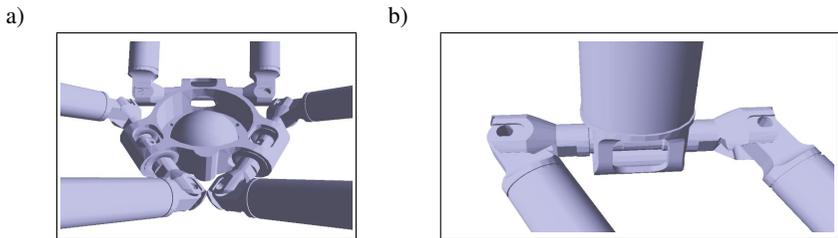


Fig. 3.12: Mechanical details of the MicroDelta robot: a) Tool-head joint construction, b) joints at “elbow” position.

joint by a revolute joint. This structure ensures that the tool-head always remains parallel to the working surface. Figure 3.11 a) shows the MicroDelta robot. A visualisation of the MicroDelta robot as displayed in an Ibx 3D graphics observer during a simulation can be seen in figure 3.11 b). The setup of joints at the tool-head is depicted in figure 3.12 a). The three revolute joints found at each “elbow” of the MicroDelta are shown in 3.12 b).

The goal of the project was to provide an environment to extensively test the control algorithms developed for the MicroDelta. The greatest challenge in this particular project was to achieve real-time behaviour at the high update frequency required by the controller. The final configuration consisted of running the controller software on external host hardware and communicating with it by means of a TCP/IP connection. This setup allows to run the Ibx simulation at a frequency of up to  $2k\text{Hz}$  while interacting with the controller in real-time.

### 3.3.3 Terrain Model

The focus of the present thesis in terms of motion planning algorithms lies in the domain of mobile robotics and more precisely rough-terrain planning. Therefore in particular the modelling of the robot-terrain interaction assumes great importance. In this section it is described how a special rigid body is designed which is used as terrain representation throughout chapter 4.

#### *Previous Work*

In robot motion planning a wide variety of terrain models have been proposed. Naturally, most thought has been given to terrain modelling issues in rough-terrain motion planning research. In [Gaw 86], a terrain model based on contour lines (“isolines”) approximated by polygons is presented. An  $A^*$  (e.g. [Nilsson 80]) algorithm is used to search a path between selected nodes of the polygons in either the same or adjacent height levels. Contour lines are transformed into a triangulated mesh surface in [Liégeois 91] and manually enhanced by inserting characteristic points such as saddle points or semantically rich locations (e.g. filling stations and points ensuring radio contact).

A digital terrain model is converted into a contour map for automatic feature extraction in [Kweon 91]. The topographic features extracted include peaks, pits, ridges and valleys. A connectivity tree is built which describes the possible trajectories between regions of the terrain. A similar approach is used in [Morlans 92] where motion planning is performed by searching the data structure using an adaptation of the  $A^*$  algorithm. Extending that notion, an irregular triangulation of a height-field based on characteristic terrain features is used in [Liégeois 93]. Additional information such as a friction coefficient and a risk coefficient is superposed as two-dimensional map.

A topology based on B-spline patches (e.g. [Mortenson 97]) is used in [Shiller 90a], [Shiller 91b], [Ben Amar 95], [Shiller 99] and [Shiller 00]. B-spline patches are parametric surfaces which offer a continuous representation of (small portions) of the terrain from which large terrains can be assembled. In [Shiller 91b], binary obstacles and regions of different physical properties are defined in rectangular areas of the parameter space which generates the surface. Navigational difficulty is characterised by a single variable  $\eta$  called “mobility factor” which ranges from  $\eta = 1$  for paved roads to  $\eta = 0$  for areas not accessible to the vehicle.

Non-triangulated polygonal terrains are used in [Siméon 91] for computing static placements of the robot. The underlying computation of the shortest path between two points on the surface of a convex-concave polyhedron had already been shown in [Mitchell 87].

A discrete height-field forms the basis of the terrain model in [Dacre-Wright 93], [Siméon 93], [Haït 96] and [Haït 99] where placements

of robots equipped with wheels passively suspended by springs are investigated to derive the free configuration space. Each patch of terrain spanned by four height data points is modelled as a generally non-planar surface which interpolates those heights. [Milési-Bellier 93] uses a similar model enhanced with generic material properties attached to each data point. In [Hebert 89], [Iagolnitzer 92] and [Nashashibi 94], the construction of such a model by means of different laser range finding devices is described.

In another approach, the 3D terrain originally given as regular height-field is approximated by a collection of tangent spheres [Jimenez 91]. Deformations of the terrain are computed as mass-spring systems of the spheres composing the terrain ([Cherif 93a], [Cherif 93b], [Jimenez 93], [Cherif 94], [Cherif 95]). Additionally, movable (detached) objects such as stones can also be included similarly. A finite state machine is applied to modelling the different types of wheel-surface adherence: no contact, gripping and sliding. In [Cherif 99a], [Cherif 99b], different terrain properties such as cohesion, deformation and friction are considered by splitting the terrain into discrete areas of constant physical properties. Binary obstacles are included as set of projected spheres on the  $xy$ -plane.

Using a wavelet decomposition, a hierarchical, multi-resolution terrain representation is computed in [Pai 98]. It is shown how the wavelet approximation error can be used as cost measure and also combined with other cost functions to determine a total cost measure.

In some related publications, no explicit terrain model is used to maintain generality. Most of these approaches rely on the definition of a regular grid of cells which are attributed a specific navigational difficulty. In [Iagnemma 99], the inclusion of uncertainty in a rough-terrain planner is emphasised. The  $A^*$  algorithm is used to initially find an optimal path. The cost function used considers the path length, “terrain unevenness” and “rover turning actions”. These measures of navigational difficulty are based on the “terrain roughness” in the surroundings of the robot.

A tip-over stability criteria for vehicles on uneven terrain is developed in [Papadopoulos 96] considering the position of the contact points which form the convex support polygon of the vehicle. Genetic algorithms are applied to compose an action plan from a discrete set of possible actions in [Farritor 98]; placements of the used robot determine the quality of actions.

While explicitly dealing with rough (Martian) terrains, [Laubach 98], [Laubach 99] and [Volpe 00] use a binary obstacle representation and omit details about vehicle-terrain interactions. In [Gennery 99], “terrain slope” and “terrain roughness” are computed similarly to the above publications.

A “traversability index” is defined on a regular grid in [Seraji 99], [Seraji 00], [Howard 01], [Seraji 02] and [Seraji 03] using fuzzy logic. Linguistic fuzzy sets for terrain slope and terrain roughness form the basis for the formulation which is

used in a fuzzy logic framework to determine navigation rules (comparable to an expert system). Physical properties of the terrain are included by defining a third fuzzy set named terrain hardness.

For application in rough terrain navigation, either a binary or continuous cost function is used on discrete regular cells of the terrain in [Yahja 00], where partially known worlds are also considered. An overall system for rough-terrain path generation and tracking is discussed in [Guo 03]. In the path generation phase, a binary distinction is made between challenging and benign terrain based on terrain roughness and terrain slope at discrete grid locations.

A number of typically sensor-based approaches exists for local navigational difficulty estimation. In [Simmons 95], a stereo vision system is used to evaluate the efficacy (expected roll and pitch angles) of travelling along local routes given by a discrete set of steering commands. Autonomous global planning capabilities are added in [Singh 00]. Local reactive behaviour based on stereo vision images is presented in [Kelly 98a] and [Kelly 98b]. Combined stereo vision and laser range-finding information is described in [Krotkov 97].

Publications which study the robot-terrain interaction in more detail include [Ben Amar 95], [Lamon 04] and [Thüer 06]. In [Ben Amar 95], a classical mechanical model of vehicle behaviour considering various types of wheel-soil interaction is derived. In [Lamon 04], a model including static, dynamic and roll friction is studied in the context of wheel torque control. While these approaches result in a high degree of fidelity, they are not applicable to the fast simulations being dealt with in this thesis due to the high computational complexity involved.

### *Ibex Terrain Model*

Terrains are represented in Ibex as polygonal mesh shapes statically anchored in space. The algorithm used to compute these special meshes is described in the following.

As in a number of publications listed above, a discrete regular height-field forms the basis of the terrain model. Terrain data is loaded from a file containing a list of 6-tuples. Each such data point consists of 3-dimensional Cartesian coordinates as well as values for static and kinetic friction coefficients ( $\mu_s, \mu_k$ ) and the coefficient of restitution  $c_r$ . Four neighbouring points span a generally non-planar rectangular surface. Each such surface can be triangulated by two different combinations of three points with either of the diagonals of the rectangle serving as common edge between two triangles. One diagonalisation direction is arbitrarily selected to generate the mesh, as is illustrated by the wire-frame representation of part of a terrain depicted in figure 3.13.

This terrain model is used to represent the environment of the robot. Nothing about the model used within the motion planner is stated here. Therefore, the arbi-

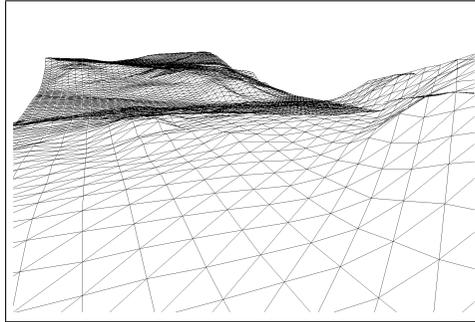


Fig. 3.13: Triangulation of the rigid body used to represent the terrain. The triangulation direction is arbitrarily chosen.

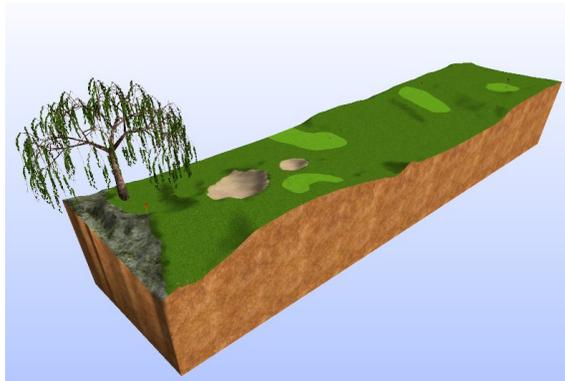


Fig. 3.14: Rigid body representing a terrain in a fictional golf course simulation. The different types of terrain (fairway, greens, rough, bunkers) used in the texture correspond to areas of different terrain properties. The tree and flags are separate rigid object.

trary direction of diagonalisation merely influences the topography of the “real” terrain. Importantly, it does not introduce any discrepancies between the “real world” model and the terrain model used by the motion planning routines.

After triangulation, the resulting surface is completed with side faces and a bottom face to yield a 3-dimensional volume as rigid object. A visualisation of such a rigid terrain body is shown in figure 3.14, where a texture has been applied to the geometry in order to reflect the various terrain properties.

The particular terrain depicted is taken from a fictional golf course simulation, hence different material properties are applied to the fairway, greens, bunkers and

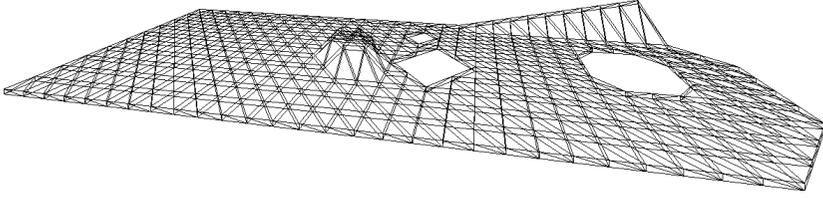


Fig. 3.15: A generated gallery mesh geometry for the rock-fall application. Protective structures can be approximated by convex-concave “roof” structures as illustrated.

the rough. Additionally, further rigid objects are also shown, such as the tree in the foreground and the flags at the holes. The material properties originally specified for each data point are mapped to the surface triangles as mean value of their constituting vertices:

$$c_i(\Delta_{p_1,p_2,p_3}) = \bar{c}_i + \delta_{c_i} = \frac{\sum_{j=1}^3 c_i(p_j)}{3} + \delta_{c_i}(\Delta_{p_1,p_2,p_3}) \Big|_{c_i \in \{c_r, \mu_k, \mu_s\}} \quad (3.28)$$

where  $c_i$  is a placeholder for the physical properties. To avoid an artificial uniformness of physical terrain parameters, optionally noise  $\delta_{c_i}$  following a random distribution can be added when computing the set of properties for a mesh triangle. A commonly used definition for the noise function is a truncated normal distribution such that

$$\delta_{c_i, \text{rand}}(\Delta_{p_1,p_2,p_3}) \sim \frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{x_i^2}{2\sigma_i^2}} \quad (3.29)$$

$$\delta_{c_i}(\Delta_{p_1,p_2,p_3}) = \begin{cases} \delta_{c_i, \text{rand}} & \text{if } 0 \leq \bar{c}_i + \delta_{c_i, \text{rand}} \leq 1 \\ 1 - \bar{c}_i & \text{if } \bar{c}_i + \delta_{c_i, \text{rand}} > 1 \\ -\bar{c}_i & \text{if } \bar{c}_i + \delta_{c_i, \text{rand}} < 0 \end{cases} \quad (3.30)$$

with  $\sigma_i$  the standard deviation for variable  $c_i$  in triangle  $\Delta_{p_1,p_2,p_3}$ . The auxiliary variable  $\delta_{c_i, \text{rand}}$  is the normal distribution representing the noise which is truncated as in equation 3.30 to ensure the convention  $0 \leq c_i \leq 1$  is fulfilled.

### Rock-Fall Application

The terrain model presented above was originally developed and validated in the course of a cooperation project with the Institute of Civil Engineering at the University of Applied Sciences of Central Switzerland. The project saw the application of Ibex to the domain of rock-slide and rock-fall simulation [Kister 05].

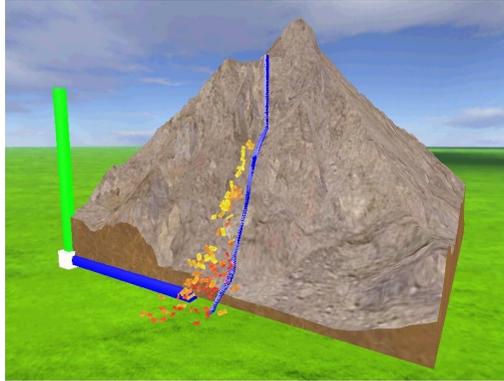


Fig. 3.16: The rock-fall application showing an example mesh of the Matterhorn with 100 cuboidal rocks - one of which draws a trace every fixed number of simulation time-steps.

The underlying motivation was the discussion triggered by the Swiss regulations for constructing protective structures for the transport network being due to expire. A series of incidents had occurred which revealed some flaw in the old directives. In some cases, protective galleries designed following the guidelines had been damaged or destroyed by events which they should have withstood. One particular issue studied by the civil engineers was the spatio-temporal interaction of rock impacts which could lead to unexpected vibration behaviour of the galleries.

To compute the trajectories of rocks, following four modes of motion were identified as relevant by the civil engineers: free fall, saltation (bouncing), rolling and sliding. A simulation application based on Ibex was developed which can be used to simulate such motions. Rocks can be specified as primitive shapes or mesh geometries. The terrain is loaded from a file as described above. Additionally, routines were developed which allow to specify the geometries of protective galleries similarly to the way terrains are loaded. Galleries are approximated by a mesh shape of constant vertical thickness. Figure 3.15 shows an example of a fictional gallery roof generated by the program.

The simulation creates a log file of the impacts registered between rocks and the gallery structure. All relevant data such as impact location, time, rock velocity, energy etc. is output to the file which is then analysed using external tools.

In the domain it still is commonplace to study the fall of a rock as isolated incident. Interactions between rocks are typically disregarded. A major benefit of the developed program compared to conventional techniques is that numerous rocks can be concurrently simulated. Additionally, simulation rates significantly faster than real-time can be achieved which is also untypical in the field.

Figure 3.16 shows a visualisation taken from the rock-fall simulation. The model loaded is the peak of the Matterhorn discretised with  $25m$  mesh size. This height-field is freely available as data sample from the Swiss Federal Office of Topography. The Cartesian coordinate file can be directly loaded into Ibex, in which case the material properties of the terrain need to be specified separately. For real analysis of a rock-fall site, the civil engineers use custom made height data with a significantly higher resolution. The simulated scene includes 100 cuboidal rocks of different physical material properties. In figure 3.16, the size of the rocks is out of proportion in relation to the terrain model (the length of the coordinate system axis representations is  $1000m$ ). While this is unrealistic, it illustrates the strength of simulations to model scenes which could not occur in reality. One of the rocks has been selected in the generic Ibex GUI to draw a marker every fixed number of time-steps; this can be seen as blue trace of its progress down the slope.

### 3.3.4 Sensor Simulation

So far, it has been described how simulations can be created which handle rigid body dynamics with constraints - namely non-penetration and joint constraints. Mechanical robot structures can be simulated with these techniques. Applying forces and torques to joints allows to simulate robot actuators. In order to model the complete control loop, a simulation of sensors must also be included. Various possibilities of sensor simulation in Ibex are presented in this section.

Localisation sensors such as satellite-based systems (e.g. Global Positioning System (GPS), Galileo) or more local solutions (e.b. trigonometry-based) can be easily simulated as “perfect sensors” by querying the position of a rigid body used to represent the sensor. To obtain a behaviour more reminiscent of reality, a noise function needs to be superimposed to model sensing errors. To compute a realistic error function, aspects such as satellite visibility or distance to beacons should be taken into consideration.

Attitude sensors like gyroscopes or a compass can be simulated likewise by accessing the orientation of some relevant rigid body. In the case of a gyroscope, the error should be accumulative, for a magnetic compass systematic or dependent on local magnetic field definitions.

Odometry sensors can be simulated by registering the accumulated rotation of a wheel. This perfect measurement (possibly discretised) is acceptable as e.g. an encoder can typically be assumed to provide accurate results. Furthermore, in reality the major source of error is caused by imperfect wheel-terrain interaction. This is automatically considered by applying an appropriate friction model in the rigid body dynamics simulation.

Contact sensors can be modelled by designing a collision geometry which represents the contact-sensitive device in the real world. Monitoring the collision ac-

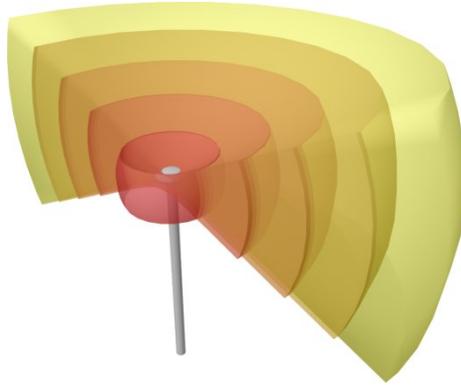


Fig. 3.17: Mesh-based trigger shapes (cutaway for illustration purposes) used to define the volumes covered by the five discrete detection ranges of a distance sensor.

tivity for that rigid body trivially approximates a contact sensor.

Synthetic vision techniques are used to simulate various types of camera. Synthetic vision was introduced in [Renault 90] for behavioural animation. The goal was to give synthetic actors a realistic model of knowledge (as opposed to e.g. global or radially restricted knowledge). This was achieved by producing an artificial image covering the field-of-view of each actor and containing distance information as well as the identities of objects seen “through” each pixel (the idea is similar to the setup shown in figure 3.19 for ray-casting). The original algorithm is extended in [Noser 95] to include a visual memory in which visually recognised bounding volumes of objects are stored for future reference. Navigation using potential-field techniques based on objects recognised in an artificially generated rendering of the world from the perspective of the actor is performed in [Blumberg 95]. A fast (real-time capable) synthetic vision system is presented in [Kuffner 99]. Saliency maps are used in [Courty 03] to design a biologically-inspired synthetic vision system.

Synthetic vision is not currently supported by Ibex. Nevertheless, since various graphics engines are included in the framework, the generated visualisations could be used as basis to implement such sensor simulations. Arrays of simulated ray-casting distance sensors (described later in this section) can already be used to generate distance images and access the identities of sensed objects.

In Ibex it is possible to simulate basic interference sensors such as light barriers or light curtains by using what is known as “trigger shapes” in the PhysX libraries. Trigger shapes are rigid bodies which only perform collision detection but no collision response. They serve to detect and report intersections and are perfectly suited to simulate sensors which produce a response to such occurrences.

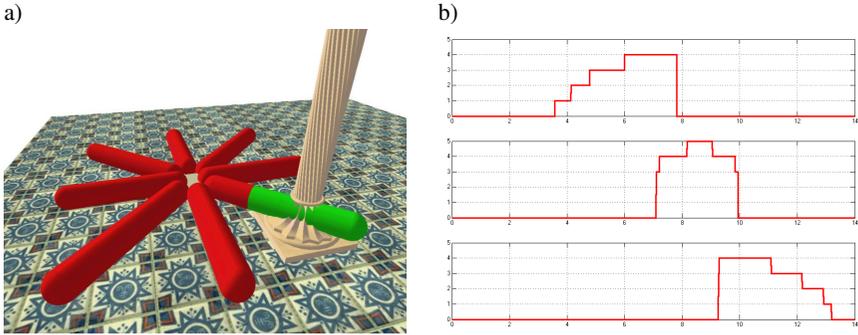


Fig. 3.18: a) Mobile robot surrounded by detection geometries of simulated ultra-sonic sensors. Each sensor consists of five discrete detection ranges specified as rigid geometries. b) Sensor output of the right-forward, right abreast and right-rear sensors as the robot passes the column from left to right in figure a).

Discrete distance sensors can be simulated by designing a number of overlapping trigger shapes which each represent a certain detection range. Figure 3.17 illustrates this concept by showing cutaway mesh shapes of such a device. In the example, five concentric trigger mesh shapes are positioned at the tip of a distance sensor. Intersections of any object with these shapes produce a signal from the penetrated triggers which indicate the (discrete) distance from sensor to object.

An example from a simulation making use of the discrete distance sensor simulation technique is shown in figure 3.18. A mobile robot has been equipped with eight sensors located around its periphery. Each sensor consists of five capsule-shaped discrete detection shapes. In figure 3.18 a), three of the right-forward triggers intersect with the column, hence generate a signal and are highlighted green in the visualisation. The plots in figure 3.18 b) show the output of the right forward, right abreast and right backward sensors as function of time while the robot passes the column from left to right seen from the perspective shown in figure 3.18 a).

Arbitrarily complex shapes can be defined for the triggers. A drawback is that the geometries themselves are static since they are simulated as rigid objects. It is conceivable (but not implemented at present) to switch between various pre-defined detection geometries, e.g. representing multiple modes of operation.

The sensor simulation based on trigger geometries approximates the behaviour of ultra-sonic sensors if the detection geometries are modelled as the detection lobe shapes of the sensors. In the next section, the focus lies on the simulation of a time-of-flight laser range-finding type of sensor. Similar arguments can be used with minor adaptations to simulate other sensors which likewise rely on wave propagation, cp. [France 99].

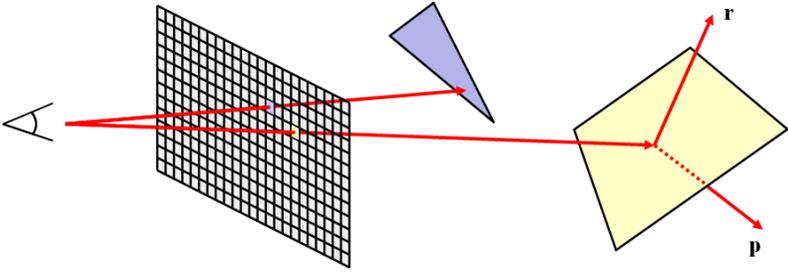


Fig. 3.19: Ray-casting principle: light rays are emitted from the position of the observer through each pixel of the rendering into the scene “behind”. The first object to be hit by a ray is the one rendered at the corresponding pixel. Light rays can also reflect ( $r$ ) and refract ( $p$ ) on surfaces.

### Ray-Casting Techniques

Ray-casting was first introduced for rendering 3-dimensional scenes in computer graphics [Appel 68]. The technique studies the trajectories of fictional light rays emitted from the position of the observer into the scene. For rendering a scene one light ray is emitted from the viewpoint through the image plane for each pixel. The first object hit by the light beam in the scene is the one displayed on that particular pixel, cp. figure 3.19. Based on the optical material properties of the surface and the positions of lights in the scene, the shading of the object is computed.

The original technique has undergone significant development since. In literature, the somewhat ambiguous distinction to ray-tracing is made. Ray-tracing was first introduced by [Whitted 80]. The idea is not to terminate the progress of the light ray at the point of first intersection but to recursively emit three types of ray at collision points. Reflected rays (vector  $r$  in figure 3.19) continue following a specular reflection law applied to shiny surfaces. The closest objects such rays encounter are displayed at the original point of impact as mirror images. Refraction rays (vector  $p$  in figure 3.19) progress similarly but pass through (and out of) transparent materials. Shadow rays are traced to light sources to determine whether the original impact point lies in shadows or is illuminated. An overview is presented in [Glassner 89], a summary of ray-object interaction algorithms in [Hanrahan 89].

Nowadays highly efficient algorithms for rendering large scenes have been developed, e.g. [Cazals 95], but for the intended sensor simulation task a simple model suffices. In the following, the terms ray-casting and ray-tracing are used interchangeably. Ray-casting has previously been applied to sensor simulation in [France 99] where it is used to approximate laser range finders, ultrasonic sensors and reflectance sensors.

### *Sensor Simulation using Ray-Casting*

Time-of-flight sensors (e.g. laser range finders, ultrasonic sensors) as well as reflectance measurement sensors (e.g. using infra-red light emitting diodes or laser light) are based on wave propagation. An emitted idealised wave follows a straight trajectory through space. It gets reflected on objects it encounters in dependence of the incidence angle and the surface properties of the object. In the following, the model used for simulating a laser range-finding device in Ibox is described.

Laser range-finders operate by modulating the emitted light wave and measuring the phase shift between the emitted and received waves. A real system is described in [Hebert 92] together with issues encountered in practice with such devices, a simulation is described in [France 99]. It is assumed that a sufficiently strong signal is received to determine the distance if the angle of incidence  $\vartheta$  at the first contact of the laser beam with an object is  $-45^\circ \leq \vartheta \leq 45^\circ$ . In Ibox, a more accurate model based on the computation of the sensed reflected light intensity is used.

An individual range-finder consists of an emitter and receiver. The emitter generates a laser beam of intensity  $I_0$  which does not display any beam divergence and has a cross-section of infinitesimally small surface (i.e. geometrically the beam is reduced to a line). The detector is co-located with the emitter, facing the same direction and is assumed to have a circular receptor surface of diameter  $d$ .

In general, reflections can be modelled as combination of specular and diffuse reflections. Perfect specular reflections (as would occur on a perfect mirror) obey the specular reflection law which states that the angle of incidence equals the angle of reflection. The beam is not dilated in the process and continues not displaying any beam divergence. In contrast, diffuse reflection spreads the incoming light over the entire hemisphere surrounding the surface of incidence.

In the implemented model, the reflection properties of the objects in the environment are computed following Lambert's cosine law, which models diffuse reflections. Specular reflections are neglected since situations in which perfect specular reflections return to the sensor in the setup described above are rare. By assuming the laser beam has an infinitesimally small cross-section, independence of the angle of incidence is achieved since the illuminated surface is reduced to a single point. Hence the reflected light can be thought to originate from a Lambertian point source located at the point of incidence. An illustration of the sensor model is shown in figure 3.20.

Lambert's law states that the light intensity  $dI$  emitted in an infinitesimally small solid angle  $d\Omega$  at an angle  $\vartheta$  from the surface normal is

$$dI = I_{\perp} \cos(\vartheta) d\Omega \quad (3.31)$$

with  $I_{\perp}$  the luminous intensity which corresponds to the intensity emitted in direc-

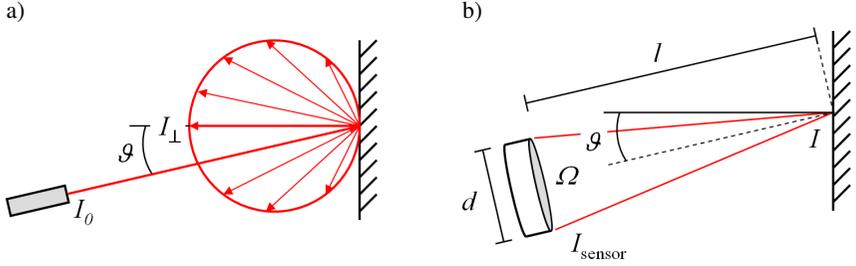


Fig. 3.20: Model of the ray-casting laser range-finding sensor simulation. A perfect laser beam is emitted and reflected on an object following Lambert's cosine law a). The light intensity received at the sensor is dependent on the distance from the point of reflection and the sensor surface  $\Omega$ .

tion of the surface normal at the point of incidence of the laser beam.  $I_{\perp}$  is the maximal intensity reflected in any direction by the Lambertian point source. The total light intensity reflected at the point of incidence is

$$I = \int_0^{\frac{\pi}{2}} I_{\perp} \cos(\vartheta) 2\pi \sin(\vartheta) d\vartheta \quad (3.32)$$

In equation 3.32, the solid angle  $d\Omega$  has been replaced by the solid angle which corresponds to a spherical segment (zone) of aperture angle  $d\vartheta$ . The integration from 0 to  $\frac{\pi}{2}$  results in the entire hemisphere of reflectance being considered.

The total intensity of the reflected light is related to the emitted intensity by

$$I = rI_0 \quad (3.33)$$

with  $r$  being the coefficient of reflection ( $0 \leq r \leq 1$ ), the only modelled optical material property of the environment. Solving equation 3.32 for  $I_{\perp}$  yields

$$I_{\perp} = \frac{I}{\pi} = \frac{rI_0}{\pi} \quad (3.34)$$

The light intensity  $I_{\text{sensor}}$  received at the sensor depends on the solid angle  $\Omega$  covered by the receptive surface in relation to the surface of a hemisphere of radius  $l$  - which is the total surface into which the light is emitted:

$$\Omega(l, d) = \frac{d^2 \frac{\pi}{4}}{\frac{1}{2} 4\pi l^2} = \frac{d^2}{8l^2} \quad (3.35)$$

In equation 3.35, the surface of the sensor is approximated by a (planar) circular surface. Combining Lambert's cosine law with equation 3.34 and equation 3.35 yields the light intensity received at the sensor:

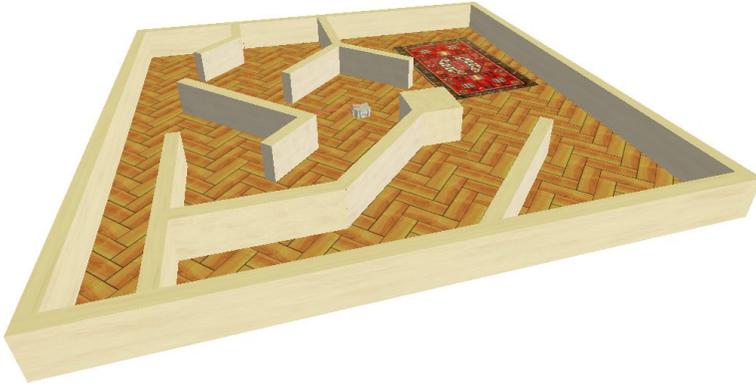


Fig. 3.21: A maze-like test environment for the laser range-finding sensor. A single sensor is positioned on a swivelling mount on top of the mobile robot in the centre of the maze.

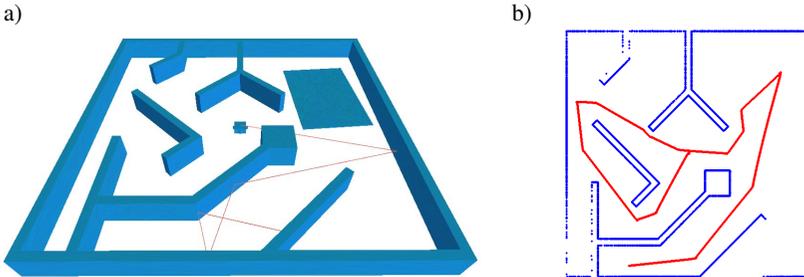


Fig. 3.22: a) Visualisation of the simulated laser beam. For illustration of the ray-casting technique, the beam is allowed to perform four specular reflections off objects in the environment. b) Map drawing performed by the robot using the physics-based model of a laser range-finder while following the plotted trajectory.

$$I_{\text{sensor}} = I_{\perp} \cos(\vartheta) \Omega = \frac{r I_0}{\pi} \cos(\vartheta) \frac{d^2}{8l^2} \quad (3.36)$$

When generating the sensor output, it is assumed that if sufficient light intensity is sensed, a correct distance value can be determined. If the intensity drops below a threshold, no output is generated. Sensing errors are not included in the model at present. Secondary reflections of the laser beam are also explicitly not considered.

A test environment for the laser range-finding sensor is depicted in figure 3.21. A single laser range-finding device is located on a swivelling mount atop the body of the robot and rotates around the vertical axis. For illustration purposes, the ray-

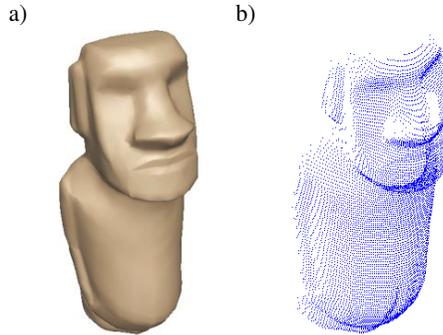


Fig. 3.23: 3-dimensional geometry reconstruction performed with the ray-casting sensor. a) The original geometry, b) a high-resolution scan taken from a single fixed sensor position.

casted beam is allowed to perform four specular reflections in figure 3.22 a) which shows the collision shapes of the setup. While the robot follows the trajectory plotted in figure 3.22 b), the measurements of the simulated laser range-finder are used to draw the map of the surroundings shown in the same figure. Each measurement of the sensor is used to draw a single point relative to the position of the robot (which is known to the robot, contrary to the environment geometry).

The same scanning technique can also be applied to probe 3-dimensional geometries. For rough-terrain motion planning, this is used to generate a sensor-based model of the terrain in the vicinity of the robot. This application is presented in section 4.2.2. An example of 3-dimensional geometry reconstruction is shown in figure 3.23 where a single ray-casting sensor has been used to scan a figurine from a static position. A high density of measurement has been taken to give a good impression of the sensor capabilities.

### 3.3.5 Simulation Content Tool-Chain

A significant bottleneck when setting up simulation experiments is caused by the lack of appropriate content development utilities. For rigid-body dynamics simulations as described in this document, the required content is often non-trivial to generate and is commonly assembled by using ad-hoc collections of various software tools each covering some specific field. Existing 3D design tools typically do not encompass the full functionality required in the simulation environment. Often, while some elements are present, they are not specified in a way which is directly useful for the simulation content generation.

A simple example is the format used to specify object orientations. While the transformations required to convert one format to another are mathematically not

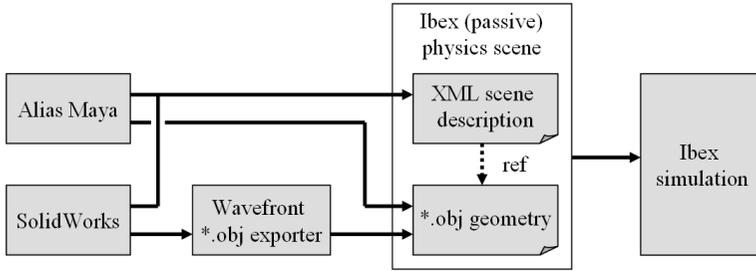


Fig. 3.24: Two main data-flow paths have been established in the Ibox tool-chain. Autodesk<sup>®</sup> Maya<sup>®</sup> as 3D modelling environment and SolidWorks<sup>®</sup> as CAD software have been extended to accommodate the design requirements posed by rigid-body dynamics simulations.

especially involved, the implementations are error-prone and the large number of similar but subtly different definitions needs to be carefully considered. Further difficulties arise when designing joints between rigid bodies. Most 3D design tools either do not contain the concept of such motion constraints or they are represented in a manner not directly applicable to the requirements of the simulation environment. Also the two “parallel” worlds of simulation and visualisation geometries described in section 3.3.2 must be represented in a user-friendly manner. In particular, physics-enabled design environments must provide an easy mechanism to create a set of collision shape primitives and assign them to a logical rigid body with an associated visualisation shape.

The fundamental elements of the simulation are rigid bodies composed of geometry descriptions and their physical characteristics. A first example of such an aggregation has been given in section 3.3.3 where it is described how a terrain rigid object is generated from dedicated files containing a list of data points. That procedure is specific to one particular type of rigid body (the terrain) and is based on a file format tailored to this purpose.

Clearly, that approach is not applicable to the generation of rigid bodies in general. For one, the individual data points cannot be assumed to be known for arbitrary shapes. Also, the triangulation which is implicitly given for a regular terrain height-field needs to be specified for general geometries (This process can potentially be automatised using e.g. a Delaunay triangulation [Delaunay 34]). Also, for terrains the creation of the physical geometry representation is trivial since the generated polygonal mesh is directly used as collision shape. A further difficulty arises from the fact that rigid bodies in general can be dynamic and hence require the specification of properties such as their density (or mass), the centre of mass position, inertia tensor etc.

To accommodate all the above mentioned design aspects, a tool-chain has been developed as part of the overall Ibex framework which considerably simplifies the generation of rigid-body dynamics simulations [Ettlín 05c]. The tool-chain consists of two major data flow paths which together cover a wide spectrum of design requirements, cp. figure 3.24. At the source of each data flow path a 3D design tool is used as host environment for Ibex-specific extensions. The two software suites which have been selected are targeted at different user groups. The first environment is Autodesk<sup>®</sup> (formerly Alias) Maya<sup>®</sup>, a 3D modelling environment which is popular in the animation industry and related fields. The second host software is SolidWorks, a major computer-aided design (CAD) package.

#### *Autodesk Maya Exporter*

Maya allows the fast creation of arbitrary polygonal geometries which can be exported natively to the wavefront object (\*.obj) file format used in Ibex for importing geometries into simulations. A number of design tools is available which makes Maya a powerful environment for post-processing \*.obj files from other sources. These features range from object triangulation and geometry reduction to the generation of UV-texture maps for more sophisticated visualisations. The direct support of \*.obj files has the additional advantage of giving precise control over the approximation degree down to the level of individual vertex positions.

The functionality of Maya is tailored to the needs of animation artists rather than mechanical engineers. A large degree of freedom is given to the modeller paired with powerful tools for generating visually appealing objects. An effective option when modelling with Maya is the use of a scripting language integrated in the package. In particular when developing repetitive designs, many modelling tasks can be automated instead of manually executed. On the downside, the kind of high-precision 3D design provided by a CAD package cannot be expected of Maya.

Basic rigid body dynamics simulation capabilities are natively contained in Maya. Since the Maya definitions differ significantly from the specifications used in Ibex, they have not been reused when designing the plug-in. The main tasks of the Maya extension are the generation of rigid-body approximation primitives as well as joint specifications and visualisation thereof in the user interface.

Figure 3.25 shows the setup depicted in figure 3.8 being designed in Maya with the Ibex extensions. The single revolute joint used to approximate the steering mechanism is visualised as well as a geometry approximation for the wheel by means of an ellipsoidal mesh shape.

#### *SolidWorks Exporter*

In the projects conducted so far with Ibex, Maya is primarily used for rapid prototype design apart from the \*.obj file post-processing described above. To of-

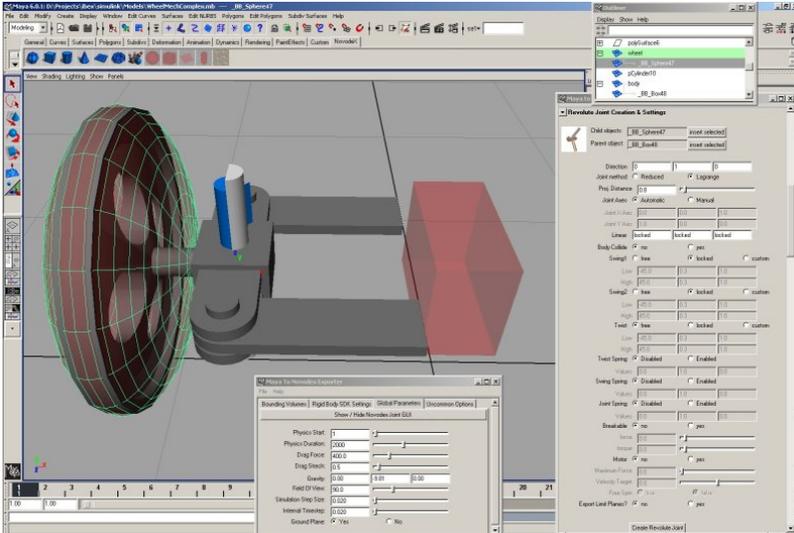


Fig. 3.25: The user interface of the Ibox-specific extensions integrated into Maya. The wheel construction depicted in figure 3.8 is shown with the simplified single axis and an ellipsoidal mesh geometry approximation for the wheel.

for the CAD functionality mechanical engineers are accustomed to, a content flow path from SolidWorks into Ibox has been developed. SolidWorks is a comprehensive CAD program which allows the precise construction of geometries defined by means of design constraints.

SolidWorks makes use of positioning constraints to precisely specify the extensions and locations of objects during the development process. These constraints conceptually reduce the degrees of freedom of the rigid bodies being designed. Therefore, it would fundamentally be possible to use these constraints for defining the free degrees of freedom of a joint between rigid bodies. From a theoretical point of view, this might seem desirable since it represents the construction procedure the designer is accustomed to. In practice though, the experience with Ibox has shown it is more convenient to distinguish between the design of the object geometries and the explicit definition of the motion constraints to be simulated. Therefore a dedicated interface has been added seamlessly into the SolidWorks GUI which allows to specify Ibox joints.

A distinction between parts and assemblies is made in SolidWorks. Parts represent the geometry of individual objects while assemblies unite multiple parts to an overall composition, e.g. a mechanical setup consisting of various components. The SolidWorks exporter by default assigns the potentially complex geometries of

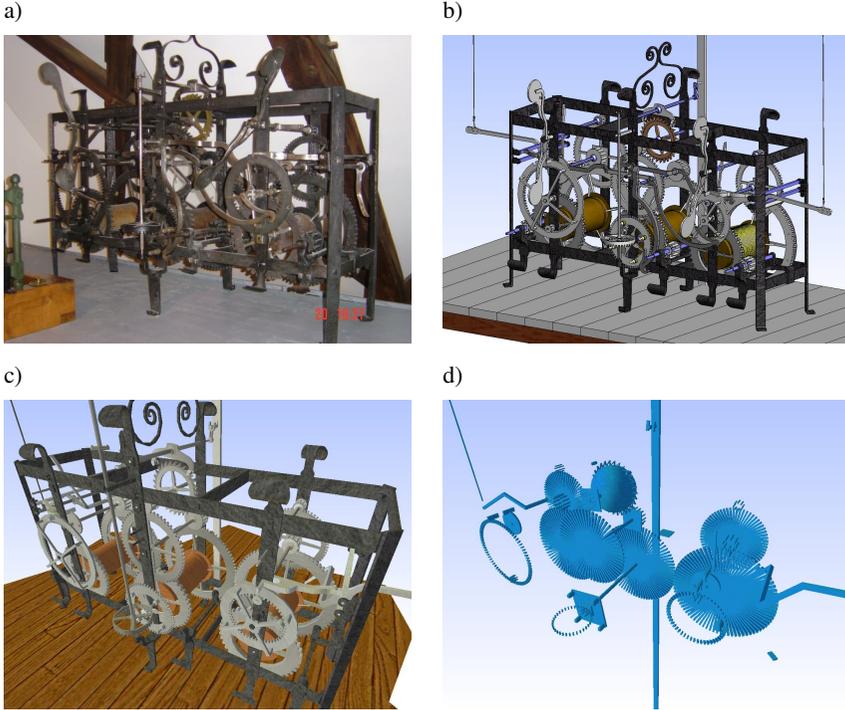
a part to the Ibex visualisation geometry. Each part is assigned a set of physics properties which include the list of collision shapes. All information generated by the Ibex extension is saved as user data in the original SolidWorks file. This approach ensures a consistent coupling of rigid body dynamics information with the logical CAD components while not affecting the further reusability of the design for other purposes. Full access to the properties of the physical shapes is given through a dedicated graphical user interface integrated into SolidWorks.

SolidWorks does not natively support the generation of wavefront \*.obj geometry files as required by Ibex. Furthermore, SolidWorks is spline-based rather than using a vertex-polygon representation like the \*.obj format. Therefore, a discretisation of the geometry needs to be performed by the Ibex extension when exporting an assembly. To this end, a simple \*.obj file exporter has been developed and integrated into SolidWorks which uses native sampling and discretisation routines.

Both the Maya and the SolidWorks exporters generate as output an XML file following a format specific to Ibex as well as the \*.obj files for geometry definitions and optionally image files for texturing objects in the visualisations. In section 3.1 it was stated that existing file formats should be supported where possible. The choice was made to nevertheless develop a dedicated format for Ibex since no existing definition fulfilled all the requirements at the time of development. For pure rigid body dynamics setups, a number of formats exists which are typically linked to an individual rigid body dynamics library. These formats lack features required in robotics simulations, such as sensor or actuator definitions. Further, it would be unwise for a simulation framework like Ibex to make a commitment to a specific rigid body dynamics library at the level of the data files as this might restrict the options of further development. An effort is being made with the open-source Collada XML standard to create an interchange format between applications for 3D assets [Col 06]. As part of version 1.4 (released in 2006, some time after the Ibex file format was specified), a physics simulation file format is included which defines basic jointed rigid body models. This development should be closely followed and taken into consideration in future extensions of Ibex.

### *Medieval Clockwork Example*

Figure 3.26 a) shows the complex mechanics of a medieval clock exhibited at the Lucerne Museum of History [HML 06]. The clockwork was originally contained in one of the towers of the town wall of Lucerne. The historically valuable artifact cannot be operated nowadays without risking damage to its structure. Therefore, the clockwork is being faithfully recreated as virtual model and simulated in Ibex in an effort to create an interactive museum exhibit which illustrates the operation of the clock to museum visitors. A model of the clockwork designed in SolidWorks and enhanced with rigid body dynamics properties using the Ibex extension



*Fig. 3.26:* a) Medieval clockwork exhibited at the Lucerne Historical Museum. b) Physics-enhanced model of the clock designed in SolidWorks using the Ibex extensions. c) The same model simulated and visualised in Ibex. d) View of the physical approximation exported from SolidWorks and visualised during an Ibex simulation.

is depicted in figure 3.26 b). A visualisation of the simulated clock in operation is shown in figure 3.26 c). The cogwheels of the real clock are of a rudimentary design, mostly containing cuboidal cogs instead of e.g. an involute profile. These simple geometries can be approximated by cuboidal and capsule shapes in Ibex - precisely modelling the original design in the relevant contact regions. The simulated physical design is shown in figure 3.26 d) where the high number of individual collision shapes can be appreciated.

The medieval clockwork relies on precisely calibrated collisions for correct operation. The successful design and simulation of such an intricate mechanism demonstrates the capabilities of the tool-chain established in Ibex. One aspect which has not been considered so far is the inclusion of active components which apply forces and torques to the system. The dynamic behaviour of active components is most conveniently modelled in dedicated dynamic modelling software. Fol-

lowing the same argument employed when integrating Ibex-specific extensions into 3D modelling tools, the following section describes how Ibex has been embedded in a dynamic modelling and control engineering software, MathWorks Simulink<sup>®</sup>.

### 3.3.6 *Integration into Industry-Standard Dynamic Modelling Environment*

So far, the simulation functionality of Ibex has been described and examples of the visualisation capabilities shown in the illustrations. This functionality is included in a software library which can be linked into stand-alone application programs.

Stand-alone programs are typically specific to a single task for which predefined physical setups are loaded and simulated. This is partly due to the fact that active components are controlled by algorithms within the compiled program code, thus limiting their flexibility. The processing of sensor data is governed by the same limitations. The user interface (which e.g. is an extension of the Ibex GUI) is typically also designed for a single application.

While such stand-alone programs are well-suited as final project outcome, they clearly lack flexibility during the development process of an overall mechatronic system consisting of a passive mechanical setup, actuators, sensors and controllers.

A well-established software for modelling and simulating dynamic systems as well as designing control algorithms is MathWorks Simulink<sup>®</sup> which is based on the MATLAB<sup>®</sup> numerical computing environment. Simulink has a visual programming interface in which blocks represent functions which operate on data flows modelled as connections between them. Numerous predefined blocks from a variety of domains such as control engineering and signal processing exist which make Simulink a versatile environment for rapidly designing and simulating dynamic systems.

Simulink is designed with a block library architecture which can be extended to include user-defined block libraries. User extensions can be written among other languages in C/C++ and the native MATLAB M language. To integrate Ibex into MATLAB/Simulink, a C++ block-set has been developed. This allows to co-simulate Ibex rigid body dynamics with all features available in Simulink simply by using the Ibex blocks in the same way native Simulink blocks are used. As part of the integration, the Ibex graphics observer based on the Nebula 2 engine has also been included into the Simulink user interface.

MathWorks markets a library named SimMechanics which also provides rigid body dynamics simulations. In comparison with Ibex, its simulation kernel lacks collision detection and dynamic collision response based on physical properties. Instead, merely kinematic constraints can be imposed on the system. SimMechanics also lacks any sensor simulation functionality beyond accessing the spatial variables of bodies and joints. To obtain visualisations comparable to those offered by Ibex, an additional MATLAB extension (the Virtual Reality Toolbox) must be in-

stalled. For simulation content generation, an exporter for SolidWorks is provided by MathWorks. It parses a CAD assembly and generates an XML configuration file which can be imported in SimMechanics. The information contained in the file includes mass and inertia properties of each part as well as joint definitions. No information about the geometries or induced motion constraints is contained. For visualisations, the design must be exported separately using a VRML format.

The Ibex Simulink integration contains various categories of blocks. System blocks contain fundamental functionality and form the basis of any Ibex simulation in Simulink. One such block contains the core simulation functionality and the XML scene file loader. A second one encapsulates the Nebula 2 graphics observer: when the Nebula 2 block is placed on a Simulink model, the graphic engine opens a window when executing the model which contains the on-line visualisation of the scene. Since the graphics engine is executed in a separate thread, the visualisation is updated and can be inspected also when the simulation is paused.

A second category of Ibex blocks gives access to simulated entities. This includes a block which represents a rigid object in the black-box abstraction described in section 3.3.2. The output of the block consists of signals which represent position, orientation as well as the linear and angular velocities. To provide easy access to the fundamental rigid body characteristics of mass and inertia, these (time-invariant) quantities are also output as signals. The rigid body can be influenced by input signals which represent forces and torques applied to the body.

The blocks giving access to joints function in a similar manner: for e.g. a revolute joint, its Cartesian position, angle (local joint position) and the orientation of the main joint axis are output. The inputs are designed in a way which allows either to directly apply a torque to the joint or use a simple controller provided as standard joint functionality by PhysX. This controller takes as inputs a desired rotational joint velocity and a maximal torque available. A further parameter specifies whether the joint should be allowed to surpass the specified velocity unhindered or whether a torque should be applied to prevent this from happening. Other joint types function similarly but do not contain any integrated controller functionality.

A next class of blocks allows to create rigid bodies from the Simulink user interface. Such blocks exist for the basic shape primitives (sphere, cuboid, capsule) as well as for general mesh-shaped rigid objects. This functionality roughly corresponds to the mechanism used in SimMechanics for creating rigid bodies, enhanced by the specification of collision geometries. It has been found that within the overall work-flow rigid body creations from the Simulink GUI are well suited for simple objects e.g. used for debugging a construction by adding obstacles. For more complex setups it has proven to be more convenient by far to use the 3D development environments discussed in section 3.3.5.

An exception are specialised blocks used for the creation of terrains and environments in buildings. The terrain creation block operates as described in section

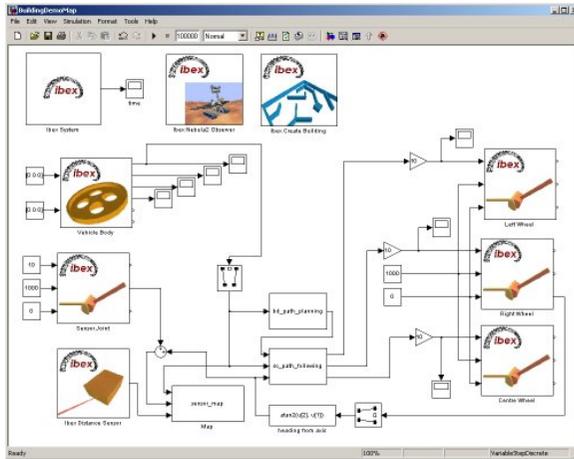


Fig. 3.27: The Simulink model used to control the map-drawing example illustrated in figure 3.22.

3.3.3 and inserts a static mesh-based rigid body representing a terrain, optionally with different material properties assigned to each triangle of the mesh shape. The building environment creation block is a convenient utility for creating a set of surfaces, walls and ramps which approximate the structure of buildings. The block parses a simple dedicated XML file and creates box-shaped rigid bodies with the specified properties. The maze-like environment shown in figure 3.21 was created using this block.

Trigger shapes (cp. section 3.3.4) can also be created from the Simulink GUI. The single output port of these blocks indicates whether the geometry of the trigger intersects with any other shape in the simulation. Optionally, a list of rigid bodies which are excluded from producing an output can be passed to trigger blocks. This can be important to avoid false reports of intersections. As example, the reporting of intersections between the sensor shapes and wheels of the robot shown in figure 3.18 a) has been suppressed.

Contact notification blocks are used to monitor a pair of rigid bodies and report contacting conditions between those objects. An example application of these blocks is the simulation of contact sensors. Another use from mobile robotics is the detection of breaking ground contact of the wheels.

Further, there exists a block which outputs the measured distance of a range-finding sensor simulation following the physical model introduced in section 3.3.4.

The Simulink model used to drive the map-generation example illustrated in figure 3.22 is shown in figure 3.27. In the top row, the three blocks are the Ibex system block, the Nebula 2 graphics observer block and a building environment

generation block respectively. Vertically aligned on the far right are three rotational joint blocks for accessing the three joint axes of the robot wheels. The block displaying the same symbol near the left margin controls the axis which supports the sensor casing atop the robot. Above it, a rigid body access block provides the spatial coordinates of the robot chassis. In the bottom-left corner of the model, a distance sensor block outputs the readings of the simulated laser range-finding device.

All other blocks in the model contain standard Simulink functionality. In particular, the path planning and trajectory tracking modules are programmed using the M language. Also the 2D plot of the sensor readings shown in figure 3.22 b) is drawn by an M-file. An advantage of this approach (and a benefit of the Simulink integration) is the rapid development resulting from using the M language. This is partly due to the simple yet powerful nature of this scripting language and partly given by the large amount of readily accessible MATLAB functionality. A drawback is the relatively slow execution time of the interpreted M language in comparison with compiled programming languages such as C/C++. Especially the MATLAB graphics functionality has proven to be a significant bottleneck to system performance. In the map drawing example, the plotting of the two-dimensional map consumes more resources than the Nebula 2 graphics engine.

In general, the control logic of a robotic system (and other user-defined functionality) can be developed in Simulink using the M, C/C++, Java, Fortran and Ada programming languages. Alternatively, the functionality can be developed using standard Simulink blocks in the graphical programming environment.

A high level of flexibility is given when co-simulating Simulink models with Ibex. Simulink can be freely configured to suit the requirements of the simulated problem. The options include fixed-step and variable-step integrations with a number of different integration schemes. Ibex operates transparently with both fixed and variable time-steps in Simulink, itself always running with fixed time-steps. The Ibex module queries the Simulink simulation time when it gets called and advances its simulation to the appropriate time-step in its own time-series. This time is selected to be the Simulink time if possible or the last Ibex time-step before it otherwise. Clearly, this scheme performs best when the Simulink step size and the Ibex step size are multiples of one another. In other cases correct simulation behaviour is also obtained but the Ibex readings do not correspond to the Simulink time whenever the time-steps do not coincide.

An important external interface is given by the Real-Time Workshop<sup>®</sup> (RTW) extension of Simulink. The RTW allows to automatically generate target platform-specific program code from a Simulink model and compile it into a real-time application for that hardware. This capability permits to e.g. design and test a controller benefitting from the Simulink functionality and directly deploy the same controller onto a real-time hardware target. Since user-defined blocks are excluded from the

RTW functionality, it is not possible to e.g. compile Ibex for a real-time target.

From a rigid-body dynamics simulation point of view, the extension of Simulink to include Ibex gives access to the complete MATLAB/Simulink functionality. In particular, being able to perform co-simulations with Simulink models allows to perform multi-domain dynamic system simulations. A typical application is the simulation of actuator characteristics in Simulink. The output of the Simulink actuator subsystem are forces and torques which are applied to the mechanical structure being co-simulated in Ibex. Sensor readings can be output from the Ibex simulation as input to the control logic simulated in Simulink, thus closing the feedback loop. This setup allows to simulate complete mechatronic systems, each component being located in an ideally suited simulation environment.

From a Simulink user's point of view, the inclusion of Ibex simulations gives access to a powerful rigid body dynamics toolbox which includes a content generation tool-chain integrated into well-established engineering solutions. As an added benefit, collision detection, collision resolution, sensor simulations and the included visualisation capabilities can be named.

### 3.4 Validations and Performance

To illustrate the kind of precision which can be expected of Ibex simulations and the included Ageia™ PhysX™ libraries in particular, a validation of simulation results is presented in the first part of this section. Thereafter, a performance analysis of various simulation setups is presented. For all testing, version 2.2.0 of the PhysX libraries has been used.

#### 3.4.1 Validation of Simulations

The validations of simulation results are performed by studying relevant physical phenomena in isolation. Simple physical setups are used to investigate unconstrained rigid body dynamics, the effects of interpenetration between two colliding bodies, the characteristics of kinetic and static friction as well as rolling motion.

##### *Unconstrained Motion*

The behaviour of unconstrained rigid bodies has been validated by analysing the ballistic trajectories of objects which are created moving at some initial velocity. For a rigid body released at a given initial linear velocity  $v_0$  with a positive elevation angle  $\alpha$ , the peak height of the ballistic trajectory  $h_{peak}$  and the horizontal distance  $x(T)$  travelled before returning to the initial height level are computed as follows assuming no aerodynamic drag:

$$h_{peak} = \frac{v_0^2 \sin^2(\alpha)}{2g} \quad (3.37)$$

$$x(T) = \frac{v_0^2 \sin(2\alpha)}{g} \quad (3.38)$$

These analytical solutions have been compared to simulation results for integration time-step values of  $\Delta t = 0.1ms, 1ms$  and  $10ms$ . The same time-step values have been used throughout the validation experiments since they cover the range of feasible settings for PhysX simulations. The simulation results obtained correspond with the analytical solution to within numerical and discretisation precision. This naturally leads to higher precision being achieved at smaller time-steps. Conservation of energy can be observed for unconstrained motion considering both linear and angular velocities:

$$E_{total} = E_{pot} + E_{kin lin} + E_{kin rot} = mgh + \frac{mv^2}{2} + \frac{I\omega^2}{2} = \text{const} \quad (3.39)$$

where  $I$  represents the inertia tensor corresponding to the rotation axis and  $\omega$  the angular velocity of the body about that axis. The simulation results for unconstrained motion of rigid bodies match the expected behaviour at a fidelity which renders superfluous further analyses. The following sections deal with the more interesting cases where dynamic constraints apply to the motion of the rigid bodies.

### *Rigid Body Collision and Rebound*

To study the behaviour of colliding rigid bodies, a setup is employed where one object is dropped onto a second one which is statically anchored in the scene. The basic setup consists of a sphere shape with radius  $r = 1m$  dropped onto a plane from a height of  $h = 10m$ . The experiment has been repeated for 20 different time-step settings ranging from  $\Delta t = 0.1ms$  to  $\Delta t = 50ms$ . The largest time-steps are outwith the practicable range but are nevertheless included to illustrate the degradation of precision.

The collision resolution behaviour is determined by the coefficient of restitution which specifies the proportion of kinetic energy remaining in the objects after the collision. Analytically, an energy conservation formulation is used, again disregarding air drag. The initial energy of the object can be computed as

$$E = E_{kin}(t_{first\ impact}) = E_{pot}(t = 0) = m \cdot g \cdot h(t = 0) \quad (3.40)$$

Similarly, the energy after the first impact can be formulated as:

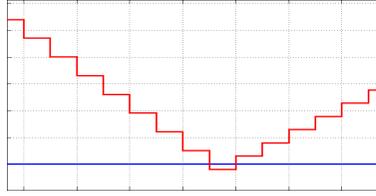


Fig. 3.28: Plot of the object's height above ground as function of time during collision resolution in the rigid body collision experiment. The general characteristics can be observed during all collision experiment runs. Importantly the interpenetration lasts no longer than one time step.

$$E' = c_r \cdot E = E_{pot}(t_1) = m \cdot g \cdot h(t_1) \quad (3.41)$$

with  $c_r$  the coefficient of restitution and  $t_1$  the time at which the object reaches the maximal height after the first rebound. In the setup described above, the coefficient of restitution directly specifies the height of the first rebound:

$$h(t_1) = \frac{c_r \cdot m \cdot g \cdot h(0)}{m \cdot g} = c_r \cdot h(0) \quad (3.42)$$

Hence, in the experiment the height of the first rebound is measured for different time-step sizes. The second measure used to determine the quality of collision resolution is the maximal amount of penetration on collision. Figure 3.28 shows a typical plot of the height of the object above ground as function of time which is found in this form for all performed collision experiments. The two bodies can be observed to intersect for no longer than one time step.

The plot shows that the penetration depth depends not only on the selected time-step size but also on the height above ground at the last time-step before the interpenetration. This height can be beneficial or detrimental to the penetration depth, depending on the relative approximation velocity of the two bodies. Figure 3.29 a) shows how the error on rebounding as well as the penetration depth deteriorate with increasing time-step size. The depicted plot has been generated from the experiment involving a sphere dropping on a plane. The equivalent experiments have also been conducted for all combinations of a mesh collision shape approximation of the sphere and a cuboidal primitive (all edge lengths  $l_i = 2m$ ) dropping on both a plane and a cuboid. The results for all experiments are virtually identical, the largest deviation in terms of rebound height being  $0.1mm$  and for the penetration  $4.5mm$ . Varying the restitution coefficients also results in equivalent outcomes of the experiment.

An interesting observation is that the rebound height is consistently too high in all measurements, on average by 1%. A possible hypothesis to explain this

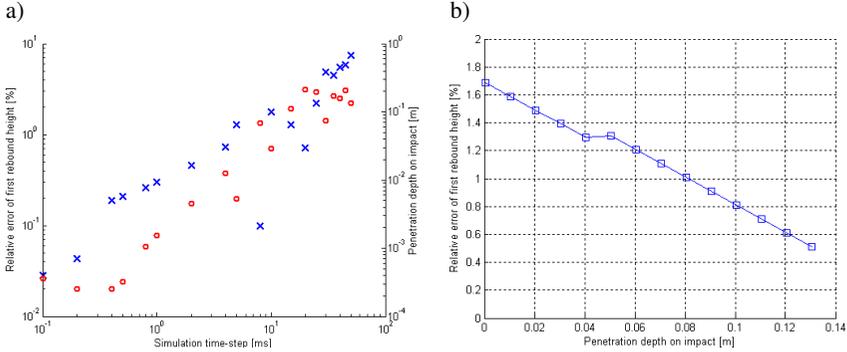


Fig. 3.29: a) Validation results for saltation test using a sphere of radius  $1m$  dropping onto a ground plane from  $10m$  height. The plot shows the relative rebound height error after the first impact (circles, left ordinate axis) and the penetration depths on impact with the ground plane (x-markers, right ordinate axis) as function of the selected simulation time-step. b) Study of relative rebound error as function of penetration depth for a time-step of  $\Delta t = 10ms$ .

energy gain is that the penalty forces applied during collision resolution are not completely compensated by the solver. To follow-up this issue, the relative rebound error is plotted against the penetration depth in figure 3.29 b) for a static time-step size of  $10ms$ . The initial height is varied minimally to control the amount of penetration. The shown plot is obtained from the sphere-on-plane experiment but is representative for all object pairs studied. The results are contrary to what might be expected, as smaller penetration depths lead to larger rebound errors. The figure also gives an idea of the variations which could be expected for the results shown in figure 3.29 a) by changing the initial height such that the penetration depth ranges between 0 and the amount travelled by the bodies towards one another in a single time-step.

### Kinetic Friction

The next physical phenomenon analysed is kinetic friction which is investigated by using an inclined ramp down which various objects are made to glide. Given the elevation angle  $\alpha$  of the slope, following forces can be analytically computed: the normal  $F_{g\perp}$  and tangential  $F_{g\parallel}$  components of the gravitational force  $F_g = m \cdot g$  as well as the (kinetic) friction force  $F_k$  opposed to the tangential force:

$$F_{g\perp} = \cos(\alpha) \cdot F_g = \cos(\alpha) \cdot m \cdot g \quad (3.43)$$

$$F_{g\parallel} = \sin(\alpha) \cdot F_g = \sin(\alpha) \cdot m \cdot g \quad (3.44)$$

$$F_k = \mu_k \cdot F_{g\perp} = \mu_k \cdot \cos(\alpha) \cdot m \cdot g \quad (3.45)$$

Assuming the body slides down the slope due to  $F_{g\parallel}$  being larger than  $F_k$  and any applying static friction, the resulting force  $F_{res}$  is parallel to  $F_{g\parallel}$  with

$$F_{res} = F_{g\parallel} - F_k = m \cdot g \cdot \sin(\alpha) - m \cdot g \cdot \cos(\alpha) \cdot \mu_k \quad (3.46)$$

The acceleration of the body down the slope can thus be computed as

$$\ddot{x} = \frac{F_{res}}{m} = g \cdot (\sin(\alpha) - \mu_k \cdot \cos(\alpha)) \quad (3.47)$$

Under the assumption that no initial positional offset or velocity are applied, the equation for the position along the slope

$$x(t) = \frac{\ddot{x} \cdot t^2}{2} = \frac{g \cdot (\sin(\alpha) - \mu_k \cdot \cos(\alpha)) \cdot t^2}{2} \quad (3.48)$$

can be solved for the kinetic friction coefficient  $\mu_k$ :

$$\mu_k = \frac{-2 \cdot x(t)}{t^2 \cdot g \cdot \cos(\alpha)} + \tan(\alpha) \quad (3.49)$$

The validation experiment for dynamic friction consists of analysing 21 objects with kinetic friction coefficients  $\mu_k = \{i \cdot 0.05 \mid i = 0..20\}$  and static friction  $\mu_s = 0$ . The time required by the objects to traverse a trajectory of given length down the slope is measured and the observed kinetic friction coefficient  $\mu'_k$  determined using equation 3.49. Three different integration time-steps have been tested:  $\Delta t \in \{0.1ms, 1ms, 10ms\}$  for 17 different inclinations  $\alpha = \{j \cdot 5^\circ \mid j = 1..17\}$ . The results for cuboidal primitives sliding down the ramp (which is also modelled as cuboid) are shown in figures 3.30 a), c) and e) for the three time-step sizes. The analogous experiments for polygonal mesh shapes of identical cuboidal geometry leads to results which correspond to within a fraction of a percentage point.

A flat cylinder modelled as mesh shape produced the somewhat different results depicted in figures 3.30 b), d) and f). They remain equal in nature to the cuboid results but display some quantitative differences. The worst correspondence is found for  $\Delta t = 0.1ms$ . There, the means of the observed kinetic friction coefficients  $\mu'_k$  corresponding to each input (PhysX) kinetic friction coefficient  $\mu_k$  differ by a maximum of  $|\mu_k - \mu'_k| < 0.09$  or 12%. The values  $\mu_k \in \{0.0, 0.05\}$  result in significantly higher relative discrepancies despite absolute deviations  $|\mu_k - \mu'_k| < 0.003$ . For time-step sizes  $\Delta t = 1ms$  and  $\Delta t = 10ms$  the deviations are  $|\mu_k - \mu'_k| < 0.015$  or 3.5% and  $|\mu_k - \mu'_k| < 0.03$  or 6% respectively.

The above mentioned comparison excludes the results for  $\Delta t = 0.1ms$  and  $\alpha = 80^\circ$  since this setup appears to represent some sort of singularity. Plots 3.49 a) and b) show the significant deviation of those measurements which only occur in

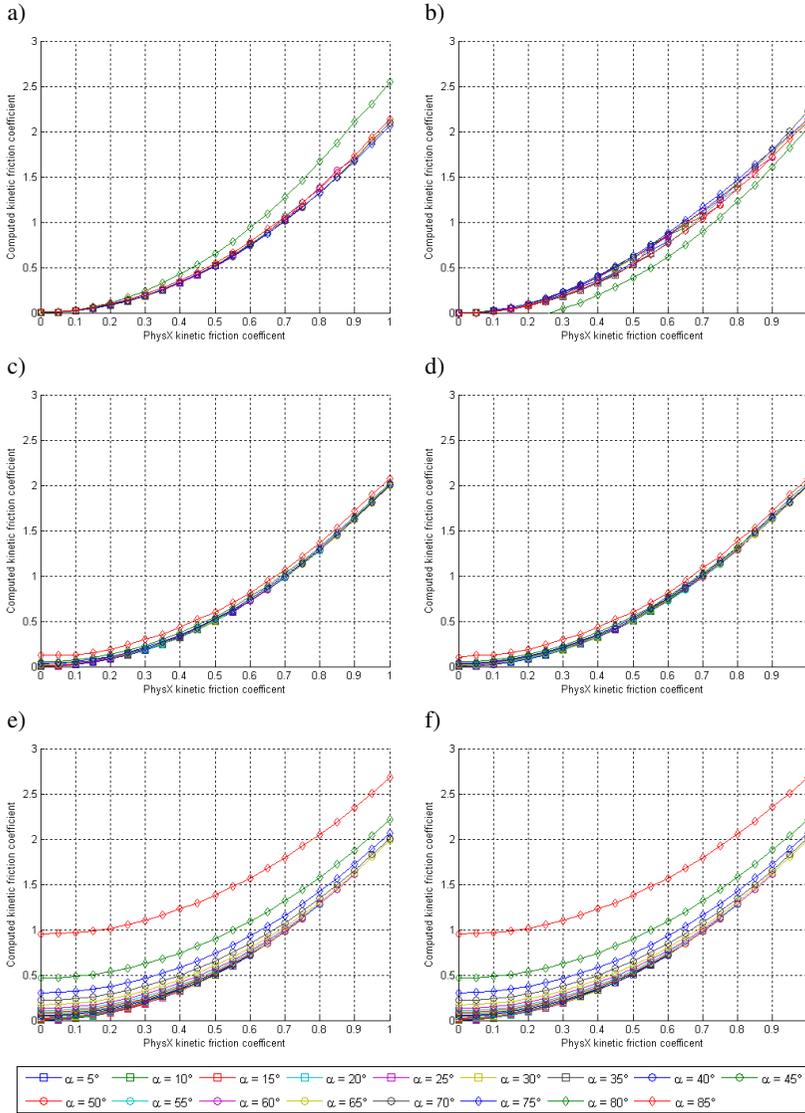


Fig. 3.30: Plots of kinetic friction validation: observed kinetic friction coefficient plotted as function of kinetic friction coefficient input into PhysX libraries. a) cuboid, time-step  $\Delta t = 0.1ms$  b) cylinder,  $\Delta t = 0.1ms$  c) cuboid,  $\Delta t = 1ms$ , d) cylinder,  $\Delta t = 1ms$ , e) cuboid,  $\Delta t = 10ms$ , f) cylinder,  $\Delta t = 10ms$ .

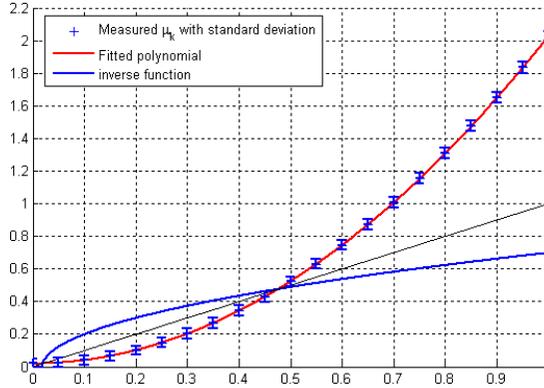


Fig. 3.31: Second-order polynomial fitting of mean measured kinetic friction values with standard deviation ( $1ms$  experiments). Inverse function which can be used to transform a desired kinetic friction value into an input to the PhysX libraries.

a neighbourhood  $79.999^\circ < \alpha < 80.001^\circ$ . To obtain a non-singular value which falls in line with the other measurements, in the analysis the value  $\mu_k(80^\circ)$  has been replaced by  $\widetilde{\mu}_k(80^\circ) = \frac{\mu_k(79.999) + \mu_k(80.001)}{2}$ .

All objects have been assigned a static friction coefficient  $\mu_s = 0$ . This leads to situations where  $\mu_k > \mu_s$  which are not considered when using traditional conventions. The result of such settings is in the PhysX libraries that some objects with  $\mu_s = 0$  nevertheless do not commence to slide down the ramp. This is presumably caused in discrete time by a potential first non-zero velocity being prevented by the kinetic friction which is already included in that time-step. Results for more combinations of  $\mu_s$  and  $\mu_k$  are presented in the following section which deals with static friction.

From the results shown in figure 3.30 it is obvious that the PhysX libraries use a different convention for dynamic friction from the one used so far in this document. If the definitions were identical, the plots would show a (linear) equality correspondence  $\mu'_k = \mu_k$ . Further, values of  $\mu'_k > 1$  have been registered, which lead to  $F_k > F_{g_\perp}$  for  $\mu_k \gtrsim 0.7$ .

This scaling as well as the non-linearity can either be accepted as part of the simulated physics model or compensated by selecting appropriate values for the input PhysX  $\mu_k$ . Figure 3.31 shows how the computed kinetic friction values can be approximated by a second-order polynomial using polynomial regression. The example shown is based on all data (cuboid, mesh cuboid and mesh cylinder) at  $\Delta t = 1ms$ . The approximation lies well within the standard deviations of the original data. Consequently the inverse function of the polynomial (also shown in the figure) can be computed and used to pre-process a desired kinetic friction value

in order to obtain the input to PhysX which leads to the expected results.

This technique could be extended to all time-steps and generalised in order to include it in a framework such as Ibox. While such a feature is easy to integrate, it has been preferred not to do so and pass on the default PhysX behaviour to the users. Given the degree of deviation in dependence of time-step size and involved object geometries, it is advisable to closely study the characteristics of a specific interaction in situations where a high degree of accuracy is desired.

### Static Friction

The static friction model has been tested in a setup consisting of 21 objects with static friction coefficients  $\mu_s = \{i \cdot 0.05 \mid i = 0..20\}$  placed on a horizontal surface. A horizontal force of a magnitude increasing linearly over time has been applied to the centre of mass of the objects. The moment when the objects start to move and the corresponding break-free force  $F_b$  have been recorded. This allows to determine the observed static friction coefficient for this setup ( $\alpha = 0$ ):

$$\mu'_s = \frac{F_b}{F_{g\perp}} = \frac{F_b}{\cos(\alpha) \cdot m \cdot g} = \frac{F_b}{m \cdot g} \quad (3.50)$$

The results of the static friction experiments are shown in figure 3.32, again for the time-step sizes  $\Delta t \in \{0.1ms, 1ms, 10ms\}$  and both the cuboidal primitive and flat polygonal mesh cylinder already used for previous validations. As in the dynamic friction validation, a mesh shape of identical geometry as the tested cuboid yielded practically indistinguishable results.

The effect described above where kinetic friction plays the role of static friction can be observed for all cases where  $\mu_k > \mu_s$ . Also, the same scaling observed in the dynamic friction validation (with  $\mu_k = 1 \rightsquigarrow \mu'_s \approx 2.03$ ) is evident. Taking into account these effects, the results for the mesh cylinder and  $\Delta t = 10ms$  shown in figure 3.32 f) approximate the desired behaviour. For  $\mu_k = 0$ , the force required to set the object into motion depends linearly on the static friction coefficient. For  $\mu_k < \mu_s$  the same force is required. If (against the usual convention)  $\mu_k > \mu_s$ , the kinetic friction coefficient prevents any starting of motion.

The other experiments reveal differing behaviour of the studied objects: The linear behaviour for  $\mu_k < \mu_s$  can be observed in many cases, but mostly  $\mu'_s$  is not independent of  $\mu_k$ . This effect can be seen well in figure 3.32 e) (cuboid,  $\Delta t = 10ms$ ) where for  $\mu_k = 0$  a linear dependence of  $\mu'_s = c \cdot \mu_s$  is evident but with a coefficient  $c \approx 1.3$  instead of  $c \approx 2$ . For increasing values of  $\mu_k$ , the corresponding coefficient increases (albeit not monotonically) such that for  $\mu_k \in \{0.8, 0.9\}$ , a value of nearly 2 is reached and for  $\mu_k = 1$  a constant  $\mu'_s \approx 2.03$  is again attained.

The experiments for  $\Delta t = 1ms$  correspond well to their  $10ms$  counterparts for  $\mu_k < 0.5$  and  $\mu'_s \lesssim 1$  as well as for  $\mu_k > 0.8$ . The intermediate interval results

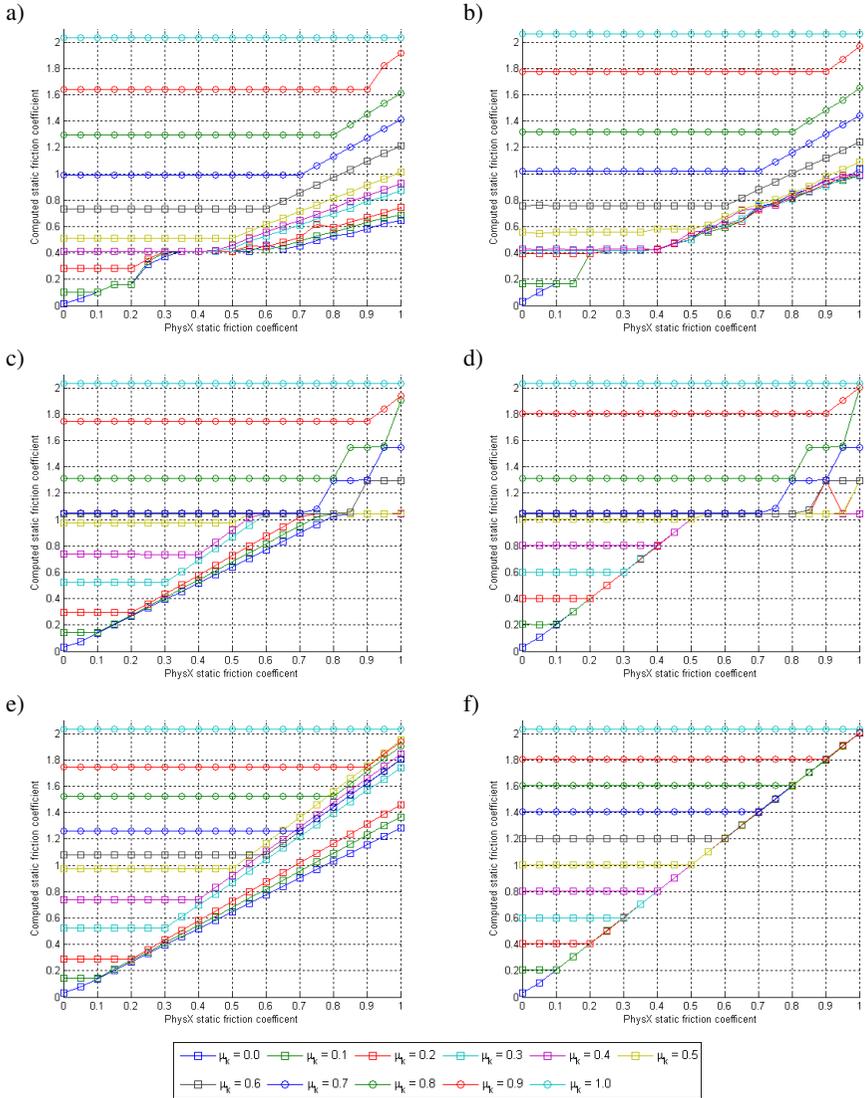


Fig. 3.32: Plots of static friction validation: observed static friction coefficient plotted as function of static friction coefficient input into PhysX libraries. a) cuboid, time-step  $\Delta t = 0.1ms$  b) cylinder,  $\Delta t = 0.1ms$  c) cuboid,  $\Delta t = 1ms$ , d) cylinder,  $\Delta t = 1ms$ , e) cuboid,  $\Delta t = 10ms$ , f) cylinder,  $\Delta t = 10ms$ .

in differing values of  $\mu'_s$  for the cases where kinetic friction prevents the motion. Furthermore, objects start to move at forces lower than those expected with the previously discussed behaviour. Also, a quantisation of required break-free forces (and hence  $\mu'_s$ ) can be observed for  $0.5 \leq \mu_k \leq 0.8$  with  $\mu'_s \gtrsim 1$ : objects of different friction coefficients commence to move at precisely the same time-step. In one case (for the mesh cylinder with  $\mu_k = 0.2$ ), an object of higher  $\mu_s$  breaks free at a lower “quantum” than its next lower neighbour.

At a time-step of  $\Delta t = 0.1ms$ , the divergence of the linear coefficient described above is even more pronounced, ranging from  $c \approx 0.65$  (cuboid, figure 3.32 a) to a similar final value of  $c \approx 2.03$  for  $\mu_k = 1$ . A distinct non-linearity is present in the range  $0.15 \lesssim \mu_s \lesssim 0.5$  where to some extent a similar effect to the quantisation described in the  $\Delta t = 1ms$  experiments can be perceived.

Contrary to the dynamic friction model, which in itself appears consistent, the combined static friction model (including both  $\mu_s$  and  $\mu_k$ ) has produced results which are difficult to map onto the conventional friction definitions. For experiments which rely on precise break-free forces it is imperative to study the concrete interaction in detail to obtain a meaningful prediction.

### Rolling Motion

The last basic interaction included in this validation deals with rolling motion. The setup used is similar to the one presented for the kinetic friction experiments: a ramp is inclined at various elevation angles  $\alpha$  and the progress of various initially resting objects studied as they move down the slope. As test objects, a sphere, a capsule primitive and a cylinder approximated as polygonal mesh (with the periphery composed of 72 planar surfaces) are used. Analytically, the tangential force  $F_{g\parallel}$  acting on a body rolling down a slope can be decomposed into a linear and an angular component:

$$F_{g\parallel} = F_{trans} + F_{rot} = m \cdot a + \frac{\tau}{r} \quad (3.51)$$

where  $\tau$  is the torque which is  $\tau = I \cdot \dot{\omega}$  with  $I$  the inertia tensor and  $\omega$  the angular velocity. The angular acceleration  $\dot{\omega}$  of a body rolling down a slope rotating about an axis at a constant distance  $r$  from the slope can be computed as  $\dot{\omega} = \frac{a}{r}$ . Equation 3.51 can hence be transformed into

$$F_{g\parallel} = \left( m + \frac{I}{r^2} \right) \cdot a \quad (3.52)$$

Inserting equation 3.44 (which is purely geometric and hence also holds true for the rolling case) into equation 3.52 and solving for the (linear) acceleration  $a$  yields:

$$a = \frac{\sin(\alpha) \cdot m \cdot g}{m + \frac{I}{r^2}} \quad (3.53)$$

For the bodies involved in the experiment, the inertia tensors (for rotations about the symmetry axis) are as follows:

$$I_{sphere} = \frac{2}{5} m_{sphere} \cdot r^2 \quad (3.54)$$

$$I_{cylinder} = \frac{1}{2} m_{cylinder} \cdot r^2 \quad (3.55)$$

$$\begin{aligned} I_{capsule} &= I_{cylinder} + I_{sphere} = \left( \frac{1}{2} m_{cylinder} + \frac{2}{5} m_{sphere} \right) r^2 \\ &= \rho_{capsule} \pi r^4 \left( \frac{h}{2} + \frac{8}{15} r \right) \end{aligned} \quad (3.56)$$

with  $\rho_{capsule}$  denoting the density of the capsule, which is assumed to be homogeneous. For the last equality in equation 3.56, the formulas for the volume of a cylinder and sphere have been used. Applying  $x = \frac{\ddot{x} \cdot t^2}{2}$ , the formula for the distance travelled by a mass under constant acceleration and solving for the time gives following expressions for the three bodies:

$$T_{sphere} = \sqrt{\frac{14 \cdot x(T)}{5 \cdot g \cdot \sin(\alpha)}} \quad (3.57)$$

$$T_{cylinder} = \sqrt{\frac{3 \cdot x(T)}{g \cdot \sin(\alpha)}} \quad (3.58)$$

$$T_{capsule} = \sqrt{\frac{x(T) \left( 3h + \frac{56}{15} r \right)}{g \cdot \sin(\alpha) \left( h + \frac{4}{3} r \right)}} \quad (3.59)$$

The validation results for the sphere are shown in figure 3.33, those for the mesh cylinder in figure 3.34 and the results for the capsule in figure 3.35. The analytical solution for rolling motion presented above applies to perfect rolling motion, i.e. when no sliding occurs. In the experiments a combination of sliding and rolling motion can often be observed. Analytically, exclusively sliding motion occurs at  $\mu_k = 0$ . As soon as some friction is present, a torque is induced in the sliding bodies which consequently commence to roll in combination with the sliding motion. At the point where the friction force  $F_k$  becomes larger than the tangential component  $F_{g\parallel}$  of the gravitation force, i.e.  $\mu_k \cdot \cos(\alpha) > \sin(\alpha)$ , the

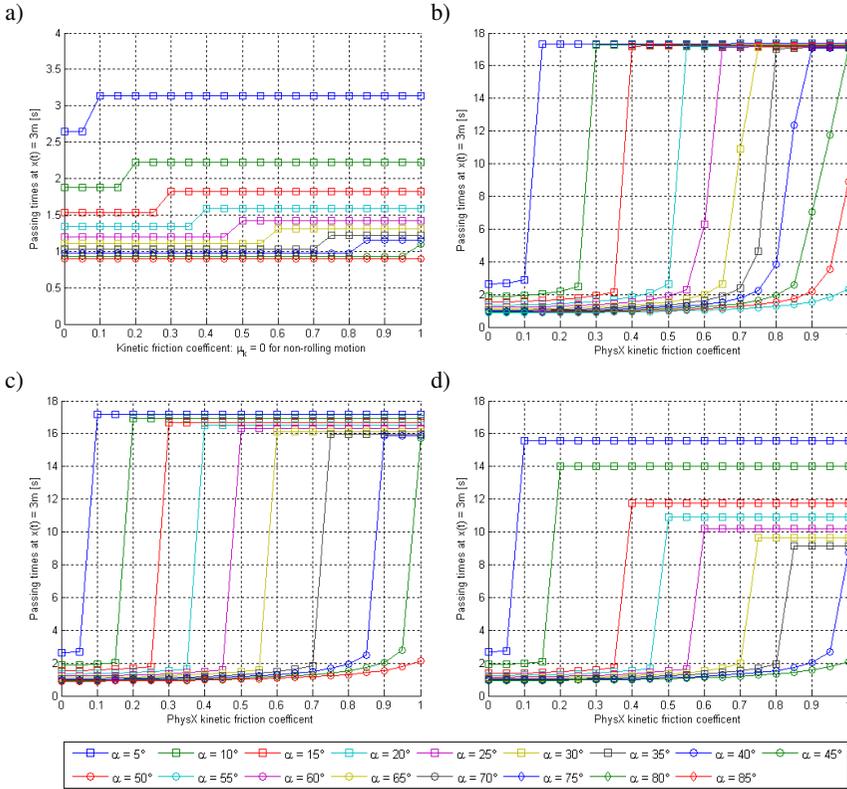


Fig. 3.33: Plots of rolling validation for sphere shape: times  $T$  required by object to traverse a trajectory of fixed length  $x(T) = 3m$ . a) Analytical solution with kinetic friction coefficient  $\mu_k$  clamped to zero for non-rolling motion. b) simulation results, time-step  $\Delta t = 0.1ms$  c)  $\Delta t = 1ms$  d)  $\Delta t = 10ms$ . Note the differing ordinate scaling of the analytical plot.

body displays an exclusively rolling motion. Figures 3.33 a), 3.34 a) and 3.35 a) show the results of a combined analytical solution for sliding and rolling motion. To improve the readability of the graphs, it has been assumed that the objects are either perfectly sliding ( $\mu_k = 0$ ) or perfectly rolling.

The value of  $\mu_k$  at which an object displays overwhelmingly rolling motion differs strongly for the three object types. This “rolling boundary” is the lowest for the capsule shape, followed by the mesh cylinder (ca.  $\Delta\mu_k = +0.05$  behind) and the sphere. The rolling boundary of the sphere increases from a similar level at small ramp inclinations to requiring  $\mu_k = 1$  to roll at  $\alpha = 45^\circ$ . At that inclination,

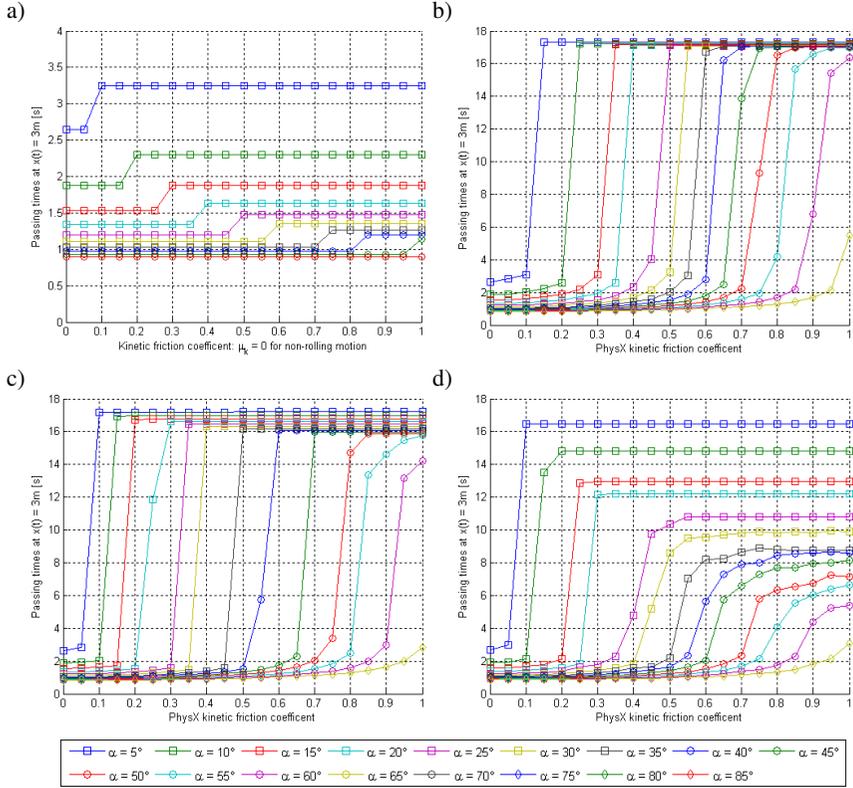


Fig. 3.34: Plots of rolling validation for cylindrical mesh shape: times  $T$  required by object to traverse a trajectory of fixed length  $x(T) = 3m$ . a) Analytical solution with kinetic friction coefficient  $\mu_k$  clamped to zero for non-rolling motion. b) simulation results, time-step  $\Delta t = 0.1ms$  c)  $\Delta t = 1ms$  d)  $\Delta t = 10ms$ . Note the differing ordinate scaling of the analytical plot.

the other shapes already commence to roll with  $\mu_k \approx 0.6$ , and continue rolling up to  $\alpha \approx 60^\circ$ . A graphical representation of the rolling boundary is shown in figure 3.36.

Considering the situations where objects do largely display rolling motion, little difference in the time required to traverse the given distance can be distinguished. For  $\Delta t \in \{0.1ms, 1ms\}$ , the times differ by less than  $|T_i - T_j|_{i,j \in \{sphere, cylinder, capsule\}} < 0.5\%$ . In particular, this means that the inertia tensor appears to play a negligible role in the computations. The absolute times measured for the objects to traverse the distance are approximately a factor 5

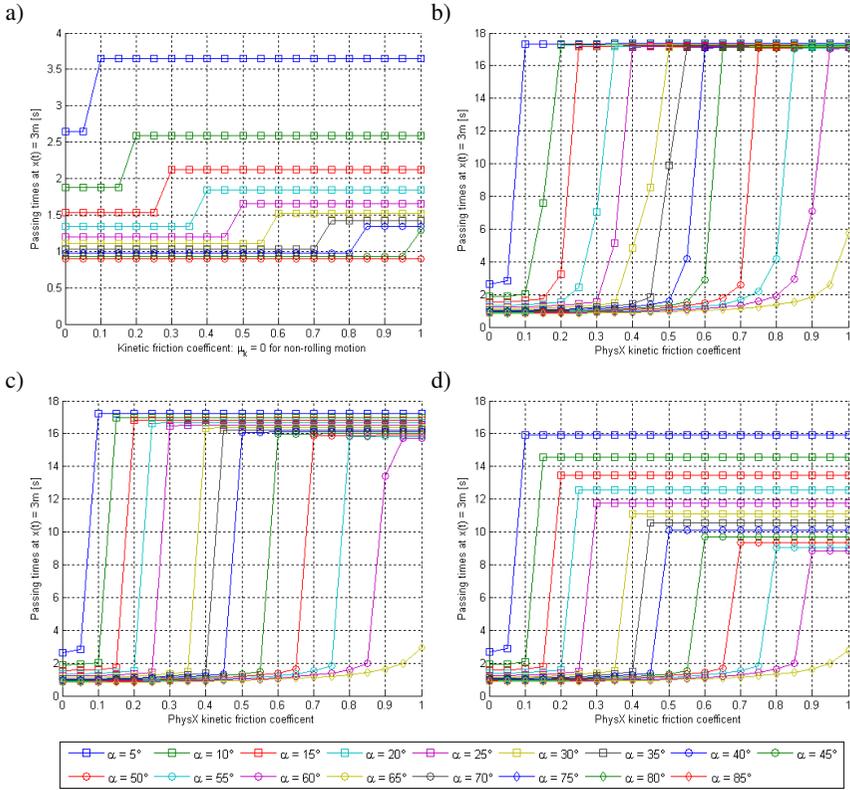


Fig. 3.35: Plots of rolling validation for capsule shape: times  $T$  required by object to traverse a trajectory of fixed length  $x(T) = 3m$ . a) Analytical solution with kinetic friction coefficient  $\mu_k$  clamped to zero for non-rolling motion. b) simulation results, time-step  $\Delta t = 0.1ms$  c)  $\Delta t = 1ms$  d)  $\Delta t = 10ms$ . Note the differing ordinate scaling of the analytical plot.

higher than expected, despite all artificial angular damping having been suppressed in the PhysX libraries.

At a simulation time-step of  $\Delta t = 10ms$ , larger discrepancies can be observed in the traversal times of the three bodies at different ramp inclinations. In particular, the mesh cylinder displays a wide spread of times. This can be explained by its periphery being composed of 72 planar surfaces which cause some amount of sliding motion, even above the nevertheless discernible rolling boundary.

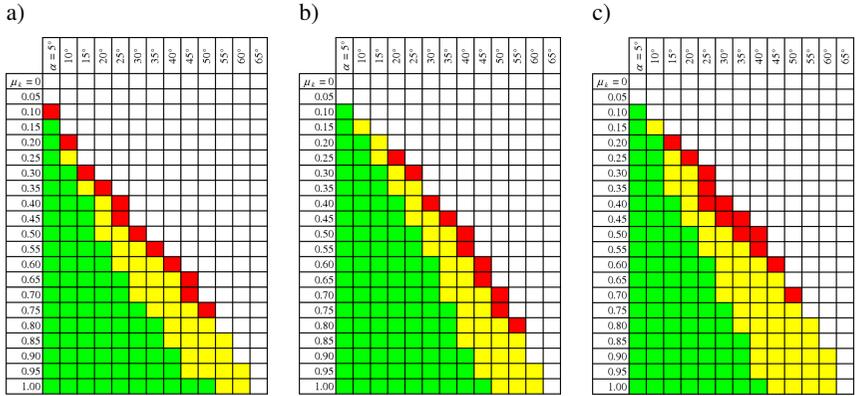


Fig. 3.36: Graphical representation of the rolling boundary for the three studied bodies. The lowest values of  $\mu_k$  (red) correspond to setups where only the capsule shape displays rolling motion to an overwhelming extent while the other shapes mostly slide. The next band (yellow) indicates situation where both the capsule and the mesh cylinder roll, but not the sphere. The highest values of  $\mu_k$  (green) indicate rolling motion for all three objects. a) time-step  $\Delta t = 0.1ms$ , b)  $\Delta t = 1ms$ , c)  $\Delta t = 10ms$ .

### 3.4.2 Performance Analysis

The most versatile configuration of Ibox is when it is used embedded in Simulink. Other than for dedicated single-application tasks it is presumably the setup most likely to be used for most simulations. Therefore, the performance tests presented in this section are based on the Ibox-Simulink integration, despite this configuration implying an overhead in terms of processing power and memory usage.

The test computer has an Intel<sup>®</sup> Centrino<sup>™</sup> 1.6GHz CPU, 1GB RAM and an ATI Mobility<sup>™</sup> Radeon<sup>®</sup> 9000 GPU with 64MB VRAM. The operating system is<sup>®</sup> Windows<sup>®</sup> XP Professional with Service Pack 2 (SP2) installed. No particular effort has been made to improve the performance of the system (e.g. by deactivating system tasks etc.).

The basic test setup for the performance measurements consists of a stack of 50 objects placed on a plane. The topmost object is positioned with a minimal horizontal offset relative to all others. When the simulation commences, this offset causes the entire stack to topple over under the effects of gravity. Such a setup leads to numerous collisions between the objects, from the average two for each object initially (with its upper and lower neighbour) to a varying number with varying collision partners as the stack collapses. For each experiment, the time required to progress the simulation 30s (roughly the time required by the objects to all be in

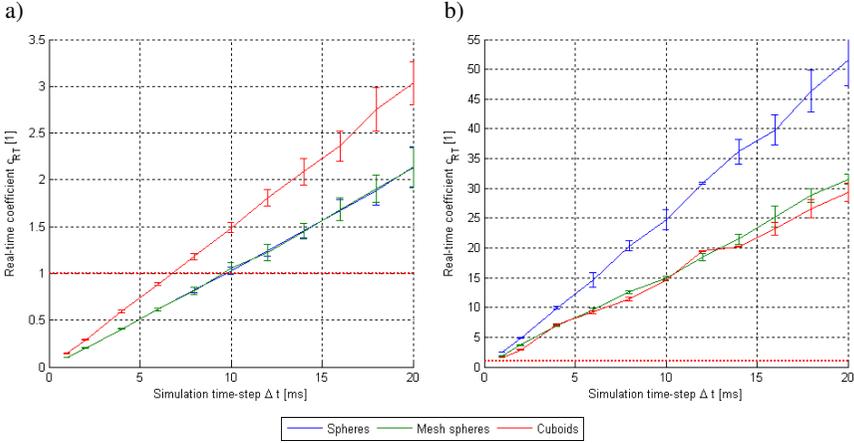


Fig. 3.37: Plots of real-time coefficients and simulation times for a collapsing stack of 50 objects simulated for 30s. a) with Nebula 2 observer window at a resolution of  $1024 \times 768$  pixels, b) without graphics output.

contact with the ground) is determined. The real-time coefficient is defined as

$$c_{RT} = \frac{t_{real}}{t_{sim}} \quad (3.60)$$

with  $t_{real}$  the duration of the process (30s) and  $t_{sim}$  the time required to simulate it. For a simulation to run in real-time, it must fulfill  $c_{RT} = 1$ . If it progresses faster, it can be artificially slowed down to achieve real-time behaviour. In the opposite case the setup cannot be simulated in real-time with the given resources. Real-time behaviour is of particular importance where other entities which run in real-time rely on interactions with the simulation. Hardware-in-the-Loop (HIL) simulations consist of real-time simulations which interact with hardware devices (such as control electronics or actuators etc.) [Hanselmann 93], [van Amerongen 03]. HIL techniques are useful to e.g. test and study the behaviour of hardware devices in a controlled environment.

The different operation modes of the simulation environment have been introduced in section 3.3.1. While not directly applying to the Simulink integration, Ibex is run at “CPU-speed”, i.e. as fast as possible for the performance tests. In all experiments, the time-steps of Ibex and Simulink have been synchronised (cp. section 3.3.6).

Figure 3.37 shows the results of simulating a collapsing stack of 50 objects. As object geometries, sphere primitives, cuboidal primitives and polygonal mesh representations approximating the sphere shape have been used. The mesh sphere

consists of 212 vertices and 420 triangles. To mitigate the impact of external influences on the performance measurements, each experiment has been repeated 5 times. Figure 3.37 shows the mean values and standard deviations of these measurements. In the experiments shown in figure 3.37 a), a Nebula 2 graphics observer window has been used to visualise the simulation at a resolution of  $1024 \times 768$  pixels. The results closely correspond for the sphere primitives and the polygonal mesh approximations thereof. This is somewhat surprising as a sphere is a significantly simpler geometry to simulate - a sphere-sphere collision detection can be reduced to determining the distance between the two sphere centres and comparing it to the sum of both radii. The cuboid primitives are simulated at the highest real-time coefficient. All objects display an approximately linear dependency on the time-step size which corresponds to the expected outcome.

Figure 3.37 b) shows the results of the same experiments without any graphical output. Abandoning the visualisation results in a significant performance boost. The setup can be simulated faster than real-time even for the smallest tested time-step of  $1ms$ . Without graphics observer, spheres are clearly simulated at the highest velocity. The time required to simulate the setup for cuboids and the mesh sphere approximations is similar. The analysis of exclusively the simulation performance reveals the reason for the unexpected performance equality of sphere primitives and polygonal mesh approximations discussed above. In the graphics engine, both the spheres and the mesh approximations are *visualised* using the same geometry. The performance of the graphics engine appears to be the determining factor of the overall performance. This causes the cuboid experiment (which involves a simpler visualisation geometry) to run faster than the two experiments involving spheres.

This hypothesis is supported by the graphics frame rates observed in the experiments visualised in figure 3.37. The amount of resources available to the graphics engine is determined by a load-balancing algorithm. In the tested configuration, the total of available resources is distributed at a constant ratio between a worker thread running the simulation and the visualisation thread. Hence (assuming the total resources remain constant during the duration of the experiment), the complexity of the visualised scene determines the graphics frame rate. Both sphere experiments run at an approximate 73FPS, while the cuboid experiment displays around 110FPS. This proportion of ca. 2:3 corresponds roughly to the performance ratio of the associated experiments. Clearly, the load-balancing scheme tested is not optimised for maximal simulation velocity - but arguably the use of a complex 3D graphical visualisation conflicts with such a target in the first place. An alternative load-balancing algorithms would behave like the described scheme at high simulation loads but limit the resources available to the graphics engine in a way that its performance does not exceed a specified upper limit, e.g. 30FPS. If more resources were available to the visualisation thread, it could renounce them, thus benefiting the simulation thread.

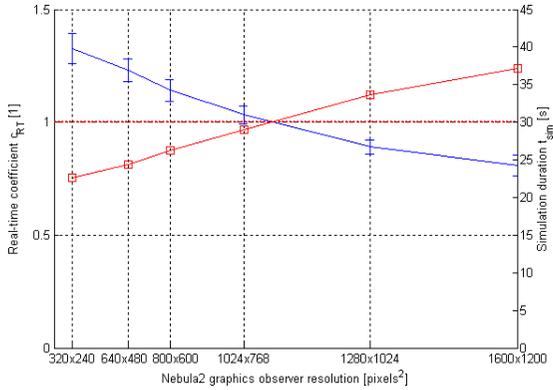


Fig. 3.38: Overall simulation performance as function of Nebula 2 graphics observer resolution. Real-time coefficient  $c_{RT}$  represented as mean and standard deviation of 5 experiments as well as corresponding mean execution times shown. Experiment setup: one stack of 50 spheres.

The influence of the visualisation resolution on the simulation performance is shown in figure 3.38. Different standard resolutions have been applied to the graphics observer window and the overall performance of the simulation measured. As experiment, a stack of 50 sphere primitives is used with the same visualisation settings as before. The frame rates of the graphics engine vary between 57FPS for the highest  $1600 \times 1200$  pixel resolution and 93FPS for the lowest  $320 \times 240$  pixel resolution. As above, the mean values and standard deviations of five simulation runs are shown together with the corresponding mean execution times. Again, it can be observed that the frame rates correspond well with the real-time coefficient due to the load-balancing algorithm employed.

The performance of Ibox simulations as function of scene complexity is visualised in figure 3.39. Departing from the original setup of 50 stacked spheres, experiments have been run which include repeated identical stacks. The stacks are positioned in a way which prevents any interaction between spheres from different stacks. This ensures that a setup containing  $n$  stacks not only consists of  $n \cdot 50$  spheres but also only leads to  $n$  times the original number of collisions. Figure 3.39 a) shows the results of simulation runs which are visualised using a Nebula 2 observer at a resolution of  $1024 \times 768$  pixels. Mean values and standard deviations for  $c_{RT}$  of five experimental runs for each setup are shown, as well as the corresponding simulation times  $t_{sim}$ . The analogous plots for the experiments without graphical output are shown in figure 3.39 b). The simulation times required by both experiments are roughly linear with the scene complexities within the tested range. The rate of increase somewhat reduces for higher complexities in the setup

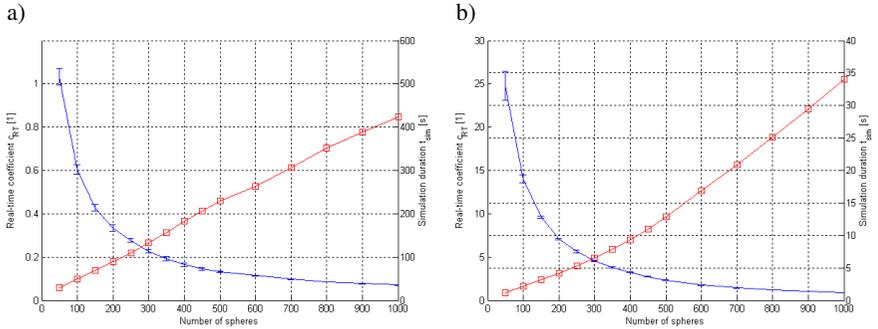


Fig. 3.39: Plots of real-time coefficients for different numbers of collapsing stacks containing 50 spheres each. Values on the abscissa indicate total number of spheres in the scene. Simulation time is  $t_{sim} = 30s$ . a) with Nebula 2 observer window at a resolution of  $1024 \times 768$  pixels, b) without graphics output.

including the graphics observer. A possible explanation therefore is that in the more complex scenes increasingly more objects are eliminated in the graphics engine by culling algorithms since they are either outwith the field of view or obscured by other objects.

The plot shown in figure 3.39 b), covers the experiments without visualisation. For scenes including more than ca. 500 spheres, a linear growth of measured execution times  $t_{sim}$  can be recognised. This trend is confirmed by experiments for 2000, 3000, 4000 and 5000 spheres not shown in the plot. For small scenes (below ca. 300 spheres) an approximately linear behaviour can also be measured, with a lower rate of increase. The transition between the two models can hypothetically be explained by the increased memory consumption, e.g. some data structure not fitting in cache memory or inducing higher management overhead from a certain scene size on.

An alternative measure for scene complexity involves not increasing the number of objects in the scene but rather the complexity of each simulated object. To this end, a setup has been simulated which includes a stack of 50 polygonal mesh cubes. In the simplest setup, the cubes are represented by their 8 corner vertices and the corresponding 12 triangles (6 triangulated quadratic faces). More complex mesh cubes are created by subdividing each face into a number of smaller triangulated quadratic surfaces. Applying the same number of subdivisions  $s$  along each axis of the cube results in  $v = 6s^2 + 2$  vertices and  $f = 2 \cdot 6s^2$  triangular faces. For  $s \rightarrow \infty$ , the proportion becomes  $\frac{f}{v} \rightarrow 2$ . The results for the experiments involving different object complexities are shown in figure 3.40. In figure 3.40 a), the setup includes a Nebula 2 graphics observer running at a resolution of  $1024 \times 768$  pixels. A linear dependency on the total object complexity - represented on the abscissa by

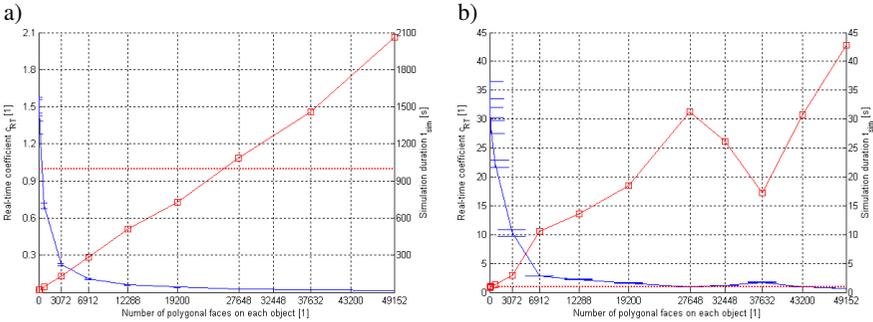


Fig. 3.40: Plots of real-time coefficients for a single collapsing stack containing 50 polygonal mesh cubes. Values on the abscissa indicate the number of polygon faces on each of the 50 objects. Simulation time is  $t_{sim} = 30s$ . a) with Nebula 2 observer window at a resolution of  $1024 \times 768$  pixels, b) without graphics output.

the number of polygons contained in each object - can be discerned. Comparing this result with figure 3.40 b) where the results for the equivalent experiments without visualisation are shown indicates to the dominating influence of the graphics engine which leads to this linear behaviour.

Without any visualisation, a somewhat surprising outcome is the non-monotonic increase of computation times with increasing object complexity. A significant minimum can be recognised around the measurement containing objects with 56 surface subdivisions (37'632 faces). The exact cause of this phenomenon could not be established without gaining access to the PhysX algorithms. Speculative explanations include that vertices on the objects are culled to a certain degree beginning at some object complexity. A further hypothetical explanation is that some aspects of memory management are altered with complex objects. Measurements for 128 subdivisions of each cube face cause the Nebula2 graphics engine to crash, they are consequently omitted in the plots. Without visualisation, a real-time coefficient of  $c_{RT} \approx 0.2$  is achieved, corresponding to 143s execution time on the test computer. On the one hand this demonstrates the capabilities of the PhysX libraries to perform fast computations of complex setups. On the other hand, this value closely corresponds to the linear extrapolation of the measurements for relatively low complexities, somewhat putting into perspective the significance of the minima at 56 subdivisions.

To round off the performance analysis, an example is presented which gives an impression of the performance achieved with a more realistic system including joints and active components. The medieval clockwork already introduced in section 3.3.5 has been embedded in a structure including a hand (At that time a single hand was used to indicate the time), the dial as well as two bells. The clockwork

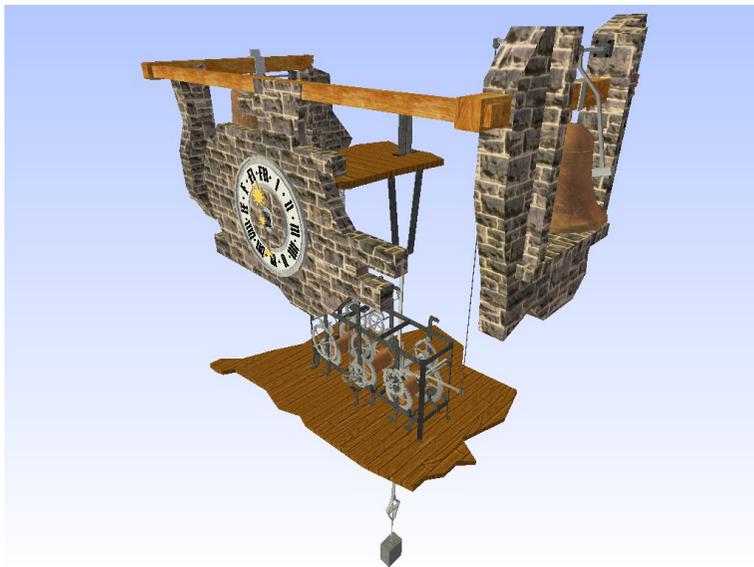


Fig. 3.41: The complete setup of the medieval clockwork visualised in an Ibex simulation.

is operated as in reality: the source of power is given by a weight attached to a rope wound around a drum. The pendulum regulates the velocity of the gears using an escapement mechanism. In the simulation the force exerted by the weight is applied directly to the drum as constant torque. This is the only source of power in the setup. The torque is sufficient to keep the pendulum swinging and the clockwork moving exclusively through the rigid body collisions between the gears and levers involved.

The complete setup is depicted in figure 3.41. The scene consists of 28 rigid objects, composed of 730 individual collision primitives, cp. figure 3.26 d) for a visualisation of the collision shapes. A total of 26 rotational joints hold the cogwheels in place and define the motion of the clockwork. On the same test computer used throughout this section, the setup can be simulated in Simulink at ca.  $27\times$  real-time using a time-step of  $0.01s$ . When adding a Nebula 2 observer for visualisation (at a resolution of  $1024\times 768$  pixels), the performance drops to  $c_{RT} \approx 0.2$  due to the complexity of the involved artwork. When running as a stand-alone application instead of embedded in Simulink, the same setup yields a performance of  $c_{RT} \approx 1.06$ , hence faster than real-time. This would allow to build a museum exhibit of the clockwork operating in real-time together with the visualisation on hardware of similar performance as the test computer.

### *Hardware-Accelerated Physics Simulation*

One important aspect to take into consideration when discussing the performance characteristics of Ibex and in particular the Ageia™ PhysX™ libraries is the availability of hardware-accelerated simulation libraries. In Spring 2006, Ageia launched its physics processing unit (PPU) also named PhysX.

The PhysX PPU is shipped embedded in an add-on PCI express card for personal computers and consists of a dedicated parallel hardware architecture optimised for computing the type of operations required by physics simulations. Physics simulations in this context include not only rigid body dynamics but also particle systems, deformable objects (e.g. cloth simulations), liquids and fractures of bodies among other things.

One manufacturer of PCI cards containing the PhysX chip, BFG Technologies ([BFG 06]) states 530 million sphere-sphere collisions per second and 533'000 convex-convex collisions per second as maximal real-time capabilities of their card. Such performance characteristics would open completely new possibilities of simulation for robot motion planning. For example it would become tractable to simulate a large number of individual pebbles as terrain model. This would offer a new level of detail when simulating the interaction of robot wheels with the terrain.

The latest PhysX libraries required to perform hardware-accelerated physics simulations have not yet been integrated into Ibex. However, the possibilities offered by hardware-supported physics make this a promising extension for the future development of the simulation framework.

#### *3.4.3 Limitations of Simulation Approach*

The possibilities of rigid body dynamics simulations have been demonstrated in this chapter by means of an introduction to the theoretical background as well as application examples of the Ibex framework. In this section some limitations of the simulation approach are discussed to complete the picture given of rigid body dynamics simulations in general and the Ibex framework in particular.

One of the major bottlenecks when attempting to create a genuine model of reality in a simulation is the availability of adequate data. Often, mechanical setups are simulated whose characteristics are not exactly available. To perform a simulation, precise values must be provided to the algorithms thus requiring some assumptions to be made about the physical system. In the case of a mechatronic system, the properties of the setup are usually known with a good precision. For a non-trivial terrain model on the other hand, data availability often is sketchy and afflicted with errors. Interpolation errors such as the arbitrary choice of triangulation direction discussed in section 3.3.3 introduce further discrepancies when modelling some specific terrain. A decrease of simulation accuracy due to limited data availability is not intrinsic to rigid body dynamics but rather a general problem of any

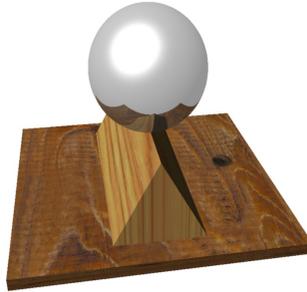
simulation approach.

In the presented rigid body dynamics techniques, discretisation errors also occur when approximating parametric curves of objects through piecewise linear polygonal geometries. The counterargument can be upheld that any arbitrary precision can be achieved with a polygonal approximation, albeit at the cost of increased computation times. This tradeoff between precise geometry approximation and computational costs is exploited when using primitive shapes to approximate complex geometries. Indeed it can be seen as an important strength of the proposed approach to specify the relative importance of computational velocity and simulation precision.

A fundamental parameter which is inherent to all discrete-time simulations is the simulation time-step. The time step usually directly corresponds to the integration step introduced in section 3.3.2. Smaller time-steps allow more accurate computations of the simulated system. Larger steps allow faster progress in time while accepting a potential reduction in accuracy. Additional deviations from the expected behaviour arise with large time-steps when the contact detection scheme allows interpenetrations of rigid bodies, i.e. no continuous collision detection is used (cp. section 3.3.2).

Discrepancies from the behaviour of a real-world system also arise from the rigidity assumption which strictly speaking does not hold true for virtually any object. Notwithstanding, many objects of interest in mechatronic systems can be approximated as rigid objects in a way which yields acceptable simulation results. As a side note, paradoxically, the two inaccuracies introduced by allowing interpenetration as consequence of discrete simulation time-steps and the rigidity assumption cause opposing errors and can cancel each other out. Jointly (and together with an appropriate restitution model) these effects can result in a level of approximation which corresponds to an elastic deformation at collision time.

Finally, the potentially chaotic behaviour of multi-body systems needs to be mentioned. In unstable configurations, small differences of local characteristics (either static properties or dynamic physical behaviour) lead to considerably different global behaviour. As an example, consider a spherical body balanced on top of a prismatic object forming an infinitesimally sharp ridge, cp. figure 3.42. Clearly this is an unstable equilibrium and the slightest perpendicular force applied to the sphere will set it in downwards motion over the side of the prism due to the gravitational pull. As a consequence of this motion, the potential energy of the sphere is transformed into kinetic energy. The moving sphere can then apply forces of a magnitude significantly larger than those required to originally set it in motion to other objects (not shown) in the setup. Furthermore, the direction of the originally applied force determines in which direction the sphere is propelled and hence where potentially the effects of its motion are felt. Considering a force applied along the direction of the ridge formed by the prism, the tiniest deviation



*Fig. 3.42: A rigid-body setup which illustrates the chaotic behaviour of multi-body systems.*

causes the sphere not to follow the ridge but to tumble down either side. Again, the issue of chaotic multi-body behaviour is not specific to rigid body dynamics simulations but is a more general problem and also occurs in the real world. It is a fact though, that unstable equilibria as described above rarely occur other than in artificially engineered setups. This is due to the principle of energy minimisation which leads to more stable configurations being adopted most of the time in the real world. In setups designed in perfect virtual environments, situations can easily be created which are near to impossible to recreate in the real world. A good example is the stack of 50 spheres used for performance testing: in the virtual environment it required the topmost body to be slightly displaced in order for the stack to collapse. In the real world it would be all but impossible to place 50 spheres on top of each other without them collapsing.

Summarising the findings of this section, it can be said that simulations as described in this document are subject to a trade-off between accuracy and computational velocity. The geometries and physical properties of objects can be approximated to any desired degree (within some practical limitations). The simplifying assumptions made when modelling a system must not be lost track of, in rigid-body dynamics simulations this fundamentally refers to the rigidity of the objects. When using the Ageia™ PhysX™ libraries appropriate settings for the simulation must be carefully chosen to model reality in the desired manner. Within the above mentioned limitations, the effects of introduced errors can be reduced to levels which make the application of the technique feasible for a multitude of setups. On the other hand, even minute errors can lead to strongly deviating results in pathological situations where the chaotic behaviour of multi-body systems potentiates the effects of small influences on the system.

## 4. ROBOT MOTION PLANNING

In chapter 3, a rigid body dynamics simulation framework has been discussed in some detail. In this chapter, the simulation environment is applied to robot motion planning tasks. The focus of the present research lies on mobile robotics motion planning and more specifically rough-terrain planning. This task is generally harder to solve than motion planning on planar surfaces, which can be considered a special case thereof. Despite the selection of examples from a single domain, it can be appreciated from the theoretical introduction to rigid body dynamics and the examples shown in chapter 3 that the Ibex framework is applicable to robot motion planning in general and to a wide variety of tasks from other domains likewise.

In the course of this chapter, three novel motion planning algorithms are presented which have been developed and tested using Ibex. The algorithms are extensions of existing approaches adapted to rough-terrain planning. The first algorithm is based on the Randomised Potential Field (RPP) approach, the other two extend  $\text{RRT}_{\text{connect}}$ , a bi-directional variant of the Rapidly Exploring Random Tree (RRT) planner. The extensions are achieved by including a new generic measure for the navigational difficulty on rough terrain.

The terrain model introduced in section 3.3.3 constitutes a convenient way of representing the distribution of terrain material properties encountered. Using such a terrain model, Ibex is well suited to perform simulations of the dynamic behaviour a robot displays when navigating on rough terrain.

Further it is outlined how the sensor simulations implemented in Ibex (cp. section 3.3.4) can be used to perform dynamic motion planning. The input provided by the sensors forms the basis of re-planning algorithms which compute new trajectories for the robot in the presence of unexpected hindrances to its progress.

The examples given in this chapter complete the picture given of the Ibex framework so far. In its entirety, an environment is described which supports the development of motion planning algorithms embedded within the larger mechatronics development process. The scope of supported activities ranges from the design of robot mechanics to the development of reactive sensor-based planning algorithms. As an added benefit, the components of the framework integrate seamlessly into existing engineering solutions, thus improving the usability and productiveness of the solution.

## 4.1 All-Terrain Motion Planning

Rough-terrain robot navigation has received a significant amount of attention recently, most prominently showcased to the broader public by the success of recent and ongoing Mars rover missions, e.g. [Cook 05], [Corneille 05], [Catling 05] and references therein. The other prominent event illustrating progress in the field is the “DARPA Grand Challenge” autonomous off-road vehicle race. In the future, enhanced autonomous capabilities will be required to accomplish increasingly ambitious planetary missions, e.g. [Stadd 04], [Winnendael 04], [Huntress 06] as well as a whole variety of Earth-bound tasks. This demand has led to the development of numerous approaches to solving the rough-terrain robot motion planning task - or more generally all-terrain motion planning. The term all-terrain motion planning is often used to describe algorithms which are well suited for rough-terrain motion planning but nevertheless can be applied to conventional (planar) motion planning tasks. Operation on a planar surface can be considered a sub-set of rough terrain navigation as a flat surface represents a degenerate case of a generally rugged one.

### Previous Work

A time-optimal planner for a point robot moving on rough terrain is presented in [Shiller 90a], [Shiller 91b]. Dynamic constraints such as avoidance of sliding and tip-over effects make use of a global friction coefficient as well as a *mobility factor* which is used to characterise ease of navigation in different regions of the terrain. The assumptions made on the terrain limit the admissible roughness to features not smaller than the size of the robot.

In [Dacre-Wright 93], the obstacle-free area for an  $n$ -wheeled robot with passive spring suspension on rough terrain is computed using placement constraints to determine stable configurations.

A measure for *traversability* is defined in [Kubota 95] and applied to a rover-like vehicle. A similar approach has been used in [Chen 96] for walking robots.

Using a wavelet decomposition, a hierarchical, multi-resolution terrain representation is computed in [Pai 98]. It is shown how the wavelet approximation error can be used as cost measure and also combined with other cost functions to determine a total cost function.

Placements are used in [Haït 99] in conjunction with constraints (robot stability, collision avoidance of the robot body, mechanical constraints and a kinematic constraint to avoid sliding motion) to determine free configurations. Terrain costs are computed in a discrete grid based on *terrain slope* and *terrain roughness*. A potential field approach [Khatib 86], [Latombe 91] is used to compute trajectories.

In [Iagnemma 99] the inclusion of uncertainty in a rough-terrain planner is emphasised. The  $A^*$  algorithm [Nilsson 98] is used to initially find an optimal path. The cost function used considers the path length, *terrain unevenness* and *rover*

*turning actions*. These measures of navigational difficulty are based on the *terrain roughness* in the surroundings of the robot.

Different terrain properties are considered in [Cherif 99b], where the terrain is split into discrete areas of constant physical properties. Static obstacles are explicitly represented in the terrain model. The 3D terrain is approximated by a collection of tangent spheres and the obstacles as set of projected spheres on the  $xy$ -plane. Deformations of the terrain are computed as mass-spring systems of the spheres composing the terrain.

A *traversability index* is established in [Seraji 99] using fuzzy logic. Linguistic fuzzy sets for *terrain slope* and *terrain roughness* form the basis for the formulation which is used in a fuzzy logic framework to determine navigation rules.

An overall system for rough-terrain path generation and tracking is discussed in [Guo 03]. In the path generation phase, a binary distinction is made between challenging and benign terrain based on *terrain roughness* and *terrain slope* at discrete grid locations. The total cost function of a single path segment is the weighted sum of the “power consumption” and the Euclidean distance between its endpoints. Only considering benign terrain locations, an  $A^*$  algorithm is used to find a path which optimises the total cost function.

### Configuration Space

When planning the high-level motions of a mobile robot, often the configuration space is reduced to a level where the robot is considered a single rigid object, disregarding details of the robot configuration, e.g. the rotational positions of the wheels. In the classic book on motion planning, [Latombe 91], this is formalised as follows: the  $N$ -dimensional configuration space is denoted by  $\mathcal{C} = \mathbb{R}^N \times SO(N) \subset \mathbb{R}^{N+N^2}$ . For three-dimensional space,  $N = 3$  leads to configurations  $\mathbf{q}$  of the robot  $\mathcal{A}$  being represented by a 3-vector  $\mathcal{T}$  holding the Cartesian position and an orientation  $\Theta \in SO(3)$ .

$\mathcal{T}$  defines the position of the robot’s frame  $\mathcal{F}_A$  with respect to the world frame  $\mathcal{F}_W$ .  $SO(N)$  denotes the *special orthogonal group* of  $N \times N$  matrices  $M \in \mathbb{R}^{N^2}$  with orthonormal columns and rows as well as determinant  $|M| = 1$ . Such matrices can be used to represent the orientation  $\Theta$  of a robot in  $N$ -dimensional space; the columns of  $M$  hold the unit vectors along the axes of  $\mathcal{F}_A$  with respect to  $\mathcal{F}_W$ . An example of  $SO(N)$  is alluded to in section 3.3.2, where the representation  $R \in \mathbb{R}^{3 \times 3}$  is introduced as rotation matrix for three-dimensional space.

In rough-terrain motion planning, the configuration space definition given above can be further reduced under the assumption that the robot remains in stable contact with the ground and the terrain is specified most generally by a many-to-one (unique) function. In these cases, a projection of the three-dimensional position  $\mathcal{T} \in \mathbb{R}^3$  onto  $\tilde{\mathcal{T}} \in \mathbb{R}^2$  can be performed:

$$\mathbf{q}_{trans} = (x, y, z)^T \mapsto \tilde{\mathbf{q}}_{trans} \equiv (x, y)^T \quad (4.1)$$

The omitted  $z$ -component which denotes the position along the vertical coordinate axis is implicitly given by the terrain topography and robot geometry. This statement holds true when using a purely kinematic physics model or assuming slow robot motion. Low robot velocities are often assumed in the domain, based on the argument of the slow velocities used by not exclusively autonomous planetary rovers due to high communication latencies.

In a similar manner, the orientation  $\Theta$  of the robot  $\mathcal{A}$  is mapped from  $SO(3)$  onto an orientation  $\tilde{\Theta} \in SO(2)$  in the  $xy$ -plane. This orientation can further be simplified to a single variable  $\theta \in [0, 2\pi)$  which represents the orientation as heading angle:

$$\mathbf{q}_{rot} \in SO(3) \mapsto \tilde{\mathbf{q}}_{rot} \equiv \theta = \tan^{-1} \left( \frac{\mathbf{e}_{\mathcal{F}_A 1y}}{\mathbf{e}_{\mathcal{F}_A 1x}} \right) \quad (4.2)$$

with  $\mathbf{e}_{\mathcal{F}_A 1}$  being the unit vector of  $\mathcal{F}_A$  which points “forward” (i.e. the heading of the robot). The  $x$  and  $y$  subscripts refer to the components of a vector in direction of the global  $x$  and  $y$  axis. Combining translation  $\tilde{T}$  and rotation  $\theta$ , a reduced configuration space  $\tilde{\mathcal{C}} \subset \mathbb{R}^3$  results for high-level motion planning. In the following, this reduced configuration space is used in most examples. To improve readability, the tilde notation is omitted consecutively.

#### 4.1.1 The “Degree of Obstacleness”

In classical motion planning literature (e.g. [Latombe 91]), often a binary obstacle definition is used. Obstacles  $\mathcal{B}_i$  represent clearly defined subspaces of the workspace  $\mathcal{W}$ . These workspace obstacles are transformed to configuration space obstacles  $\mathcal{CB}_i = \{\mathbf{q} \in \mathcal{C} \mid \mathcal{A}(\mathbf{q}) \cap \mathcal{B}_i \neq \emptyset\}$ , with  $\mathcal{A}(\mathbf{q})$  the subspace of  $\mathcal{W}$  occupied by the robot  $\mathcal{A}$  at configuration  $\mathbf{q}$ . This in turn leads to the definition of free space  $\mathcal{C}_{free} = \mathcal{C} \setminus \bigcup_{i=1}^q \mathcal{CB}_i$ .

The constant factor with all rough-terrain motion-planning lies in the underlying characteristics of the rough terrain itself. By the very nature of the task, binary obstacle definitions cannot be exclusively applied to rough-terrain motion planning, cp. [Iagnemma 99]. The boundary between navigable and impassable terrain is hard to define. Moreover, while in “difficult” terrain navigation might be possible, it can be desirable to display a preference for “easier” regions. Each configuration of the robot operating on the terrain has a characteristic difficulty associated with its attainment. Depending on the properties of the task being studied, different aspects of the robot/terrain interaction assume high relevance. These factors are consequently included in the terrain abstraction while other aspects are typically

chosen to be omitted. Nevertheless, independently of the terrain model used, there remains the specific difficulty associated with reaching a particular configuration.

To achieve independence of any concrete definition of navigational difficulty, following definition of the degree of presence of an obstacle (“obstacleness” for short) is proposed [Ettlin 06b], [Ettlin 05a]:

$$0 \leq o(\mathbf{q}_{motion}) \leq 1 \quad \forall \mathbf{q}_{motion} \in \mathcal{C}_{motion} \quad (4.3)$$

In equation 4.3,  $o(\mathbf{q}_{motion}) = 0$  corresponds to the absence of any terrain-induced hindrance to motion of the robot. A value of  $o(\mathbf{q}_{motion}) = 1$  represents impassable terrain i.e. it corresponds to the presence of an obstacle in a binary definition environment. The obstacleness of any configuration  $\mathbf{q}_{motion} \in \mathcal{C}_{motion}$  can be determined by evaluating the obstacleness function at that configuration.

To allow more concise algorithm formulations following shorthand notation has been introduced to specify the set of configurations which fulfill some specific condition of the associated obstacleness.

$$\mathcal{C}_{[o \bullet x]} = \{\mathbf{q} \in \mathcal{C}_{motion} \mid o(\mathbf{q}) \bullet x\} \mid_{\bullet \in \{=, <, >, \leq, \geq\}} \quad (4.4)$$

The motion planning configuration space  $\mathcal{C}_{motion}$  has been implicitly introduced to allow for a distinction between the configuration properties relevant to rough-terrain navigation and the configuration space  $\mathcal{C}_{robot}$  of the robot itself.  $\mathcal{C}_{motion}$  is composed of selected configuration parameters identified as being relevant to the specific rough-terrain navigation application. For example, the reduced configuration space  $\tilde{\mathcal{C}}$  introduced above can be used. In the following, the *motion* subscript is mostly omitted. The notation  $\mathcal{C}$  refers to the configuration space used for motion planning where not explicitly stated otherwise.

### Concrete Obstacleness Definition Composition

The definition of obstacleness given in equation 4.3 represents a generic measure for the degree of presence of an obstacle at a configuration of the robot on the terrain. This general concept needs to be concretised for every specific class (environment and robot characteristics) of motion planning problems. Such a concrete obstacleness definition results from the combination of several components  $K_\psi(\mathbf{q})$  corresponding to phenomena  $\psi$  relevant for the class of problems:

$$o(\mathbf{q}) = f(\{K_\psi(\mathbf{q}) \mid \psi \in \Psi\}) \quad (4.5)$$

This combination is represented as function  $f$  in equation 4.5 with  $\Psi$  being the set of relevant phenomena for the problem class. To ensure consistent behaviour of  $f$ , phenomena components  $K_\psi$  are required to be normalised, i.e.  $0 \leq K_\psi \leq 1$ . In the following, examples of possible phenomena components  $K_\psi$  are given.

### Terrain Topography

The inclination of the terrain at a particular location is an important measure for the ease of navigation at that position on the terrain. Let  $T(x, y)$  denote the terrain height data at position  $(x, y)$  with  $x_{min} \leq x \leq x_{max}$  and  $y_{min} \leq y \leq y_{max}$ . The obstacleness component  $K_{incl}$  representing terrain inclination can be defined as

$$K_{incl}(x, y) = \frac{|\nabla T(x, y)|}{\max_{\{p, q\}}(|\nabla T(p, q)|)} \quad (4.6)$$

under the assumption that  $\max_{\{p, q\}}(|\nabla T(p, q)|) > 0$ .

Another useful definition of  $K_{incl}$  is based on the elevation angle of the terrain at each location and is subsequently normalised to fulfil the requirements of an obstacleness component:

$$K'_{incl}(x, y) = \frac{2 \cdot \tan^{-1}(|\nabla T(x, y)|)}{\pi} \quad (4.7)$$

Alternatively to the above definitions, it may be numerically desirable to guarantee the full dynamics  $0 \leq K_{incl} \leq 1$  are exploited. For the definition shown in equation 4.6 e.g., this can be done by defining  $K_{incl}$  as:

$$K''_{incl}(x, y) = \frac{|\nabla T(x, y)| - \min(|\nabla T|)}{\max(|\nabla T|) - \min(|\nabla T|)} \quad (4.8)$$

where it is required that  $\max(|\nabla T|) - \min(|\nabla T|)$  be non-zero.

Figure 4.1 a) shows a rendering of a sample terrain, a contour-line representation is depicted in figure 4.1 b). A contour plot of the corresponding inclination obstacleness component  $K_{incl}$  is shown in figure 4.2.

Additionally to a obstacleness component  $K_{incl}$  dealing with terrain inclination, a dedicated term for lateral inclinations of the terrain  $K_{lat}$  in relation to the robot has proven to be useful. The need has arisen when simulating tracked vehicles using the terrain model described in section 3.3.3 and tracks composed of rigid bodies and linked with revolute joints (cp. figures 4.3 and 4.4). It has been found that such vehicles can negotiate significantly steeper terrain inclinations in direction of motion than in slopes perpendicular to their heading. Inadequate lateral inclinations cause the vehicle to enter an uncontrolled sideways slide. Such a situation deviates the robot from the planned path and can potentially be costly to recover from or even endanger the task outcome. By using a lateral inclination term, such configurations can be assigned a higher degree of obstacleness and thus avoided when computing the motion plan. While it can be argued that in reality the behaviour of a tracked vehicle may differ from the described situation, this example serves to illustrate the versatility of a generic obstacleness definition which can be adapted to arising requirements in a flexible manner.

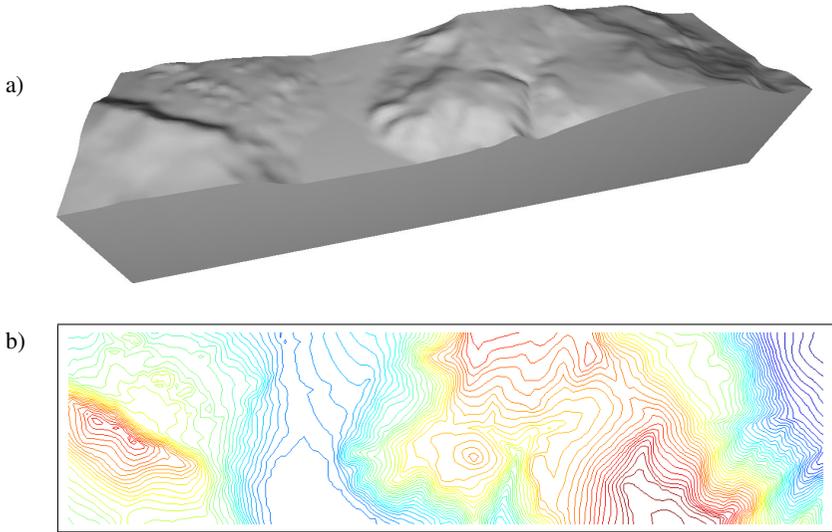


Fig. 4.1: Sample terrain consisting of 7401 vertices and 13994 triangles a) as solid mesh visualisation, b) contour-line representation.

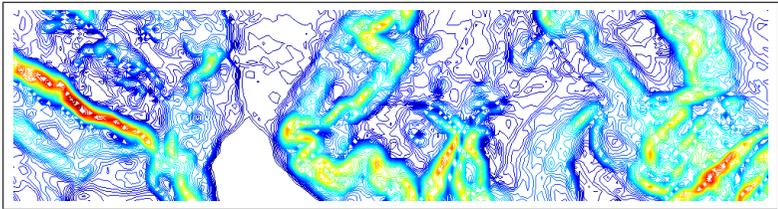


Fig. 4.2: Contour-line representation of the inclination obstacleness component  $K_{incl}$  computed for the sample terrain shown in figure 4.1.

Contrary to the inclination terms introduced above, the lateral inclination term  $K_{lat}$  is not independent of the robot orientation  $\theta$ . Similarly to  $\theta$ , the direction of terrain inclination  $\Phi(x, y)$  can be computed as single angle about the vertical axis:

$$\Phi(x, y) = \tan^{-1} \left( \frac{\frac{\partial T(x, y)}{\partial y}}{\frac{\partial T(x, y)}{\partial x}} \right) \quad (4.9)$$

Based thereon and on the normalised magnitude of the inclination  $K_{incl}$  defined as in either equation 4.6 or equation 4.8, the lateral inclination component can be specified as:

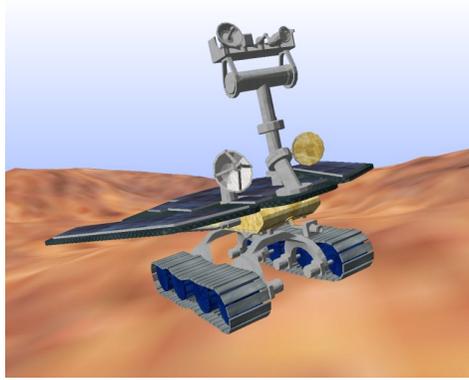


Fig. 4.3: View of a fictional tracked vehicle used in the motion planning experiments operating on rough terrain.

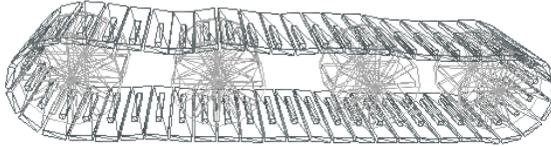


Fig. 4.4: Detail wire-frame representation of a single track used in the model shown in figure 4.3. Each track consists of 50 track elements which are simulated as rigid bodies connected by revolute joints. It remains in place with respect to the wheels exclusively through collisions between the rigid bodies.

$$K_{lat}(x, y, \theta) = c_{lat}(x, y, \theta) \cdot K_{incl}(x, y) \quad (4.10)$$

The coefficient  $c_{lat}$  used to scale the inclination term  $K_{incl}(x, y)$  in equation 4.10 can be defined in a number of ways. In a first attempt, it can be defined proportional to the deviation from a parallel orientation of the robot with the terrain gradient:

$$c_{lat \ lin}(x, y, \theta) = \left| \frac{2}{\pi} \left( \left( (\Phi(x, y) - \theta) - \frac{\pi}{2} \right) \bmod \pi \right) - 1 \right| \quad (4.11)$$

Figure 4.5 shows the different definitions of  $c_{lat}$ . It is apparent, that the linear definition  $c_{lat \ lin}$  is algorithmically not ideal, since its derivative has discontinuities

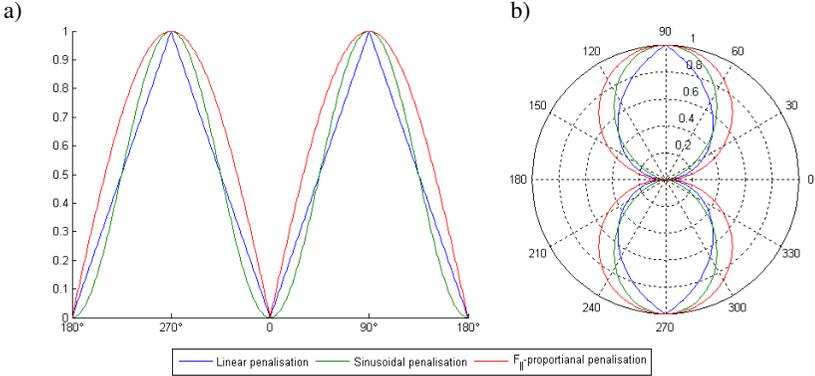


Fig. 4.5: Characteristics of lateral obstacleness component coefficient as linear plot a) and polar plot b). The linear penalisation results in discontinuities of the derivative and hence unstable behaviour, which are avoided with the sinusoidal definition. As alternative, the physics-based variant proportional to  $F_{g_{\parallel}}$  is depicted.

at  $\Phi(x, y) - \theta = i \cdot \frac{\pi}{2} | i \in \mathbb{Z}$ . This can lead to unstable behaviour of the robot when operating near those relative orientations. To avoid such discontinuities, a sinusoidal function  $c_{lat\ sin}$  can be designed:

$$c_{lat\ sin}(x, y, \theta) = \frac{-\cos(2(\Phi(x, y) - \theta)) + 1}{2} \tag{4.12}$$

Such a definition maximally penalises perpendicular terrain gradients with respect to the robot while disappearing for gradients aligned or opposed to the direction of motion. Compared to a linear dependency, the sinusoidal coefficient results in a wider range of relative orientations which are lightly penalised, thus benefitting the allowed manoeuvrability of the robot.

Figure 4.6 shows a visualisation of  $K_{lat\ sin}$  for the sample terrain depicted in figure 4.1. The ground plane corresponds to the dimensions of the terrain, the vertical axis denotes the robot heading  $\theta$ . Solid areas in the plot represent subspaces with lateral inclination values  $K_{lat\ sin} \geq k$ , where  $k = 0.3$  has been arbitrarily selected to highlight the characteristics of the function. The  $\pi$ -periodicity as consequence of the penalisation of lateral inclinations can be clearly recognised. Consider the steep slope which is clearly visible on the left side of figure 4.2 as area of locally maximal gradient, coloured red. The slope is inclined in direction NNE<sup>1</sup>, with its extension stretching in direction ESE. Following the motivation given above, the robot should avoid orientations around ESE and WNW. This can be recognised in

<sup>1</sup> North-North-East, assuming North at the top of the page in 2D figures like e.g. figures 4.1, 4.2. Orientations in 3D figures are as indicated in figure 4.6. Further cardinal points: S (South), W (West).

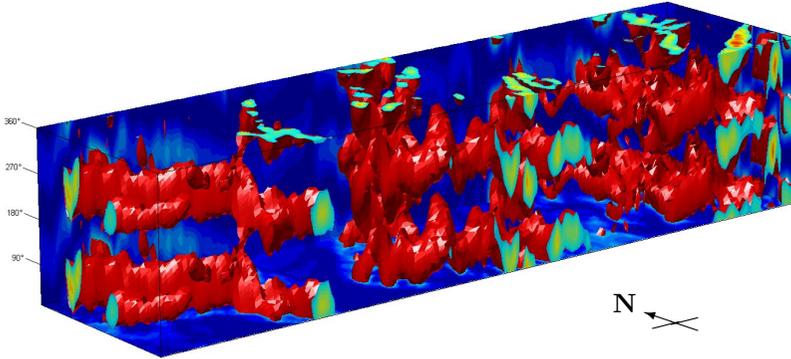


Fig. 4.6: Visualisation of the lateral inclination obstacleness component  $K_{lat\ sin}$  for the sample terrain depicted in figure 4.1. The vertical axis denotes the heading of the robot. Solid areas represent subspaces  $K_{lat\ sin} \geq 0.3$ .

figure 4.6 as double red volume at the corresponding  $xy$ -coordinates. The inhibited heading angles can be recognised to be around ESE and WNW, as required. The other steep slope in the terrain, located in the SE (bottom-right-hand) corner, runs roughly NE with a SE inclination. This navigational hindrance produces high lateral obstacleness regions around NE and SW orientations, as expected. Other features of the terrain can analogously be recognised.

On the downside, the sinusoidal definition  $K_{lat\ sin}$  is devoid of any real physical meaning. Therefore as alternative a coefficient is proposed which scales with the proportion of the tangential component  $F_{g_{\parallel}}$  of the gravitation force acting perpendicularly on the robot. This is simply expressed by

$$c_{lat\ F_{\parallel}} = \sin(|\Phi(x, y) - \theta|) \quad (4.13)$$

Unfortunately, the absolute value of the relative orientation again introduces a discontinuity at the important situation of parallel alignment.

### Terrain Constitution

Apart from the purely topographical properties covered by the inclination and lateral inclination obstacleness components, it is important to consider physical terrain material properties. These material properties determine to a high degree the ease of navigability on the terrain.

In the terrain abstraction discussed in section 3.3.3, it is possible to assign physical material properties to every point of the terrain to model the contact behaviour. The ability to simulate different physical material properties at each triangle of a rigid body represented as polygonal mesh geometry is one of the key strengths of

the PhysX libraries used in Ibex. It allows to simulate rough terrain navigation tasks in a rigid body dynamics environment with an unprecedented degree of faithfulness. While the obstacleness definition is independent of the simulation model, the following arguments are based on the terrain abstraction used in Ibex to simplify the use of the simulation environment for illustrating the algorithm functionality.

In Ibex, physical material properties are represented by means of the coefficient of restitution  $c_r$  as well as static and kinetic friction coefficients ( $\mu_s$  and  $\mu_k$ ). It is assumed that the three above mentioned coefficients are defined such that:

$$0 \leq c \leq 1 \mid c \in \{c_r, \mu_s, \mu_k\} \quad (4.14)$$

The corresponding obstacleness components  $K_{phys}$  must be defined to reflect the increased difficulty of navigation caused by low values of the friction coefficients  $\mu_s$  and  $\mu_k$  as well as the opposite effect of  $c_r$ . Given the ranges as in equation 4.14, the obstacleness components can be trivially defined as:

$$K_{phys}(x, y) = \begin{cases} K_\mu(x, y) = (1 - \mu(x, y)) & |_{\mu \in \{\mu_s, \mu_k\}} \\ K_r(x, y) = c_r(x, y) \end{cases} \quad (4.15)$$

In cases with small variations of the physical properties over the entire terrain it can prove to be numerically advantageous to opt for a representation of  $K_{phys}$  which maps the actual values occurring in the terrain to the entire interval  $[0, 1]$ . This can be done analogously to the terrain topography component definition given in equation 4.8 at the cost of the definitions then being dependent on the terrain characteristics.

The friction components  $K_\mu$  in equation 4.15 are independent of robot orientation, allowing to model isotropic friction. If an anisotropic friction model were used (which is supported by the PhysX libraries but not currently in Ibex), the definitions would have to be updated in a way resembling the definition of the lateral inclination component  $K_{lat}$  in equations 4.11 - 4.13. The precise definition depends on the properties of the anisotropic friction model.

### Combination Function

Departing from the general equation 4.5, a concrete combination function  $f$  can be defined to compute  $o(\mathbf{q})$  for a specific motion planning scenario. Even more so than with the concrete obstacleness component definitions given above, the combination function needs to be designed for a concrete motion planning setting. Therefore the combination functions shown in the following serve mainly for illustration purposes. Based on the introduced component definitions, a basic combination function with coefficients  $a_1$  to  $a_4$  can be designed as

$$o(\mathbf{q}) = \frac{a_1 \cdot K_{incl} + a_2 \cdot K_{\mu_s} + a_3 \cdot K_{\mu_k} + a_4 \cdot K_{lat}}{a_1 + a_2 + a_3 + a_4} \quad (4.16)$$

Such a function is reminiscent of the potential functions used in artificial potential-field motion planning. This resemblance is discussed and expanded in section 4.1.2. Importantly, the obstacleness components are independent of one another and their relative importance can be taken into consideration by the choice of the coefficients  $a_i$ .

For many applications more powerful combination functions lead to better results. In equations 4.11 - 4.13, a lateral inclination component has been introduced to help avoid configurations in which the robot enters an unwanted lateral sliding motion. More stringently, it can be desirable to avoid configurations  $\mathbf{q}$  which would cause the robot to start sliding altogether. The required condition (in analogy to the geometrical force computations in equations 3.43 - 3.46) can be modelled as

$$F_s > F_{g\parallel} \rightarrow \mu_s \cdot \cos(\alpha) \cdot F_g > \sin(\alpha) \cdot F_g \quad (4.17)$$

with  $F_s$  the static friction force and  $F_{g\parallel}$  the component of the gravitational force  $F_g$  parallel to the terrain surface (and opposed to the friction force).  $\alpha$  stands for the elevation angle of the terrain. Inequality 4.17 can be rewritten to yield the condition  $\mu_s > \tan(\alpha)$ . Based thereon, the function  $s(\mathbf{q})$  can be defined to express the degree of compliance in terms of obstacleness components:

$$s = \frac{\tan(\frac{\pi}{2} \cdot K'_{incl})}{(1 - K_{\mu_s}) \circ \mu_{s \text{ vehicle}}} \quad (4.18)$$

The static friction coefficient of the parts of the vehicle in contact with the terrain is denoted by  $\mu_{s \text{ vehicle}}$ .  $K'_{incl}$  is the scaled elevation angle (cp. equation 4.7) as inclination component.  $K_{\mu_s}$  is the static friction obstacleness component (used here to “retrieve” the friction coefficient of the terrain at the current configuration). Finally,  $\circ$  is the operator used to combine the friction coefficients of the vehicle and the terrain. Using these definitions, the obstacleness combination function which prevents sliding of the robot can be written as:

$$o(\mathbf{q}) = \begin{cases} s(\mathbf{q}) & \text{if } s(\mathbf{q}) < 1 \\ 1 & \text{else} \end{cases} \quad (4.19)$$

### Relation to Cost Functions

Cost functions can be defined to optimise the path generation according to some criteria. These criteria can cover purely geometric (e.g. path length), temporal and physics-based aspects (e.g. energy optimisation).

The complete system for rough-terrain path generation and tracking of [Guo 03] makes a binary distinction between challenging and benign terrain based on *terrain roughness*  $\phi$  and *terrain slope*  $k$  at discrete grid locations. Terrain roughness is defined as variation of terrain elevations within a circular domain centred at the

position of the robot. The cost in terms of “power consumption”  $P$  is based on  $\phi$  and  $k$ . The total cost function of a single path segment is the weighted sum of the “power consumption” and the Euclidean distance between its endpoints. Only considering benign terrain locations, an  $A^*$  algorithm [Nilsson 98] is used to find a path which optimises the total cost function  $f_{pp}$  for the whole path:

$$f_{pp} = \alpha_1 \sum P(\phi, k) + \alpha_2 \sum D \quad (4.20)$$

In [Iagnemma 99] the inclusion of uncertainty in a rough-terrain planner is emphasised. Again, the  $A^*$  algorithm is used to initially find an optimal path. The cost function used considers the path length, *terrain unevenness* and *rover turning actions*. Terrain unevenness is computed specifically for the discussed class of robots and is based on the *terrain roughness*  $S$ . Contrary to  $\phi$  in [Guo 03],  $S$  is defined using a rectangle instead of a circular area. Rover turning actions is a term only included in the weighted sum of the cost function for path configurations where the robot is required to change its direction of motion. It is computed similarly to terrain roughness but making use of the circular area maximally swept during a skid steering point turn by the rectangle used for  $S$ .

Individual cost function terms like terrain roughness, terrain slope or the cost of rover turning actions can be represented as obstacleness components. A discretised configuration space  $\mathcal{GC}$  [Latombe 91] can be defined to perform a graph search using e.g.  $A^*$  for computing an optimal path based on obstacleness in the sense of the above mentioned publications.

Obstacleness as general measure of the navigational difficulty associated with each configuration can be used as basis for formulating many rough-terrain motion planning algorithms. In the following, two families of algorithms are described, the first centred on randomised potential fields (RPPs), the second on rapidly growing random trees (RRTs).

#### 4.1.2 Randomised Potential Field (RPP) Approach

Each obstacleness component is a scalar function defined over the entire configuration space of the robot (possibly reduced to  $\mathcal{C}_{motion}$ ). Moreover, regions of low obstacleness define areas which are preferred as compared to high obstacleness areas. These properties make obstacleness a related concept to the repulsive potential used in potential-field based motion planning approaches.

In this section, first a brief introduction to artificial potential-field based motion planning is given. Thereafter, a concrete RPP motion planning algorithm is presented which makes use of the obstacleness definition to become applicable to rough-terrain planning.

### Potential-Field Based Motion Planning

In artificial potential field robot motion planning methods (e.g. [Khatib 86], [Latombe 91]), the robot is conceptually considered as a body to which artificially generated forces are applied in order to determine its motions. The artificial forces  $\vec{F}(\mathbf{q})$  are produced by an artificially generated differentiable potential function  $\mathbf{U} : \mathcal{C}_{free} \mapsto \mathbb{R}$  with  $\vec{F}(\mathbf{q}) = -\vec{\nabla}\mathbf{U}(\mathbf{q})$ .  $\vec{\nabla}\mathbf{U}(\mathbf{q})$  is the gradient vector of  $\mathbf{U}$  at configuration  $\mathbf{q}$ , e.g.  $\vec{\nabla}\mathbf{U}(\mathbf{q}) = (\frac{\partial U}{\partial x}, \frac{\partial U}{\partial y}, \frac{\partial U}{\partial z})^T$  for three-dimensional translational motion.

The potential field  $\mathbf{U}(\mathbf{q})$  is generated as superposition of two components: an attractive potential  $\mathbf{U}_{att}(\mathbf{q})$  and a repulsive potential  $\mathbf{U}_{rep}(\mathbf{q})$ . The attractive potential is associated with  $\mathbf{q}_{goal}$  and is classically e.g. defined as parabolic well [Latombe 91]:

$$\mathbf{U}_{att}(\mathbf{q}) = \frac{1}{2}\xi\rho_{goal}^2(\mathbf{q}) \quad (4.21)$$

with  $\xi$  a positive scaling factor and  $\rho_{goal}(\mathbf{q}) = \|\mathbf{q} - \mathbf{q}_{goal}\|$  the Euclidean distance to the goal. Such a definition results in good stabilising characteristics, as the force displays linear convergence to 0 when approaching the goal:

$$\vec{F}_{att}(\mathbf{q}) = -\vec{\nabla}\mathbf{U}_{att}(\mathbf{q}) = \xi\rho_{goal}(\mathbf{q})\vec{\nabla}\rho_{goal}(\mathbf{q}) = -\xi(\mathbf{q} - \mathbf{q}_{goal}) \quad (4.22)$$

Traditionally, potential field methods are applied in binary obstacle settings [Hwang 92], [Barraquand 92], [Bouilly 95], [Feder 97]. In such cases, the repulsive potential  $\mathbf{U}_{rep}(\mathbf{q})$  is associated with the C-obstacle<sup>2</sup> region  $\mathcal{C}_{obs} = \bigcup_i \mathcal{CB}_i$  and is independent of the goal configuration. The intention of the repulsive potential is to create a potential barrier around the C-obstacles but not to affect the motion of the robot when it is distant thereof. Following definition is used in [Latombe 91]:

$$\mathbf{U}_{rep} = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\rho(\mathbf{q})} - \frac{1}{\rho_0}\right)^2 & \text{if } \rho(\mathbf{q}) \leq \rho_0 \\ 0 & \text{if } \rho(\mathbf{q}) > \rho_0 \end{cases} \quad (4.23)$$

with  $\eta$  a positive scaling factor,  $\rho(\mathbf{q}) = \min_{\mathbf{q}' \in \mathcal{CB}} \|\mathbf{q} - \mathbf{q}'\|$  the distance from the C-obstacle region and  $\rho_0$  the *distance of influence*.

Potential field approaches can be very efficient compared to other methods but since they use algorithms involving fastest descent optimisation, they display a tendency to get trapped in local minima of the potential function. Conceptually, either potentials without local minima (*navigation functions*) need to be designed or mechanisms to escape from local minima included in the planner.

<sup>2</sup> A C-Obstacle  $\mathcal{CB}$  is the region in  $\mathcal{C}$  onto which a (binary) workspace obstacle  $\mathcal{B} \in \mathcal{W}$  is mapped.

Navigation functions, (cp. [Rimon 92] for a summary of the original algorithms), can be costly to create and can in practice only be computed analytically for low dimensions  $N \lesssim 3$  and geometrically simple C-obstacles. As computationally attractive alternative, the configuration space  $\mathcal{C}$  can be discretised into a grid  $\mathcal{GC}$  [Barraquand 91]. For the case of  $\mathcal{C} = \mathbb{R}^2 \times \theta$  described above, the discretised configuration space  $\mathcal{GC}$  consists of configurations  $(k_x \partial x, k_y \partial y, k_\theta \partial \theta)$  with  $k_x, k_y, k_\theta \in \mathbb{Z}$  and a mod  $2\pi$  arithmetic being applied to the orientation additionally. The discrete configuration grid is searched for a free path under the assumption that the initial and goal configurations are  $\{\mathbf{q}_{init}, \mathbf{q}_{goal}\} \subset \mathcal{GC}$ . Further, it is assumed that two  $p$ -neighbours<sup>3</sup>  $\{\mathbf{q}_1, \mathbf{q}_2\} \subset \mathcal{GC}$  which lie in free space are connected in  $\mathbb{R}^N$  by a straight line segment which also lies in free space, i.e. the discretisation is well-suited to the nature of the obstacles.

In discretised configuration spaces, *wavefront expansion/propagation* algorithms can be applied to compute navigation functions, e.g. [Barraquand 92]. Starting from  $\mathbf{q}_{goal}$ , the potential for all 1-neighbours is recursively computed by summing up the distance from the goal (in terms of the number of discrete configurations lying in between). This distance metric is known as *Manhattan* or  $L^1$  distance. If the wavefront does not reach  $\mathbf{q}_{init}$ , the goal is not reachable from there (at the selected discretisation resolution). Otherwise, the resulting path is minimal in  $L^1$  distance. Additionally the algorithm causes the solution path to display a tendency to follow obstacle borders.

Since the computation of navigation functions is costly, even in discretised configuration spaces, a number of mechanisms has been designed to escape local minima encountered in a straight-forward depth-first search instead. In best-first potential-guided path planning (cp. [Barraquand 90b]), a tree is iteratively constructed with configurations of a discretised configuration space  $\mathcal{GC}$  as nodes. The tree root is  $\mathbf{q}_{init}$ . At each iteration, the  $p$ -neighbours of the leaf of  $T$  with the lowest potential are examined. Those neighbours whose potential value is below some (high) threshold (to avoid inserting configurations near or even within obstacles) are inserted into the tree as children of the current node. The algorithm terminates when  $\mathbf{q}_{goal}$  has been reached or the entire reachable free subset of  $\mathcal{GC}$  has been explored (failure). Conceptually this algorithm “fills up” local minima till a saddle point is reached and “escape” is possible. Best-first planning is only practical for  $N \lesssim 5$ , for larger values filling up wells is not tractable since the number of discretised configurations in a well grows exponentially with  $N$ .

To overcome this limitation, randomised algorithms have been developed. Randomised Potential-field Planners (RPP) [Barraquand 90a], [Barraquand 90b] are by themselves independent of the potential function used but are described here using a discrete grid potential as introduced above. During the algorithm execution, a

---

<sup>3</sup>  $p$ -neighbours of a configuration  $\mathbf{q}$  in  $\mathcal{GC}$  are configurations in  $\mathcal{GC}$  with at most  $p$  coordinates differing from those of  $\mathbf{q}$  and differing by exactly one discretisation step. Defined for  $1 \leq p \leq N$ .

graph  $G$  is constructed with local minima of  $U$  as nodes and links joining minima for which a path between them has been found. Commencing at  $\mathbf{q}_{init}$ , best-first motions are executed (steepest-descent, i.e. selecting the neighbour with the lowest potential in the discrete case) till a minimum  $\mathbf{q}_{loc}$  is reached. The potential field is defined in a way that if  $U(\mathbf{q}_{loc} = 0)$ , the goal configuration has been reached and the problem is solved ( $\mathbf{q}_{loc} = \mathbf{q}_{goal}$ ). Otherwise a number of similar algorithms exist which attempt to escape the local minima by executing a series of random motions at  $\mathbf{q}_{loc}$  before recommencing the execution of best-first moves. If the next minimum is different/lower than  $\mathbf{q}_{loc}$ , it is inserted as its successor in  $G$  and the above steps repeated. When a path to the goal configuration has been found, it leads there from the initial configuration via a series of local minima and potentially erratic random motions. Therefore, often some post-processing is applied to obtain a smoother path.

#### *Obstacle-aware RPP Planner (RPP<sub>obst</sub>)*

Potential-field planning is conventionally performed in binary obstacle settings. As such it is not directly applicable to rough-terrain motion planning. Potential fields have been applied to solve all-terrain navigation tasks in [Green 94], where nevertheless binary obstacle definitions are used. A control law is presented which includes the effects of detected obstacles for on-line obstacle avoidance tasks. In [Haddad 98], the probability of configuration space cells containing an obstacle is estimated using Bayesian inference based on sensor data and a knowledge base. A threshold is applied to these probabilities and a repulsive potential generated by the cells identified as obstacles - in a now binary sense. Constrained optimal paths (optimal paths passing through a given way-point) are used in [Shiller 04] to compute *cost maps*. A cost map is generated by assigning each discrete configuration the cost of the constrained optimal path going from the initial configuration to the goal configuration through it. Two single-source searches from  $\mathbf{q}_{init}$  and  $\mathbf{q}_{goal}$  are required to create a cost map. Cost maps are likened to navigation functions where the gradient at each configuration points to the closest local optimal path. This is used to generate a set of homotopic classes of paths<sup>4</sup> which are all locally optimal. In [Shimoda 05], a potential field is used for high-speed (i.e. dynamic) motion planning on rough terrain. Instead of using a classical definition in Cartesian space, “trajectory space” [Spenko 04] is used which allows to express certain dynamic conditions (e.g. side slip, roll-over) as potential function. The trajectory space is a two-dimensional space composed of the current path curvature of the vehicle and its longitudinal velocity. This space allows to determine actuator settings, throttle setting and steering angle in the example. The presence of obstacles is also

---

<sup>4</sup> Two paths are homotopic, if they can be deformed onto one another while keeping the end-points fixed and not “leaping over any obstacles” - this notion is somewhat broadened in [Shiller 04]

considered, again making use of a binary obstacle definition.

Continuous obstacle definitions for rough-terrain planning can be integrated into a potential field algorithm by using an obstacleness measure of navigational difficulty. An obstacleness-aware randomised potential-field planner (named  $RPP_{\text{obst}}$  for convenience) has been designed using  $o(\mathbf{q})$  as basis for defining the repulsive potential  $\mathbf{U}_{rep}$ . An attractive potential  $\mathbf{U}_{att}$  is created centred at the goal configuration  $\mathbf{q}_{goal}$  and the total potential computed as the superposition of  $\mathbf{U}_{rep}$  and  $\mathbf{U}_{att}$  [Ettlin 05a].

Conceptually the main difference of such an approach is that the repulsive potential is not generated by (the border of) a set of binary obstacles but contributed to by each configuration of the (discretised) terrain. Using the terrain model introduced in section 3.3.3, the configuration space is already inherently discrete along the translational  $x$  and  $y$  axes, due to the discrete nature of the raw point-based data. For practical purposes, the orientation axis  $\theta$  is also discretely sampled, resulting in the overall discrete grid potential  $\mathcal{GC}$ .

Additionally, the use of obstacleness transparently allows to consider different aspects contributing to the overall function characterising the navigational difficulty by defining corresponding components. Every individual component can be considered to generate a component potential field in the domain of  $\mathcal{GC}$ . The total repulsive potential  $\mathbf{U}_{rep\ total}$  is then a function of the component potentials, in analogy to the definition of the overall obstacleness function in equation 4.5:

$$\mathbf{U}_{\psi}(x, y, \theta) = K_{\psi}(x, y, \theta) \quad (4.24)$$

$$\mathbf{U}_{rep\ total}(x, y, \theta) = f'(\{U_{\psi}(x, y, \theta) \mid \psi \in \Psi\}) \quad (4.25)$$

As an example an application on the sample terrain shown in figure 4.1 is given. The intention of the presented example obstacleness is to prevent the robot from entering sliding motion, as introduced in equations 4.18 and 4.19. Additionally, lateral inclinations are to be discouraged, with angles  $\alpha_{\perp} > \alpha_{\perp\ max}$  to be avoided altogether. The obstacleness component which avoids all sliding motion for low robot velocities can be written as

$$K_{slide}(x, y) = \begin{cases} \frac{|\nabla T(x, y)|}{\mu_s(x, y) \circ \mu_s\ vehicle} & \text{if } |\nabla T(x, y)| < \mu_s(x, y) \circ \mu_s\ vehicle \\ 1 & \text{else} \end{cases} \quad (4.26)$$

The lateral inclination of the robot on the terrain can be expressed as

$$\alpha_{\perp}(x, y, \theta) = \tan^{-1}(\sin(|\Phi(x, y) - \theta|) \cdot |\nabla T(x, y)|) \quad (4.27)$$

which is illustrated in figure 4.7 for a terrain inclination angle  $\alpha$ . This allows to define following obstacleness component to fulfill the lateral inclination condition:

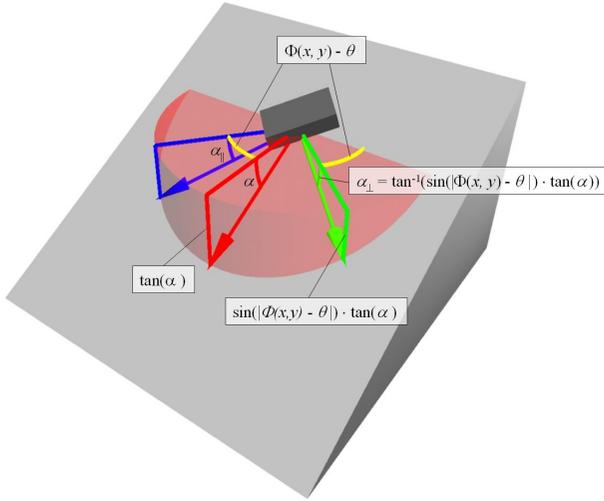


Fig. 4.7: Lateral inclination angle ( $\alpha_{\perp}$ ) illustration based on terrain inclination  $\alpha$ .

$$K_{lat}(x, y, \theta) = \begin{cases} \frac{\alpha_{\perp}(x, y, \theta)}{\alpha_{\perp max}} & \text{if } \alpha_{\perp} < \alpha_{\perp max} \\ 1 & \text{else} \end{cases} \quad (4.28)$$

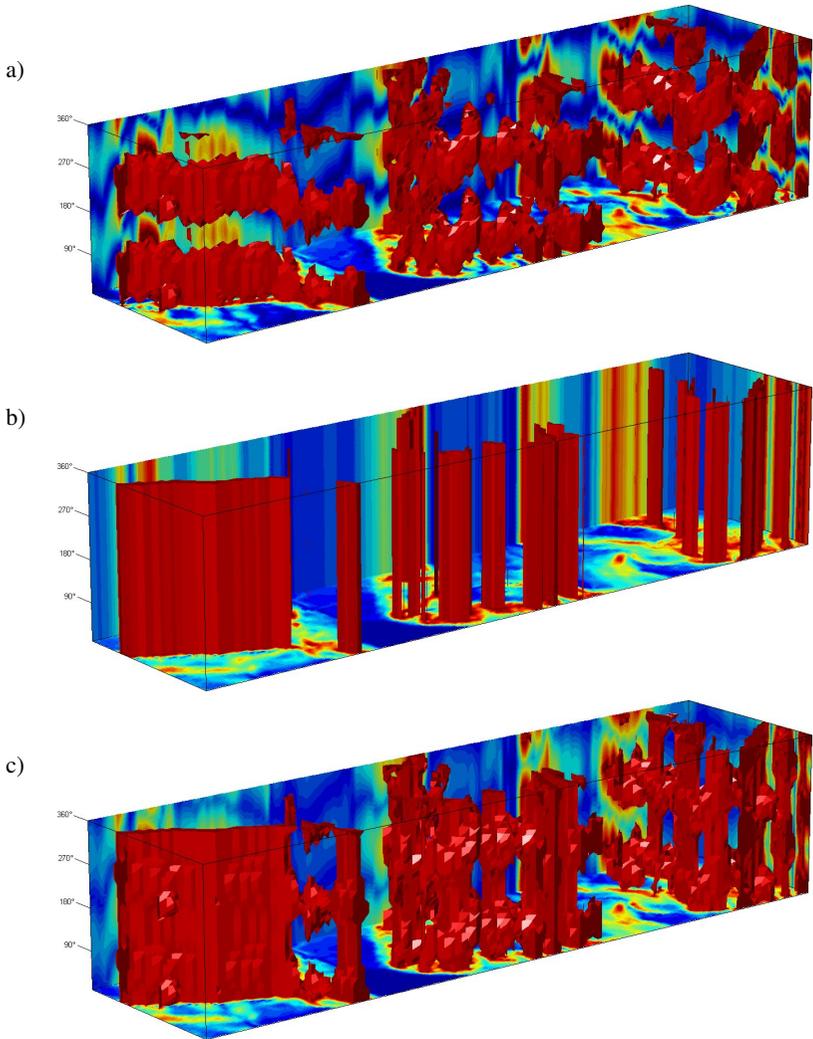
The obstacleness combination function can thus e.g. be written by computing the average of both involved obstacleness components when neither of them is forbidding (i.e. equal to one) and resolving to one otherwise:

$$o(x, y, \theta) = \begin{cases} \frac{K_{slide}(x, y) + K_{lat}(x, y, \theta)}{2} & \text{if } (K_{slide} < 1) \wedge (K_{lat} < 1) \\ 1 & \text{else} \end{cases} \quad (4.29)$$

This obstacleness definition translates directly into a repulsive potential function being defined over the same domain:

$$\mathbf{U}_{rep total}(x, y, \theta) = o(x, y, \theta) \quad (4.30)$$

Optionally, the influence of each discrete configuration on the obstacleness (and hence repulsive potential) can be defined in some neighbourhood of that configuration. Such considerations become important when the size of the robot is large in relation to the distance between discrete configurations. Taking into consideration the relative sizes of robot and terrain resolution, a kernel function for each data point can be defined as e.g. an  $N$ -dimensional Gaussian distribution. This



*Fig. 4.8:* Example of obstacleness component and combination function computation. For the terrain depicted in figure 4.1, obstacleness components a) preventing lateral inclinations  $\alpha_{\perp} > 20^{\circ}$  and b) preventing sliding motion at slow velocities are computed. c) The combination function given in equation 4.29. Solid areas in all figures indicate prohibited areas (e.g.  $o(\mathbf{q}) = 1$ ) according to the depicted function.

corresponds to applying the relevant kernel function as filter to the entire repulsive potential.

Figure 4.8 shows the obstacleness components and the combination function given above for the sample terrain shown in figure 4.1. In figure 4.8 a),  $K_{lat}$  is depicted for  $\alpha_{\perp max} = 20^\circ$ . All figures use solid areas to denote function values equal to one, i.e. prohibited areas with respect to the plotted function. Figure 4.8 b) shows  $K_{slide}$ , the obstacleness component which avoids sliding motion of the robot in static settings. To improve the readability of the graph, a constant value for the static friction between the robot and the terrain has been assumed over the entire workspace. The orientation independence of  $K_{slide}$  can be recognised in the constant obstacleness values for all robot headings at every Cartesian position. The obstacleness combination function given in equation 4.29 is shown in figure 4.8 c). One characteristic of this obstacleness definition which can be seen in the plot is that a robot located in the South-Western corner of the terrain (above the steep slope) can only escape this region by traversing the slope in a NE direction. Such a course of action leads the robot to steer approximately parallel to the terrain gradient as required by the stated conditions.

An example of the  $RPP_{obst}$  planner in action is depicted in figure 4.9. It is based on the terrain shown in figure 4.1 and makes use of the obstacleness function illustrated in figure 4.8 c) as repulsive potential  $U_{rep}$ . The attractive potential is defined as linear function  $U_{att} = \|\mathbf{q} - \mathbf{q}_{goal}\|$  which generates a force  $F_{att} = \nabla U_{att}$  of constant magnitude over the entire configuration space. A linear attractive potential does not possess the desirable convergence properties of a parabolic attractive potential (cp. equation 4.21). Nevertheless, this can be neglected in the example since the velocity of the robot is not computed as consequence of the total artificial force conceptually applied to it when generating the trajectory.

Figure 4.9 a) shows a contour map of the terrain with the robot configurations along the trajectory superimposed. The initial configuration is located in the South-Western corner, the goal in the North-East. It can be recognised how the planner steers a direct course to the goal where little hindrance to progress is present. In the region labelled “1”, the trajectory encounters the prohibitively steep slope which generates the barrier of  $o(\mathbf{q}) = 1$  visible in the SW corner of figure 4.8 c). The attractive potential has a strong component in direction East. This causes a series of zigzag motions (involving a number of point-skid-turns) being executed when repelled by the obstacleness barrier. Eventually a configuration is reached where the barrier can be traversed. After crossing the large plane in the South, the trajectory reaches the high-obstacleness region caused by the prominent plateau mountain at position 2. This leads to a local minimum which is escaped by performing a series of random moves. While performing random motions, the configurations of the robot are visualised in blue. Random moves are computed in the example by advancing a step in the current direction with a fixed probability of  $p_{move} = 0.4$  or turning a fixed angle  $\pm\Delta\theta = 10^\circ$  with probability  $p_{turn left} = p_{turn right} = 0.3$ . Importantly, a potential random motion is tested for validity (i.e. the new config-

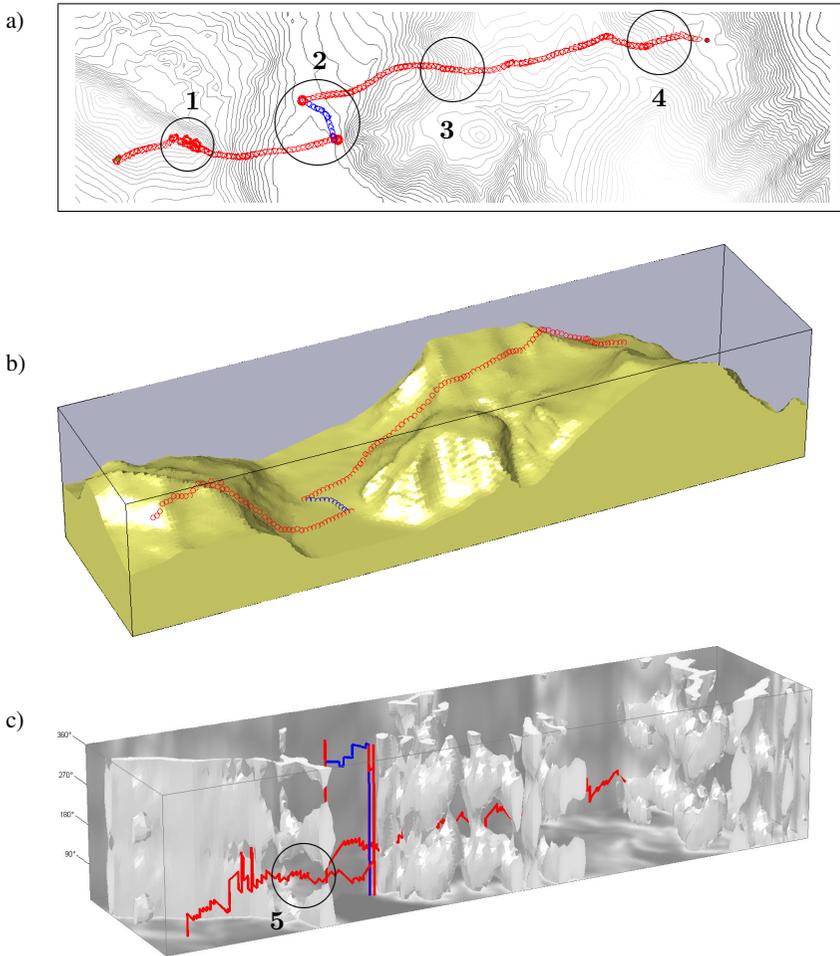


Fig. 4.9: Example of the obstacleness-aware RPP planner  $RPP_{\text{obst}}$  showing the trajectory produced for the example terrain depicted in figure 4.1. a) Contour map showing individual robot positions and orientations along the trajectory. b) Scaled 3D mesh showing the trajectory c) Configuration space showing high-obstacleness ( $o(\mathbf{q}) > 0.9$ ) regions solid. Blue trajectory segments indicate RPP random moves.

uration must fulfill  $o(\mathbf{q}_{\text{new}}) < 1$ ). After the random motions, the robot performs a point-turn and proceeds in the general direction of the target. The effects of the lateral inclination component  $K_{\text{lat}}$  can be recognised in positions **3** and **4**, where the trajectory swerves off the direct line to climb a slope more perpendicularly.

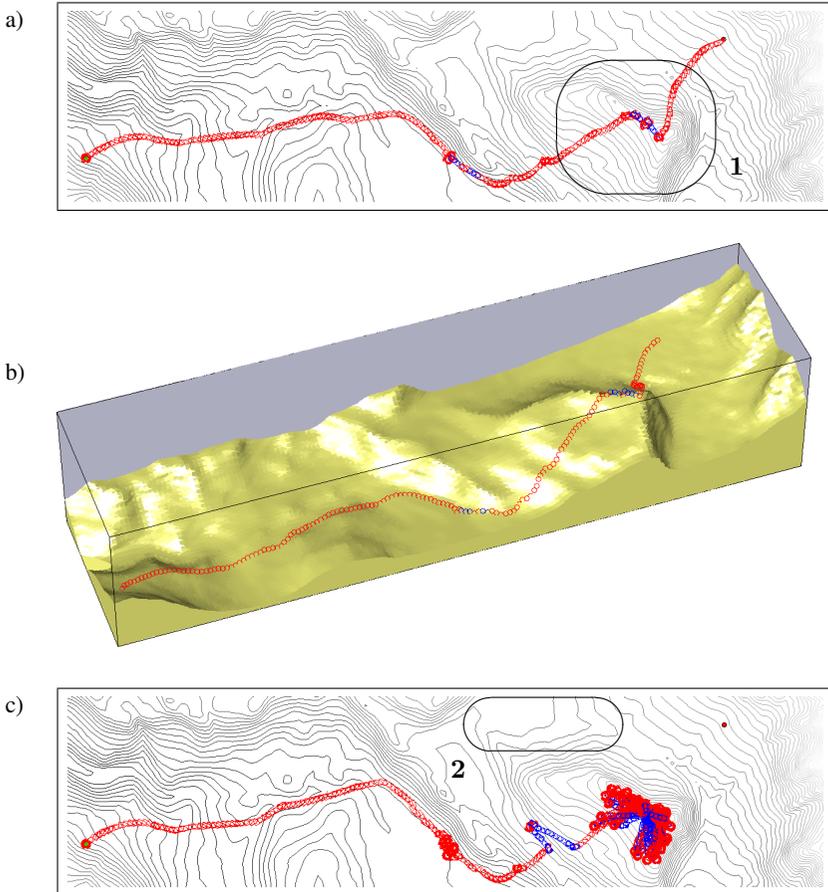


Fig. 4.10: Illustration of the local minima problem encountered in RPP planners.

The progress of the robot is shown on a scaled 3D representation of the terrain ( $z$ -axis scaling factor = 2) in figure 4.9 b) to illustrate the plausibility of the computed route. The progress through the obstacleness function in configuration space is visualised in figure 4.9 c). The solid regions indicate  $o(\mathbf{q}) > 0.9$  compared to  $o(\mathbf{q}) > 1$  in figure 4.8 c). Additionally, some smoothing has been performed which can be recognised by the lack of sharp edges visible in figure 4.8 c). At the position marked 5 it can be seen how the path is guided through the gap in the obstacleness function generated by the slope in the South-East. The long vertical segments along the orientation axis at the beginning of the random walk are created by the periodicity of the orientation.

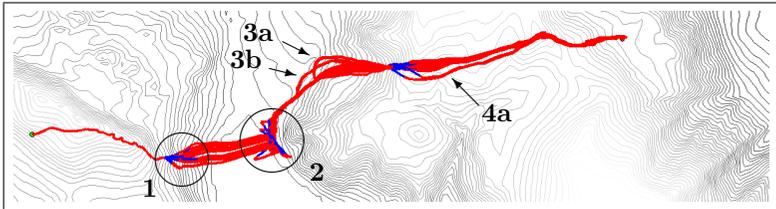


Fig. 4.11: Results of 20 obstacle-aware  $RPP_{\text{obst}}$  experiments with identical initial conditions to illustrate exploration behaviour.

The potential-field approach is well suited to perform obstacle-avoidance tasks and navigate through regions of low obstacleness in simple topographies as the one shown in the example. For more complex settings, the detection and evasion of local minima becomes a major difficulty. Figure 4.10 illustrates this limitation: figures 4.10 a) and 4.10 b) show the same algorithm described above in operation on a different terrain. The remarkable feature of this topography is the elevation marked 1 which has a comparatively gentle slope when approaching from the South-West but steep faces to the East and North-East. As can be seen in figure 4.10 a) and 4.10 b), the obstacleness-aware  $RPP_{\text{obst}}$  planner finds an escape, making a perpendicular descent down the North-Eastern face.

Figure 4.10 c) on the other hand shows the effects of applying a more stringent obstacleness function which leads to an obstacleness barrier being created at the steep slopes of mountain “1”. This causes the trajectory to get trapped in the resulting local minimum and even numerous random sequences failing at freeing the robot. In the figure, 5000 steps have been performed, where a normal path to the goal configuration consists of approximately 500 steps. Applying longer random sequences might help, but would erode the meaning of the obstacleness/potential function and does not fundamentally solve the problem - pathological cases can easily be constructed. The core of the issue is that  $RPP$  planners are local in their nature, the only feature being considered is the gradient of the potential function at the configuration of the robot. In the example,  $RPP_{\text{obst}}$  thus fails to find the easily navigable path through the plain labelled 2.

To illustrate the exploration characteristics of the  $RPP_{\text{obst}}$  planner, 20 experiments with identical initial conditions have been conducted and visualised in figure 4.11. The setting is the sample terrain shown in figure 4.1, with lower friction coefficients compared to the experiments illustrated in figures 4.8 and 4.9. This leads to additional local minima along the trajectory of the robot. The mean number of steps on a path is 458, with a standard deviation of 94. This high deviation is mainly due to two runs which follow erratic random paths when meeting the obstacleness barrier at the base of the plateau mountain (position 2). The maximal length of a

path is 826 steps, the minimal length 382 steps.

An interesting observation is that the interaction of random sequences and the obstacleness function leads to a discrete number of variant routes which are approximately followed by the different runs. After the first random sequence (labelled **1**), two branches of equal strength occur. Later on the trajectory, the route marked **3a** was followed by 7 runs, route **3b** by 3 runs, the “main” route by the remaining 10 runs. Branch **4a** was followed on two occasions. The discrete number of relatively narrow corridors is caused by the repulsive potential field which leads to valleys in its topology being followed. In a binary obstacle setting this fact is explicitly exploited in some implementations (e.g. [Barraquand 90b], [Hwang 92]) to generate a roadmap representation of the configuration space using the resulting approximation of a Voronoi diagram as graph edges and the intersections as nodes.

Region **2** has the character of a major “watershed”, where one branch of motion plans heads South-East and off the map - for that reason they could not be included in the summary. In the experiment, 5 runs interleaved with the 20 registered experiments were observed. Following such a trajectory would lead to a globally strongly differing solution if there were sufficient terrain extension available. Nevertheless, this must not be confounded with real exploration. The attractive potential emitted from the goal strongly influences all non-random motions to have some component in direction goal. This can be clearly seen in the local minima example shown in figure 4.10 c).

For global trajectory computation the local nature of potential-field approaches is generally disadvantageous. On the other hand, locality can be highly desirable in situations where global data is not available or afflicted with uncertainties. Such situations are common in practice and e.g. typically arise in sensor-based planning.

In the following section a motion planner based on rapidly-exploring random trees (RRTs) is presented. It uses obstacleness as basis for encouraging locally desired behaviour but has a global scope of the search. This allows the planner to successfully negotiate more complicated terrain topologies than the potential-field based planner discussed in this section.

### 4.1.3 Rapidly-Exploring Random Trees (RRT) Approach

The RRT family of algorithms which forms the basis of the algorithms proposed in this section was not specifically designed for rough-terrain planning but is of greater generality in motion planning. The original RRT algorithm [LaValle 98a] has demonstrated powerful exploration abilities in the configuration space. The key contribution of RRT algorithms is the way in which a configuration space is searched, with exploration being biased towards unexplored regions of  $\mathcal{C}$ . An algorithm outline is showed in figure 4.12. The Rapidly Exploring Random Tree  $\mathcal{T}$  is built by starting with the initial configuration  $\mathbf{q}_{init}$  as root node and repeat-

---

```

BUILD_RRT( $\mathbf{q}_{init}$ )
1   $\mathcal{T}.\text{init}(\mathbf{q}_{init});$ 
2  for  $k = 1$  to  $K$  do
3       $\mathbf{q}_{rand} \leftarrow \text{RANDOM\_CONFIG}();$ 
4       $\text{EXTEND}(\mathcal{T}, \mathbf{q}_{rand});$ 
5  Return  $\mathcal{T};$ 

```

---

```

EXTEND( $\mathcal{T}, \mathbf{q}$ )
1   $\mathbf{q}_{near} \leftarrow \text{NEAREST\_NEIGHBOUR}(\mathbf{q}, \mathcal{T});$ 
2  if  $\text{NEW\_CONFIG}(\mathbf{q}, \mathbf{q}_{near}, \mathbf{q}_{new})$ 
3       $\mathcal{T}.\text{add\_vertex}(\mathbf{q}_{new});$ 
4       $\mathcal{T}.\text{add\_edge}(\mathbf{q}_{near}, \mathbf{q}_{new});$ 
5      if  $\mathbf{q}_{new} = \mathbf{q}$ 
6          Return Reached;
7      else
8          Return Advanced;
9  Return Trapped;

```

---

Fig. 4.12: The rapidly-exploring-random-tree (RRT) algorithm.

edly calling the EXTEND function. EXTEND attempts to grow the tree in direction of a randomly generated configuration  $\mathbf{q}_{rand}$ . First, the nearest configuration  $\mathbf{q}_{near} \in \mathcal{T}$  is determined and the NEW\_CONFIG function called. NEW\_CONFIG creates a new node  $\mathbf{q}_{new}$  which is located one discrete step  $\varepsilon$  away from  $\mathbf{q}_{near}$  in direction of  $\mathbf{q}_{rand}$ . The new node is consequently attempted to be added to the tree and a value returned which encodes the success of the operation - either the random node has been reached (*Reached*), a step in its direction taken (*Advanced*) or the operation has failed due to being trapped at an obstacle (*Trapped*). Figure 4.13 illustrates the operation of the EXTEND function.

RRTs naturally expand to solving kinodynamic problems [LaValle 99], where a probabilistically complete kinodynamic motion planner is presented.

Introducing a bias towards expanding a single RRT in direction of the goal configuration increases convergence importantly [LaValle 01]. This is done by replacing the random configuration the extension is directed to with the goal state in a small fraction of the EXTEND operations.

Extending this concept, heuristic biasing of RRT expansion has been proposed in [Urmson 03]. The likelihood of selecting a node for expansion is computed

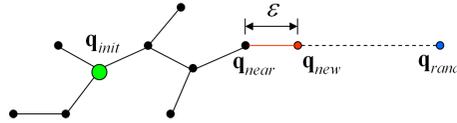


Fig. 4.13: The EXTEND function of RRT algorithms. The nearest node of the RRT  $q_{near}$  is extended one discrete step  $\epsilon$  in direction of a randomly generated configuration  $q_{rand}$ .

based on both the size of its Voronoi region (as in the original algorithm) and the integrated cost of the path leading to it. The cost measure used can either be discrete or continuous, making this approach directly applicable to rough-terrain planning.

In [Bruce 02], the RRT algorithm is applied to mobile robotics and extended in two ways. Firstly, a biased distribution is employed to generate the random configurations. For on-line trajectory planning it is assumed that the configuration space only changes slowly in the application example. Therefore, when a solution has been found, a number of configurations on the path are stored in a cache as hints for computing future solutions. Instead of always expanding the tree towards a random configuration as in the original algorithm, a configuration from the cache or the goal configuration is used with a certain probability. The second modification concerns the distance metric which is used to select the nearest neighbour in the algorithm. It is not only computed based on the distance to the target state (random, goal or waypoint) but also takes into account the distance from the initial state. By modifying this bias, a trade off can be achieved between shorter paths from root to leaves and the amount of exploration performed by the tree.

For single-query motion planning tasks, bidirectional RRT growth has proven to be effective, e.g. [LaValle 99], [LaValle 01]. The extension of both trees is performed alternately towards random configurations and the other tree. A more aggressive bias towards joining the two trees can be achieved using the “connect” heuristic proposed in [Kuffner 00], where the single extension step is repeated until either the random state/opposite tree or an obstacle is encountered.

In [Strandberg 04] it is demonstrated how the difficulties RRT-based algorithms encounter with narrow passages can be mitigated through the use of additional (“local”) trees. The basic idea is that if a random sample cannot be connected to either one of the “global” trees or another local tree, it becomes the root of a local tree itself. Local trees are expanded like standard RRTs. A renewed connection attempt with other trees is made based on the growth of the bounding-box of a tree.

In the context of probabilistic roadmaps (PRM), a similar approach has been proposed in [Akinc 03]. In the traditional PRM approach [Kavraki 96], milestones are generated in the free configuration space and connected using a local planner. This local planner is replaced by a single-query sampling-based tree method, where

Expansive Space Trees [Hsu 02] and RRTs are explicitly demonstrated.

RRTs have also proven to be useful in a wide variety of domains, including the computation of collision-free human reaching movements [Kallmann 03a], kinodynamic path planning for a blimp robot [Kim 03] and humanoid robots [Kuffner 03].

Previous applications of Rapidly Exploring Random Trees to rough-terrain navigation include [Kobilarov 04], where an approximately time-optimal rough-terrain path planner is presented which considers dynamic constraints of the robot. The planner is based on a purely geometric terrain representation and does not use any other measure for the navigational difficulty.

### *Obstacleness-aware RRT Planner (RRT<sub>obst</sub>)*

In [Ettlin 06b], the RRT<sub>obst</sub> rough terrain motion planner based on rapidly exploring random trees is presented which makes use of the obstacleness measure of navigational difficulty. RRT<sub>obst</sub> guides the search of two RRTs rooted at the initial and goal configurations towards regions of low obstacleness as well as towards one another.

An outline of the RRT<sub>obst</sub> algorithm is shown in figure 4.14. Three functions are described: the motion planner itself, the EXTEND procedure in analogy to the basic RRT algorithm and CONNECT inspired by RRT<sub>connect</sub> [Kuffner 00]. As compared to RRT<sub>connect</sub>, an additional condition related to obstacleness has been introduced. Where the original algorithms perform a test of passed configurations against  $\mathcal{C}_{free}$ , the proposed solution checks the obstacleness at the passed configuration against a value which is adapted in the course of the algorithm execution.

As in the original algorithm, the CONNECT procedure is a greedy function which repeats the EXTEND step multiple times. In [Kuffner 00], three exit conditions of CONNECT exist: 1.) *Reached* when the target configuration  $\mathbf{q}$  lies within  $\varepsilon$  of the RRT and can be added directly. 2.) *Advanced* when a step of length  $\varepsilon$  leads to  $\mathbf{q}_{new} \in \mathcal{C}_{free}$  which can be added to the tree. 3.) *Trapped* when the new configuration lies outside the free area and is rejected. In the proposed solution, the second and third conditions have been modified to check for  $\mathbf{q}_{new} \in \mathcal{C}_{[0<\xi]}$ , a more stringent condition.

NEAREST\_NEIGHBOUR remains unchanged and (approximately) determines the node of the passed tree which lies closest to the passed configuration by some metric  $\rho$ .

RANDOM\_CONFIG returns a random position  $\mathbf{q}_{rand} \in \mathcal{C}_{free}$  in the original algorithms. In the obstacleness aware RRT planner, a parameter  $\xi$  is additionally passed to the function which only returns configurations  $\mathbf{q}_{rand} \in \mathcal{C}_{[0<\xi]}$ .

NEW\_CONFIG( $\mathbf{q}$ ,  $\mathbf{q}_{near}$ ,  $\mathbf{q}_{new}$ ) performs a motion of constant length  $\varepsilon$  from  $\mathbf{q}_{near}$  towards  $\mathbf{q}$  and returns the new configuration  $\mathbf{q}_{new}$  on success. The condition for success has been modified from  $\mathbf{q}_{new} \in \mathcal{C}_{free}$  to  $\mathbf{q}_{new} \in \mathcal{C}_{[0<\xi]}$ .

---

```

EXTEND_OBST( $\mathcal{T}$ ,  $\mathbf{q}$ ,  $\xi$ )
1   $\mathbf{q}_{near} \leftarrow \text{NEAREST\_NEIGHBOUR}(\mathbf{q}, \mathcal{T})$ ;
2  if NEW_CONFIG( $\mathbf{q}$ ,  $\mathbf{q}_{near}$ ,  $\mathbf{q}_{new}$ ,  $\xi$ )
3       $\mathcal{T}.\text{add\_vertex}(\mathbf{q}_{new})$ ;
4       $\mathcal{T}.\text{add\_edge}(\mathbf{q}_{near}, \mathbf{q}_{new})$ ;
5      if  $\mathbf{q}_{new} = \mathbf{q}$ 
6          Return Reached;
7      else
8          Return Advanced;
9  Return Trapped;

```

---

```

CONNECT_OBST( $\mathcal{T}$ ,  $\mathbf{q}$ ,  $\xi$ )
1  repeat
2       $S \leftarrow \text{EXTEND}(\mathcal{T}, \mathbf{q}, \xi)$ 
3  until not ( $S = \text{Advanced}$ )
4  Return  $S$ ;

```

---

```

RRT_OBST( $\mathbf{q}_{init}$ ,  $\mathbf{q}_{goal}$ )
1   $\mathcal{T}_1.\text{init}(\mathbf{q}_{init})$ ;  $\mathcal{T}_2.\text{init}(\mathbf{q}_{goal})$ ;
2   $\xi \leftarrow \xi_{init}$ ;
3  for  $k = 1$  to  $K$  do
4       $\mathbf{q}_{rand} \leftarrow \text{RANDOM\_CONFIG}(\xi)$ ;
5      if not ( $\text{EXTEND\_OBST}(\mathcal{T}_1, \mathbf{q}_{rand}, \xi) = \text{Trapped}$ )
6          if ( $\text{CONNECT\_OBST}(\mathcal{T}_2, \mathbf{q}_{new}, \xi) = \text{Reached}$ )
7              Return PATH( $\mathcal{T}_1$ ,  $\mathcal{T}_2$ );
8      SWAP( $\mathcal{T}_1$ ,  $\mathcal{T}_2$ );
9       $\xi \leftarrow \text{UPDATE\_LIMIT}(k, K, \xi_{init})$ ;
10 Return Failure;

```

---

Fig. 4.14: RRT<sub>obst</sub>, a local obstacle-aware RRT<sub>connect</sub> algorithm

The motion planning function RRT\_OBST controls the behaviour of the RRT expansion by changing the value  $\xi$ . This variable represents the limit of obstacle-ness which is acceptable for new configurations being added to the RRT. Initially

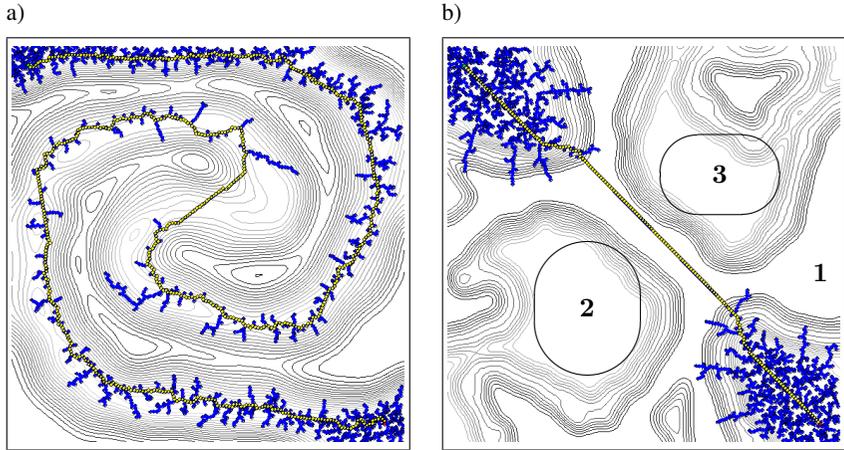


Fig. 4.15: a) The  $\text{RRT}_{\text{obst}}$  algorithm finds a solution in cases which would be impossible to solve using the RPP planner. b) The locality of the  $\text{RRT}_{\text{obst}}$  algorithm can lead to situations where favourable low-obstacle regions are not benefitted from. Contour lines shown refer directly to the obstacle function, not the underlying terrain features.

the value can be set using some heuristic, e.g. as function of the possibly known global obstacle minima.

The new function `UPDATE_LIMIT` controls the expansion of the motion planner by updating  $\xi$  in dependence of the number of iterations executed. In the early stages of the search, a low value of  $\xi$  ensures that only favourable regions of the configuration space are searched. If this yields no solution to the problem, allowing areas of increasingly higher obstacle expands the search. At this stage, the RRT being constructed is already biased towards regions of low obstacle, only evading to high obstacle regions where inevitable.

Using a non-decreasing `UPDATE_LIMIT` function,  $\text{RRT}_{\text{obst}}$  displays the desired behaviour locally. In more complex obstacle topologies it degenerates to the original  $\text{RRT}_{\text{connect}}$  algorithm. In particular, local minima are disregarded as a consequence of the non-decreasing behaviour of `UPDATE_LIMIT`. In algorithms based on potential-fields, local minima of the total potential function are problematic for the planner and are best avoided, as described above. To the contrary, local minima of the obstacle function are desirable regions in obstacle-aware RRT-based algorithms since low-obstacle areas are easily navigable.

The  $\text{RRT}_{\text{obst}}$  algorithm is shown in operation in figure 4.15. To improve legibility, contour lines of the obstacle function are shown instead of terrain features. The grown RRT-trees are shown in blue, the solution path is marked yellow. In

figure 4.15 a), the algorithm finds a solution in a non-trivial obstacleness topology, a double spiral of low obstacleness with the initial and goal configurations at either end. The obstacleness increases towards the central section of the spiral. The high-obstacleness ridges between low-obstacleness regions are (nearly) impassable, i.e.  $\mathcal{C}_{[o \approx 1]}$ . With the RPP planner, such an obstacleness function would be virtually impossible to navigate due to the pronounced local minima which would result in the total potential.

The locality of the  $\text{RRT}_{\text{obst}}$  algorithm is illustrated in figure 4.15 b). The initial and goal configurations are located in low-obstacleness regions surrounded by high-obstacleness barriers. Those barriers are linked by a ridge of equally high obstacleness forming the connected equipotential area labelled **1**. To both sides of the ridge are two significant obstacleness depressions (marked **2** and **3**) which would be beneficial to traverse when navigating from  $\mathbf{q}_{\text{init}}$  to  $\mathbf{q}_{\text{goal}}$ . Due to this obstacleness topology and the use of the “connect” heuristic, once the planner has found a path out of the two low-obstacleness areas at  $\mathbf{q}_{\text{init}}$  and  $\mathbf{q}_{\text{goal}}$ , a path following a straight line between the two nearest nodes of the tree is planned. This path lies on the high-obstacleness ridge while even a small deviation to either side would result in significantly lower obstacleness values along the central section of the route. Depending on the concrete obstacleness function adopted, this would allow e.g. faster, less risky or more energy efficient progress to be made.

The  $\text{RRT}_{\text{obst}}$  algorithm applied to the problem which caused the  $\text{RPP}_{\text{obst}}$  planner to get stuck in a local minimum is illustrated in figure 4.16. The distance metric  $\rho$  for the RRT algorithms in the configuration space  $\mathcal{C}_{\text{motion}} = \tilde{\mathcal{C}} \subset \mathbb{R}^3$  has been defined as

$$\rho = \rho_{\text{rot } 1} + \rho_{\text{trans}} + \rho_{\text{rot } 2} \quad (4.31)$$

where  $\rho_{\text{rot } 1}$  denotes the number of purely rotational actions required to align the robot with the direction of motion defined by a straight line to the target configuration. The number of discrete translational motion steps required to reach the Cartesian position of the target is  $\rho_{\text{trans}}$ . The number of rotations to achieve the orientation of the target configuration after the translation is  $\rho_{\text{rot } 2}$ . Such a definition corresponds to a motion strategy in which the robot bridges the gap between two configurations by first performing a point-turn to attain an orientation heading towards the target configuration. Next, the robot follows a straight trajectory to the target followed by another point turn to reach the orientation of the target configuration.

Figures 4.16 a) and b) show the resulting trajectory on the terrain, in which some straight line segments generated by the “connect” heuristic can be recognised (e.g. stretches marked **1**). The area which caused the local minimum in the total potential of the  $\text{RPP}_{\text{obst}}$  planner (labelled **2**, cp. figure 4.10 c) for the  $\text{RPP}_{\text{obst}}$  behaviour) is circumnavigated by the planner. Figure 4.16 shows a visualisation

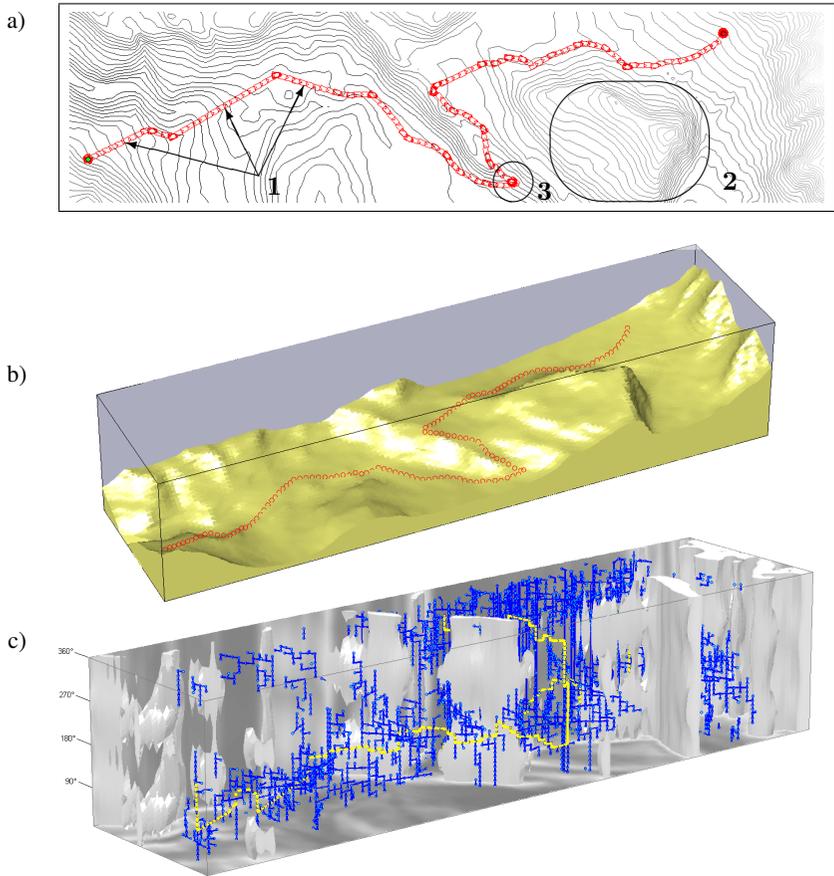


Fig. 4.16: The  $RRT_{obst}$  planner operating on the scenario which caused the RPP planner to get trapped in a local minimum, cp. figure 4.10 c). Figures a) and b) show the computed route which evades the “trap” caused by the mountain labelled **2** and benefits of the easily navigable plane to the North thereof. c) shows the computed RRT-tree plotted in  $\mathcal{C} \subset \mathbb{R}^3$  with the heading represented along the vertical axis.

of the configuration space with the heading plotted along the vertical axis. The two trees are shown in blue; all segments are connected considering the  $\text{mod}(2\pi)$  arithmetic of the orientation. The solution path is highlighted in yellow. The two trees grown from the initial and goal configurations are joined at the location of the highest obstacleness along the route, labelled **3**. Since this is the only major obstacleness maxima along the route, the  $RRT_{obst}$  algorithm is adequate for planning this itinerary. If more such high-obstacleness regions existed between  $\mathbf{q}_{init}$  and

$\mathbf{q}_{goal}$ , the locality of the algorithm would lead to the issue of the intermediate low-obstacleness regions not being taken advantage of, as illustrated in figure 4.15 b).

*Obstacleness-aware RRT Planner with Way-points ( $RRT_{obst\ way}$ )*

To design a motion planner which displays equally advantageous behaviour in more complex obstacleness functions, it would be possible to adopt a more sophisticated control of  $\xi$ , i.e. the regions of allowed exploration. This would inevitably lead to topological considerations on the obstacleness function which has been considered an unjustifiably complicated approach. Instead, an algorithm named  $RRT_{obst\ way}$  has been developed [Ettlin 06a] which combines the benefits of using local trees as in [Strandberg 04] with the additional guidance provided by a cost function. As measure of cost the degree of obstacleness is used to maintain a high degree of generality. The local trees are implemented as additional RRTs rooted at random configurations  $\mathbf{q} \in \mathcal{C}_{motion}$ . A similar approach was first suggested in [LaValle 01], where it is also outlined how a probabilistic roadmap (PRM) planner [Kavraki 96] can be emulated by growing an RRT at each random configuration generated during the growing process.

The proposed algorithm differs from that approach in that a constant number of RRT root nodes are generated at the beginning of the planning process, rooted at locations following a random distribution in  $\mathcal{C}_{motion}$ . To date, a uniform distribution has been applied but it would be of interest to investigate in which situations a semantically richer distribution is advantageous. The purpose of the additional nodes is to ensure that each significant local minima in the obstacleness function is explored by a structure of RRTs before it is connected to RRTs outside the minima. This approach leads to solution paths which pass through areas of low obstacleness wherever possible.

The  $RRT_{obst\ way}$  algorithm is outlined in figure 4.17. Like in  $RRT_{obst}$ , the areas of allowed exploration are controlled by the variable  $\xi$  which is updated non-decreasingly by UPDATE\_LIMIT. All root nodes located in an area  $\mathcal{C}_{[o < \xi]}$  are considered active RRTs and extended in turn. After every extension, an attempt is made to connect a tree to its nearest neighbour, emulating the greedy heuristic of  $RRT_{connect}$ . When such a connection is achieved, the two trees are merged and treated as one thereafter. To implement this functionality, two new functions have been added to the algorithm.

NEAREST\_TREE determines which tree in the list of active RRTs is closest to the passed point. The nearest tree as well as the nearest node within that tree are returned. As with the NEAREST\_NEIGHBOUR function itself, it is possible to use an approximate nearest neighbour algorithm (e.g. [Atramentov 02], [Indyk 98], [Arya 98]) to potentially significantly improve computation times. A benefit of  $RRT_{connect}$  is that frequently long path segments can be created with a

---

```

RRT_OBST_WAY( $\mathbf{q}_{init}, \mathbf{q}_{goal}$ )
1   $\mathcal{T}_1$ .init( $\mathbf{q}_{init}$ );  $\mathcal{T}_2$ .init( $\mathbf{q}_{goal}$ );
2   $\xi \leftarrow \xi_{init}$ ;
3  for  $n = 3$  to  $N$  do
4       $\mathbf{q}_n \leftarrow \text{RANDOM\_CONFIG}(1)$ ;
5       $\mathcal{T}_n$ .init( $\mathbf{q}_n$ );
6  for  $k = 1$  to  $K$  do
7      for  $n = 1$  to  $N$  do
8           $\mathbf{q}_{rand} \leftarrow \text{RANDOM\_CONFIG}(\xi)$ ;
9          if not ( $\text{EXTEND}(\mathcal{T}_n, \mathbf{q}_{rand}, \xi) = \text{Trapped}$ )
10              $\{m, \mathbf{q}_{near_m}\} \leftarrow \text{NEAREST\_TREE}(\mathbf{q}_{new})$ ;
11              $\mathbf{q}_{near_n} \leftarrow \text{NEAREST\_NEIGHBOUR}(\mathbf{q}_{near_m}, \mathcal{T}_n)$ ;
12             if ( $\text{CONNECT}(\mathcal{T}_m, \mathbf{q}_{near_n}, \xi) = \text{Reached}$ )
13                 if ( $\{m, n\} = \{1, 2\}$ )
14                     Return  $\text{PATH}(\mathcal{T}_1, \mathcal{T}_2)$ ;
15                 else
16                      $\text{MERGE\_TREES}(\mathcal{T}_m, \mathcal{T}_n)$ ;
17              $\xi \leftarrow \text{UPDATE\_LIMIT}(k, K, \xi_{init})$ ;
18  Return Failure;

```

---

Fig. 4.17: Outline of  $\text{RRT}_{\text{obst way}}$ , a global obstacleness-aware  $\text{RRT}_{\text{connect}}$  algorithm using waypoints at which local  $\text{RRT}_{\text{obst}}$  trees are grown.

single call to the costly  $\text{NEAREST\_NEIGHBOUR}$  routine. While this advantage remains in  $\text{RRT}_{\text{obst way}}$ , attention must be paid to select an efficient implementation for the expensive  $\text{NEAREST\_TREE}$  function, e.g. based on bounding volumes as in [Strandberg 04].

The second new function is  $\text{MERGE\_TREES}$  which is called to join the RRTs whenever a connection attempt between two trees has succeeded. In the algorithm outline shown in figure 4.17, the initial configuration is the root of  $\mathcal{T}_1$  and the goal configuration the root of  $\mathcal{T}_2$ . In order to recognise the success condition on line 13,  $\text{MERGE\_TREES}$  is assumed to merge the trees and store them in the tree with the lower index.

Figure 4.18 illustrates the operation of the  $\text{RRT}_{\text{obst way}}$  algorithm. A continuous obstacleness function has been defined in  $\mathcal{C}_{\text{motion}} \subset \mathbb{R}^2$  and plotted as contour lines for illustration purposes. Regions of higher obstacleness are visualised by

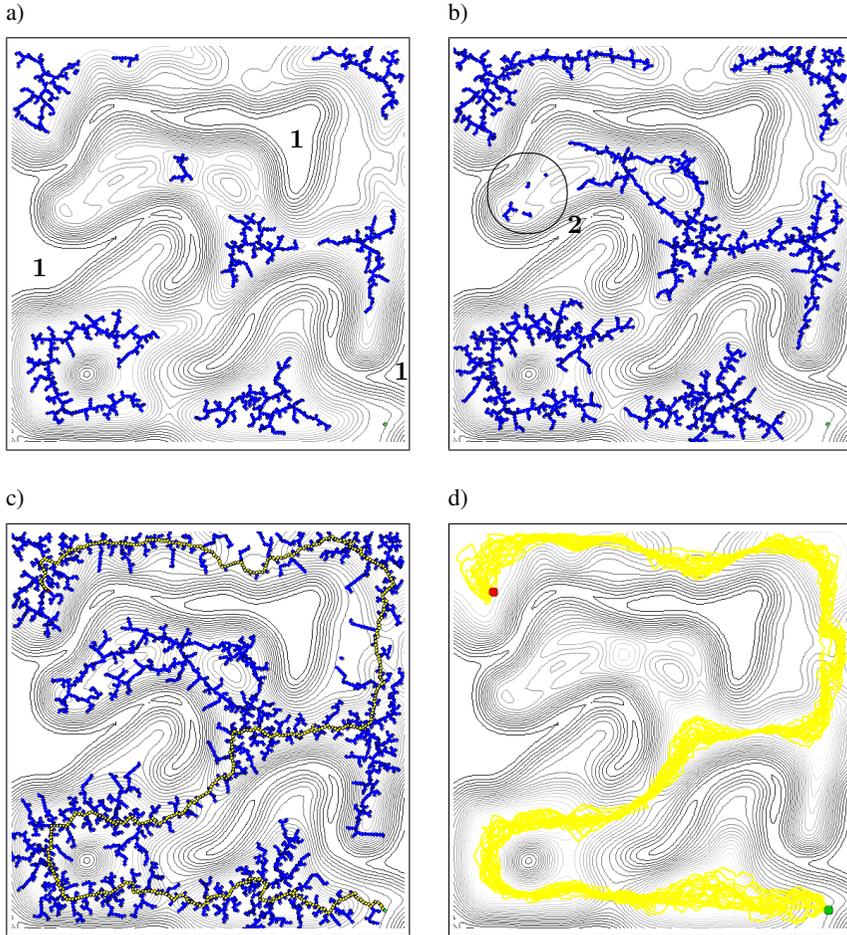


Fig. 4.18: a)-c) Different stages of the RRT<sub>obst way</sub> algorithm during a single motion planning task. In figure c), the solution path has been found and is highlighted in yellow. d) Overlay of solution paths of 20 runs with identical starting conditions.

darker contour lines. The contour lines corresponding to the highest obstacleness values denote impassable areas  $\mathcal{C}_{[o=1]}$ , e.g. regions labelled **1**.  $\mathbf{q}_{init}$  and  $\mathbf{q}_{goal}$  are located in the South-Eastern and North-Western corners, respectively. The goal configuration is located in an area of relatively low obstacleness ( $o(\mathbf{q}_{goal}) \approx 0.24$ ), while the initial configuration is at  $o(\mathbf{q}_{init}) \approx 0.72$ . In this experiment 100 random configurations have been generated as potential local tree roots following a uniform distribution over  $\mathcal{C}_{motion}$ . The UPDATE.LIMIT procedure has been programmed

to linearly increase the value of  $\xi$  during the operation.

Figure 4.18 a) shows an early stage in which  $\xi$  limits the generation of new configurations to areas of low obstacleness. Rooted at random configurations, eight trees have been grown. The greedy behaviour of  $\text{RRT}_{\text{connect}}$  can be recognised in the straight line segments included in most of the trees. At this stage, the initial configuration has not yet been reached since  $\xi < o(\mathbf{q}_{\text{init}})$  while  $\mathbf{q}_{\text{goal}}$  is already connected to its own tree.

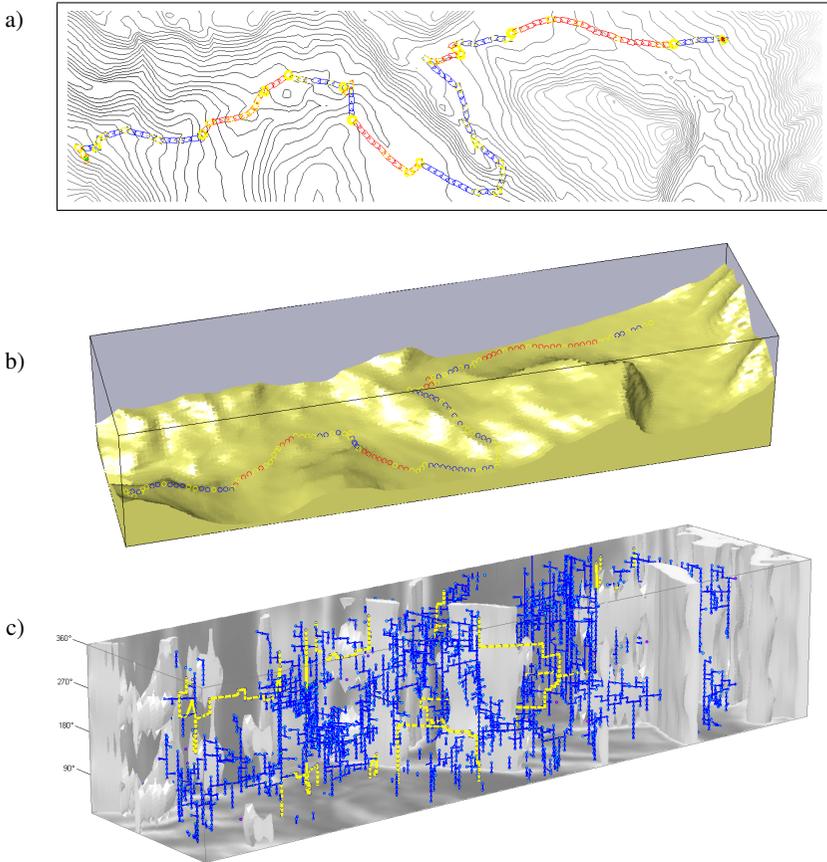
In figure 4.18 b),  $\xi$  has been sufficiently increased to allow the central three trees to have merged. Likewise, the two trees in the North-West have been combined. Also, four new trees have just emerged at the far end of the central obstacleness valley (area labelled 2). Still, the initial configuration is outwith the explored areas; in this particular example, there exists a path  $\tau$  between  $\mathbf{q}_{\text{init}}$  and  $\mathbf{q}_{\text{goal}}$  with  $\forall \mathbf{q} \in \tau : o(\mathbf{q}) \leq o(\mathbf{q}_{\text{init}})$ .

The final tree and solution path is shown in figure 4.18 c).  $\xi$  has been increased sufficiently to allow access to  $\mathbf{q}_{\text{init}}$ . In the illustrated case,  $o(\mathbf{q}_{\text{init}})$  is the highest obstacleness encountered on the path. It can easily be shown that given an infinitesimally fine control of  $\xi$ , a solution is found as soon as the allowed areas of exploration connect  $\mathbf{q}_{\text{init}}$  with  $\mathbf{q}_{\text{goal}}$ . The maximum obstacleness  $\lceil o(\mathbf{q}) \rceil = \max_{\mathbf{q} \in \tau} (o(\mathbf{q}))$  on such a path corresponds to the configuration where the allowed areas of  $\mathcal{C}_{\text{motion}}$  are last connected. If  $\lceil o(\mathbf{q}) \rceil$  is located in the terrain, a narrow passage problem arises when connecting the allowed areas at that point. Therefore in practice a heuristic tradeoff between growth rate of  $\xi$  and minimisation of  $\lceil o(\mathbf{q}) \rceil$  needs to be considered. In the experiment illustrated in figure 4.18,  $\lceil o(\mathbf{q}) \rceil = o(\mathbf{q})_{\text{init}}$ . If a potential tree root lies at  $\lceil o(\mathbf{q}) \rceil$ , the narrow passage problem does not arise since that node is expanded as soon as  $\xi$  allows such a step and is hence immediately connected to the rest of the merged trees.

Applying  $\text{RRT}_{\text{obst way}}$  to the problem showing the locality of  $\text{RRT}_{\text{obst}}$  depicted in figure 4.15 b) leads to a trajectory being planned which benefits from the local obstacleness minima along the route. The trajectory passes through one of the obstacleness minima to either side of the ridge traversed by  $\text{RRT}_{\text{obst}}$ . Either side is traversed with a similar probability in dependency on the random tree node locations.

A trajectory computed by the  $\text{RRT}_{\text{obst way}}$  algorithm operating on the same terrain shown in figures 4.10 and 4.16 (for the  $\text{RPP}_{\text{obst}}$  and  $\text{RRT}_{\text{obst}}$  planners, respectively) is visualised in figure 4.19. In the example 50 randomly distributed potential RRT roots have been created following a uniform distribution. The influence of the random points can be clearly recognised, by leading to discernible “way-points” along the trajectory.

Since the RRTs have been designed without explicitly encoding the direction of motion, some stretches of the trajectory are computed to be traversed in backwards motion. Such path segments are highlighted by blue configurations in figures 4.19



*Fig. 4.19:* The raw output of the  $RRT_{\text{obst way}}$  planner operating on the same scenario depicted in figures 4.10 and 4.16. 50 potential random tree roots are created in the configuration space following a uniform random distribution. Due to the heading being included in the configuration space and the nature of RRTs, some stretches are planned as backwards motions (plotted in blue in figures a) and b)). Pure rotational motions are plotted in yellow. c) The path through configuration space.

a) and b). Configurations where exclusively rotational motions are performed are visualised in yellow.

The stretches to be traversed in backwards motion are a result of the way the local RRTs are grown and connected: the goal is to join trees by reaching the same configuration. In the illustrated case, a configuration consists exclusively of  $\mathbf{q} \in \{x, y, \theta\}$  with  $\theta$  encoding the orientation of the robot and not the direction of

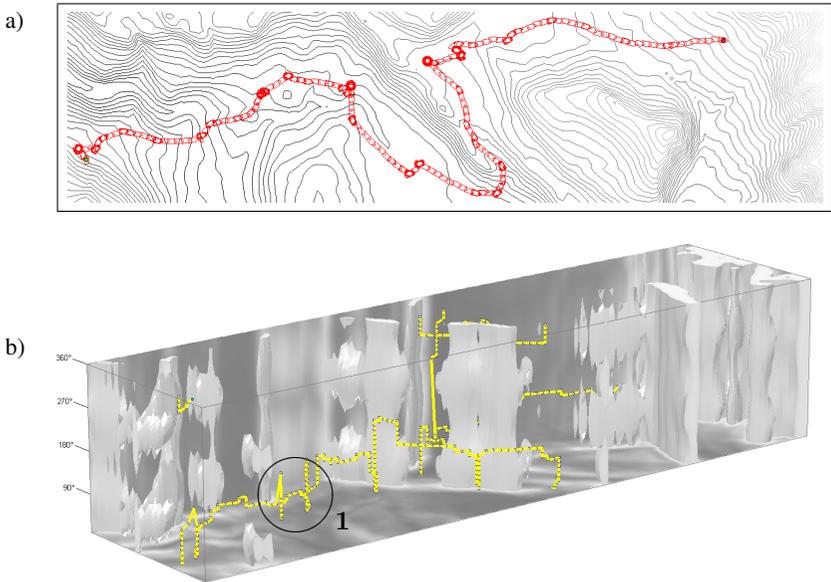


Fig. 4.20: Post-processed output of the  $RRT_{\text{obst way}}$  trajectory shown in figure 4.19. All path segments are traversed in forward motion; rotations have been adapted and are in agreement with the obstacleness constraints.

motion. Figure 4.19 c) shows the computed trajectory in the configuration space with the visualised obstacleness function. The formation of reverse stretches can be perceived in the way trees are joined in  $\mathcal{C}$ . In  $RRT_{\text{obst}}$ , the same phenomenon can be observed when joining  $\mathcal{T}_{\text{init}}$  and  $\mathcal{T}_{\text{goal}}$ . There, the issue can be trivially diverted by growing  $\mathcal{T}_{\text{goal}}$  “backwards”, using opposite orientations to the direction of growth. This option is barred in  $RRT_{\text{obst way}}$  since the connectivity of the local trees is not known at the time of their growth.

Given the symmetry of the employed obstacleness components (in particular  $K_{\text{lat}}$ , the lateral inclination component), a trajectory as shown in figures 4.19 a) and b) can be directly employed for robots which are capable of reverse motions. Nevertheless, it can be desirable to avoid backwards motions. To do so, either the configuration space and RRT growth/connectivity needs to be adapted to explicitly include the direction of motion or the path post-processed.

Again, given the symmetry of the obstacleness components, a possible post-processing scheme consists of inverting the backwards stretches and recomputing the pure rotational motions (highlighted in yellow in figure 4.19 a)) linking stretches of translational motion. When recomputing the rotations, the obstacleness needs to be taken into consideration to ensure the motions are adequate for the given task.

The resulting trajectory is visualised in figure 4.20. In figure 4.20 b) it becomes obvious that further post-processing of the trajectory might lead to better results. The locations where a number of rotations is performed in one direction (e.g. in the area labelled 1), only to be compensated after a small number of translational motions are candidate situations to be optimised. It would need to be investigated in each case if better results can be achieved under the constraints of the obstacleness conditions.

#### 4.1.4 Performance Comparison

Classifying the discussed algorithms of the RRT family,  $\text{RRT}_{\text{obst way}}$  can be considered the most general.  $\text{RRT}_{\text{obst}}$  can be understood as a specialisation thereof which employs no local trees.  $\text{RRT}_{\text{connect}}$  in turn represents the specialisation of  $\text{RRT}_{\text{obst}}$  using a binary obstacle/obstacleness definition. In the first experiment, these algorithms are compared operating on the sample terrain depicted in figure 4.1 to solve the task illustrated in figure 4.9. The second and third experiments consists of solving the task shown in figure 4.10. In experiment 2, the obstacleness function is selected in a way which allows the randomised potential-field algorithms to conclude successfully, as shown in figure 4.10 a) and b). Conversely, the third experiment is run with the obstacleness function which leads to the pronounced local minimum in the total potential function of the RPP algorithms, cp. figure 4.10 c). RPP and the obstacleness-aware  $\text{RPP}_{\text{obst}}$  consequently cannot solve the task.

The terrains used in experiments 1-3 consist of relatively simple topologies. The advantages of using the  $\text{RRT}_{\text{obst way}}$  algorithm become apparent when operating on more complex terrain topologies. The fourth experiment therefore is run on the task shown in figure 4.18. Again the RPP-based algorithms cannot find a solution due to local minima caused by the higher terrain complexity.

For the analysis, a bi-directional variant of the original RRT named  $\text{RRT}_{\text{Ext Ext}}$  in [LaValle 01] is also included. The algorithm alternately performs EXTEND steps with both trees,  $\text{RRT}_{\text{connect}}$  is consequently called  $\text{RRT}_{\text{Ext Con}}$ . Since both RRT and  $\text{RRT}_{\text{connect}}$  both do not take into account any information about navigational difficulty of the traversed terrain (i.e. obstacleness), the experimental setup is biased towards  $\text{RRT}_{\text{obst}}$  and  $\text{RRT}_{\text{obst way}}$ . The known algorithms should therefore be considered only as reference. The same holds true for the RPP implementation compared to the obstacleness-aware  $\text{RPP}_{\text{obst}}$  planner.

The obstacleness-aware RRT planners have been executed using different parameter sets. The individual algorithms are labelled as “ $\text{RRT}_{\text{obst},n,k}$ ” and “ $\text{RRT}_{\text{obst way},i,n,k}$ ”.  $n$  is the number of iterations (in thousands) required by the UPDATE\_LIMIT function to increase  $\xi$  (the variable controlling the allowed areas of exploration) from the initial value to  $\xi = 1$ . This value does not correspond to the number of executed iterations since the algorithms terminate at lower values of

$\xi$ , as soon as a solution has been found.  $l$  denotes the number of potential local tree roots randomly generated in  $RRT_{\text{obst way}}$ .

Each algorithm has been executed 20 times to take into consideration the effects of randomisation on all algorithms. The mean values and standard deviations of the registered characteristics are given in the result tables, denoted by prefixes  $\mu$  and  $\sigma$  respectively. Following measures are used to characterise the solution paths:

- $|\tau|$  : Path length.
- $\Sigma o(\mathbf{q})$  : Total obstacleness on path.
- $\mu_{o(\mathbf{q})}$  : Mean obstacleness on path.
- $[o(\mathbf{q})]$  : Maximal obstacleness on path.

Results for the first experiment operating on the task shown in figure 4.9 are summarised in table 4.1. As compared to the obstacleness function shown in figure 4.9 c), the task has been made somewhat harder by lowering the allowed amount of lateral inclination  $\alpha_{\perp}$  (cp. equations 4.27 - 4.29). This modification leads to narrower passages of admissible obstacleness along the route, thus better highlighting the differences between the algorithms.

Comparing RPP to  $RPP_{\text{obst}}$ , the obstacleness-aware algorithm produces longer trajectories, which is an expected outcome. The total cost of the trajectories is lower in  $RPP_{\text{obst}}$  by a small margin. As a consequence, the mean obstacleness on a trajectory configuration is significantly lower. The maximal obstacleness on the path is practically identical but lower in  $RPP_{\text{obst}}$ . In this example, the RPP algorithms need to perform an average of four series of random motions which

Algorithm	$\mu_{ \tau }$	$\sigma_{ \tau }$	$\mu_{\Sigma o(\mathbf{q})}$	$\sigma_{\Sigma o(\mathbf{q})}$	$\mu_{\mu_{o(\mathbf{q})}}$	$\sigma_{\mu_{o(\mathbf{q})}}$	$\mu_{[o(\mathbf{q})]}$	$\sigma_{[o(\mathbf{q})]}$
RPP	403.4	25.6%	201.6	32.4%	0.494	10.5%	0.950	1.7%
$RPP_{\text{obst}}$	497.9	26.9%	197.7	19.5%	0.407	13.9%	0.942	1.6%
RRT	149.9	4.4%	56.7	7.0%	0.378	7.8%	0.888	5.2%
$RRT_{\text{connect}}$	152.7	8.5%	61.2	11.0%	0.401	7.2%	0.874	5.8%
$RRT_{\text{obst}10k}$	186.4	8.3%	74.7	8.1%	0.402	6.6%	0.848	5.6%
$RRT_{\text{obst}50k}$	204.5	7.3%	70.7	12.9%	0.345	8.5%	0.705	10.9%
$RRT_{\text{obst}100k}$	214.4	7.5%	70.4	9.3%	0.328	6.7%	0.664	16.6%
$RRT_{\text{obst way}100 10k}$	369.6	20.2%	105.1	16.7%	0.288	9.3%	0.893	12.6%
$RRT_{\text{obst way}100 50k}$	396.1	12.7%	98.1	15.1%	0.248	6.5%	0.860	15.8%
$RRT_{\text{obst way}100 100k}$	402.1	14.6%	97.4	16.4%	0.242	8.7%	0.754	14.8%
$RRT_{\text{obst way}500 10k}$	474.0	13.7%	120.0	25.1%	0.250	11.3%	0.866	9.2%
$RRT_{\text{obst way}500 50k}$	514.3	17.3%	125.2	21.3%	0.243	10.1%	0.894	13.8%
$RRT_{\text{obst way}500 100k}$	509.1	14.9%	113.3	18.5%	0.222	8.9%	0.852	16.3%

Tab. 4.1: Comparison of algorithm performance for the task illustrated in figure 4.9.

Algorithm	$\mu \tau $	$\sigma \tau $	$\mu\Sigma o(\mathbf{q})$	$\sigma\Sigma o(\mathbf{q})$	$\mu\mu_o(\mathbf{q})$	$\sigma\mu_o(\mathbf{q})$	$\mu\lceil o(\mathbf{q}) \rceil$	$\sigma\lceil o(\mathbf{q}) \rceil$
RPP	342.3	20.9%	192.2	31.8%	0.552	11.1%	0.970	1.2%
RPP <sub>obst</sub>	354.4	24.0%	187.0	34.4%	0.518	11.0%	0.962	1.5%
RRT	175.8	7.8%	66.9	10.1%	0.381	6.7%	0.895	9.0%
RRT <sub>connect</sub>	183.8	9.7%	71.2	13.1%	0.387	7.7%	0.915	9.0%
RRT <sub>obst10k</sub>	185.8	8.0%	62.4	12.6%	0.335	7.9%	0.892	17.4%
RRT <sub>obst50k</sub>	202.4	10.2%	57.3	10.8%	0.284	6.1%	0.693	28.7%
RRT <sub>obst100k</sub>	228.3	9.1%	62.0	11.7%	0.271	6.1%	0.562	21.7%
RRT <sub>obst way100 10k</sub>	426.1	23.5%	91.3	23.3%	0.216	7.8%	0.828	18.0%
RRT <sub>obst way100 50k</sub>	435.1	12.9%	86.3	12.8%	0.199	5.5%	0.720	19.5%
RRT <sub>obst way100 100k</sub>	403.3	13.6%	74.2	12.0%	0.185	6.1%	0.706	23.5%
RRT <sub>obst way500 10k</sub>	582.0	20.8%	124.6	21.5%	0.215	13.5%	0.895	12.9%
RRT <sub>obst way500 50k</sub>	552.0	15.1%	102.5	16.6%	0.186	8.1%	0.816	21.7%
RRT <sub>obst way500 100k</sub>	535.1	17.1%	93.8	16.9%	0.176	9.8%	0.829	22.1%

Tab. 4.2: Comparison of algorithm performance for the task shown in figures 4.10 a) and b).

is reflected in the high standard deviations of path length and total cost. These random sequences in high-obstacleness areas lead to the similar results and only small benefits of including an obstacleness measure. The total path costs, mean and maximal obstacleness values of both RPP and RPP<sub>obst</sub> are significantly higher than with the RRT-based approaches. While being a topologically simple terrain, the complexity is too high to yield good results with the gradient-following approach of potential field algorithms.

The shortest routes and lowest total costs are achieved by the RRT and RRT<sub>connect</sub> algorithms in this example. Selecting a direct route compensates the higher mean costs in this case. This is particularly true in comparison with RRT<sub>obst way</sub>, where an increased number of random configurations leads to disadvantageous detours being made in the simple topology of the experiment. This serves as good example to show that too many local trees lead to unnecessarily complicated trajectories. This effect is amplified when using fast increase rates of  $\xi$ , allowing premature access to many random configurations which are consequently connected. The trade-off between better computation times and higher-quality trajectories can be recognised within the RRT algorithms containing an equal number of local trees. In the straightforward obstacleness topology of the first example, the best suited algorithms are RRT<sub>obst</sub> which lead to relatively short trajectories with acceptable total costs and low values for the mean and maximal obstacleness on the route.

The second experiment summarised in table 4.2 deals with the task illustrated in figure 4.10 a) and b). Again a simple overall terrain topology leads to similar results as in the first experiment. The solutions for the potential field algorithms

Algorithm	$\mu \tau $	$\sigma \tau $	$\mu_{\Sigma o(\mathbf{q})}$	$\sigma_{\Sigma o(\mathbf{q})}$	$\mu_{\mu_o(\mathbf{q})}$	$\sigma_{\mu_o(\mathbf{q})}$	$\mu_{\lceil o(\mathbf{q}) \rceil}$	$\sigma_{\lceil o(\mathbf{q}) \rceil}$
RPP/RPP <sub>obst</sub>	-	-	-	-	-	-	-	-
RRT	198.5	6.5%	74.3	5.3%	0.375	7.5%	0.894	12.7%
RRT <sub>connect</sub>	194.8	5.3%	73.5	7.3%	0.377	5.8%	0.851	12.6%
RRT <sub>obst10k</sub>	216.0	9.9%	71.3	16.0%	0.329	9.6%	0.870	12.7%
RRT <sub>obst50k</sub>	239.6	11.7%	63.8	14.7%	0.266	6.2%	0.626	24.5%
RRT <sub>obst100k</sub>	238.3	12.4%	59.5	16.2%	0.249	8.3%	0.554	22.1%
RRT <sub>obst way100 10k</sub>	431.8	13.2%	109.4	19.8%	0.252	11.9%	0.908	10.9%
RRT <sub>obst way100 50k</sub>	421.5	13.3%	86.0	15.5%	0.204	8.4%	0.834	15.2%
RRT <sub>obst way100 100k</sub>	418.9	12.3%	79.1	14.8%	0.189	7.5%	0.782	15.5%
RRT <sub>obst way500 10k</sub>	611.5	17.0%	139.6	21.9%	0.227	8.9%	0.947	8.3%
RRT <sub>obst way500 50k</sub>	530.3	15.7%	105.1	15.7%	0.199	9.2%	0.908	16.5%
RRT <sub>obst way500 100k</sub>	508.4	11.4%	94.6	11.0%	0.187	8.5%	0.883	16.2%

Tab. 4.3: Comparison of algorithm performance for the task illustrated in figure 4.10 c).

are significantly worse than for the approaches based on RRTs. In this setup, the somewhat higher topological complexity leads to a clearer advantage for RRT<sub>obst</sub> which produces the lowest values for total obstacleness and interestingly also the maximal obstacleness on the route. Again, the local trees do not yield satisfying results since they produce too long trajectories. Their benefit lies in producing a higher degree of exploration of the algorithm. This effort is counterproductive on simple topologies as the presented example.

Table 4.3 shows the results for the third experiment which considers the same task with the lower allowed maximal lateral inclination angle, the outcome of which is depicted for RPP<sub>obst</sub> in figure 4.10 c). The potential-field approaches fail to find a solution. Conversely, the other algorithms perform similarly to the second experiment. Again, the RRT<sub>obst</sub> approach performs best.

Adding locally rooted random trees to the algorithm was proposed in [Ettlin 06a] to handle complex terrain (and hence obstacleness) topologies. This benefit can be seen in the fourth experiment, which is summarised in table 4.4. Contrary to the first three experiments, significant local obstacleness minima are present which are benefitted from by exploiting the additional trajectories created by the local trees. Once again, the path length increases with obstacleness awareness and the number of random trees included. Additionally, finer control of the allowed exploration regions also increases path length.

Due to the higher complexity of the obstacleness topology, the total obstacleness on a path is reduced by the introduction of local trees. As could be observed in experiments 1 to 3, too many local trees are detrimental to the results. In experiment 4, the optimal setting within the listed examples consists of 100 local trees. Using 500 randomly placed RRTs reduces the mean obstacleness on a path but not the to-

Algorithm	$\mu \tau $	$\sigma \tau $	$\mu\Sigma_o(\mathbf{q})$	$\sigma\Sigma_o(\mathbf{q})$	$\mu\mu_o(\mathbf{q})$	$\sigma\mu_o(\mathbf{q})$	$\mu\lceil o(\mathbf{q}) \rceil$	$\sigma\lceil o(\mathbf{q}) \rceil$
RPP/RPP <sub>obst</sub>	-	-	-	-	-	-	-	-
RRT	356.3	3.54%	266.2	6.93%	0.747	5.05%	0.997	0.0%
RRT <sub>connect</sub>	346.6	4.73%	263.7	4.34%	0.762	5.27%	0.997	0.0%
RRT <sub>obst10k</sub>	370.3	3.1%	179.1	7.7%	0.483	6.5%	0.729	4.7%
RRT <sub>obst50k</sub>	379.6	2.4%	167.7	4.8%	0.442	4.2%	0.625	0.7%
RRT <sub>obst100k</sub>	392.5	3.5%	167.3	4.4%	0.427	4.8%	0.614	0.6%
RRT <sub>obst way100 10k</sub>	456.6	10.1%	159.3	11.5%	0.349	6.9%	0.718	9.9%
RRT <sub>obst way100 50k</sub>	491.1	7.1%	142.6	5.3%	0.291	7.8%	0.624	4.9%
RRT <sub>obst way100 100k</sub>	488.8	7.2%	136.1	5.3%	0.279	6.3%	0.612	1.9%
RRT <sub>obst way500 10k</sub>	554.0	10.5%	194.1	11.4%	0.351	6.8%	0.722	8.9%
RRT <sub>obst way500 50k</sub>	565.8	6.3%	157.2	4.5%	0.279	5.5%	0.618	2.0%
RRT <sub>obst way500 100k</sub>	577.4	7.1%	153.1	3.9%	0.266	5.6%	0.611	0.8%

Tab. 4.4: Comparison of algorithm performance for the task illustrated in figure 4.18.

tal cost - the path displays a disadvantageous tendency to linger in low-obstacleness regions. In terms of the maximal obstacleness on a path, all algorithms using obstacleness achieve similar results with a finer control of the exploration leading to better results.

The experiments listed in this section demonstrate how the inclusion of obstacleness into motion planning algorithms leads to the expected benefits compared to algorithms using a binary obstacle definition in rough-terrain settings. In simple environments RRT<sub>obst</sub>, an obstacleness-aware RRT<sub>connect</sub> algorithm displays the best results. Where a higher complexity of the obstacleness distribution is present creating additional local trees as proposed in RRT<sub>obst way</sub> leads to further improvements. A potential disadvantage of the latter approach is that the number of randomly generated local tree roots is somewhat dependent on the complexity of the obstacleness function. This introduces an additional parameter which needs to be considered when employing the algorithm to a specific problem.

The presented motion planners do not consider any system dynamics. Due to the typically slow operational velocities of planetary rovers such an omission is widespread in the related motion planning literature, e.g. [Dacre-Wright 93], [Haït 99], [Seraji 99] and [Iagnemma 99] among others. Nevertheless it would be strongly desirable to consider dynamic effects at the level of motion plan generation. The extension of obstacleness-based reasoning about the traversed terrain into a kinodynamic planner is arguably one of the most important future extensions of the proposed algorithms.

In particular the RRT approach is ideally suited to include dynamic behaviour of the robot [LaValle 99]. When designing a kinodynamic RRT-based planner, the  $n$ -dimensional configuration space  $\mathcal{C}$  is replaced by the  $2n$ -dimensional state space

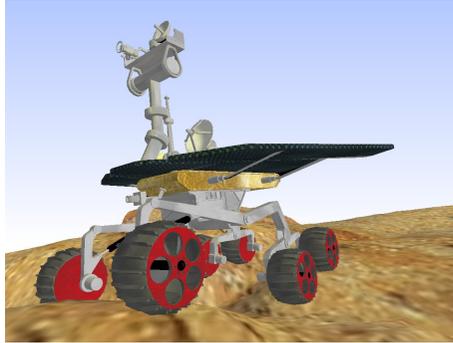


Fig. 4.21: Design of a fictional planetary rover used for rough-terrain motion planning simulations.

$\mathcal{X}$ . The RRT-trees are grown in  $\mathcal{X}$ , each EXTEND operation is based on the set of possible control inputs (and constraints) at the present configuration. To include the obstacleness measure, the navigational difficulty would need to be evaluated accordingly in  $\mathcal{X}$ . The increased complexity of the explored space suggests that in particular the RRT<sub>obst way</sub> algorithm could lead to good results.

#### 4.1.5 Trajectory Following

Trajectory following is not a primary field of interest in the present research. Nevertheless it represents a crucial link between the simulation environment discussed in chapter 3 and the motion planning algorithms presented in this chapter.

As robot model for the simulations, a fictional design similar to current Mars rover models has been adopted. The robot has been developed using the simulation content generation tool-chain presented in section 3.3.5. A picture of the simulated rover is shown in figure 4.21, the collision-shape approximation can be seen in figure 4.26. The robot design is inspired by the rocker-bogie structure for wheel suspension [Bickler 92]. Such a design allows all six wheels to remain in contact with the terrain while navigating over obstacles larger than one wheel diameter. When specifying the collision shapes, the largest degree of simplification has been adopted for the wheels of the rover which are approximated as spheres. In the implemented design, steering is achieved by performing skid-turns, i.e. operating the wheels on one side at a different rate than the wheels on the other side. This technique allows the robot to perform point turns - as required e.g. by the trajectories computed by the planners discussed in sections 4.1.2 and 4.1.3.

The trajectories to follow have been computed without considering system dynamics and are represented as a sequence of discrete configurations to attain. Given

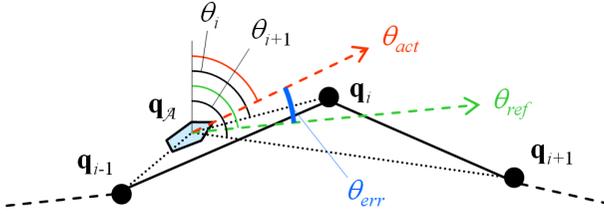


Fig. 4.22: Path smoothing scheme adopted for trajectory following.

such an input, conceptually the ideal behaviour of the robot would be to travel between way-points at a constant velocity, then perform a point turn and continue on the next line segment. Such behaviour cannot be achieved in either the real world or the simulation environment which both contain dynamic effects.

The primary goal of the present work is to illustrate the Ibex simulation environment as support for rough-terrain robot motion planning. To demonstrate a complete motion planning system, a simple path tracking scheme has been implemented. The scheme allows the robot to approximately follow the paths computed by the motion planners in the simulation while using physical parameters resembling those expected in reality for the rover and terrain.

The implemented path tracking algorithm considers exclusively the orientational deviation of the rover as error measure. The orientational error  $\theta_{err}$  is defined as the deviation between the current heading  $\theta_{act}$  and the direction to the desired position  $\theta_{ref}$  following a straight line:  $\theta_{err} = \theta_{ref} - \theta_{act}$ .

To execute the purely kinematic motion plan in a dynamic environment, some degree of path smoothing has been implemented at the level of reference value generation. The reference orientation  $\theta_{ref}$  is computed taking into account the next two way-points on the trajectory instead of just the next. Following weighting scheme is used to determine the desired heading in dependency of the headings  $\theta_i$  and  $\theta_{i+1}$  to the configurations  $\mathbf{q}_i$  and  $\mathbf{q}_{i+1}$  of the next two way-points:

$$\theta_{ref} = (1 - p_i) \cdot \theta_i + p_i \cdot \theta_{i+1} \quad (4.32)$$

where  $p_i$  represents the proportion of distance travelled on the path segment  $\mathbf{q}_{i-1} \rightarrow \mathbf{q}_i$ . Considering that the robot may deviate from the ideal path,  $p_i$  is computed as

$$p_i = \frac{|\mathbf{q}_A - \mathbf{q}_{i-1}|}{|\mathbf{q}_A - \mathbf{q}_{i-1}| + |\mathbf{q}_i - \mathbf{q}_A|} \quad (4.33)$$

The path smoothing scheme is illustrated in figure 4.22. This approach ensures a smooth transition of the desired orientation when passing way-points. Addition-

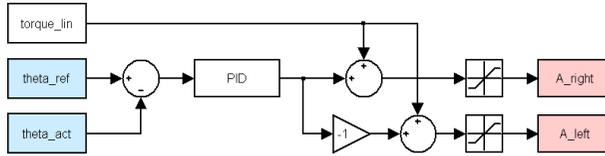


Fig. 4.23: Basic trajectory tracking scheme used with the simulated planetary rover.

ally, some tolerance is introduced such that the robot is assumed to have reached a way-point as soon as it is within a small distance thereof.

The basic implemented path-tracking scheme uses a PID controller to minimise the orientational error of the rover. The output of the controller generates commands for the motors of the robot, in inverted direction for one of the sides to produce the desired skid-turns. Additionally, constant motion commands are generated which induce a (slow) forward motion by themselves. These two types of motion commands are superposed before being sent as input to the actuators. An upper bound is applied to the torque exerted by each motor in the simulation to emulate the dynamic restrictions of the actuators employed. A graphical representation of this control scheme is shown in figure 4.23. Actuators are shaded red, sensors blue, including the reference value which can be considered a value derived from sensor data. In the implementation, when generating the motion commands both actuator inputs are proportionally reduced if one desired torque exceeds the maximal torque of the actuator. This measure ensures the steering behaviour is not distorted by truncating an actuator command above the allowable maximal torque.

A more sophisticated version of the trajectory tracker additionally controls the motion commands which generate the forward motion in dependency of the orientational error. The reasoning behind this measure is that the robot should only progress if it is heading in the right direction. A schematic representation of this

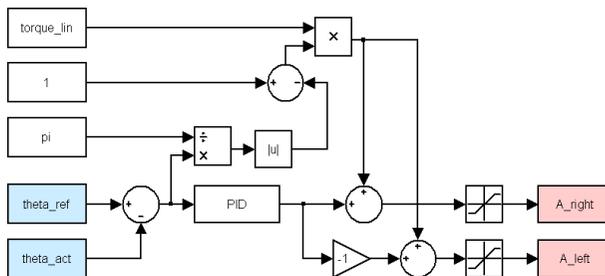


Fig. 4.24: Trajectory tracking scheme used with the simulated planetary rover which reduces the forward velocity in dependence of the orientational error.

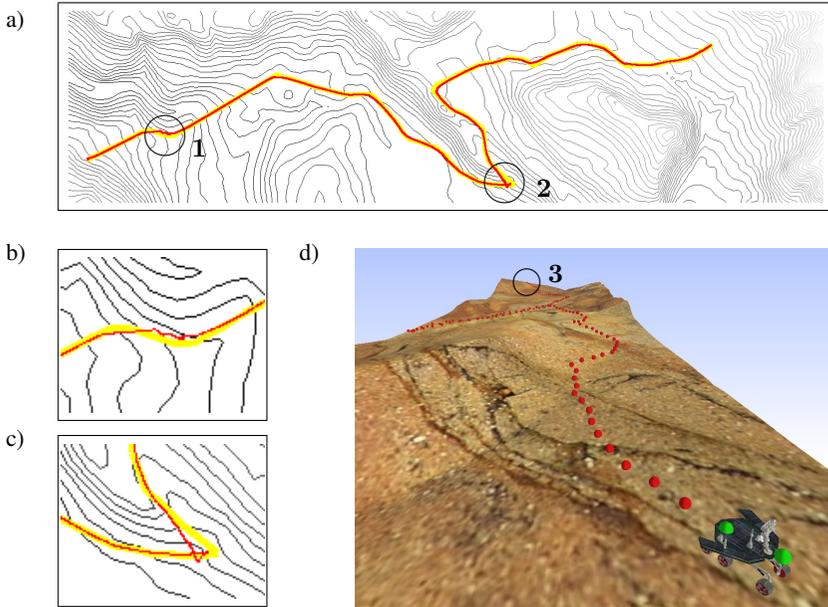


Fig. 4.25: a) Computed path as output of motion planner (yellow) and superposed actual path of the robot (red). b) and c) Detail views of areas labelled 1 and 2. d) The rover reaching the goal configuration after following a trajectory which started near the horizon at the position labelled 3.

controller is shown in figure 4.24.

These path trackers do not exactly reflect the computed motion plan (which would call for perfect point turns at the way-points). Nevertheless, they allow to closely follow of the trajectory in an environment where dynamic effects are explicitly in action. The simple trajectory trackers are presented here to complete an overall motion planning and execution system within Ibox. More sophisticated path tracking algorithms are e.g. presented in [Jiang 97], [Koh 99], [Guo 03] as well as the references therein.

An example of the second controller described above in action is illustrated in figure 4.25. The followed trajectory compared to the original motion plan is depicted in figure 4.25 a). The output of the motion planner is shown as thick yellow line, the position of the robot during the simulation as superposed thin red line. Some path-tracking errors can be discerned: at the position marked 1, the perpendicular slope is not avoided with sufficient clearance leading to some deviation from the path. A detail view is shown in figure 4.25 b). More importantly, the steepest terrain inclination on the path at position 2 leads the rover to slide down-

hill while performing the sharp point-turn, as shown in detail in figure 4.25 c). At that position, the deviation from the path is approximately  $\frac{1}{3}$  of the rover length. At all other locations only insignificant deviations can be observed. Some of the tracking errors are caused by the look-ahead scheme of considering the next two way-points, this effect can also be recognised in both figures 4.25 b) and c).

The phenomenon encountered at location 2 has been observed in a number of experiments: while performing skid-turns, the adhesion of the rover drops sharply as compared to when performing predominantly translational motion. This effect was not considered when computing the original motion plan. On the one hand, this highlights the shortcomings of using a motion planner which does not consider dynamic effects. On the other hand it serves as example to illustrate the usefulness of the Ihex framework to uncover such deficiencies.

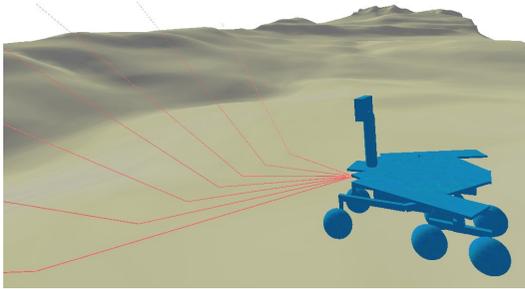
A visualisation of the robot reaching the goal configuration after following the trajectory in figure 4.25 a) is shown in figure 4.25 d). The initial configuration is located near the horizon at the position labelled 3. The individual way-points are rendered as trigger shapes, allowing the size of the robot to be appreciated in relation to the distance between way-points. The radius of the trigger spheres approximately corresponds to the tolerance allowed when reaching a way-point. The reference point on the robot is located near the centre of the vehicle body.

Naturally, in research where the dynamic behaviour of the robot is studied, a more sophisticated control-theoretic approach would need to be implemented for path tracking. In particular, an approach in which an orientational controller and a velocity controller are superposed would increase the robustness of the system. The PID controller used in the example displays difficulties in steep terrain where the robot tends to deviate from the reference trajectory. Using more advanced controllers specifically suited for the system dynamics could not only increase the overall performance but also lead to a more stable and robust system. Examples of more advanced controllers include state-feedback controllers (e.g. Model Predictive Controller, [García 89]) and adaptive controllers, e.g. [Ioannou 96].

## 4.2 *Dynamic Environments*

### 4.2.1 *Scenario Description*

The scenario which is adopted for modelling dynamic environments is often encountered in rough-terrain robot motion planning. It is assumed some prior knowledge about the operation environment is known to the motion planner, in a real scenario e.g. by evaluating remote sensing data of the environment. An initial motion plan is computed based on the available information. During the operation of the robot, sensor data is acquired and compared with the known terrain characteristics. If the divergence lies within some error bound, the original motion



*Fig. 4.26:* Collision shape approximation of planetary rover shown in figure 4.21. The laser range-finding sensor array used for dynamic motion planning is illustrated with specular reflections for the individual sensor beams bouncing off the terrain.

plan is retained and executed. If larger discrepancies are detected, some form of re-planning algorithm needs to be executed.

The original motion plan is computed with the techniques described in section 4.1. This requires a large amount of knowledge, in particular about the physical material properties used to compute the obstacleness of the terrain. Terrain properties can be estimated e.g. by evaluating remote sensing data to derive the physical characteristics. Such techniques are generally afflicted with high estimation errors. Even when operating on the terrain it is an error-prone process to predict the terrain properties in the vicinity of the robot for updating the obstacleness function.

This section illustrates the capabilities of the Ibx simulation framework to create an environment for dynamic sensor-based rough-terrain motion planning. The emphasis therefore does not lie on the employed sensing techniques or methods of terrain characteristics estimation. Much rather, dynamic planning is demonstrated using the laser range-finding sensor introduced in section 3.3.4. More involved approaches could e.g. benefit from analysing the visual appearance of the terrain to derive material properties in the surroundings of the rover. Sensor fusion would then lead to a more comprehensive model of the expected terrain properties.

#### 4.2.2 *Integration of Sensor Information*

The planetary rover depicted in figure 4.21 has been equipped with an array of range-finding devices to acquire information about the local terrain topography. Figure 4.26 shows the collision-shape approximation of the fictional planetary rover used for the experiments together with the mounted sensor array. Seven laser range-finding sensors are positioned at the nose of the vehicle covering an arc of  $90^\circ$  with a depression angle of  $10^\circ$  relative to the robot chassis. The specular reflections of the sensor beams are included exclusively for illustration purposes - sensor data is

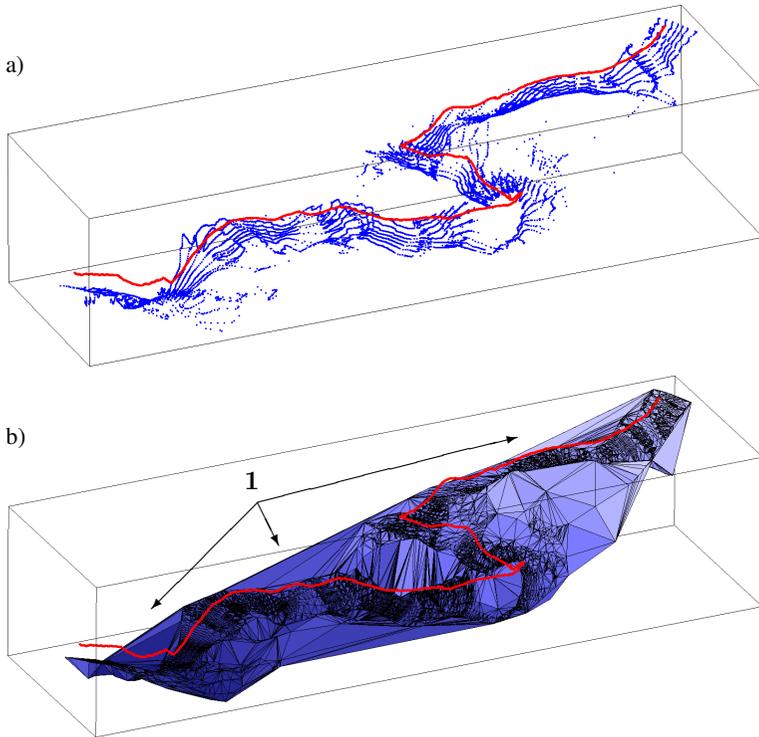


Fig. 4.27: a) Sensor values gathered by the rover while following the trajectory. b) Terrain reconstruction using a Delaunay triangulation. Height scaled by a factor of 5.

gathered from the first point of impact, cp. section 3.3.4.

During the operation of the rover, the sensors gather distance data which can be transformed into world-frame data points of the terrain given the position and orientation of the robot. Figure 4.27 a) shows the trajectory followed by the sensor array mounted on the robot as well as the sensor points gathered. The height axis is scaled by a factor of 5. The path of the sensor differs somewhat from the one traced by the robot itself since the sensor emitter is mounted approximately half a vehicle length ahead and slightly below the reference point of the robot. The outputs of each individual sensor device can be distinguished in the sections of the trajectory where the rover follows approximately a straight line.

Based on the sensor data gathered, it is possible to perform a reconstruction of the sampled terrain. A Delaunay triangulation [Delaunay 34], [Rippa 90], [Kallmann 03b] (and references therein) of the data points is shown together with the sensor trajectory in figure 4.27 b). Where a high density of data points is avail-

able, the reconstruction corresponds well with the original terrain model. On the fringes of the reconstruction some artifacts can be observed, e.g. positions marked **1**. The artifacts are caused by the employed Delaunay algorithm which triangulates the projections of all data points onto the  $xy$ -plane within a convex hull. Delaunay triangulation is only one possibility of generating a polygonal model from data points, for a comparison and survey see [Garland 95].

Given sensor data of the terrain, it is possible to detect geometric discrepancies with the model used for generating the original motion plan. This is illustrated in figure 4.28 where the trajectory of the sensor emitter as well as the gathered sensor data is shown together with original terrain information. Red points denote sensor readings, blue points the height of the original terrain at the equivalent location. The employed superposition leads to sensor readings being covered by original data points where both values are identical. Additionally, the terrain polygons which are “hit” by a sensor beam are visualised in green. The height of the terrain is scaled by a factor of 5.

In figure 4.28 a), the navigated terrain is identical with the one used for motion planning. The superposition produces almost exclusively visual readings for the original terrain heights, i.e. correct values are sensed. Some few rounding errors can be recognised in the valley labelled **1**.

Some divergencies have been introduced for the experiment shown in figures 4.28 b) and c). The terrain geometry has been modified as shown in the encircled areas **2**, **3** and **4** of figure 4.28 b). The trajectory of the rover passes to the left of the three modifications as seen from the perspective shown in the figure. The sensors pick up the elevation differences as visualised by the presence of numerous red points in figure 4.28 c). The sensed height differences are highlighted by vertical lines. It can be seen how the obstacles have shielded some areas from being scanned by the sensors, e.g. in the areas labelled **5** and **6**. The jagged edge of obstacle **4** can be discerned in the plot at the position marked **7**.

In the context of obstacleness-based motion planning, the ability to detect geometric discrepancies between the expected and sensed terrain can be used to update topographic obstacleness components such as those dealing with inclination. To update the obstacleness components associated with the physical material properties, other sensing techniques need to be applied. For a real-world legged robot, a method is e.g. presented in [Caurin 94] to analyse the terrain properties directly with one leg of the robot. Such an approach is not directly applicable in the simulation due to the non-deformable geometry of the terrain, but could be approximated by returning the terrain properties where “touched” by the sensing leg. Synthetic vision could also be applied to approximately estimate terrain characteristics based on the visual aspect of the terrain.

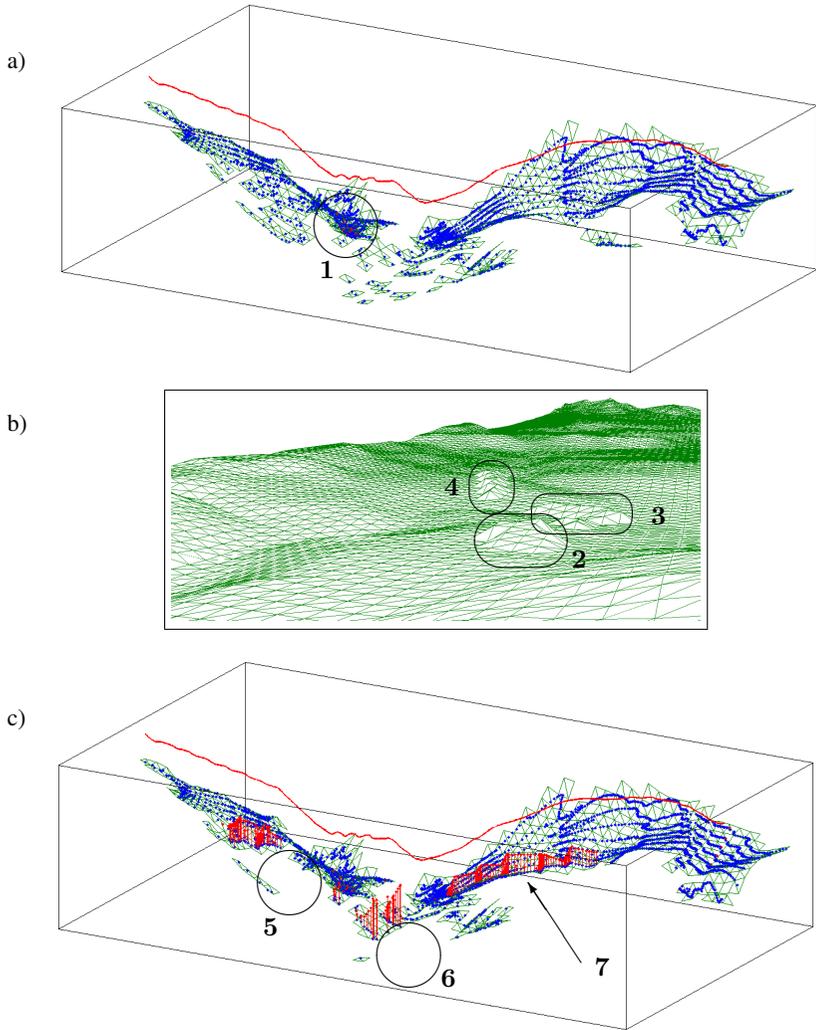


Fig. 4.28: a) Sensor readings for a terrain which corresponds to the expected model. b) Topographical differences introduced into the terrain and c) sensor readings highlighting the detection of the divergencies.

### 4.2.3 Dynamic Re-Planning

Independently of the model used to detect and represent deviations of the sensed terrain from the original description, situations occur where the motion plan needs

to be modified. Depending on the nature of the detected deviations, local re-planning algorithms can be applied or the original (global) motion planner must be invoked. In this section only the first case is considered, since global re-planning using new terrain data does not differ algorithmically from the procedures described earlier in this chapter.

A noteworthy exception are the potential-field based algorithms which are intrinsically local planners. A simple sensor-based re-planning approach can be realised by modifying the repulsive potential of e.g. the  $RPP_{\text{obst}}$  planner to include the sensed divergencies from the originally assumed terrain. In fact, artificial potential-field algorithms were originally developed to perform on-line obstacle avoidance rather than global motion planning tasks [Khatib 86], [Latombe 91]. The avoidance of moving obstacles in a 2D environment is shown in [Feder 97]. In [Brock 99], a method is presented which continually deforms a planned trajectory as information about obstacles evolves while following the trajectory.

To demonstrate the applicability of Ibex to sensor-based dynamic re-planning of a motion plan, a scenario is assumed where the terrain model itself remains unchanged from the original but additional obstacles are introduced. The obstacles are inserted into the scene as static rigid bodies. The easiest way of adding such obstacles is to work with the Simulink integration of Ibex and use the Ibex blocks which create rigid bodies in the simulation directly from the Simulink GUI.

In the adopted scenario, detected additional objects are considered to represent a high risk and are hence treated as  $\mathcal{C}_{[0=1]}$ , i.e. “binary” obstacles. Such an approach simplifies the inclusion of sensor data into the re-planning algorithms by eliminating the need for complex obstacle identification and characterisation routines. The simplification is legitimate at this point since approaches required to perform a more subtle distinction of obstacle presence have been illustrated above.

The implemented dynamic re-planning algorithm computes a bounding circle around the projection of the detected obstacle coordinates onto the  $xy$ -plane and adds a safety margin which defines an area to be avoided. All way-points of the original trajectory which lie within this forbidden area are cancelled and new way-points along the perimeter of the circle added instead. The new way-points are computed along the shorter arc connecting the points where the original path intersects the circle. As new obstacle points are detected, a lazy approach is applied to updating the position and radius of the designated danger area. The representation of the obstacle is only updated when the error between the current obstacle representation and the sum of sensed data points surpasses some threshold.

When all sensors break contact with the obstacle, a similar approach is used to decay the validity of known obstacle measurements. This technique allows the system to treat separate obstacles as new hazards if they are detected sufficient time after contact is broken with earlier obstacles. Obviously, this method only performs well in environments where obstacles are scarce.

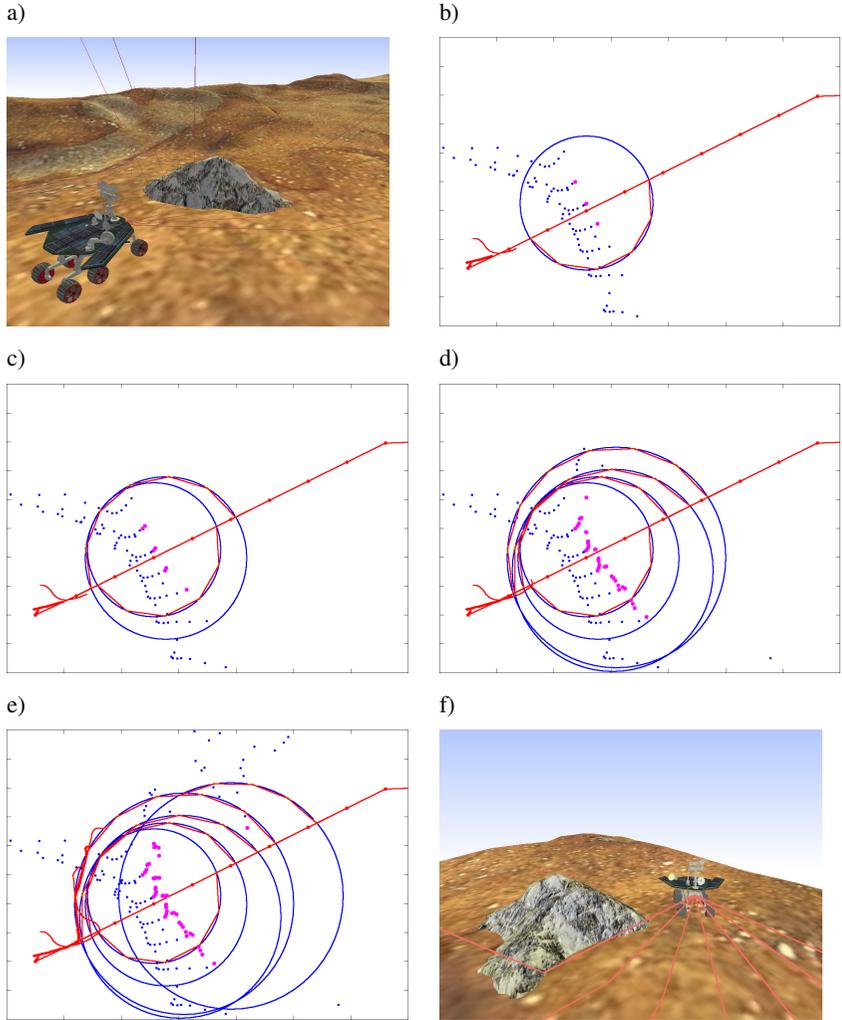
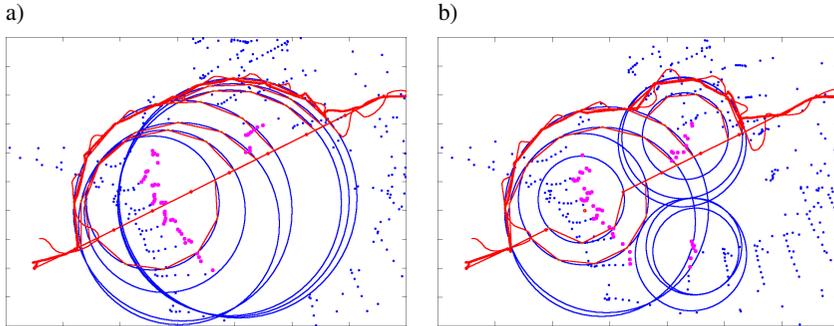


Fig. 4.29: The planetary rover executing a dynamic re-planning algorithm on detecting an unexpected obstacle while navigating on rough terrain.

Figures 4.29 and 4.30 illustrate the dynamic re-planning algorithm in action. The initial situation with the rover approaching an uncharted obstacle is depicted in figure 4.29 a). Three of the sensor beams have just picked up the unexpected obstacle - this can be recognised by their specular reflections being cast steeply upwards.



*Fig. 4.30:* a) The final trajectory in the situation illustrated in figure 4.29. b) Alternate approach in which the decay of obstacle validity is faster compared to the algorithm shown in figure 4.29. This leads to far parts of the same obstacle being represented as separate danger areas.

A representation of the data available to the robot at that point is shown in figure 4.29 b). The three sensor readings (marked by purple points in the plot) at the centre of the circle denoting the dangerous area can be recognised directly in the original path plotted in red. The thick red line marks the position of the rover itself, the thin line represents the position of the distance sensor. The overall length of the rover is slightly larger than the distance between two original way-points; cp. also figure 4.25 d). The modified path based on the sensor data can be seen following way-points on the periphery of the computed bounding circle.

In figure 4.29 c), new sensor readings become available near the known locations and an additional point is detected to the South-East. As a consequence, the location and size of the danger zone are updated. Interestingly, now the shorter arc leads the rover to circumnavigate the obstacle to the North instead of the Southerly route computed after the first detections.

The situation has progressed several detection steps further in figure 4.29 d). The radius has been further increased to reflect the new estimated size of the obstacle. The robot has reached the point where the deviation from the original route commences.

Following the newly computed route, the rover breaks sensor contact with the obstacle by heading North. In figure 4.29 e), rounding the second way-point of the detour leads to a sensor reading being picked up from the protruding extension of the obstacle. The same point in time is depicted in an Ixex observer as shown in figure 4.29 f). This image also gives an idea of the clearance adopted to navigate around obstacles.

The final trajectory followed by the rover around the obstacle is plotted in figure 4.30 a). When re-joining the original route, a relatively large path tracking error

(a deviation of approximately  $\frac{1}{4}$  of the vehicle length) can be recognised. This deviation is caused by the inclination of the terrain which can be recognised on the right margin of figure 4.29 f). The situation is aggravated by the rover having to perform a sharp point-turn during which the simple path-tracking scheme employed cannot satisfyingly control the position of the rover. The influence of the look-ahead scheme employed (cp. section 4.1.5) can also be recognised.

Figure 4.30 b) shows the effects of reducing the amount of time before eliminating detected obstacle data points from memory. Discarding known obstacle locations does not influence the path which has already been altered by those obstacles. This procedure serves to reduce the granularity of obstacles which are not detected concurrently. To handle multiple obstacle detections within a short time span or even simultaneously, more powerful clustering techniques need to be applied, such as e.g. k-means clustering [MacQueen 67].

A powerful extension of the proposed scenario which is easy to implement using Ibex is to simulate moving obstacles such as other vehicles. For simplicity of control, they can be made to follow a predefined trajectory in a purely kinematic manner. It is also conceivable to implement reactive external objects which behave in dependency of the actions taken by the primary robot.

A more active approach to replanning would consist of interrupting the progress of the rover when an obstacle is detected and analysing the situation. This could e.g. be done by sensing the terrain in the surroundings of the obstacle using a sensor which can be pointed at specific locations. Such a procedure corresponds to the scanning of the statue illustrated in figure 3.23. Knowing the precise geometry of an obstacle before the re-planning stage can provide crucial information to design better trajectories.

### 4.3 Results

In this chapter various robot motion planning algorithms have been presented with a focus on rough-terrain planning. Three new algorithms have been introduced which are adaptations of known approaches to rough-terrain planning. This allows to benefit of the known advantages of these approaches for rough-terrain navigation. As basis for the proposed extensions, a generic measure for the degree of presence of an obstacle called “obstacleness” has been defined [Ettlin 06b], [Ettlin 05a]. Such a definition is required to characterise the navigational difficulty of the terrain which is inherently incompatible with classical binary obstacle definitions.

The first proposed algorithm, which extends Randomised Potential Field Planners (RPP) to include a continuous terrain-induced obstacle measure is called RPP<sub>obst</sub> [Ettlin 05a]. The measure for navigational difficulty of the terrain is used directly as basis for defining the repulsive potential in the algorithm. While still displaying the typical greedy behaviour of potential-field based planners this allows

$RPP_{\text{obst}}$  to compute routes biased towards areas of easy navigation.

RPP motion planners are intrinsically local approaches which iteratively consider exclusively the immediate surroundings of the current configuration when computing the motion plan. Local approaches are advantageous in situations where little or imprecise information about the environment is known. Such situations are common in reality and e.g. occur typically in sensor-based motion planning. Potential-field based planners are also ideally suited to perform on-line obstacle avoidance tasks. The involved computations are relatively inexpensive which is in itself an advantage and also allows to e.g. consider fast moving obstacles. On the other hand, local planners increasingly display difficulties finding a global trajectory when the complexity of the environment is increased. In the case of potential-field planners the main problem consists of the algorithm getting trapped in local potential minima; this effect is illustrated in an example as motivation for the need of more sophisticated planners.

The other proposed motion planners are based on the Rapidly Exploring Random Trees (RRT) approach. An RRT is a tree data structure which displays highly favourable exploration capabilities in the configuration space. The second proposed algorithm extension called  $RRT_{\text{obst}}$  is based on the bi-directional  $RRT_{\text{connect}}$  heuristic [Ettlin 06b]. The proposed algorithm biases the exploration of the RRTs towards areas of easy navigation. The algorithm yields good results in simple obstacleness topologies which contain at most one local maxima (i.e. typically an obstacleness saddle point) on the route between initial and goal configuration. In more complex topologies the performance degenerates towards the uninformed  $RRT_{\text{connect}}$ .

The third suggested algorithm,  $RRT_{\text{obst way}}$ , is designed to benefit from the desirable local behaviour of  $RRT_{\text{obst}}$  while being applicable to arbitrarily complex obstacleness topologies [Ettlin 06a]. This is achieved by maintaining a dynamic list of additional local trees in the configuration space. The local trees are grown using the  $RRT_{\text{obst}}$  heuristic. This approach ensures all local obstacleness minima are explored and consequently benefitted from when combining the local trees to a global solution. One issue which remains to be investigated in more detail is how to select the optimal number of local trees. Examples have shown that a high number of local trees grown in simple obstacleness topologies are detrimental to the path quality. In complex topologies the approach produces the best results of the studied algorithms. This is of particular interest when extending the presented approaches to include system dynamics since kinodynamic motion planners typically operate in more complex, higher-dimensional spaces.

To link the motion planning algorithms to the Ibx simulation environment for testing, a simple path following algorithm has been implemented. While not fulfilling any optimality criteria, the approach is adequate to allow a simulated fictional planetary rover to approximately follow the computed trajectory.

The integration of simulated sensor data into the motion planning algorithms has been demonstrated with an example. An array of simulated range-finding sensors has been mounted on the simulated planetary rover and used to gather terrain data while following a pre-computed trajectory. Examples have been given of terrain reconstruction by interpolation of the acquired data as well as the detection and quantification of discrepancies between the originally assumed terrain model and the sensed terrain geometry.

A sensor-based dynamic re-planning algorithm has been presented which performs basic reasoning on obstacle characteristics using the acquired sensor data. Based on this obstacle representation which is continually updated as more data becomes available, the original trajectory is deformed to allow the rover to circumnavigate the obstacle safely.

The path-tracking and dynamic re-planning algorithms presented are basic implementations intended to demonstrate the capabilities of the Ibex framework rather than represent optimal solutions in their domain. Overall this chapter serves to illustrate the possibilities offered by the Ibex simulation framework to support the development of sophisticated robot motion planning algorithms. The simulation framework can be used to help develop kinodynamic motion planning algorithms in complex environments. Additionally, on-line sensor data can be accessed to implement reactive behaviour of the robot. A wide range of possibilities exists to include additional obstacles into the scenario. These obstacles can be either static, follow kinematic trajectories or implement their own dynamic behaviour. The same techniques presented here can also be used to develop cooperative motion planning algorithms with multiple robots operating in the same simulated scene.



## 5. CONCLUSION

In this work, a concept of a complete simulation environment for rigid body dynamics and motion planning is developed and worked out in detail. This concept has been realised and demonstrated as the so-called Ibex simulation environment.

A general introduction to robot motion planning is given in chapter 2. After an overview of the historic development in the field, the role of a motion planning module within a complete robotic system is highlighted. These introductory thoughts serve to illustrate the context in which present-day motion planners operate. Based thereon the motivation for supporting motion planning algorithm development with a simulation environment is established.

Chapter 3 deals with various aspects of the simulation environment designed to support the development of motion planning algorithms. First, a list of requirements is established for the simulation framework based on the findings of chapter 2. After a brief review of related simulation environments, the developed concept is elaborated.

The framework consists of a simulation component and various additional modules which integrate the simulation with third-party engineering tools. The simulation component makes use of rigid body dynamics simulation algorithms which are included in the form of the Ageia™ PhysX™ libraries.

The simulation component itself is not only available as software library which can be linked into stand-alone programs but also in the form of an extension for MathWorks Simulink®. In the latter configuration the rigid body dynamics simulation fits seamlessly into the host user interface making its use straightforward for users familiar with Simulink.

The proposed simulation environment can be used to support motion planning with non-holonomic constraints and kinodynamic motion planning. This is achieved directly through the inclusion of rigid body dynamics algorithms which encompass collision detection and collision response routines as well as a number of possible motion constraints between bodies, as described in section 3.3.2. The simulation kernel is deterministic, thus fulfilling a further requirement. This determinism is not compromised when co-simulating a rigid body dynamics setup with a Simulink model.

To support sensor-based motion planning it has been shown how various types

of sensors can be simulated to gather information about the robot and its environment. The range reaches from basic sensor simulations based on the spatial variables of some rigid body to the physics-based model of sensors based on wave propagation which is developed in section 3.3.4.

For simulations supporting motion planning algorithm development it is important to be able to simulate the complete control loop found in mechatronics. Apart from the sensors, the control loop includes actuators which influence the robot and its environment as specified by the control logic. The most convenient way of simulating actuators with the proposed solution is to design a dynamic model of the actuator in Simulink and apply the resulting forces to the mechanical structure co-simulated in the proposed simulation environment.

The simulation has proven to operate at velocities which make it practicable to include multiple robots. Generally speaking the hardware requirements for the simulation consist of a PC capable of running the latest operating systems. Naturally, better simulation rates can be achieved with a high-end PC equipped with a fast CPU, plentiful RAM and a powerful graphics card. Even such a system is nowadays considered a consumer product and costs significantly less than typical dedicated simulation hardware. One interesting possibility to enhance the simulation speed is opened by the newly available hardware-accelerated physics simulations: a dedicated extension card is added to a PC which takes over the physics computations at a rate significantly outpacing conventional simulations running on the CPU. Ageia was the first company to produce such a so-called physics processing unit (PPU). To benefit from the hardware-accelerated simulations, the proposed solution would simply need to be updated to include a version of the PhysX libraries which supports the PPU.

To enable an on-line inspection of the simulation progress, various graphics engines have been included in the simulation module, notably one based on the Nebula 2 engine which is also available in the Simulink integration. The visualisations produced using this graphics engine can be appreciated throughout this document. Interactions with the simulation are performed in a physically consistent manner by applying forces and torques to rigid objects or joints.

Full access to simulation data is given through the Ibex API to application programmers when using Ibex as library. The physical variables of simulated entities are accessible as signals on the Simulink user interface. Signals are the standard data flow representation in Simulink, where they can be processed and analysed using the wealth of functionality provided. The MATLAB/Simulink environment also includes a series of import/export routines which allow to exchange data with other environments.

The control logic of the robot can be conveniently developed by either directly using the Simulink graphical programming language or creating “blocks” therein which encompass user program code. In either case, the information made avail-

able to the motion planner can be precisely controlled through the Simulink signal routing functions. User-defined blocks can be implemented using a number of programming languages. For fast prototype development, the native MATLAB M scripting language is well suited. On the other end of the spectrum, C/C++ is supported which allows to include only slightly modified versions of code developed for the robot hardware using these languages. Fortran, Ada and Java are also supported. An interesting option is offered by the Real-Time Workshop<sup>®</sup> extension of Simulink which allows to generate source code for target hardware from a Simulink model. This feature allows to test precisely the algorithms which will be deployed on the real robot without any error-prone intermediate manual steps.

To support the development of simulation content a versatile tool-chain has been established for the proposed simulation framework, cp. section 3.3.5. Again following the principle of benefitting from existing solutions in a domain, two 3D design tools have been extended: Autodesk<sup>®</sup> Maya<sup>®</sup> and SolidWorks<sup>®</sup>. The two programs satisfy the needs of different user groups. Maya is in widespread use in the 3D modelling community and animation industry. SolidWorks on the other hand is a comprehensive CAD package which offers high-precision constraint-based design methods and tools required when engineering a mechanical structure.

The application of the simulation environment to the development of robot motion planning algorithms is demonstrated in chapter 4 where three new motion planning algorithms are proposed. The examples shown are drawn from mobile robotics and in particular from rough-terrain motion planning.

In rough-terrain planning the interaction between the robot and the terrain is of fundamental importance. To characterise the navigational difficulty associated with traversing some specific part of the terrain, a generic measure named the “degree of obstacleness” is introduced in section 4.1.1. Using such a generic measure allows to decouple the formulation of rough-terrain planners from any concrete application example. When applying such a general motion planner to a specific scenario, a concrete obstacleness function is defined which formulates how the navigational difficulty is computed for the task at hand. Guiding the computation of motion plans by means of obstacleness allows to precisely define how various aspects of navigational difficulty should be considered when computing the trajectory.

The first new motion planner proposed is based on the Randomised Potential Field (RPP) technique and is called  $RPP_{\text{obst}}$ . The navigational difficulty of the terrain is employed to define the repulsive potential used in the approach. This allows the planner to divert the computed path to easily navigable terrain while in general following the locally greedy approach of potential-field planners.  $RPP_{\text{obst}}$  is particularly well suited for simple terrain topologies with simple associated obstacleness functions. When the obstacleness distribution becomes more complicated RPP planners display a tendency to get trapped in local minima of the computed potential due to the locality of the approach. For the same reason, RPP planners are

not well suited for tasks which involve higher-dimensional configuration spaces.

To overcome the locality of RPP planners, the other two proposed rough-terrain motion planning algorithms are based on Rapidly-Exploring Random Trees (RRTs). RRTs have been shown to display good global exploration capabilities, also in high-dimensional configuration spaces. A further benefit is that RRTs are designed to easily accommodate non-holonomic constraints and perform kinodynamic motion planning.

$RRT_{\text{obst}}$ , the first novel algorithm proposed based on RRTs extends the bi-directional  $RRT_{\text{connect}}$  heuristic. The algorithm greedily grows two trees rooted at the initial and goal configurations towards each other while guiding their expansion to easily navigable terrain.  $RRT_{\text{obst}}$  produces good results in terrains with at most one local obstacleness maxima along the computed trajectory. In more complex obstacleness topologies, the algorithm fails to benefit from easy terrain in parts of the motion plan which lie between such local maxima.

In order to maximally benefit from local obstacleness minima along the route, the second proposed RRT algorithm,  $RRT_{\text{obst way}}$  grows additional local RRTs at random locations in the configuration space. The local trees as well as the two original trees at the initial and goal configurations are grown according to the  $RRT_{\text{obst}}$  heuristic. All these trees which locally compute trajectories in easily navigable regions are successively merged to a global solution which consequently also follows easily traversable terrain.  $RRT_{\text{obst way}}$  computes the best trajectories of the compared algorithms on complex terrains. On the other hand it has been shown that including excessive numbers of local trees in simple terrains leads to a deterioration of path quality. The reduction in quality is caused by detrimental meandering of the trajectory (i.e. “too much exploration”). It remains to be investigated how the number of local trees should be optimally chosen in dependency of the terrain complexity.

To test the described algorithms in the simulation environment, fictional planetary terrains have been used with a geometry consisting of well over 10'000 mesh polygons. Each polygon is assigned its own set of physical material properties. Based on the terrain topography and the distribution of physical properties, a concrete obstacleness measure is defined for the simulated fictional model of planetary rover. The motion planners described above are used to compute an original motion plan. A simple trajectory tracking scheme has been implemented which allows the robot to approximately follow the trajectory.

An example has been presented in which the motion planner makes use of an array of simulated distance sensors to scan the terrain ahead of the robot. It is shown how deviations from the originally assumed terrain can be detected using the simulated sensors. A dynamic re-planning algorithm has been demonstrated in the simulation environment which allows the motion planner to incrementally derive a model of the detected obstacle and consequently devise a route to by-pass it.

These examples illustrate the type of experiments which are possible to perform with the Ibex environment to support robot motion planning. System dynamics, non-holonomic constraints and dynamic re-planning based on sensor data are covered by the examples. What has not been explicitly shown is the use of multiple robots. Importantly, for the simulation environment the inclusion of a further robot does not make any conceptual difference. There is no limitation which would hinder the inclusion of dozens of robots other than the reduction of computational velocity.

The application of Ibex to motion planning has been demonstrated in a rough-terrain setting. Nevertheless it can be appreciated from the examples how the simulation framework is of broader applicability and can serve to help develop a whole range of state-of-the-art motion planning algorithms. The simulation framework and in particular its integration in Simulink can also be applied to a wide range of tasks outwith motion planning. Some hint of the possible scope is given by the application examples shown throughout this document.



## 6. OUTLOOK

The work described in this document includes aspects from numerous different research fields. In particular the developed simulation environment is designed to operate in the intrinsically interdisciplinary field of mechatronics. The same holds true for the presented simulation of an overall motion planning system embedded in a robot. When describing such complex systems, a comprehensive list of possible extensions can typically be drawn up. In this final chapter of this thesis some possible directions of future research are presented.

An obvious extension of the proposed simulation environment is the integration of rigid body dynamics libraries which support hardware-accelerated physics simulations (such as the latest versions of Ageia<sup>TM</sup> PhysX<sup>TM</sup>). Using a hardware-accelerated simulation kernel opens up a whole series of new possibilities.

On the one hand the complexity of scenes which can be run at an acceptable simulation rate is significantly increased. For motion planning this means more complex robots and environments can be simulated. In particular, multi-robot motion planning would benefit from the increase in velocity. The increase in feasible simulation complexity would also allow to study more advanced robot-terrain interactions. Individual loose boulders can already be simulated with the present implementation. Taking this idea one step further, hardware-accelerated rigid body dynamics would allow to simulate extremely high numbers of small, simple objects in order to create gravel-like terrain behaviour.

On the other hand, numerous new features are being added to what increasingly are not only rigid body dynamics libraries. These extensions are often computationally expensive and only feasible at larger scales using the computational power offered by hardware acceleration. Some of the additional features can prove to be of interest for simulating new classes of motion planning environment as described in the following.

Deformable object simulations may be applicable to motion planning for non-rigid geometries - either in the robot or the environment. One important application of deformable objects in the environment arises in medical simulations where human tissue might be simulated using such techniques.

Fluid simulations based on particle systems might be used to simulate liquid obstacles (e.g. puddles, river-crossings etc.). Potentially, amphibious robots could

be simulated using the same technique.

Fracturable objects might be included to reflect obstacles which are damaged if the robot does collide with them. After such a collision, fragments of the original obstacle would be present in the scene and would need to be considered in future motion plans.

Importantly it needs to be investigated in each case if the simulation model of a particular feature matches the requirements for the motion planning task at hand. Some of the new features being included in physics engines are focussed on achieving visual effects rather than representing a faithful physical model. Nevertheless, such visual effects can also be of interest to integrate, especially when developing synthetic vision sensors. In the mentioned example of medical simulations, it might be of interest to visually represent body fluids which disturb the visibility onto the underlying tissue. Smoke and haze are other typical applications of particle systems. In mobile robot motion planning, they could be employed to reduce the visibility of some point of interest. A crucial aspect when studying the inclusion of such novel features into motion planners is that using a simulation full control over the studied effect can be exerted.

An extension of the present functionality which can be achieved with the current simulation kernel is the development of further simulated sensor types. Doing so would broaden the scope of motion planning tasks which can be studied. A whole range of real-world sensors can be approximated by adapting the wave-propagation sensor model described in this document. Completely new types of simulated sensor include synthetic vision which would allow to support the development of computer-vision based motion planning algorithms. In this context, the possibilities offered by simulated synthetic stereo vision would also be of interest to explore.

In section 3.3.4 a series of additional robotic sensors are listed which can be implicitly used in the proposed simulation environment. Possibilities include localisation sensors, attitude sensors, odometry sensors and contact sensors. Using various concepts of rigid body dynamics, perfect forms of these sensors can be used at present. As has been done for the ray-casting range-finding sensor, a physical model could be added to these implicit sensor simulations. Encapsulating the resulting sensor abstractions and making them accessible explicitly to the user would result in a library covering a variety of sensor types. Such a library would significantly increase the scope of the simulation environment.

Improvements can also be made to the simulation content generation tool-chain. It would be desirable to have a more comprehensive integration of sensor and actuator specifications in the interface used to develop the mechanical model. Especially if a sensor library is established as proposed above, a possibility should be provided in the modelling tools to include and configure sensors in the design. The same holds true for the definition of actuators in the mechanical model. The goal

of such a development would be to provide a single user interface which allows to design all components contained in a simulation. Such a single solution would significantly increase the comfort of developing simulation content.

At the same time, the flexibility offered at present of co-simulating arbitrary dynamic models with the rigid bodies from within the Simulink user interface must not be compromised. A solution should be found which offers both the comfort of predefined and easily configurable components available within a single user interface as well as the power to freely specify the behaviour of the components when desired.

For future tool-chain developments, emerging file format standards should be taken into consideration to describe simulation scenes. A promising approach for rigid body dynamics setups is the open-source Collada standard. It needs to be studied how such standards are applicable to the needs of robotics simulations including sensors and actuators.

In terms of the presented motion planning algorithms, arguably the highest-priority extension is to include dynamic effects at the planning level. The resulting algorithms can then be used to perform kinodynamic all-terrain motion planning. Additionally, more general non-holonomic constraints induced by the steering mechanism should be considered. The RRT family of algorithms is ideally suited to perform kinodynamic planning with non-holonomic constraints. In such cases, the  $n$ -dimensional kinematic configuration space  $\mathcal{C}$  is replaced by a  $2n$ -dimensional state space  $\mathcal{X}$ . The RRT-trees are grown in  $\mathcal{X}$ , each step of tree growth is computed based on the set of possible control inputs (and constraints) at the present configuration.

In kinodynamic motion planning, the obstacleness function needs to be defined accordingly in state space. Depending on the dynamic behaviour of the system, the influence of terrain characteristics on the navigational difficulty associated with each configuration needs to be evaluated. Thereafter, the kinodynamic extensions of  $\text{RRT}_{\text{obst}}$  and  $\text{RRT}_{\text{obst way}}$  can be applied to finding low-obstacleness trajectories in state space. The increased complexity of the explored space suggests that the  $\text{RRT}_{\text{obst way}}$  algorithm which benefits from local obstacleness minima could lead to good results.

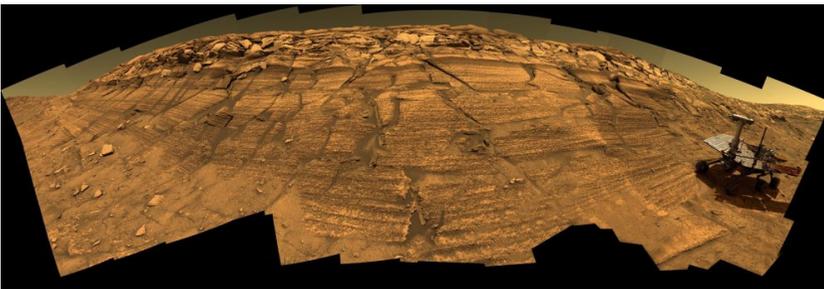
The sensor-based dynamic re-planning example shown in section 4.2 demonstrates how such algorithms can be developed with the help of the proposed simulation framework. The obstacle model departs from a binary obstacle definition while operating within a continuous obstacle environment. An alternative which should be explored is to re-evaluate the obstacleness of the terrain when deviations from the original terrain are reported by the sensors. Such a re-evaluation can easily be performed for the terrain topography based on the principles shown in section 4.2.2. Depending on the scenario assumptions, a re-evaluation of the physical ter-

rain properties can be harder to achieve.

Given sensor-based obstacleness information of the terrain allows to design a re-planning scheme which operates on the same form of data used to compute the original motion plan. Such an approach is a logical further development of the presented motion planning algorithms. It would allow to not only sense obstacles but also to detect areas which allow easier navigation than following the original motion plan.

In this document it has been shown what kind of simulations are possible with the present implementation of the proposed simulator. In this chapter some possible extensions have been illustrated which give an idea of what can be achieved in the near future. All in all, a simulation framework has been presented which can simulate scenarios to support motion planning in a wide variety of domains of active research interest.

The application examples in this document have been drawn mainly from rough-terrain motion planning. Impressive advances have been made in autonomous rough-terrain navigation as illustrated by the success of current Mars rover missions. Still being far from true autonomy, a great deal still remains to be researched in the field. The long journey ahead is sure to be riddled with numerous successes as well as failures. It can be hoped that simulations and algorithms like the ones proposed in this document can help prevent some of the worst pitfalls.



*Fig. 6.1:* The long journey ahead... synthetic view of NASA's "Opportunity" Mars Exploration Rover created from 46 individual images taken by Opportunity in the course of 8 martian days and a photorealistic model of the rover. Image Credit: NASA/JPL-Caltech/Cornell.

# APPENDIX



## A. DEFINITIONS

### A.1 List of Symbols

$a$	: Acceleration (e.g. of a rigid body)
$\mathcal{A}$	: The robot
$\mathcal{B}$	: Binary workspace obstacle
$c_r$	: Coefficient of restitution
$c_{RT}$	: Real-time coefficient
$\mathcal{C}$	: Configuration space of a robot
$\mathcal{CB}$	: Binary configuration space obstacle
$\delta$	: Artificial noise function
$\epsilon$	: Error / accuracy term
$\varepsilon$	: Approximation step size in RRT algorithms
$\eta$	: Scaling factor of repulsive potential in potential-field planning
$f$	: Force, emphasising locality at a single point of an object
$F$	: Force (e.g. applied to a rigid body)
$\mathcal{F}$	: Frame / coordinate system
$G$	: A graph
$\mathcal{GC}$	: Discretised configuration space of a robot
$I$	: Inertia tensor of a rigid body
	: Light intensity
$J$	: Impulse (e.g. applied to a rigid body)
$K_\psi$	: Obstacleness component corresponding to phenomenon $\psi \in \Psi$
$L$	: Angular momentum (e.g. of a rigid body)
$m$	: Mass (e.g. of a rigid body)
$M$	: Mass, emphasised as sum/integral over a rigid body
$\mu_k$	: Kinetic friction coefficient
$\mu_s$	: Static friction coefficient
$N$	: Dimensionality of the configuration space of a robot
$o$	: Degree of obstacleness, cp. section 4.1.1
$\omega$	: Angular velocity (e.g. of a rigid body)
$p$	: Contact point between two rigid bodies
$P$	: Linear momentum (e.g. of a rigid body)

$\Phi$	: Direction of inclination of terrain represented as single angle
$\Psi$	: Set of relevant phenomena for motion planning task
$q$	: Orientation explicitly represented as quaternion
$\mathbf{q}$	: Configuration of a robot
$r$	: Coefficient of wave reflection
$r_i$	: Local coordinates of point $i$ within its rigid body
$R$	: Orientation explicitly represented as rotation matrix
$\rho$	: Density (e.g. of a rigid object)
	: Euclidean distance to specific configuration in potential-field planning
	: Distance metric used in RRT-based motion planning
$t$	: Time
$T$	: Terrain (height data)
$\mathcal{T}$	: (Cartesian) position of a robot
	: Tree data structure in RRT motion planning
$\tau$	: Torque (e.g. applied to a rigid body)
$\theta$	: Orientation reduced to a single heading angle
$\Theta$	: Orientation (e.g. of a rigid body or robot), cp. $q$ , $R$
$U$	: Artificial potential function used in potential-field motion planning
$v$	: Linear velocity (e.g. of a rigid body)
$\mathcal{W}$	: Workspace of a robot
$x$	: Position (e.g. of a rigid body)
$\xi$	: Scaling factor of attractive potential in potential-field planning
	: Variable controlling allowed areas of exploration in obstacle-aware RRT-based algorithms

Tab. A.1: List of symbols

---

## A.2 List of Abbreviations

AABB	:	Axis-aligned bounding box
AI	:	Artificial intelligence
API	:	Application programming interface
CAD	:	Computer aided design
CPU	:	Central processing unit
DOF	:	Degree/degrees of freedom
FPS	:	Frames per second
GPS	:	Global Positioning System
GPU	:	Graphics processing unit
GUI	:	Graphical user interface
HIL(S)	:	Hardware-in-the-loop (simulation)
IP	:	Internet Protocol
MER	:	Mars Exploration Rover
OBB	:	Object-aligned bounding box
PC	:	Personal Computer
PCI	:	Peripheral Component Interconnect
PID	:	Proportional-integral-derivative (controller)
PPU	:	Physics processing unit
PRM	:	Probabilistic road-maps
RAM	:	Random access memory
RPP	:	Randomised potential-field (planner)
RRT	:	Rapidly-growing random tree
TCP	:	Transmission Control Protocol
VRAM	:	Video RAM
VRML	:	Virtual reality modeling language
XML	:	Extensible markup language

Tab. A.2: List of abbreviations



## BIBLIOGRAPHY

- [Akinc 03] M. Akinc, K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plaku & L. E. Kavraki.  
*Probabilistic Roadmaps of Trees for Parallel Computation of Multiple Query Roadmaps.*  
In Proceedings 11<sup>th</sup> International Symposium of Robotics Research, ISRR, 2003.
- [Alami 95] R. Alami, F. Robert, F. Ingrand & S. Suzuki.  
*Multi-Robot Cooperation through Incremental Plan-Merging.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 3, pages 2573–2579, 1995.
- [Amato 96] Nancy M. Amato & Yan Wu.  
*A Randomized Roadmap Method for Path and Manipulation Planning.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 113–120, 1996.
- [Appel 68] Arthur Appel.  
*Some Techniques for Shading Machine Renderings of Solids.*  
In Proceedings AFIPS Spring Joint Computer Conference, volume 32, 1968.
- [Arya 98] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman & A. Y. Wu.  
*An Optimal Algorithm for Approximate Nearest Neighbor Searching Fixed Dimensions.*  
ACM, 1998.
- [Atramentov 02] A. Atramentov & S.M. LaValle.  
*Efficient Nearest Neighbor Searching for Motion Planning.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, 2002.
- [Baciu 03] George Baciu & Wingo S. K. Wong.  
*Image-Based Techniques in a Hybrid Collision Detector.*  
IEEE Transactions on Visualization and Computer Graphics, vol. 9, no. 2, pages 254–271, 2003.
- [Bandi 95] Srikanth Bandi & Daniel Thalmann.  
*An Adaptive Spatial Subdivision of the Object Space for Fast Collision Detection of Animated Rigid Bodies.*  
Computer Graphics Forum, vol. 14, no. 3, pages 259–270, 1995.

- [Baraff 89] David Baraff.  
*Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies.*  
Computer Graphics, vol. 23, no. 3, pages 223–232, 1989.
- [Baraff 93] David Baraff.  
*Non-Penetrating Rigid Body Simulation.*  
In Proceedings 1<sup>st</sup> Eurographics Workshop on Virtual Environments, 1993.
- [Baraff 94] David Baraff.  
*Fast Contact Force Computation for Nonpenetrating Rigid Bodies.*  
In Proceedings 21<sup>st</sup> Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH, pages 23–34, 1994.
- [Baraff 97] David Baraff.  
*An Introduction to Physically Based Modeling: Rigid Body Simulation 2 - Nonpenetration Constraints.*  
In SIGGRAPH Course Notes, 1997.
- [Barequet 96] Gill Barequet, Bernard Chazelle, Leonidas J. Guibas, Joseph S. B. Mitchell & Ayellet Tal.  
*BOXTREE: Hierarchical Representation for Surfaces in 3D.*  
Computer Graphics Forum, vol. 15, no. 3, pages 387–396, 1996.
- [Barraquand 90a] J. Barraquand & J.-C. Latombe.  
*A Monte-Carlo Algorithm for Path Planning with Many Degrees of Freedom.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 3, pages 1712–1717, 1990.
- [Barraquand 90b] Jérôme Barraquand, Bruno Langlois & Jean-Claude Latombe.  
*Robot Motion Planning with many Degrees of Freedom and Dynamic Constraints.*  
In Proceedings 5<sup>th</sup> International Symposium of Robotics Research, ISRR, pages 435–444, 1990.
- [Barraquand 91] J. Barraquand & J.-C. Latombe.  
*Robot Motion Planning: A Distributed Representation Approach.*  
International Journal of Robotics Research, vol. 10, no. 6, pages 628–649, 1991.
- [Barraquand 92] J. Barraquand, B. Langlois & J.C. Latombe.  
*Numerical Potential Field Techniques for Robot Path Planning.*  
IEEE Transactions on Robotics and Automation, Man and Cybernetics, vol. 22, no. 2, 1992.
- [Barraquand 93] J. Barraquand & J.-C. Latombe.  
*Nonholonomic Multibody Mobile Robots: Controllability and Motion Planning in the Presence of Obstacles.*  
Algorithmica, vol. 10, no. 2–4, pages 121–155, 1993.
- [Barzel 88] Ronen Barzel & Alan H. Barr.  
*A Modeling System Based on Dynamic Constraints.*

- In Proceedings 15<sup>th</sup> Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH, pages 179–188, 1988.
- [Beckmann 90] N. Beckmann, H.-P. Kriegel, R. Schneider & B. Seeger.  
*The R\*-tree: An Efficient and Robust Access Method for Points and Rectangles.*  
In Proceedings ACM SIGMOD International Conference on Management of Data, pages 322–331, 1990.
- [Bellman 61] Richard E. Bellman.  
Adaptive control processes.  
Princeton University Press, 1961.
- [Ben Amar 95] F. Ben Amar & Ph. Bidaud.  
*Dynamic Analysis of Off-Road Vehicles.*  
In Proceedings 4<sup>th</sup> International Symposium on Experimental Robotics, 1995.
- [BFG 06] *BFG Technologies*<sup>®</sup> : <http://www.bfgtech.com/>, 2006.
- [Bickler 92] D.B. Bickler.  
*A New Family of JPL Planetary Surface Vehicles.*  
In Proceedings International Conference on Missions, Technologies, and Design of Planetary Mobile Vehicles, pages 301–306, 1992.
- [Blumberg 95] Bruce M. Blumberg & Tinsley A. Galyean.  
*Multi-Level Direction of Autonomous Creatures for Real-Time Virtual Environments.*  
In Proceedings 22<sup>nd</sup> Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH, pages 47–54, 1995.
- [Bobrow 85] J.E. Bobrow, S. Dubowsky & J.S. Gibson.  
*Time-Optimal Control of Robotic Manipulators Along Specified Paths.*  
International Journal of Robotics Research, vol. 4, no. 3, pages 3–17, 1985.
- [Bouilly 95] B. Bouilly, T. Siméon & R. Alami.  
*A Numerical Technique for Planning Motion Strategies of a Mobile Robot in Presence of Uncertainty.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 1327–1332, 1995.
- [Brock 99] O. Brock & O. Khatib.  
*Elastic Strips: A Framework for Integrated Planning and Execution.*  
In Proceedings International Symposium on Experimental Robotics, volume 250 of *Lecture Notes in Control and Information Sciences*, pages 328–338. Springer Verlag, 1999.
- [Brooks 83] R. Brooks & T. Lozano-Peréz.  
*A Subdivision Algorithm in Configuration Space for Findpath with Rotation.*  
In Proceedings 8<sup>th</sup> International Joint Conference on Artificial Intelligence, ICAI, pages 799–806, 1983.

- [Bruce 02] J. Bruce & M. Veloso.  
*Real-Time Randomized Path Planning for Robot Navigation.*  
In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2002.
- [Büchler 05] P. Büchler, A. Ettlín & B.J. Nelson.  
*Ibex - A Framework for Hardware in the Loop Simulation.*  
In Proceedings 18<sup>th</sup> International Congress of Mechanical Engineering, COBEM, 2005.
- [Catling 05] D. C. Catling.  
*Twin Studies on Mars.*  
Nature, vol. 436, pages 42–43, 2005.
- [Caurin 94] G.A.P. Caurin & N. Tschichold-Gurman.  
*The Development of a Robot Terrain Interaction System for Walking Machines.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 2, pages 1013–1018, 1994.
- [Cazals 95] Frédéric Cazals, George Drettakis & Claude Puech.  
*Filtering, Clustering and Hierarchy Construction: a New Solution for Ray-Tracing Complex Scenes.*  
Computer Graphics Forum, vol. 14, no. 3, pages 371–382, 1995.
- [Chang 94] Cheol Chang, Myung Jin Chung & Bum Hee Lee.  
*Collision Avoidance of Two General Robot Manipulators by Minimum Delay Time.*  
IEEE Transactions on Systems, Man and Cybernetics, vol. 24, no. 3, pages 517–522, 1994.
- [Chang 95] H. Chang & T.-Y. Li.  
*Assembly Maintainability Study with Motion Planning.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 1012–1019, 1995.
- [Chen 96] C.H. Chen & V. Kumar.  
*Motion Planning of Walking Robots in Environments with Uncertainty.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 3277–3282, 1996.
- [Cherif 93a] M. Cherif, C. Laugier, C. Mil'esi-Bellier & B. Faverjon.  
*Combining Physical and Geometric Models to Plan Safe and Executable Motions for a Rover Moving on a Terrain.*  
In Proceedings 3<sup>rd</sup> International Symposium on Experimental Robotics, 1993.
- [Cherif 93b] Moëz Cherif & Christian Laugier.  
*Using Physical Models to Plan Safe and Executable Motions for a Rover Moving on a Terrain.*  
In Proceedings 1<sup>st</sup> Workshop on Intelligent Robot Systems, 1993.
- [Cherif 94] Moëz Cherif & Christian Laugier.

- Dealing with Vehicle/Terrain Interactions when Planning the Motions of a Rover.*  
In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 1994.
- [Cherif 95] Moëz Cherif & Christian Laugier.  
*On Physically-Based Modeling and Off-Road Vehicle Motion Planning.*  
In Proceedings 4<sup>th</sup> International Conference on Intelligent Autonomous Systems, 1995.
- [Cherif 99a] Moëz Cherif.  
*Kinodynamic Motion Planning for All-Terrain Wheeled Vehicles.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, 1999.
- [Cherif 99b] Moëz Cherif.  
*Motion Planning for All-Terrain Vehicles: A Physical Modeling Approach for Coping with Dynamic and Contact Interaction Constraints.*  
IEEE Transactions on Robotics and Automation, vol. 15, no. 2, pages 202–218, 1999.
- [Clark 03] C.M. Clark, S.M. Rock & J.C. Latombe.  
*Motion Planning for Multiple Mobile Robot Systems Using Dynamic Networks.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 3 of 4222–4227, 2003.
- [Clavel 88] Reymond Clavel.  
*Delta, a Fast Robot with Parallel Geometry.*  
In Proceedings of the 18<sup>th</sup> International Symposium on Industrial Robots, pages 91–100, 1988.
- [Codourey 04] A. Codourey, A. Steinecker, B. Sprenger, M. Honegger, M. Thurner, A.-C. Pliska, U. Gubler & C. Bosshard.  
*A Flexible Desktop Robotic System and Dedicated Tools for Delicate Tasks in Microassembly.*  
Technical report, CSEM, MechRob Special Session, 2004.
- [Cohen 95] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha & Madhav K. Ponamgi.  
*I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments.*  
In Proceedings Symposium on Interactive 3D Graphics, pages 189–196, 1995.
- [Col 06] *COLLADA - Digital Asset Exchange Schema for Interactive 3D:*  
<http://www.khronos.org/collada/>, 2006.
- [Collett 05] Toby H.J. Collett, Bruce A. MacDonald & Brian P. Gerkey.  
*Player 2.0: Toward a Practical Robot Programming Framework.*

- In Proceedings of the Australasian Conference on Robotics and Automation, 2005.
- [Cook 05] Richard A. Cook.  
*The Mars Exploration Rover Project.*  
Acta Astronautica, vol. 57, no. 2–8, pages 116–120, 2005.
- [Corke 96] P.I. Corke.  
*A Robotics Toolbox for MATLAB.*  
IEEE Robotics and Automation Magazine, vol. 3, no. 1, pages 24–32, 1996.
- [Corneille 05] Philip Corneille.  
*Extended Mission for MER Twins.*  
Spaceflight, vol. 47, no. 9, pages 339–343, 2005.
- [Courty 03] Nicolas Courty, Éric Marchand & Bruno Araldi.  
*A New Application For Saliency Maps: Synthetic Vision Of Autonomous Actors.*  
In Proceedings International Conference on Image Processing, ICIP, volume 3, pages 1065–1068, 2003.
- [Dacre-Wright 93] B. Dacre-Wright & T. Siméon.  
*Free Space Representation for a Mobile Robot Moving on a Rough Terrain.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 37–43, 1993.
- [Delaunay 34] Boris Nikolaevich Delaunay.  
*Sur la Sphère Vide.*  
Izvestija Akademii Nauk Sojuza Sovetskikh Socialističeskich Respublik. Otdelenie Matematičeskich i Estestvennych Nauk. Serija 7, pages 793–800, 1934.
- [Donald 93] Bruce Randall Donald, Patrick G. Xavier, John F. Canny & John H. Reif.  
*Kinodynamic Motion Planning.*  
ACM, vol. 40, no. 5, pages 1048–1066, 1993.
- [Donald 95] B. Donald & P. Xavier.  
*Provably Good Approximation Algorithms for Optimal Kinodynamics Planning: Robots with Decoupled Dynamic Bounds.*  
Algorithmica, vol. 14, no. 6, pages 443–479, 1995.
- [Erdmann 87] Michael Erdmann & T. Lozano-Pérez.  
*On Multiple Moving Objects.*  
Algorithmica, vol. 2, no. 4, pages 477–521, 1987.
- [Ettlin 04] Alan Ettlin, Patrick Büchler, Rafal Trzebiatowski, Ronald Vuillemin & Hannes Bleuler.  
*An integrated modular environment for cooperative e-learning.*  
In Proceedings 33<sup>rd</sup> International Symposium IGIP / IEEE / ASEE, pages 424–429, 2004.

- 
- [Ettlin 05a] Alan Ettlin, Patrick Büchler & Hannes Bleuler.  
*Rough-Terrain Robot Motion Planning Based on Topology and Terrain Constitution.*  
In Proceedings 18<sup>th</sup> International Congress of Mechanical Engineering, COBEM, 2005.
- [Ettlin 05b] Alan Ettlin, Patrick Büchler & Hannes Bleuler.  
*A Simulation Environment for Robot Motion Planning.*  
In Proceedings 5<sup>th</sup> International Workshop on Robot Motion and Control, RoMoCo, 2005.
- [Ettlin 05c] Alan Ettlin, Patrick Büchler, Roman Schnarrwiler, Xander Schorno & Hannes Bleuler.  
*Physics Simulation Tool-Chain Development as Interdisciplinary Integration Project.*  
In Proceedings 34<sup>th</sup> International Symposium IGIP / IEEE / ASEE, 2005.
- [Ettlin 05d] Alan Ettlin, Patrick Büchler, Rafal Trzebiatowski, Ronald Vuillemin & Hannes Bleuler.  
*Real-Time Physics Simulations in Education.*  
In Proceedings Global Congress on Engineering and Technology Education, 2005.
- [Ettlin 06a] Alan Ettlin & Hannes Bleuler.  
*Randomised Rough-Terrain Robot Motion Planning.*  
In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2006.
- [Ettlin 06b] Alan Ettlin & Hannes Bleuler.  
*Rough-Terrain Robot Motion Planning based on Obstacle-ness.*  
In Proceedings IEEE 9<sup>th</sup> International Conference on Control, Automation, Robotics and Vision, ICARCV, 2006.
- [Farritor 98] S. Farritor, H. Hacot & S. Dubowsky.  
*Physics-Based Planning For Planetary Exploration.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 278–283, 1998.
- [Feder 97] H.J.S. Feder & J.-J.E. Slotine.  
*Real-Time Path Planning using Harmonic Potentials in Dynamic Environments.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 1, pages 874–881, 1997.
- [Ferbach 98] Pierre Ferbach.  
*A Method of Progressive Constraints for Nonholonomic Motion Planning.*  
IEEE Transactions on Robotics and Automation, vol. 14, no. 1, pages 172–179, 1998.
- [France 99] Laure France, Alain Girault, Jean-Dominique Gascuel & Bernard Espiau.

- Sensor Modeling for a Walking Robot Simulation.*  
In Proceedings Eurographics Workshop on Computer Animation and Simulation, CAS, pages 189–198, 1999.
- [García 89] Carlos E. García, David M. Prett & Manfred Morari.  
*Model Predictive Control: Theory and Practice.*  
Automatica, vol. 25, no. 3, pages 335–348, 1989.
- [Garland 95] Michael Garland & Paul S. Heckbert.  
*Fast Polygonal Approximation of Terrains and Height Fields.*  
Technical report CMU-CS-95-181, Computer Science Department, Carnegie Mellon University, 1995.
- [Gaw 86] D. Gaw & A. Meystel.  
*Minimum-Time Navigation of an Unmanned Mobile Robot in a  $2\frac{1}{2}D$  World with Obstacles.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 1670–1677, 1986.
- [Gennery 99] Donald B. Gennery.  
*Traversability Analysis and Path Planning for a Planetary Rover.*  
Autonomous Robots, vol. 6, pages 131–146, 1999.
- [Glassner 89] Andrew S. Glassner.  
An introduction to ray tracing.  
Academic Press Ltd., 1989.
- [Goldstein 02] Herbert Goldstein, Charles P. Poole & John Safko.  
Classical mechanics [international ed.].  
Addison Wesley, 3<sup>rd</sup> edition, 2002.
- [Gonzalez-Baños 98] H.H. Gonzalez-Baños, L.J. Guibas, J.C. Latombe, S.M. LaValle, D. Lin, R. Motwani & C. Tomasi.  
*Motion Planning with Visibility Constraints: Building Autonomous Observers.*  
In Proceedings 8<sup>th</sup> International Symposium of Robotics Research, ISRR, pages 95–101, 1998.
- [Gottschalk 96] S. Gottschalk, M. C. Lin & D. Manocha.  
*OBTree: Hierarchical Structure for Rapid Interference Detection.*  
Computer Graphics, vol. 30, pages 171–180, 1996.
- [Gourret 89] J.-P. Gourret, N. M. Thalmann & D. Thalmann.  
*Simulation of object and human skin formations in a grasping task.*  
In Proceedings 16<sup>th</sup> Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH, pages 21–30, 1989.
- [Green 94] D. N. Green, J. Z. Sasiadek & G. S. Vukovich.  
*Path Tracking, Obstacle Avoidance, and Position Estimation by an Autonomous, Wheeled Planetary Rover.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 1300–1305, 1994.
- [Guo 02] Y. Guo & L. Parker.

- A Distributed and Optimal Motion Planning Approach for Multiple Mobile Robots.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 2612–2619, 2002.
- [Guo 03] Y. Guo, L. E. Parker, D. L. Jung & Z. Dong.  
*Performance-Based Rough Terrain Navigation for Nonholonomic Mobile Robots.*  
In Proceedings 29<sup>th</sup> Annual Conference of the IEEE Industrial Electronics Society, IECON, page 2811, 2003.
- [Haddad 98] H. Haddad, M. Khatib, S. Lacroix & R. Chatila.  
*Reactive Navigation in Outdoor Environments using Potential Fields.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 1232–1237, 1998.
- [Haït 96] A. Haït & T. Siméon.  
*Motion Planning on Rough Terrain for an Articulated Vehicle in Presence of Uncertainties.*  
In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pages 1126–1132, 1996.
- [Haït 99] A. Haït, T. Siméon & M. Taïx.  
*Algorithms for Rough Terrain Trajectory Planning.*  
Advanced Robotics, vol. 14, no. 6, 1999.
- [Hanrahan 89] Pat Hanrahan.  
An introduction to ray tracing, chapter A Survey of Ray-Surface Intersection Algorithms, pages 79–119.  
Academic Press Ltd., 1989.
- [Hanselmann 93] H. Hanselmann.  
*Hardware-In-The-Loop Simulation as a Standard Approach for Development, Customization, and Production Test.*  
SAE Technical Papers, 1993.
- [Hav 06] *Havok Physics<sup>TM</sup>*: <http://www.havok.com/>, 2006.
- [Hebert 89] Martial Hebert, C. Caillas, Eric Krotkov, In So Kweon & Takeo Kanade.  
*Terrain Mapping for a Roving Planetary Explorer.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 2, pages 997–1002, 1989.
- [Hebert 92] Martial Hebert & Eric Krotkov.  
*3D Measurements from Imaging Laser Radars: How Good are they?*  
Image and Vision Computing, vol. 10, no. 3, pages 170–178, 1992.
- [Held 95] Martin Held, James T. Klosowski & Joseph S.B. Mitchell.  
*Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs.*  
In Proceedings 7<sup>th</sup> Canadian Conference on Computational Geometry, pages 205–210, 1995.

- [HML 06] *Lucerne Museum of History*: <http://www.hmluzern.ch/>, 2006.
- [Howard 01] Ayanna Howard & Homayoun Seraji.  
*An Intelligent Terrain-Based Navigation System for Planetary Rovers*.  
IEEE Robotics and Automation Magazine, vol. 8, no. 4, pages 9–17, 2001.
- [Hsu 98] D. Hsu, L. E. Kavraki, R. Motwani & S. Sorkin.  
*On Finding Narrow Passages with Probabilistic Roadmap Planners*.  
In Proceedings International Workshop on Algorithmic Foundations of Robotics, WAFR, 1998.
- [Hsu 99] D. Hsu, J.-C. Latombe & R. Motwani.  
*Path Planning in Expansive Configuration Spaces*.  
International Journal of Computational Geometry and Applications, vol. 9, no. 4 & 5, pages 495–512, 1999.
- [Hsu 02] D. Hsu, R. Kindel, J.C. Latombe & S. Rock.  
*Randomized Kinodynamic Motion Planning with Moving Obstacles*.  
International Journal of Robotics Research, vol. 21, no. 3, pages 233–255, 2002.
- [Hubbard 96] P. M. Hubbard.  
*Approximating Polyhedra with Spheres for Time-Critical Collision Detection*.  
ACM Transactions on Graphics, vol. 15, no. 3, 1996.
- [Huntress 06] W. Huntress, D. Stetson, R. Farquhar, J. Zimmerman, B. Clark, W. O’Neil, R. Bourke & B. Foing.  
*The Next Steps in Exploring Deep Space—A Cosmic Study by the International Academy of Astronautics*.  
Acta Astronautica, vol. 58, no. 6–7, pages 304–377, 2006.
- [Hwang 92] Y. K. Hwang & N. Ahuja.  
*A Potential Field Approach to Path Planning*.  
IEEE Transactions on Robotics and Automation, vol. 8, no. 1, pages 23–32, 1992.
- [Iagnemma 99] Karl Iagnemma, Frank Génot & Steven Dubowsky.  
*Rapid Physics-Based Rough-Terrain Rover Planning with Sensor and Control Uncertainty*.  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 3, pages 2286–2291, 1999.
- [Iagolnitzer 92] M. Iagolnitzer, F. Richard, J.F. Samson & P. Tournassoud.  
*Locomotion of an All-Terrain Mobile Robot*.  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 1, pages 104–109, 1992.
- [Indyk 98] P. Indyk & R. Motwani.  
*Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality*.

- In Proceedings of 30<sup>th</sup> ACM Symposium on Theory of Computing, pages 604–613, 1998.
- [Ioannou 96] Petros A. Ioannou & Jing Sun.  
Robust adaptive control.  
Prentice-Hall, 1996.
- [Irr 06] *Irrlicht graphics engine*: <http://irrlicht.sourceforge.net/>, 2006.
- [Jiang 97] Z.-P. Jiang & H. Nijmeijer.  
*Tracking Control of Mobile Robots: A Case Study in Backstepping*.  
Automatica, vol. 33, no. 7, pages 1393–1399, 1997.
- [Jimenez 91] Stéphane Jimenez, Annie Luciani & Christian Laugier.  
*Physical Modeling as an Help for Planning the Motions of a Land Vehicle*.  
In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pages 1461–1466, 1991.
- [Jimenez 93] Stéphane Jimenez, Annie Luciani & Christian Laugier.  
*Predicting the Dynamic Behaviour of a Planetary Vehicle using Physical Modeling*.  
In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pages 345–351, 1993.
- [Kallmann 03a] M. Kallmann, A. Aubel, T. Abaci & D. Thalmann.  
*Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping*.  
In Proceedings Eurographics, volume 22, 2003.
- [Kallmann 03b] Marcelo Kallmann, Hanspeter Bieri & Daniel Thalmann.  
*Fully Dynamic Constrained Delaunay Triangulations*.  
In Geometric Modelling for Scientific Visualization, pages 241–257, 2003.
- [Kalra 95] Devendra Kalra.  
*A General Formulation of Rigid Body Assemblies for Computer Graphics Modeling*.  
Technical report HPL-95-70, Hewlett-Packard Laboratories, 1995.
- [Kavraki 96] Lydia Kavraki, Petr Svestka, Jean-Claude Latombe & Mark Overmars.  
*Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces*.  
IEEE Transactions on Robotics and Automation, vol. 4, no. 12, pages 566–580, 1996.
- [Kelly 98a] Alonzo Kelly & Anthony Stentz.  
*An Approach to Rough Terrain Autonomous Mobility*.  
Autonomous Robots, 1998.
- [Kelly 98b] Alonzo Kelly & Anthony Stentz.  
*Rough Terrain Autonomous Mobility - Part 1: A Theoretical Analysis of Requirements*.

- Autonomous Robots, no. 5, pages 129–161, 1998.
- [Khatib 86] Oussama Khatib.  
*Real-time Obstacle Avoidance for Manipulators and Mobile Robots.*  
International Journal of Robotics Research, vol. 5, no. 1, pages 90–98, 1986.
- [Kim 03] Jongwoo Kim & James P. Ostrowski.  
*Motion Planning of Aerial Robot using Rapidly-Exploring Random Trees with Dynamic Constraints.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 2200–2205, 2003.
- [Kister 05] Bernd Kister, Cornelia Heldner & Alan Ettl.  
*Der Einsatz Numerischer Berechnungsverfahren zur Simulation der Steinschlagproblematik.*  
In Luzerner Baukolloquium Geotechnik, 2005.
- [Klosowski 98] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral & K. Zikan.  
*Efficient Collision Detection Using Bounding Volume Hierarchies of  $k$ -DOPs.*  
IEEE Transactions on Visualization and Computer Graphics, vol. 4, no. 1, pages 21–36, 1998.
- [Kobilarov 04] Marin Kobilarov & Gaurav S. Sukhatme.  
Time optimal path planning on outdoor terrain for mobile robots under dynamic constraints.  
unpublished research paper from the USC Center for Robotics and Embedded Systems lab, 2004.
- [Koenig 04] N. Koenig & A. Howard.  
*Design and use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator.*  
In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, volume 3, pages 2149–2154, 2004.
- [Koh 99] K. C. Koh & H. S. Cho.  
*A Smooth Path Tracking Algorithm for Wheeled Mobile Robots with Dynamic Constraints.*  
Journal of Intelligent and Robotic Systems, vol. 24, no. 4, pages 367–385, 1999.
- [Krotkov 97] Eric Krotkov, Martial Hebert, Lars Henriksen, Paul Levin, Mark Maimone, Reid Simmons & James Teza.  
*Field Trials of a Prototype Lunar Rover under Multi-Sensor Safe-guarded Teleoperation Control.*  
In American Nuclear Society 7<sup>th</sup> Topical Meeting on Robotics and Remote Systems, pages 575–582, 1997.
- [Kubota 95] T. Kubota, I. Nakatani & T. Yoshimitsu.  
*Path Planning for Planetary Rover based on Traversability Probability.*

- In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 739–744, 1995.
- [Kuffner 99] J. Kuffner & J. Latombe.  
*Fast Synthetic Vision, Memory, and Learning Models for Virtual Humans.*  
In Proceedings IEEE International Conference on Computer Animation, 1999.
- [Kuffner 00] James J. Kuffner & Steven M. LaValle.  
*RRT-Connect: An Efficient Approach to Single-Query Path Planning.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, 2000.
- [Kuffner 03] James Kuffner, Koichi Nishiwaki, Satoshi Kagami, Masayuki Inaba & Hirochika Inoue.  
*Motion Planning for Humanoid Robots.*  
In Proceedings 11<sup>th</sup> International Symposium of Robotics Research, ISRR, 2003.
- [Kweon 91] I.S. Kweon & T. Kanade.  
*Extracting Topographic Features for Outdoor Mobile Robots.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 3, pages 1992–1997, 1991.
- [Lafleur 91] B. Lafleur, N. Magnenat-Thalmann & D. Thalmann.  
*Cloth Animation with Self-Collision Detection.*  
In Modeling in Computer Graphics : proceedings of the IFIP WG 5.10 Working Conference, 1991.
- [Lamon 04] P. Lamon, A. Krebs, M. Lauria, S. Shooter & R. Siegwart.  
*Wheel Torque Control for a Rough Terrain Rover.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 5, pages 4682–4687, 2004.
- [Latombe 91] Jean-Claude Latombe.  
Robot motion planning, volume SECS 124 of *The Kluwer Int. series in engineering and computer science.*  
Kluwer, cop., 1991.
- [Latombe 99] Jean-Claude Latombe.  
*Motion Planning: A Journey of Robots, Molecules, Digital Actors, and Other Artifacts.*  
International Journal of Robotics Research, vol. 18, no. 11, pages 1119–1128, 1999.
- [Laubach 98] Sharon L. Laubach, Joel W. Burdick & Larry Matthies.  
*An Autonomous Path Planner Implemented on the Rocky7 Prototype Microrover.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 292–297, 1998.
- [Laubach 99] Sharon Laubach & Joel Burdick.

- An Autonomous Sensor-Based Path-Planner for Planetary Micro-rovers.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, 1999.
- [Laue 06] T. Laue, K. Spiess & T. Röfer.  
*SimRobot - A General Physical Robot Simulator and Its Application in RoboCup.*  
In RoboCup 2005: Robot Soccer World Cup IX, number 4020 in Lecture Notes in Artificial Intelligence, pages 173–183. Springer, 2006.
- [Laumond 94] J.-P. Laumond, P.E. Jacobs, M. Taix & R.M. Murray.  
*A Motion Planner for Nonholonomic Mobile Robots.*  
IEEE Transactions on Robotics and Automation, vol. 10, no. 5, pages 577–593, 1994.
- [LaValle 98a] S. M. LaValle.  
*Rapidly-Exploring Random Trees: A New Tool for Path Planning.*  
Technical report TR 98-11, Computer Science Dept., Iowa State University, 1998.
- [LaValle 98b] S.M. LaValle & S.A. Hutchinson.  
*Optimal Motion Planning for Multiple Robots having Independent Goals.*  
IEEE Transactions on Robotics and Automation, vol. 14, no. 6, pages 912–925, 1998.
- [LaValle 99] S. M. LaValle & J. J. Kuffner.  
*Randomized Kinodynamic Planning.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 473–479, 1999.
- [LaValle 01] S. M. LaValle & J. J. Kuffner.  
Algorithmic and computational robotics: New directions, chapter Rapidly-Exploring Random Trees: Progress and Prospects, pages 293–308.  
A K Peters, 2001.
- [Leger 00] Chris Leger.  
Darwin2k: An evolutionary approach to automated design for robotics.  
The International Series in Engineering and Computer Science. Springer, 2000.
- [Liégeois 91] Alain Liégeois & Christophe Moignard.  
*Minimum-Time Motion Planner for Mobile Robots on Uneven Terrains.*  
In Proceedings Robotic Systems: Advanced Techniques and Applications, EURISCON, pages 271–278, 1991.
- [Liégeois 93] Alain Liégeois & Christophe Moignard.

- Optimal Motion Planning of a Mobile Robot on a Triangulated Terrain Model.*  
 In Geometric Reasoning for Perception and Action, volume 708 of *Lecture Notes in Computer Science*, pages 51–68, 1993.
- [Liu 89] Y.-H. Liu, S. Kuroda, T. Naniwa, H. Noborio & S. Arimoto.  
*A Practical Algorithm for Planning Collision-Free Coordinated Motion of Multiple Mobile Robots.*  
 In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 3, pages 1427–1432, 1989.
- [Lozano-Pérez 76] Tomás Lozano-Pérez.  
*The Design of a Mechanical Assembly System.*  
 Technical report AI-TR 397, MIT AI Lab, 1976.
- [Lozano-Pérez 79] Tomás Lozano-Pérez & Michael A. Wesley.  
*An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles.*  
 Communications of the ACM, vol. 22, no. 10, pages 560–570, 1979.
- [Lozano-Pérez 83] Tomás Lozano-Pérez.  
*Spatial Planning: A Configuration Space Approach.*  
 IEEE Transactions on Computers, vol. C-32, no. 2, pages 108–120, 1983.
- [Lozano-Perez 84] T. Lozano-Perez, M. Mason & R. Taylor.  
*Automatic Synthesis of Fine-Motion Strategies for Robots.*  
 International Journal of Robotics Research, vol. 3, no. 1, pages 3–24, 1984.
- [Lumelsky 90] V. Lumelsky & T. Skewis.  
*Incorporating Range Sensing in the Robot Navigation Function.*  
 IEEE Transactions on Systems, Man, and Cybernetics, vol. 20, no. 5, pages 1058–1069, 1990.
- [Lumelsky 95] Vladimir J. Lumelsky & Andrei M. Shkel.  
*Incorporating Body Dynamics into the Sensor-Based Motion Planning Paradigm. The Maximum Turn Strategy.*  
 In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 1637–1642, 1995.
- [MacQueen 67] J. B. MacQueen.  
*Some Methods for classification and Analysis of Multivariate Observations.*  
 In Proceedings 5<sup>th</sup> Berkeley Symposium on Mathematical Statistics and Probability, volume 1, pages 281–297, 1967.
- [Maver 93] J. Maver & R. Bajcsy.  
*Occlusions as a Guide for Planning the Next View.*  
 IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 15, no. 5, pages 417–433, 1993.
- [McKenna 90] Michael McKenna & David Zeltzer.  
*Dynamic Simulation of Autonomous Legged Locomotion.*

- In Proceedings 17<sup>th</sup> Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH, pages 29–38, 1990.
- [Michel 04] Olivier Michel.  
*Webots: Professional Mobile Robot Simulation*.  
International Journal of Advanced Robotic Systems, vol. 1, no. 1, pages 39–42, 2004.
- [Milési-Bellier 93] C. Milési-Bellier, C. Laugier & B. Faverjon.  
*A Kinematic Simulator for Motion Planning of a Mobile Robot on a Terrain*.  
In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 1993.
- [Mitchell 87] Joseph S. B. Mitchell, David M. Mount & Christos H. Papadimitriou.  
*The Discrete Geodesic Problem*.  
SIAM Journal on Computing, vol. 16, no. 4, pages 647–668, 1987.
- [Moore 88] Matthew Moore & Jane Wilhelms.  
*Collision Detection and Response for Computer Animation*.  
In Proceedings 15<sup>th</sup> Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH, pages 289–298, 1988.
- [Morlans 92] R. Morlans & A. Liégeois.  
*A DTM-Based Path Planning Method for Planetary Rovers*.  
In Missions, Technologies and Design of Planetary Mobile Vehicles, 1992.  
Paper 4-10.
- [Mortenson 97] Michael E. Mortenson.  
Geometric modeling (2<sup>nd</sup> ed.).  
John Wiley & Sons, Inc., 1997.
- [Nashashibi 94] F. Nashashibi, P. Fillatreau, B. Dacre-Wright & T. Siméon.  
*3-D Autonomous Navigation in a Natural Environment*.  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 1, pages 433–439, 1994.
- [Naylor 90] Bruce Naylor, John Amanatides & William Thibault.  
*Merging BSP Trees Yields Polyhedral Set Operations*.  
In Proceedings 17<sup>th</sup> Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH, pages 115–124, 1990.
- [Neb 06] *The Nebula Device 2 graphics engine: <http://nebuladevice.cubik.org/>, 2006.*
- [New 06] *Newton<sup>TM</sup> Game Dynamics: <http://www.newtondynamics.com/>, 2006.*
- [Nilsson 69] N.J. Nilsson.  
*A Mobile Automation: An Application of Artificial Intelligence Techniques*.  
In Proceedings 1<sup>st</sup> International Joint Conference on Artificial Intelligence, pages 509–520, 1969.

- 
- [Nilsson 80] Nils J. Nilsson.  
Principles of artificial intelligence.  
Tioga Publishing Company, 1980.
- [Nilsson 98] N. J. Nilsson.  
Artificial intelligence: A new synthesis.  
Morgan Kaufmann Publishers, 1998.
- [Noborio 89] H. Noborio, S. Fukuda & S. Arimoto.  
*Fast Interference Check Method using Octree Representation*.  
Advanced Robotics, vol. 3, pages 193–212, 1989.
- [Noser 95] Hansrudi Noser, Olivier Renault, Daniel Thalmann & Nadia Magnenat-Thalmann.  
*Navigation for Digital Actors Based on Synthetic Vision, Memory, and Learning*.  
Computers and Graphics, vol. 19, no. 1, pages 7–19, 1995.
- [ODE 06] *Open Dynamics Engine<sup>TM</sup>*: <http://www.ode.org/>, 2006.
- [O'Donnell 89] P.A. O'Donnell & T. Lozano-Perez.  
*Deadlock-Free and Collision-Free Coordination of Two Robot Manipulators*.  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 1, pages 484–489, 1989.
- [Ope 06] *OpenSim, a 3D simulator for autonomous robots*: <http://opensimulator.sourceforge.net/>, 2006.
- [Pai 98] D.K. Pai & L.M. Reissell.  
*Multiresolution Rough Terrain Motion Planning*.  
IEEE Transactions on Robotics and Automation, vol. 1, 1998.
- [Papadopoulos 96] E.G. Papadopoulos & D.A. Rey.  
*A New Measure of Tipover Stability Margin for Mobile Manipulators*.  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, pages 3111–3116, 1996.
- [Phy 06] *Ageia<sup>TM</sup> PhysX<sup>TM</sup> rigid body dynamics libraries*: <http://www.ageia.com>, 2006.
- [Pressman 00] R. S. Pressman & D. Ince.  
Software engineering: A practitioner's approach, european adaptation.  
McGraw-Hill Publishing Company, 2000.
- [Quinlan 94] Sean Quinlan.  
*Efficient Distance Computation Between Non-Convex Objects*.  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, 1994.
- [Redon 02] S. Redon, A. Kheddar & A. Coquillart.  
*Fast Continuous Collision Detection between Rigid Bodies*.  
In Proceedings Eurographics, volume 21, 2002.
- [Reif 97] J. Reif & H. Wang.

- Algorithms for robotic motion and manipulation, chapter Non-Uniform Discretization Approximations for Kinodynamic Motion Planning and its Applications, pages 97–112.  
A K Peters, 1997.
- [Renault 90] Olivier Renault, Nadia Magnenat-Thalmann & Daniel Thalmann.  
*A Vision-based Approach to Behavioral Animation.*  
Journal of Visualization and Computer Animation, vol. 1, no. 1, pages 18–21, 1990.
- [Rimon 92] E. Rimon & D.E. Koditschek.  
*Exact Robot Navigation using Artificial Potential Functions.*  
IEEE Transactions on Robotics and Automation, vol. 8, no. 5, pages 501–518, 1992.
- [Rippa 90] Samuel Rippa.  
*Minimal Roughness Property of the Delaunay Triangulation.*  
Computer Aided Geometric Design, vol. 7, no. 6, pages 489–497, 1990.
- [Schwartz 83a] Jacob T. Schwartz & Micha Sharir.  
*On the Piano Movers' Problem: II: General Techniques for Computing Topological Properties of Real Algebraic Manifolds.*  
Advances in Applied Mathematics, vol. 4, pages 298–351, 1983.
- [Schwartz 83b] Jacob T. Schwartz & Micha Sharir.  
*On the Piano Movers' Problem: III Coordinating the Motion of Several Independent Bodies: the Special Case of Circular Bodies Moving Amidst Polygonal Barriers.*  
International Journal of Robotics Research, vol. 2, no. 3, pages 46–75, 1983.
- [Sekhavat 98] S. Sekhavat, P. ˇ Svestka, J.-P. Laumond & M. Overmars.  
*Multi-level path planning for nonholonomic robots using semi-holonomic subsystems.*  
International Journal of Robotics Research, vol. 17, no. 8, pages 840–857, 1998.
- [Seraji 99] Homayoun Seraji.  
*Traversability Index: a new Concept for Planetary Rovers.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, 1999.
- [Seraji 00] Homayoun Seraji.  
*Fuzzy Traversability Index: A new Concept for Terrain-Based Navigation.*  
Journal of Robotic Systems, vol. 17, no. 2, pages 75–91, 2000.
- [Seraji 02] Homayoun Seraji & Ayanna Howard.  
*Behavior-Based Robot Navigation on Challenging Terrain: A Fuzzy Logic Approach.*  
IEEE Transactions on Robotics and Automation, vol. 18, no. 3, pages 308–321, 2002.

- 
- [Seraji 03] Homayoun Seraji.  
*New Traversability Indices and Traversability Grid for Integrated Sensor/Map-Based Navigation.*  
Journal of Robotic Systems, vol. 20, no. 3, pages 121–134, 2003.
- [Seugling 06] Axel Seugling & Martin Rölin.  
Evaluation of physics engines and implementation of a physics module in a 3d-authoring tool.  
Master's thesis, Umeå University, Dept. of Computing Science, 2006.
- [Shiller 90a] Z. Shiller & J.C. Chen.  
*Optimal Motion Planning of Autonomous Vehicles in Three-Dimensional Terrains.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, 1990.
- [Shiller 90b] Z. Shiller & H.-H. Lu.  
*Robust Computation of Path Constrained Time Optimal Motions.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 1, pages 144–149, 1990.
- [Shiller 91a] Z. Shiller & S. Dubowsky.  
*On Computing the Global Time-Optimal Motions of Robotic Manipulators in the Presence of Obstacles.*  
IEEE Transactions on Robotics and Automation, vol. 7, no. 6, pages 785–797, 1991.
- [Shiller 91b] Z. Shiller & Y.R. Gwo.  
*Dynamic Motion Planning of Autonomous Vehicles.*  
IEEE Transactions on Robotics and Automation, vol. 7, no. 2, 1991.
- [Shiller 99] Z. Shiller.  
*Motion Planning for Mars Rover.*  
In Proceedings 1<sup>st</sup> International Workshop on Robot Motion and Control, RoMoCo, 1999.
- [Shiller 00] Z. Shiller.  
*Obstacle Traversal for Space Exploration.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, 2000.
- [Shiller 04] Z. Shiller, Y. Fujita, D. Ophir & Y. Nakamura.  
*Computing a Set of Local Optimal Paths through Cluttered Environments and over Open Terrain.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 5, pages 4759–4764, 2004.
- [Shimoda 05] Shingo Shimoda, Yoji Kuroda & Karl Iagnemma.  
*Potential Field Navigation of High Speed Unmanned Ground Vehicles on Uneven Terrain.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, 2005.
- [Shin 84] K.G. Shin & N.D. McKay.

- Open Loop Minimum Time Control of Mechanical Manipulators.*  
American Control Conference, pages 1231–1236, 1984.
- [Shkel 97] Andrei M. Shkel & Vladimir J. Lumelsky.  
*The Jogger's Problem: Control of Dynamics in Real-Time Motion Planning.*  
Automatica, vol. 33, no. 7, pages 1219–1233, 1997.
- [Shoemake 85] Ken Shoemake.  
*Animating Rotation with Quaternion Curves.*  
In Proceedings 12<sup>th</sup> Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH, 1985.
- [Siméon 91] T. Siméon.  
*Motion Planning for a Nonholonomic Mobile Robot on 3-Dimensional Terrains.*  
In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pages 1455–1450, 1991.
- [Siméon 93] T. Siméon & B. Dacre-Wright.  
*A Practical Motion Planner for All-Terrain Mobile Robots.*  
In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 1993.
- [Siméon 02] Thierry Siméon, Stephane Leroy & Jean-Paul Laumond.  
*Path Coordination for Multiple Mobile Robots: a Resolution-Complete Algorithm.*  
IEEE Transactions on Robotics and Automation, vol. 18, no. 1, pages 42–49, 2002.
- [Simmons 95] Reid Simmons, Eric Krotkov, Lonnie Chrisman, Fabio Cozman, Richard Goodwin, Martial Hebert, Lalitesh Katragadda, Sven Koenig, G. Krishnaswamy, Y. Shinoda, William Red L. Whitaker & P. Klarer.  
*Experience with Rover Navigation for Lunar-like Terrains.*  
In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pages 441–446, 1995.
- [Singh 99] A.P. Singh, J.C. Latombe & D.L. Brutlag.  
*A Motion Planning Approach to Flexible Ligand Binding.*  
In Proceedings 7<sup>th</sup> International Conference on Intelligent Systems for Molecular Biology, pages 252–261, 1999.
- [Singh 00] Sanjiv Singh, Reid Simmons, Trey Smith, Anthony Stentz, Vandt Verma, Alex Yahja & Kurt Schwehr.  
*Recent Progress in Local and Global Traversability for Planetary Rovers.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 2, pages 1194–1200, 2000.
- [Snyder 95] John M. Snyder.  
*An Interactive Tool for Placing Curved Surfaces without Interpenetration.*

- 
- In Proceedings 22<sup>nd</sup> Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH, pages 209–218, 1995.
- [Son 00] Wookho Son, Kyunghwan Kim & Nancy M. Amato.  
*An Interactive Generalized Motion Simulator (GMS) in an Object-Oriented Framework.*  
In Proceedings Computer Animation, pages 176–181, 2000.
- [Spenko 04] Matthew J. Spenko, Karl D. Iagnemma & Steven Dubowsky.  
*High-Speed Hazard Avoidance for Mobile Robots in Rough Terrain.*  
In Proceedings SPIE - Unmanned ground vehicle technology VI, volume 5422, pages 439–450, 2004.
- [Stadd 04] Courtney Stadd & Jeff Bingham.  
*The US Civil Space Sector: Alternate Futures.*  
Space Policy, vol. 20, no. 4, pages 241–252, 2004.
- [Stentz 94] Anthony Stentz.  
*Optimal and Efficient Path Planning for Partially-Known Environments.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 4, pages 3310–3317, 1994.
- [Strandberg 04] M. Strandberg.  
*Augmenting RRT-Planners with Local Trees.*  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, volume 4, pages 3258–3262, 2004.
- [Svestka 95] P. Svestka & M.H. Overmars.  
*Motion Planning for Car-like Robots using a Probabilistic Learning Approach.*  
International Journal of Robotics Research, vol. 16, no. 2, pages 119–143, 1995.
- [Svestka 98] P. Svestka & M.H. Overmars.  
*Coordinated Path Planning for Multiple Robots.*  
Robotics and Autonomous Systems, vol. 23, no. 4, pages 125–133, 1998.
- [Takeda 94] H. Takeda, C. Facchinetti & J.-C. Latombe.  
*Planning the Motions of a Mobile Robot in a Sensory Uncertainty Field.*  
IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 16, no. 10, pages 1002–1017, 1994.
- [Taylor 88] R.H. Taylor & V.T. Rajan.  
*The Efficient Computation Of Uncertainty Spaces For Sensor-based Robot Planning.*  
In Proceedings IEEE International Workshop on Intelligent Robots, pages 231–236, 1988.
- [Thüer 06] T. Thüer, A. Krebs & R. Siegwart.  
*Comprehensive Locomotion Performance Evaluation of All-Terrain Robots.*

- In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2006.
- [Tok 06] *Tokamak Game Physics SDK*: <http://www.tokamakphysics.com/>, 2006.
- [Tombropoulos 99] R. Tombropoulos, J.R. Adler & J.C. Latombe.  
*Carabeamer: A Treatment Planner for a Robotic Radiosurgical System with General Kinematics*.  
Medical Image Analysis, vol. 3, no. 3, pages 237–264, 1999.
- [Urmson 03] C. Urmson & R. Simmons.  
*Approaches for Heuristically Biasing RRT Growth*.  
In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2003.
- [van Amerongen 03] J. van Amerongen & P. Breedveld.  
*Modelling of Physical Systems for the Design and Control of Mechatronic Systems*.  
Annual Reviews in Control, vol. 27, no. 1, pages 87–117, 2003.
- [Vaughan 03] Richard T. Vaughan, Brian Gerkey & Andrew Howard.  
*On Device Abstractions For Portable, Resuable Robot Code*.  
In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pages 2121–2427, 2003.
- [Vazquez 02] P.-P. Vazquez, M. Feixas, M. Sbert & A. Llobet.  
*Viewpoint Entropy: A new Tool for Obtaining Good Views for Molecules*.  
In Proceedings Eurographics/IEEE Data Visualisation, 2002.
- [Volpe 00] Richard Volpe, Tara Estlin, Sharon Laubach, Clark Olson & J. Balaram.  
*Enhanced Mars Rover Navigation Techniques*.  
In Proceedings IEEE International Conference on Robotics and Automation, ICRA, 2000.
- [Whitted 80] Turner Whitted.  
*An Improved Illumination Model for Shaded Display*.  
Communications of the ACM, vol. 23, no. 6, pages 343–349, 1980.
- [Winnendael 04] M. Van Winnendael, B. Gardini, R. Roumeas & J. Vago.  
*The ExoMars Mission of ESAs Aurora Programme*.  
In Engineering, Construction, and Operations in Challenging Environments: Earth & Space, pages 991–998, 2004.
- [wxW 06] *wxWidgets library*: <http://www.wxwidgets.org/>, 2006.
- [Yahja 00] Alex Yahja, Sanjiv Singh & Anthony Stentz.  
*An Efficient On-line Path Planner for Outdoor Mobile Robots*.  
Robotics and Autonomous Systems, vol. 32, pages 129–143, 2000.
- [Yob 06] *Yobotics! Simulation Construction Set*: <http://yobotics.com/simulation/simulation.htm>, 2006.

## CURRICULUM VITAE

Alan Ettlín was born on 10<sup>th</sup> June 1977 in Lucerne, Switzerland. He went to school in England and Switzerland, obtaining a Scientific High School Diploma (Matura Typus C) from Kantonsschule Alpenquai Luzern, Switzerland.

He studied at the Swiss Federal Institute of Technology in Zurich (Eidgenössische Technische Hochschule Zürich, ETHZ) and received the equivalent of an M.Sc. Computing Science (Dipl. Informatik-Ing. ETH) in April 2003. From September 2001 to September 2002, he studied at University of Aberdeen, Scotland. The master thesis “Scene Understanding using Bayesian Networks” written in the Perceptual Computing and Computer Vision Group (PCCV) at ETHZ deals with spatial reasoning about clusters of objects recognised using computer vision techniques.

After contributing to an industry project in the PCCV, he started work as researcher at the Institute of Electronics (IfE) at University of Applied Sciences of Central Switzerland (Fachhochschule Zentralschweiz, FHZ; Hochschule für Technik und Architektur, HTA) in October 2003. In February 2004, he joined the Robotics Systems Laboratory 1 (Laboratoire de systèmes robotiques 1, LSRO1) as an external PhD student while continuing work at IfE on various industry projects related to the research topics of the present work. This cooperation led to the doctoral thesis “Rigid Body Dynamics Simulation for Robot Motion Planning”. He passed his PhD exam on 29<sup>th</sup> September 2006.

