Technische Universität München
Lehrstuhl für Echtzeitsysteme und Robotik

Efficient Motion Planning for Intuitive Task Execution in Modular Manipulation Systems

Markus Rickert

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Michael Beetz, Ph.D.

Prüfer der Dissertation:
1. Univ.-Prof. Dr. Alois Knoll
2. Prof. Dr. Herman Bruyninckx, Katholieke Universiteit Leuven/Belgien

Die Dissertation wurde am 30. Juni 2010 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 8. Mai 2011 angenommen.

Abstract

The design of complex systems in robotics requires modern engineering approaches and intuitive task paradigms. Approaches based on teach-in concepts and textual programming languages with steep learning curves result in long development cycles that are not suitable for flexible and modular factory environments. Application specialists require accessible robotics interfaces in order to create new custom solutions compared to the expert knowledge in both domains that is currently required. The removal of safety fences and the availability of light-weight robot structures leads to the desire for direct interaction with a human operator and raises the issue of advanced and multimodal command paradigms. Modularity, fault-tolerance, and redundancy require easy integration of different sensor and actuator components as well as fast replacement of similar devices.

This work introduces an instruction paradigm that goes beyond typical programming interfaces and toward a more intuitive interaction style with a robot system. A suitable system architecture and a general programming paradigm are necessary design goals in such a concept. Motion planning algorithms can be used in order to automate tasks such as pick-and-place operations, especially for frequently adjusted target positions in changing scenarios. However, complex environments may provoke inefficient and drastically varying paths. Computationally efficient motion planning must avoid exhaustive exploration of configuration space and actively balance the exploitation of available information. A newly developed motion planner actively balances these two opposing forces in order to increase efficiency and achieve significant performance improvements compared to other state-of-the-art sampling-based planners.

Verbal and non-verbal communication play an important role when instructing complex systems in an intuitive way. Accessible descriptions are realized by a combination of task-based control algorithms with multimodal interfaces and cognitive reasoning modules. The developed online control architecture for distributed systems also addresses safety aspects by integrating various sensor components. Multiple constraints can be included and sorted by priority in order to achieve desired behaviors patterns. Abstraction of kinematics, dynamics, geometry, and hardware aspects result in flexible systems with replaceable components. Reengineering bet-

ween systems is reduced due to the development of an open source framework with common robotics algorithms.

In summary, this work provides valuable contributions to the field of motion planning algorithms, a task-driven instruction paradigm independent of a specific robot system, and an open source framework introducing concepts of model-based engineering.

Zusammenfassung

Der Entwurf komplexer Robotiksysteme erfordert moderne Entwicklungsansätze und intuitive Aufgabenbeschreibungen. Ansätze, die auf Teach-In-Konzepten und textuellen Programmiersprachen mit steilen Lernkurven basieren, führen zu langen Entwicklungszyklen, die nicht für flexible und modulare Fabrikumgebungen geeignet sind. Applikationsspezialisten benötigen zugängliche Robotikschnittstellen um neue angepasste Lösungen zu erschaffen, verglichen mit dem derzeit benötigten Expertenwissen in beiden Domänen. Die Entfernung von Sicherheitszäunen und die Verfügbarkeit von Leichtbauarmen führen zum Wunsch nach direkter Interaktion mit einem menschlichen Benutzer und eröffnen die Frage nach fortgeschrittenen und multimodalen Befehlsparadigmen. Modularität, Fehlertoleranz und Redundanz benötigen einfache Integration von verschiedenen Sensor- und Aktorkomponenten, sowie die schnelle Austauschbarkeit von ähnlichen Geräten.

Diese Arbeit führt ein neues Instruktionsmodell ein, das über klassische Programmiermethoden hinaus geht in Richtung einer intuitiven Zusammenarbeit mit einem Roboter. Eine geeignete Systemarchitektur und ein generelles Programmiermuster sind nötige Designziele in solch einem Konzept. Bahnplanungsalgorithmen können eingesetzt werden um Aufgaben wie Pick-and-Place Operationen zu automatisieren, insbesondere für häufig angepasste Zielpositionen in sich verändernden Szenarios. Komplexe Umgebungen können jedoch ineffiziente und stark variierende Pfade hervorrufen. Berechnungseffiziente Bahnplanung muss vollständige Konfigurationsraum-Exploration vermeiden und aktiv die Ausnutzung verfügbarer Informationen ausbalancieren. Ein neuentwickelter Bahnplanungsalgorithmus balanciert diese beiden gegenwirkenden Kräfte aktiv, um eine Effizienzsteigerung und signifikante Leistungssteigerungen gegenüber anderen modernen stichprobenbasierten Planern zu erreichen.

Verbale und nichtverbale Kommunikation spielen eine wichtige Rolle in der Instruierung komplexer Systeme auf intuitive Weise. Zugängliche Beschreibungen werden durch eine Kombination von aufgabenorientierten Kontrollalgorithmen mit multimodalen Schnittstellen und kognitiven Entscheidungsmodulen erreicht. Die entwickelte Online-Kontrollarchitektur für verteilte Systeme beschäftigt sich auch mit Sicherheitsaspekten durch die Integration verschiedener Sensorkomponenten.

Mehrere Bedingungen können berücksichtigt und nach Priorität sortiert werden, um gewünschte Verhaltensmuster zu erreichen. Abstraktion von Kinematik-, Dynamik-, Geometrie- und Hardwareaspekten führt zu flexiblen Systemen mit austauschbaren Komponenten. Neuentwicklungen unter Systemen werden vereinfacht durch eine quelloffene Bibliothek mit verbreiteten Robotikalgorithmen.

Zusammenfassend leistet diese Arbeit wertvolle Beiträge im Bereich Bahnplanungsalgorithmen, ein aufgabengetriebenes Instruktionsmuster das unabhängig von spezifischen Robotersystemen ist, sowie ein quelloffenes Framework das Konzepte modellbasierter Entwicklung einführt.

Acknowledgments

First of all, I would like to thank my advisor Prof. Dr. Alois Knoll for the opportunity to work in many interesting and challenging projects as well as for his encouragement and support. I would also like to thank Prof. Dr. Hermann Bruyninckx for investing the time to get to know my work and acting as second advisor.

Furthermore, I would like to thank Prof. Dr. Oliver Brock, Prof. Dr. Darius Burschka, and Prof. Dr. Jürgen Schmidhuber for many interesting, helpful, and inspiring discussions over the past few years.

Special thanks to Dr. Gerhard Schrott, Dr. Reinhard Lafrenz, Monika Knürr, Amy Bücherl, and Gisela Hibsch for the constant support and motivation.

Many thanks to Simon Barner, Dr. Christian Buckl, Dr. Mary Ellen Foster, Michael Geisinger, Manuel Giuliani, Markus Huber, Michael Kaßecker, Claus Lenz, Elmar Mair, Dr. Stefan Riesner, Thorsten Röder, Oliver Ruepp, and all the other colleagues and students I had the pleasure to work with during the past few years.

Finally, I would like to thank my friends and family, especially my parents, for the constant support and encouragement during the work on this thesis.

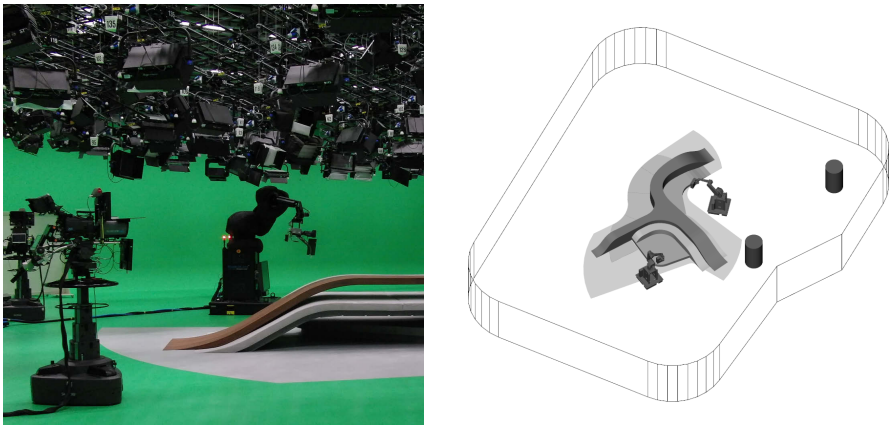
Contents

Chapter 1
Introduction

Robotic applications are among today's most complex systems, with solutions overlapping a vast amount of diverse domains and requirements. Robot systems are often deployed in critical areas where any downtime in production cannot be tolerated. Critical real-time constraints have to be met during execution and the system must include precautionary measures in case of failures or even provide redundancy. Operators have to possess expert knowledge in specific production or application domains as well as in the field of robotics. Environments easily managed by humans present tremendous challenges for robots and the realization of all requirements demands the compliance with the most recent methods and principles of software and systems engineering.

Current industrial settings typically prevent direct cooperation between human workers and robot systems due to safety regulations and heavy machinery. The scenarios in which manipulators are used often involve large production volumes with repeated movements. These programs are largely developed and optimized over a number of weeks and integration in installations is often reduced to a minimum, with a limited set of input/output signals as a means of synchronization between different components. New developments however show that more flexible programming can support the development of individual solutions and small batch production models. Interaction between several robotic installations as well as with human coworkers can further enhance production efficiency.

Systems like the television studio application presented in Fig.1.1 demonstrate how advanced sensor integration can simplify automation and help overcome uncertainties in detection and positioning of manipulation objects. Similar shots are used in every broadcast and mobile manipulators can be used in order to replace human camera operators. In this example, traditional solutions such as distance sensors are infeasible due to the green screen of the virtual studio. The walls cannot be marked in any way for the same reason and localization has to be highly accurate. In order to accomplish this, the floor is equipped with a number of extremely accurate sensors and the platform's manipulator can be used in order to compensate for position errors. The large number of obstacles in the environment and the ability for operators to create new custom movements requires the availability of collision de-



(a) (b)

Fig. 1.1 Robotics application in a television studio. Operators are able to create new movements with a simple interface. Sensors in the floor are used in order to locate the mobile manipulator. **a** real-world overview including multiple obstacles in the environment (Photograph provided by Robotics Technology Leaders GmbH), **b** synchronized representation used for collision detection and visualization

tection systems due to safety aspects. The challenge lies in integrating all hardware and software components of such a complex distributed system (Fig. 1.2) while still providing an intuitive way of programming it.

A lot of fundamental solutions and approaches in robotics have been well-known for decades, with many available textbooks on the specification and calculation of kinematics and dynamics [117, 89, 38, 137] alone. After introduction of the first algorithms in this area a few decades ago, a lot of time was invested into creating more efficient and extremely specialized versions of these routines due to limited hardware performance. Yet, even today, hardly any concrete specifications and implementations are available or have been adopted into industry standards. Hardware manufacturers implement their own set of proprietary instructions based on these foundations. However, flexible solutions that can replace different components without monopolies and production downtime are desired. Most projects—especially in research—start from scratch, time and again, therefore often resulting in poorly conceived, fragile, and in particular untested implementations that cannot be shared with the community. There are no standardized hardware drivers available and many incompatible middleware solutions with proprietary network protocols are introduced rather than focusing on proper interface specifications. Efforts for standardization are in many cases unknown to larger audiences or fail to be adopted among different domains. The complexity of these solutions typically prevents current industrial operators from creating new custom applications. Special training with a steep learning curve is required in order to gather expert knowledge and implement even the most basic tasks.

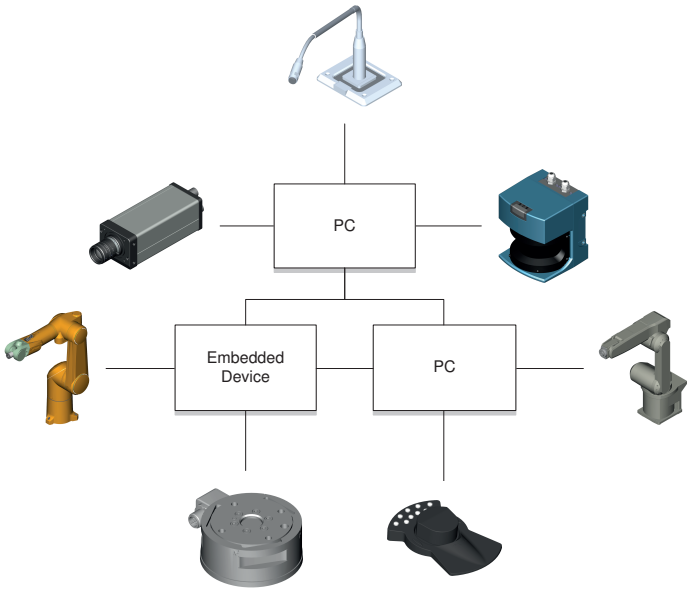


Fig. 1.2 Example of a distributed target system with a series of actuators and sensors. Each component typically uses different interfaces and protocols, thus requiring custom device drivers. The network communication between machines and the used operating systems should respect potential real-time requirements

1.1 Software Development

A nearly infinite amount of libraries and solutions for different areas are available in robotics. The bulk of the projects is focused on the field of middlewares and discusses the encapsulation of different parts in a system and how these can communicate efficiently according to their individual requirements. Typically this results in the introduction of isolated components and the developer's task is to fill in the necessary functionality. Little help is given in the actual development of these components; the vast amount of incompatible vector and matrix classes in high-level programming languages is just one example. Uniform specifications, compatible data structures, basic algorithms, and mature hardware drivers are often scarce and have to be implemented from scratch. This lack of standardization also presents a large barrier for industrial partners and therefore only intensifies these points.

Only the introduction of open and consistent standards leads to mass-market acceptance and adoption across academic and industrial borders. Proprietary specifications in 3D computer graphics may serve as an example to this dilemma. Early manufacturer dependent interfaces that were optimized for specialized hardware components allowed programmers to realize their ideas faster and with higher performance than by writing their own rudimentary implementations. However, only the adoption of open standards enabled software engineers to run their code on the

graphics hardware of their choice and led to an open market for manufacturers. A similar development can now be observed with physics simulations and other general calculations running on this type of hardware. Standardized interface specifications can endorse the development of a large variety of solutions to a single problem and more efficient algorithms can thus be introduced and actually compared to each other. This also includes proper abstraction layers for hardware components and operating systems.

Even today, industrial applications are usually limited to teach-in methods. Typically, every hardware manufacturer maintains its own proprietary programming language and these are only suitable to fulfill a very limited set of task requirements. Special training and modifications are required after switching to different equipment due to incompatible interfaces, thus facilitating vendor lock-in. On the other hand, few techniques developed in research are mature enough to be adopted outside of laboratories and the implementations are often not stable enough for operation in production settings. Required however are generative programming, including code generation, verification, automatic deployment, reusability, and simple configuration. With this in place, task description can switch to a higher level of abstraction and may allow for programming by less specialized staff.

1.1.1 Current Software Engineering Approaches in Robotics

Most of the current approaches to software engineering in robotics can be categorized into three main categories. The first class tries to provide functionality of a specific domain in the form of software libraries that can be used in manual implementations. Many open solutions for a broad spectrum of specialized fields are available, including simultaneous localization and mapping (SLAM) [108], motion planning [121], as well as kinematics, dynamics, hardware control, or Bayesian filtering [25]. These frameworks are usually designed as an aid for programmers when writing specific programs, a lack of common data structures and interfaces however often complicates the cooperation between different libraries.

By far the largest and most common category of robotics software addresses the definition of new communication patterns. Similar to a multitude of already existing approaches such as CORBA, DCOM, SOAP, and Ice, these middlewares provide interfaces for data exchange between distributed computer systems and remote procedure calls [35, 106, 24, 49, 124]. The developer is often given no support in the creation of the individual modules in such a system and most implementations therefore are created from scratch. During these efforts some may also revert to a selection of frameworks if the required functionality is available as part of these libraries. More often than not however the specification of consistent interfaces is omitted, thus resulting in a large variety of incompatible device drivers and basic algorithms. Due to this, compatible modules in the same area cannot be exchanged for one another, including motion planning, SLAM, object and speech recognition,

as well as hardware components such as cameras, force and range sensors, optical trackers, or motors.

Of these middleware solutions, only a few try to offer advanced functionality in the form of visual programming, simulation, code generation, debugging, or deployment [4, 81]. The goal of these integrated development approaches is to provide a common tool chain for creating robotics applications. While some efforts have been started to standardize communication patterns, common interfaces for hardware components and algorithms of the same type are still rare. If available, code generation, debugging, and deployment is very limited and proper automated verification is usually completely absent. This also includes many real-time considerations and abstract descriptions of sensor information, together with physical properties of robot models such as kinematics, dynamics, and geometry data. The tool chains also by no means provide complete environments for the whole development process.

While not actually part of any classic category, the object-oriented modeling of mechatronic systems as presented in [114] provides interesting solutions for many basic issues. Mainly used for simulation purposes, this type of modeling can describe components of technical systems from mechanical, electrical, hydraulic, control, and numerous other domains. By adding definitions in the form of differential, algebraic, and discrete equations, these open specifications are all compatible and can be combined across several areas. Code can be automatically generated based on these equations and early attempts are made to utilize this feature outside of simulation [1, 43].

1.1.2 Toward Model-Driven Engineering

In contrast to a traditional development process, model-driven engineering focuses on the supply of reusable parts that can be combined with each other. Instead of trying to solve problems all over again, mature components are created that are compatible and can build on top of each other. From a robotics perspective, these models should incarnate two important points of view: the available hardware components and the tasks they are being used for (Fig. 1.3).

One side represents the physical properties, including the type of the robot with its specific kinematics, dynamics, and mechanics. Furthermore, all available sensors in the robot itself and the ones placed in the surrounding environment, together with a classification of the type of information made available by them. It also involves the exact shape of available geometries, hardware drivers required for controlling specific actuator and sensor components, and special requirements regarding the operating system, the target architecture, and basic time constraints. The other side contains algorithmic components, where the actual programs are provided and eventually run on existing systems. Ideally, this is implemented in a way that allows for the exchange of different solutions. Many different algorithms and calculations are supplied, including basic mathematical operations, signal processing, simple image

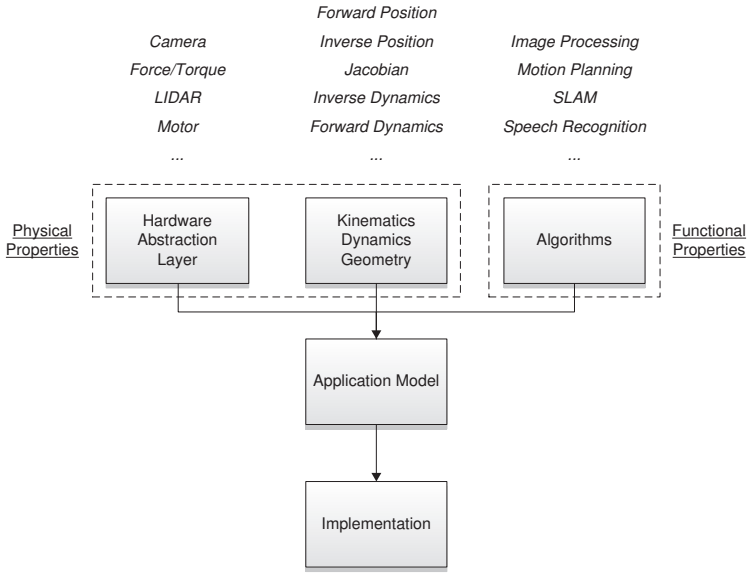


Fig. 1.3 Model-based development process with various modules that can be used in the manual creation of an application. Physical properties of the system are given by device drivers for the hardware components as well as kinematics, dynamics, and geometry representations. A library of algorithms and calculations provides the actual program logic or functional properties

processing, interpolation methods, trajectory optimization techniques, digital filters, collision detection, or more advanced concepts such as object tracking, behavior based programs, and abstract task descriptions.

The actual program logic or application is assembled from a selection of these components. All elements possess dedicated input and output channels for specific data types, as well as a collection of more sophisticated service ports. A motor control component for instance will provide interfaces for reading and writing position, velocity, acceleration, and torque data, whereas a camera will supply the recently acquired raw image and functions for changing parameters such as resolution, frame rate, or gain. The kinematics of the used robot type is responsible for conversions from joint to operational space with respect to a chosen world frame, while a tracking module delivers position data of an observed object relative to the same coordinate system. In a complete and well-tested library of components, advanced algorithms can be added by hierarchically combining a selection of already existing implementations.

Apart from the saved effort of having to reimplement these components from scratch all over again, this also leads to simplifications in code generation and in particular verification. Fully reengineered programs also have to be completely retested and verified, whereas the use of already inspected components can at the least immensely reduce the overall costs. In addition, proper interfaces to target architectures may enable automatic optimization of runtime performance, for instance by

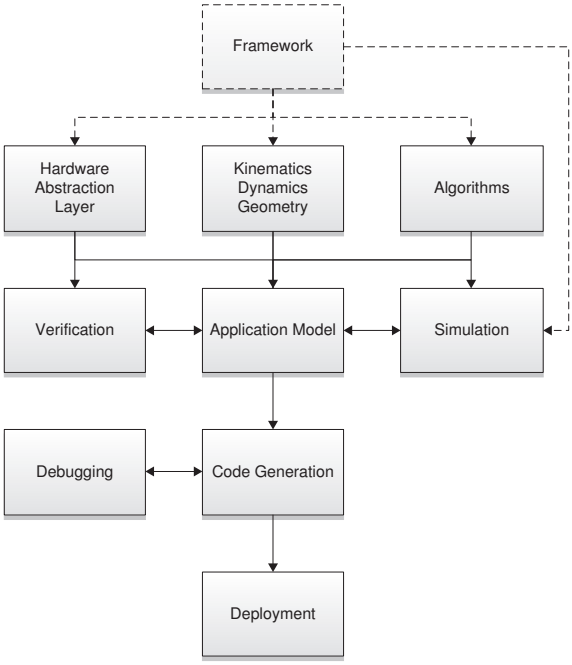


Fig. 1.4 Refined model-driven development process. Application code is automatically generated and deployed to a target system. Simulation, verification, and debugging complete the development process. Frameworks may assist in the creation of modules and tools in the workflow

introducing parallel versions of existing algorithms. Deployment can also utilize compatible patterns by adapting communication channels between components to the target architecture's requirements. A system target comprised of a single unit can take full advantage of direct communication, whereas a distributed system has to pay special attention to the time constraints of its modules. The testing of new components can be vastly improved through the introduction of proper debug and simulation specifications and interfaces, especially with regards to safety aspects. A complete and proper physical description of the robot is not only useful in applications, but can also contribute to a proper simulation of the system.

With the availability of specifications and interfaces in the form of free and open standards, industrial partners can decide to provide compatible hardware drivers and model specifications or even algorithm components. A unified development process also allows for the design and evolution of various compatible applications in a robotic tool chain. This can include editors for kinematics and dynamics parameters or 3D models, but also graphical user interfaces for application design, simulation, and diagnostics. Free and open reference implementations have to be available for every step in the workflow.

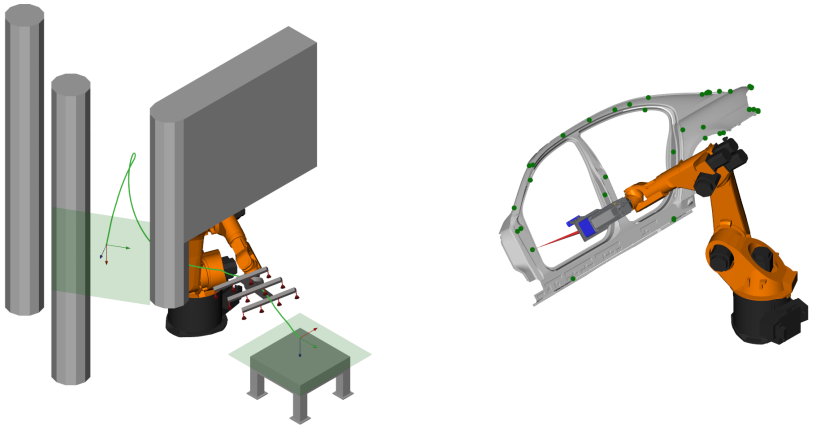
The development process (Fig. 1.4) of a new robot system starts with the analysis of the intended application and the acquirement of suitable components. If the

manufacturer does not supply adequate specifications, models, and drivers for these types of actuators or sensors, the next step comprises the creation of proper kinematics, dynamics, and geometry data. This also includes proper implementations for control of the individual hardware components and abstractions for the intended target architecture and its operating system. Ideally, the set of existing algorithmic foundations already provides a large collection of different modules. New components are added for special requirements that are not fulfilled by the existing set of base elements. The complete target application is then assembled based on these functional parts, specific hardware and operating system abstractions, and matching time and communication constraints. Proper verification of new components is required before deployment, in order to guarantee safety aspects in the overall system. Internet channels can be used for the distribution of new models and implementations, while community ratings and peer review can help with quality assurance. Contributions that have been extensively tested and are verified by experts can be marked accordingly. Extensive testing in simulators can also reduce failures and hardware damage, while debug and deployment components are useful during development.

1.2 From Teach-In to Abstract Task Description

Robot systems in industrial settings are often instructed to perform operations such as pick-and-place, assembly, welding, painting, or cutting over and over again. Typically, these applications are created by an operator using procedures such as the teach-in method, where a series of instructions are written in a vendor-specific programming language. These programs can include basic movements in joint and operational space of the robot, as well as commands for controlling corresponding equipment such as pneumatic valves, grippers, and various input/output signals. Some control architectures may include limited support for sensor equipment, often in the form of force/torque devices.

Although some language constructs may successfully conceal certain workarounds, for instance when picking up objects from a table, there is no direct correlation between the written program and the actual task. Depending on the shape of the object and the surface, the robot usually has to approach the area from a right angle. In order to account for obstacles in the robot's workspace, the operator will have to include evasive movements that are dependent on the current environment or trajectories of other devices and may have to be changed in the future. These additional instructions could also be misinterpreted by a different programmer and thus can result in collisions when modifications to the original program are introduced. In addition, new target coordinates generally require a redesign of the original program. Collision detection and motion planning algorithms can be used in order to create these movements between task points, as long as a correct environment model can be provided (Fig. 1.5a). Depending on the complexity of the task and the environment, finding a collision free path may be extremely difficult and time-consuming.



(a) (b)

Fig. 1.5 Abstract task descriptions without manually programmed movements and largely independent of a specific kinematic structure. **a** pick-and-place operation defined by a start and goal position, **b** laser welding application with a series of target points

In addition, the quality of the solutions from sampling-based planners can also vary and produce solutions that are not well-suited for time-critical operations.

Creating programs for complex manipulation tasks such as welding and cutting is very time-consuming and requires a large amount of instructions and trajectory points. Manual optimization for time-optimal trajectories also becomes increasingly difficult with the number of intermediate points and level-of-detail. This also includes aspects such as time synchronization between multiple manipulators and dynamic optimization with respect to hardware constraints (e.g., joint limits, maximum velocity, acceleration, torque). Some manufacturers and industry solutions have already introduced new offline programming concepts that allow the direct specification of trajectories using CAD models [156]. Rather than specifying a collection of programming instructions with robot movements, the operator can select points, lines, or curves of a target object in order to create corresponding trajectories of the end effector. Various algorithms can then be used in order to optimize this movement with respect to the parameters of the robot and its selected tool. However, these approaches still only generate a series of offline trajectories based on the same set of instructions used in the original teach-in method with no direct access to the underlying control architecture. In contrast to an online control program, integration of additional sensor information required for visual-servoing operations or dynamic environments for reactive obstacle avoidance is typically not supported.

Especially with increasing cooperation between robot systems and human workers and the versatility of manipulators for individual solutions and small batch series, more flexible and accessible solutions for task description are required. Compared to traditional settings with fences and no cooperation, new light-weight manipulator systems allow for direct interaction between human and robot. In such a setup, an

operator can simply touch and move the manipulator to an intended configuration rather than issuing the command by entering coordinates or joint angles. Similarly, speech and gestures are common input methods between human coworkers and a basic request for handing over a cup from a table with a combination of these two is easily understood. A task description for a robot system however will require several additional aspects in order to accomplish the same goal. First, the object's position and orientation in space has to be determined. Various sensors and object recognition strategies can be applied in order to accomplish this together with an appropriate model of the environment. Actual grasping of the cup in question and proper strategies for this are also based on these estimations and collected information, including grasp points for similar objects. This also includes data about affordances [62], such as fixed orientation constraints for containers with liquids that are important during transportation or parameters such as correct contact pressure. Different grasp types might be appropriate depending on the later form of interaction with the object. In case of a welding application, the task description would include data such as welding points/lines, entry cones, material properties, or exposure time (Fig. 1.5b). The actual program and execution of the task is dependent on the chosen robot (e.g., stationary manipulator, mobile platform, humanoid) and end effector type (e.g., gripper, welding torch). Additional constraints such as proper balance, preferred postures, or obstacle avoidance can be added accordingly [133].

1.3 Architectures for Cognitive Systems

In contrast to robot applications that only execute predetermined offline programs and feature no direct contact with human partners, interactive scenarios with human-robot collaboration require some form of artificial intelligence. Humans employ various forms of verbal and non-verbal communication during cooperation and are able to infer goals and intentions based on the actions of their partner and the context. A robot system in such a setting has to be able to perform in a similar fashion in order to be efficient and reliable. The system also has to be able to respond to new or unforeseen events in a robust manner.

Various approaches to this type of problem can be found, each with different benefits and drawbacks [141, 87, 142, 96, 88]. The traditional symbolic method uses explicit representations, where a number of symbols together with specific rules and a working memory is used in order to determine proper reactions to certain events. The symbols and rules have to be written and designed by a programmer and thus suffer from an inherent lack of completeness. This approach is often used for natural language processing and can represent complex relationships. On the other hand, the greatest strengths of sub-symbolic systems lie in their ability to handle inconsistent and noisy data together with autonomous learning. Similar to the concept of neurons in the brain, a collection of simple nodes in a network is used to generate decisions. The function of such a network can be trained using either supervised, unsupervised, or reinforcement learning, resulting in adjusted weights of the connections between

its nodes. Hybrid models try to combine the advantages of both methods in order to handle problems that cannot be addressed by one approach alone.

1.4 Contributions and Structure

This work focuses on the limitations of current programming and design concepts in the field of robotics. Modern engineering approaches and advanced task concepts are necessary in order to create modular systems that can be instructed with a more intuitive interaction style. These issues are addressed through the development of a more efficient motion planning algorithm that avoids exhaustive search and actively balances configuration space exploration and the exploitation of available information, an instruction paradigm that does not require expert knowledge in the robotics domain and is independent of a particular robot system, and an underlying open source framework that reduces the need of complex reengineering.

The robotics foundations in Chap. 2 describe important aspects in the engineering of these systems. They support the core components in a modular design and provide the basis for later concepts.

Chapter 3 begins with an overview of the current motion planning field and discusses various approaches and their strengths and limitations in the context of completeness versus efficiency. It introduces the concept of effective motion planning through a deliberate balance between exploration and exploitation. Experimental results demonstrate the improvements in planning performance with a variety of difficult scenarios.

Distributed architecture, task-based control, and the framework of Chap. 4 are demonstrated in two intuitive and modular systems for human-robot interaction. One scenario focuses on a hybrid model with symbolic dialog management, subsymbolic goal inference and error management, as well as multimodal input and output. The second system shows similar concepts in an industrial setting with a detailed look at the integration of sensor information and safety aspects.

Chapter 2
Robotics Foundations and Model Description

Before addressing topics for programming robot systems such as motion planning and task-driven control paradigms, a few basic principles need to be introduced. Many of these aspects are considered expert knowledge and build the core of all implementations in the robotics domain. They are vital elements in the design of abstract robot systems and easy access to this functionality should be provided. An understanding of these foundations and related design goals will help in later discussions. This chapter can be considered as a tutorial and is also often referred to in later parts of this thesis.

Traditional robot systems consist of a number of rigid bodies connected by a collection of joints that allow relative motion in between these elements [117, 89, 38, 137]. Such a model is defined by its geometric shapes, their corresponding mass and inertia, the placement of its links and joints compared to each other, and its placement in the surrounding environment. Position and orientation of the individual components is specified relative to a common world coordinate system or frame. The location of the robot with respect to this world reference is given by a transformation to its base frame, whereas the configuration of each joint provides displacements and rotations for the corresponding links relative to its parent coordinate system in the robot's structure. Many different devices such as grippers or cameras can usually be attached to the last element in a robot's structure of links, which is typically referred to as its end effector and describes the tool frame. Other frames can be introduced in order to describe objects and similar features in the scenario.

Serial chains, kinematic trees, and closed-loop systems are amongst the possible architectures of a robot system and the number of end effectors is directly related to the mechanical structure. The flexibility of an architecture is defined by the number of joints and their individual degrees of freedom (DOF), with categories including revolute, prismatic, helical, cylindrical, spherical, and 6-DOF joints. They are typically controlled using position, velocity, or torque commands and enclosed sensors can measure corresponding feedback values. A typical industrial manipulator is built using a serial chain of six joints with one end effector. The n total parameters in a system required to describe the robot's current configuration or posture span a joint

space of the same dimension. Its end effector's location in physical or operational space is defined by the position and orientation of the corresponding frame.

2.1 Kinematics and Dynamics

A description of the position, velocity, acceleration and related higher-order derivatives of a rigid body is handled by a robot's individual kinematics equations. In the most basic instance this deals with the task of calculating link frames based on the current joint configuration or the inverse problem of creating position variables matching an end effector's given target position and orientation. Mappings from joint to operational space and vice versa are required in order to create adequate trajectories for different types of applications [117, 89, 38, 48, 137].

Dynamics on the other hand focuses on the relationship of forces, joint torques, and the algorithms required to generate motion. A robot system with low-level access can typically not be given commands in the form of joint accelerations. Torques in the joint's motors have to counteract forces from gravity and friction while inducing the correct amount of control output that produces the desired trajectory. Translation of joint torques to accelerations on the other hand is useful in creating proper simulations of the system. This also includes proper handling of torques and forces between joint and operational space.

2.1.1 Position and Orientation

A coordinate system is defined by an origin and an orientation [117, 89, 38, 137]. Each of the two components requires three degrees of freedom for a complete description. In an n-dimensional Euclidean space, the origin is given by a vector $\boldsymbol{p} \in \mathbb{R}^n$, with $n = 3$ for the physical universe. Its elements p_x, p_y, and p_z define distances along the axes of a corresponding reference frame such that

$$\boldsymbol{p} = p_\mathrm{x}\,\hat{\boldsymbol{\imath}} + p_\mathrm{y}\,\hat{\boldsymbol{\jmath}} + p_\mathrm{z}\,\hat{\boldsymbol{k}} = \begin{bmatrix} p_\mathrm{x} & p_\mathrm{y} & p_\mathrm{z} \end{bmatrix}^\mathrm{T} , \tag{2.1}$$

where $\hat{\boldsymbol{\imath}}$, $\hat{\boldsymbol{\jmath}}$, and $\hat{\boldsymbol{k}}$ are unit vectors and versors corresponding to the axes of the Cartesian coordinate system. Similar descriptions can be provided by cylindrical or spherical coordinates. A vector ${}^{j}\boldsymbol{p}_i$ can be used in order to describe the position of the origin of a coordinate system i relative to a reference frame j. In case of frames with equal orientations, translations can be calculated using vector addition and multiple displacements can be combined in the same way:

$${}^{j}\boldsymbol{u} = {}^{i}\boldsymbol{u} + {}^{j}\boldsymbol{p}_i \tag{2.2}$$

$${}^{k}\boldsymbol{p}_i = {}^{j}\boldsymbol{p}_i + {}^{k}\boldsymbol{p}_j \,. \tag{2.3}$$

In projective space $\mathbb{RP}^n$, the corresponding homogeneous coordinate representation can be represented by a vector of dimension $n + 1$. This is achieved through the inclusion of an extra coordinate w that can also be viewed as a scaling factor and is usually set to 1:

$$\begin{bmatrix} wp_x & wp_y & wp_z & w \end{bmatrix}^T . \tag{2.4}$$

The special case for infinite direction vectors uses a value of 0 for the last element and the 4×1 null vector is undefined.

Orientation of a right-handed coordinate system in a space of dimension n can be defined by the special orthogonal group

$$\mathrm{SO}(n) = \left\{ \boldsymbol{R} : \boldsymbol{R} \in \mathbb{R}^{n \times n} , \ \boldsymbol{R}\boldsymbol{R}^T = \mathbf{1} , \ \det(\boldsymbol{R}) = +1 \right\} , \tag{2.5}$$

where SO(3) is the rotation group for three-dimensional space. With only three parameters necessary in order to describe the angular degrees of freedom in contrast to the nine elements of a matrix

$$\boldsymbol{R} = \begin{bmatrix} \hat{\boldsymbol{x}} & \hat{\boldsymbol{y}} & \hat{\boldsymbol{z}} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} , \tag{2.6}$$

its columns are composed of three mutually orthogonal unit vectors $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}$, and $\hat{\boldsymbol{z}}$, thereby imposing six independent constraints. This can also be verified when a proper orthonormal matrix is written in the form

$$\boldsymbol{R} = (\mathbf{1} - \boldsymbol{S})^{-1} (\mathbf{1} + \boldsymbol{S}) , \tag{2.7}$$

where $\boldsymbol{S}$ is a skew-symmetric matrix that can be described by only three parameters s_x, s_y, and s_z such that

$$\boldsymbol{S} = -\boldsymbol{S}^T = \begin{bmatrix} 0 & -s_z & s_y \\ s_z & 0 & -s_x \\ -s_y & s_x & 0 \end{bmatrix} \tag{2.8}$$

and whose transpose is equal to its negative. The components of a vector define its projection onto the unit directions of its reference frame. Therefore, the orientation of a frame i with respect to a frame j is described by the matrix

$${}^j\boldsymbol{R}_i = \begin{bmatrix} {}^j\hat{\boldsymbol{x}}_i & {}^j\hat{\boldsymbol{y}}_i & {}^j\hat{\boldsymbol{z}}_i \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{x}}_i \cdot \hat{\boldsymbol{x}}_j & \hat{\boldsymbol{y}}_i \cdot \hat{\boldsymbol{x}}_j & \hat{\boldsymbol{z}}_i \cdot \hat{\boldsymbol{x}}_j \\ \hat{\boldsymbol{x}}_i \cdot \hat{\boldsymbol{y}}_j & \hat{\boldsymbol{y}}_i \cdot \hat{\boldsymbol{y}}_j & \hat{\boldsymbol{z}}_i \cdot \hat{\boldsymbol{y}}_j \\ \hat{\boldsymbol{x}}_i \cdot \hat{\boldsymbol{z}}_j & \hat{\boldsymbol{y}}_i \cdot \hat{\boldsymbol{z}}_j & \hat{\boldsymbol{z}}_i \cdot \hat{\boldsymbol{z}}_j \end{bmatrix} , \tag{2.9}$$

where the elements are the dot products of each frame's unit vectors. The two reference frames can be interchanged by the inverse of this rotation matrix. Because it is a proper orthogonal matrix (2.5), its inverse is equal to its transpose:

$${}^i\boldsymbol{R}_j = {}^j\boldsymbol{R}_i^{-1} = {}^j\boldsymbol{R}_i^T . \tag{2.10}$$

For coordinate systems with the same origin, a rotation matrix can be used in order to describe a vector in a frame with different orientation:

$$^{j}\boldsymbol{u} = {}^{j}\boldsymbol{R}_{i}\,{}^{i}\boldsymbol{u}\,. \tag{2.11}$$

Multiple rotations can be combined through non-commutative matrix multiplication, such that the orientation of coordinate system i as seen from k is established by the matrix

$$^{k}\boldsymbol{R}_{i} = {}^{k}\boldsymbol{R}_{j}\,{}^{j}\boldsymbol{R}_{i}\,. \tag{2.12}$$

The rotation matrices about the three versors of the Cartesian coordinate system by an angle θ are given by

$$\boldsymbol{R}(\hat{\boldsymbol{\imath}},\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}, \tag{2.13}$$

$$\boldsymbol{R}(\hat{\boldsymbol{\jmath}},\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \tag{2.14}$$

$$\boldsymbol{R}(\hat{\boldsymbol{k}},\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.15}$$

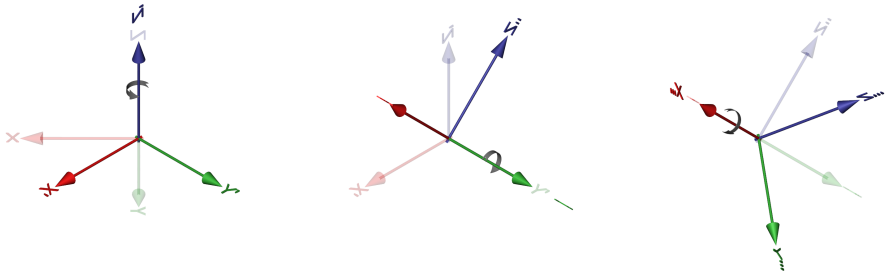
Any orientation in three-dimensional space can be described by properly combining three of these basic rotations in succession. Depending on the order of multiplication, these rotations are either applied to an axis of the original coordinate system or of a newly-created one in each step. The former method is often referred to as fixed angle representation, while the three parameters of the latter convention are known as Euler angles. In total, this leads to a number of twelve unique orders of rotation for the two approaches as each fixed angle version can be mapped to an equivalent Euler angle one and vice versa. The three angles ϕ, θ, and ψ in the X-Y-Z fixed angle representation

$$\boldsymbol{R}(\hat{\boldsymbol{k}},\psi)\,\boldsymbol{R}(\hat{\boldsymbol{\jmath}},\theta)\,\boldsymbol{R}(\hat{\boldsymbol{\imath}},\phi) \tag{2.16}$$

are sometimes called *yaw*, *pitch*, and *roll*. Due to the same order of multiplication, this series of rotation is identical to the one used in the Z-Y′-X″ Euler angle order (Fig. 2.1). Both conventions are subject to singularities when two of the three angles describe rotations around the same axis. The combined rotation matrix

$$\boldsymbol{R}(\hat{\boldsymbol{k}},\psi,\hat{\boldsymbol{\jmath}},\theta,\hat{\boldsymbol{\imath}},\phi) = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \tag{2.17}$$

equivalent to (2.16) shows that the singularity appears when the second rotation parameter θ approaches $\pm 90°$, leading to a loss of one degree of freedom ($\cos\theta$ and $\sin\theta$ are abbreviated as c_θ and s_θ respectively). This is also referred to as *gimbal*



(a) (b) (c)

Fig. 2.1 Series of rotations using Z-Y′-X″ Euler angles, identical to rotations using X-Y-Z fixed angles. **a** first rotation with 45° about original $\hat{\boldsymbol{z}}$ axis, **b** second rotation with −45° about new $\hat{\boldsymbol{y}}$ axis, **c** final rotation with −45° about new $\hat{\boldsymbol{x}}$ axis

lock and can be compensated when calculating angles from a rotation matrix by arbitrarily choosing one of the other two angles. This representation of rotations has the disadvantage that small changes in rotation can result in large angular changes in the proximity of singularities. Other common angle orders in robotics include Z-X′-Z″ and Z-Y′-Z″.

Unit quaternions [136, 97, 41] describe an alternative parameterization of rotation that can be used in order to prevent these issues. While they present an increase in performance and numerical stability, their main disadvantage however is that four parameters are now required in order to describe three angular degrees of freedom. In a robot system with spherical joints such as free-flying robots this has to be considered in some way. A quaternion $\boldsymbol{h} \in \mathbb{H}$ with a real part h_0 and an imaginary part h_1, h_2, and h_3 is defined in an extension of complex numbers as

$$\boldsymbol{h} = h_0 + h_1\,\mathrm{i} + h_2\,\mathrm{j} + h_3\,\mathrm{k}\,, \tag{2.18}$$

with

$$\mathrm{i}^2 = \mathrm{j}^2 = \mathrm{k}^2 = \mathrm{i}\,\mathrm{j}\,\mathrm{k} = -1\,. \tag{2.19}$$

This is similar to the definition of vectors in Euclidean space (2.1). In fact, these vectors can also be represented in quaternion notation by defining a value of 0 for the h_0 component. Every quaternion $\boldsymbol{h}$ has a conjugate

$$\bar{\boldsymbol{h}} = h_0 - h_1\,\mathrm{i} - h_2\,\mathrm{j} - h_3\,\mathrm{k}\,, \tag{2.20}$$

such that

$$\boldsymbol{h}\bar{\boldsymbol{h}} = \bar{\boldsymbol{h}}\boldsymbol{h}\,, \tag{2.21}$$

with its norm $\|\boldsymbol{h}\|$ given by the corresponding square root and the result of this product being equal to 1 for unit quaternions. Quaternions are often described in $\mathbb{R}^4$, with a scalar $w \in \mathbb{R}$ equal to the real part and a vector $\boldsymbol{v} \in \mathbb{R}^3$ consisting of the imaginary components. Addition is then defined by adding the respective scalar

and vector parts, whereas the multiplication of two quaternions $\boldsymbol{h}_1$ and $\boldsymbol{h}_2$ is more complex and not commutative:

$$\boldsymbol{h}_1 \boldsymbol{h}_2 = (w_1, \boldsymbol{v}_1)(w_2, \boldsymbol{v}_2) = (w_1 w_2 - \boldsymbol{v}_1 \cdot \boldsymbol{v}_2, w_1 \boldsymbol{v}_2 + w_2 \boldsymbol{v}_1 + \boldsymbol{v}_1 \times \boldsymbol{v}_2) \,. \quad (2.22)$$

Rotation of a vector is accomplished by multiplication with a quaternion on its right and the conjugate on its left or vice versa for the inverse operation:

$$^{j}\boldsymbol{u} = {}^{j}\boldsymbol{h}_i \, {}^{i}\boldsymbol{u} \, {}^{j}\bar{\boldsymbol{h}}_i \quad (2.23)$$

$$^{i}\boldsymbol{u} = {}^{j}\bar{\boldsymbol{h}}_i \, {}^{j}\boldsymbol{u} \, {}^{j}\boldsymbol{h}_i \,, \quad (2.24)$$

although this representation is not unique. A rotation about a quaternion $\boldsymbol{h}$ is identical to the one with $-\boldsymbol{h}$ due to the fact that a change in orientation about an angle θ and an axis $\hat{\boldsymbol{n}}$ is equal to the same operation with its negated counterparts. Angle and axis description can be converted to a unit quaternion

$$\boldsymbol{h} = (w, \boldsymbol{v}) = \left(\cos(\tfrac{1}{2}\theta), \hat{\boldsymbol{n}} \sin(\tfrac{1}{2}\theta)\right), \quad (2.25)$$

where its components are then referred to as Euler parameters. Furthermore a unit quaternion can be mapped to a rotation matrix

$$\boldsymbol{R}(\boldsymbol{h}) = \begin{bmatrix} 1 - 2\,(h_2^2 + h_3^2) & 2(h_1\,h_2 + h_3\,h_0) & 2\,(h_1\,h_3 - h_2\,h_0) \\ 2\,(h_1\,h_2 - h_3\,h_0) & 1 - 2\,(h_1^2 + h_3^2) & 2\,(h_2\,h_3 + h_1\,h_0) \\ 2\,(h_1\,h_3 + h_2\,h_0) & 2\,(h_2\,h_3 - h_1\,h_0) & 1 - 2\,(h_1^2 + h_2^2) \end{bmatrix}, \quad (2.26)$$

which can be shown to fulfill all requirements of SO(3) and that can also be translated back to quaternion elements without any issues. Direct multiplication by quaternions is however more efficient, as fewer operations are required compared to traditional 3×3 matrices.

For coordinate systems where origin as well as orientation of two frames differs, a rotation (2.11) is always followed by a translation (2.2) in this representation. Both have to be performed in order to describe a vector in frame i relative to frame j:

$$^{j}\boldsymbol{u} = {}^{j}\boldsymbol{R}_i \, {}^{i}\boldsymbol{u} + {}^{j}\boldsymbol{p}_i \quad (2.27)$$

These two operations can however be combined in the special Euclidean group

$$\mathrm{SE}(n) = \left\{ \begin{bmatrix} \boldsymbol{R} & \boldsymbol{p} \\ \boldsymbol{0} & 1 \end{bmatrix} : \boldsymbol{R} \in \mathrm{SO}(n) \,, \quad \boldsymbol{p} \in \mathbb{R}^n \right\}, \quad (2.28)$$

with a homogeneous transformation $\boldsymbol{T} \in \mathrm{SE}(3)$ for three-dimensional space. Its components $\boldsymbol{R}$ and $\boldsymbol{p}$ may be chosen independent from each other and can also be assigned an identity matrix or a null vector, thereby resulting in no rotation or translation respectively. Computation with separated translation and rotation parts is however more efficient. The same vector in homogeneous coordinate representation (2.4) can now be transformed between the two frames by a single multiplication

with the matrix

$$^{j}\boldsymbol{T}_{i} = \begin{bmatrix} ^{j}\boldsymbol{R}_{i} & ^{j}\boldsymbol{p}_{i} \\ \boldsymbol{0} & 1 \end{bmatrix} . \tag{2.29}$$

Scaling and perspective transformations are defined by similar 4×4 matrices. Origin and orientation of frame j relative to frame i are given by the inverse of this matrix, resulting in

$$^{i}\boldsymbol{T}_{j} = {^{j}\boldsymbol{T}_{i}}^{-1} = \begin{bmatrix} ^{j}\boldsymbol{R}_{i}^{\mathrm{T}} & -^{j}\boldsymbol{R}_{i}^{\mathrm{T}}\,^{j}\boldsymbol{p}_{i} \\ \boldsymbol{0} & 1 \end{bmatrix} \tag{2.30}$$

and multiple transformations can be composed through non-commutative multiplication such that

$$^{k}\boldsymbol{T}_{i} = {^{k}\boldsymbol{T}_{j}}\,{^{j}\boldsymbol{T}_{i}} . \tag{2.31}$$

This can for instance be used when expressing the position and orientation of a robot manipulator's end effector—in this case a serial chain with n degrees of freedom and a gripper—relative to its world frame in a homogeneous transformation

$$^{\text{world}}\boldsymbol{T}_{\text{tool}} = {^{\text{world}}\boldsymbol{T}_{\text{base}}}\,{^{\text{base}}\boldsymbol{T}_{0}}\,{^{0}\boldsymbol{T}_{1}}\,\cdots\,{^{n-1}\boldsymbol{T}_{n}}\,{^{n}\boldsymbol{T}_{\text{tool}}} , \tag{2.32}$$

with $^{\text{base}}\boldsymbol{T}_{0}$ as the mapping from base frame to the first joint coordinate system, matrices $^{0}\boldsymbol{T}_{1}, \ldots, {}^{n-1}\boldsymbol{T}_{n}$ representing the individual transformations by each of the robot's joints in their current configuration and a description of the currently attached tool $^{n}\boldsymbol{T}_{\text{tool}}$. Such a calculation is commonly referred to as forward position kinematics.

2.1.2 Metric Spaces and Interpolation

Distance in Euclidean space can be visualized by connecting two points with a straight line and measuring the respective length. Planar map representations of the surface of the earth have borders and the western and eastern sections on the other hand appear to be disconnected, while the actual regions on its sphere are close by. These cases can be described and applied to other scenarios such as kinematic configurations with the more general concepts of topological and metric spaces.

A topological space is a set X together with a collection of open subsets for which both the empty set and X are open subsets, as well as the union of any number of open subsets and the intersection of a finite number of open subsets [97, 99]. In case such a space locally resembles the n-dimensional Euclidean space, it is known as a manifold of the same dimension. This property is similar to that of the earth's surface which—while actually round—appears to be flat for an observer standing on top of it. Two manifolds are considered to be identical when they belong to the same homotopy class, which means they can be deformed into each other. A well-known example for this is the surface of a coffee cup with a handle, which is in the same homotopy class as the one from a donut shaped object. The most basic example of a one-dimensional manifold is given in the form of $\mathbb{R}^1$. It can for example be used

for a prismatic or revolute joint with upper and lower bounds. Multiple homeomorphic manifolds can be combined, hence the space of $\mathbb{R}^1 \times \mathbb{R}^1 \times \mathbb{R}^1$ is equal to the Euclidean space $\mathbb{R}^3$. A number of manifolds that are not homeomorphic to $\mathbb{R}^1$ are given by a circle and its more general parameterization in the form of n-spheres

$$\mathbb{S}^n = \left\{ x \in \mathbb{R}^{n+1} : \|x\| = 1 \right\}, \tag{2.33}$$

where a circle is the one-dimensional case. In particular, the circle representation is important for revolute joints with no limits that cannot be represented by $\mathbb{R}^1$, whereas the latter is the correct behavior for these joints under restricted angles. Without this wraparound, the calculated distance between two joint configurations might be too large. Quaternions and in the same way true three-dimensional rotations belong to the $\mathbb{S}^3$ class, but because the n-sphere manifolds are not homeomorphic to each other, this representation is not identical to the $\mathbb{S}^1 \times \mathbb{S}^1 \times \mathbb{S}^1$ manifold described by Euler angles. Although rotations with quaternions are not unique due to the negated angle and axis problematic (2.25), this can be avoided by choosing the real projective space $\mathbb{RP}^3$ instead, which is the set of all lines in three-dimensional space passing through the origin. Every possible orientation is thus counted only once and the correct manifold for a free-flying robot is given by $\mathbb{R}^3 \times \mathbb{RP}^3$. For spherical joints with angular limitations this is replaced by combinations of $\mathbb{R}^1$ and $\mathbb{S}^1$ respectively.

A metric space has a global distance function that returns the distance between two points as a non-negative real number and must satisfy several conditions: the distance between equal points is returned as zero, the two points must be interchangeable, and it must fulfill the triangle inequality [97, 93, 99]. For the Euclidean space, where this value is given by the length of the straight line connecting these points, this function is defined by the Euclidean distance

$$\mathrm{d}_2(\boldsymbol{u}, \boldsymbol{v}) = \|\boldsymbol{u} - \boldsymbol{v}\|_2 = \sqrt{(u_1 - v_1)^2 + \ldots + (u_n - v_n)^2} \tag{2.34}$$

and the associated norm is referred to as Euclidean norm. An alternative metric that uses the absolute sum of the differences of each coordinate axis is given by the Manhattan distance

$$\mathrm{d}_1(\boldsymbol{u}, \boldsymbol{v}) = \|\boldsymbol{u} - \boldsymbol{v}\|_1 = |u_1 - v_1| + \ldots + |u_n - v_n| \tag{2.35}$$

and its respective norm. This variant is similar to moving along a grid layout. These metrics and norms can also be generalized for $\mathbb{R}^n$ in the form of

$$\|\boldsymbol{u}\|_p = \left(\sum_{i=1}^{n} |u_i|^p \right)^{1/p}. \tag{2.36}$$

A distance metric for a circle representation in $\mathbb{S}^1$ cannot use the previous functions, as they would not correctly relate to the length of the traveled distance. An adjusted approach can be based on the respective angles θ_1 and θ_2 of the two points instead; however it has to consider the fact that there are two possible solutions and should

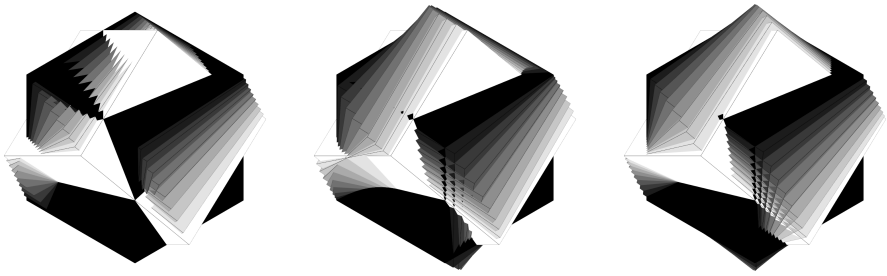


Fig. 2.2 Interpolation of a cube's orientation, rotated 45° in each world axis. The 10 individual steps are visualized from black to white. **a** linear with matrix representation, **b** linear with Euler angles, **c** spherical linear with quaternions

thus always select the smallest one:

$$\mathrm{d}_{\mathbb{S}^1}(\theta_1, \theta_2) = \min\bigl(\|\theta_1 - \theta_2\|, 2\pi - \|\theta_1 - \theta_2\|\bigr)\,. \tag{2.37}$$

While this can be used together with Euler angles in order to compute an approximated distance for three-dimensional orientations, a proper distance metric for $\mathbb{RP}^3$ should rather be based on quaternions. With Euler angle representations, there are multiple solutions for a single rotation and the separate angle components might not represent the actual overall difference. As with the circle example, there are also two possible paths in $\mathbb{S}^3$ that have to be compensated for due to quaternions not being unique:

$$\mathrm{d}_{\mathbb{S}^3}(\boldsymbol{h}_1, \boldsymbol{h}_2) = \min\bigl(\|\boldsymbol{h}_1 - \boldsymbol{h}_2\|, \|\boldsymbol{h}_1 + \boldsymbol{h}_2\|\bigr)\,. \tag{2.38}$$

For unit quaternions, the angle of rotation θ between them is defined by the equation

$$\theta = \arccos(\boldsymbol{h}_1 \cdot \boldsymbol{h}_2)\,, \tag{2.39}$$

leading to the alternate representation

$$\mathrm{d}_{\mathbb{S}^3}(\boldsymbol{h}_1, \boldsymbol{h}_2) = \min\bigl(\arccos(\boldsymbol{h}_1 \cdot \boldsymbol{h}_2), \arccos(\boldsymbol{h}_1 \cdot -\boldsymbol{h}_2)\bigr)\,. \tag{2.40}$$

In a distance metric with translational and rotational components, a proper weight factor should be introduced in order to balance the different units.

Interpolation within a range $t \in [0, 1]$ between two vectors that form a straight line in space can be achieved by the linear interpolation operation

$$\mathrm{lerp}(\boldsymbol{u}_1, \boldsymbol{u}_2, t) = (1 - t)\,\boldsymbol{u}_1 + t\boldsymbol{u}_2\,. \tag{2.41}$$

This procedure can however not directly be applied to three-dimensional rotations [136, 41]. The linear interpolation of the individual elements of a rotation matrix leads to undesired results, as the implicit constraints of the orthonormal matrix are not considered. As a 3×3 matrix can also be used in order to represent

transformations such as scaling, this operation leads to the effects seen in Fig. 2.2a, where a cube's orientation is interpolated between two states. The intermediate steps are furthermore not equally distributed as is the case with position vectors. Linear interpolation of three Euler angles (Fig. 2.2b) is not suited for this kind of application as well. There is no direct relationship between the individual angles and the overall rotation, even with proper circle metrics. Similar to the matrix variant, the angular velocity in this procedure does not stay constant throughout the calculation. Applying this interpolation method directly to quaternions does not result in satisfying results either because the intermediate steps do not produce proper unit quaternions. It can actually be seen as a straight line connection between the two points rather than a proper path on the surface of the corresponding unit sphere. While normalizing these points results in the best solution so far, its velocity profile still suffers from the aforementioned shortcut. The spherical linear interpolation

$$\mathrm{slerp}(\boldsymbol{h}_1, \boldsymbol{h}_2, t) = \frac{\sin\big((1-t)\,\theta\big)}{\sin\theta}\,\boldsymbol{h}_1 + \frac{\sin(t\theta)}{\sin\theta}\,\boldsymbol{h}_2 \tag{2.42}$$

on the other hand divides the angle θ between the two orientations into equal sections and therefore follows a great arc on the unit sphere (Fig. 2.2c). It features a constant angular velocity throughout the whole interpolation range and can also be written in the form

$$\mathrm{slerp}(\boldsymbol{h}_1, \boldsymbol{h}_2, t) = \boldsymbol{h}_1\,(\bar{\boldsymbol{h}}_1\,\boldsymbol{h}_2)^t = \boldsymbol{h}_1^{1-t}\,\boldsymbol{h}_2^t\,. \tag{2.43}$$

As orientation descriptions with quaternions are not unique, the shortest possible interpolation path is given by calculating the minimal angular distance between the two quaternions (2.40) and selecting the proper negated representation when it becomes necessary.

2.1.3 Velocity and Force

A robot is a moving system composed of several joints and links. Besides a static form of position and orientation, it therefore becomes necessary to be able to express velocities and accelerations as well as static forces and moments in order to describe movement and the effect of external influences [117, 89, 38, 137].

Linear and angular velocities are given by differentiation as

$$\dot{\boldsymbol{p}} = \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{\boldsymbol{p}(t+\Delta t) - \boldsymbol{p}(t)}{\Delta t} \tag{2.44}$$

$$\dot{\boldsymbol{R}} = \frac{\mathrm{d}\boldsymbol{R}}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{\boldsymbol{R}(t+\Delta t) - \boldsymbol{R}(t)}{\Delta t}\,. \tag{2.45}$$

If the frame in which the linear velocity vector is described differs from the one in which it is differentiated, this change in reference can be described by including

a rotation matrix similar to (2.11). While a constant position vector can likewise be transformed between two coordinate systems i and j with the same origin, this is not the case if the latter frame is rotating with an angular velocity. A look at the properties of an orthonormal rotation matrix shows that the product with its transpose is equal to the identity matrix (2.5) and by differentiation together with the definition of skew-symmetric matrices (2.8) this results in

$$\dot{\boldsymbol{R}}\boldsymbol{R}^{\mathrm{T}} + \boldsymbol{R}\dot{\boldsymbol{R}}^{\mathrm{T}} = \dot{\boldsymbol{R}}\boldsymbol{R}^{\mathrm{T}} + (\dot{\boldsymbol{R}}\boldsymbol{R}^{\mathrm{T}})^{\mathrm{T}} = \boldsymbol{S} + \boldsymbol{S}^{\mathrm{T}} = \boldsymbol{0}\,. \tag{2.46}$$

The derivative of a constant vector moving with an angular velocity is thus given by

$${}^{j}\boldsymbol{v} = {}^{j}\dot{\boldsymbol{p}} = {}^{j}\dot{\boldsymbol{R}}_i\,{}^{i}\boldsymbol{p} = {}^{j}\dot{\boldsymbol{R}}_i\,{}^{j}\boldsymbol{R}_i^{\mathrm{T}}\,{}^{j}\boldsymbol{p} = {}^{j}\boldsymbol{S}_i\,{}^{j}\boldsymbol{p}\,. \tag{2.47}$$

In a different interpretation, the $\sin\theta$ and $\cos\theta$ values in (2.13), (2.14), and (2.15) respectively approach $\mathrm{d}\theta$ and 1 for the differential rotation $\mathrm{d}\theta$. With the X-Y-Z fixed angle representation (2.17), this also results in the skew-symmetric matrix definition and is often referred to as angular velocity matrix. In this case, its three parameters describe the angular velocity vector $\boldsymbol{\omega}$ and together with the description of linear velocity $\boldsymbol{v}$ the two can now be defined as

$$\boldsymbol{v} = v_{\mathrm{x}}\,\hat{\boldsymbol{\imath}} + v_{\mathrm{y}}\,\hat{\boldsymbol{\jmath}} + v_{\mathrm{z}}\,\hat{\boldsymbol{k}} \tag{2.48}$$

$$\boldsymbol{\omega} = \omega_{\mathrm{x}}\,\hat{\boldsymbol{\imath}} + \omega_{\mathrm{y}}\,\hat{\boldsymbol{\jmath}} + \omega_{\mathrm{z}}\,\hat{\boldsymbol{k}}\,. \tag{2.49}$$

Multiplication of a screw-symmetric matrix with a vector is equal to the cross product with a vector of its three defining components. The description of linear and angular velocities can thus be transformed between two frames with different origins and orientations by the equations

$${}^{j}\boldsymbol{v} = {}^{j}\boldsymbol{R}_i\,({}^{i}\boldsymbol{v} + {}^{i}\boldsymbol{\omega} \times {}^{i}\boldsymbol{p}_j) \tag{2.50}$$

$${}^{j}\boldsymbol{\omega} = {}^{j}\boldsymbol{R}_i\,{}^{i}\boldsymbol{\omega}\,. \tag{2.51}$$

When both linear and angular velocity are combined into a single vector of size 6×1, this relationship can also be written in the form

$$\begin{bmatrix} {}^{j}\boldsymbol{\omega} \\ {}^{j}\boldsymbol{v} \end{bmatrix} = {}^{j}\boldsymbol{X}_i \begin{bmatrix} {}^{i}\boldsymbol{\omega} \\ {}^{i}\boldsymbol{v} \end{bmatrix}. \tag{2.52}$$

Considering the fact that a cross product between two vectors is equivalent to a skew-symmetric matrix multiplication this gives

$${}^{j}\boldsymbol{X}_i = \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} \\ \boldsymbol{S}({}^{j}\boldsymbol{p}_i) & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} {}^{j}\boldsymbol{R}_i & \boldsymbol{0} \\ \boldsymbol{0} & {}^{j}\boldsymbol{R}_i \end{bmatrix} = \begin{bmatrix} {}^{j}\boldsymbol{R}_i & \boldsymbol{0} \\ \boldsymbol{S}({}^{j}\boldsymbol{p}_i)\,{}^{j}\boldsymbol{R}_i & {}^{j}\boldsymbol{R}_i \end{bmatrix} \tag{2.53}$$

and its inverse

$${}^{j}\boldsymbol{X}_i^{-1} = {}^{i}\boldsymbol{X}_j = \begin{bmatrix} {}^{i}\boldsymbol{R}_j & \boldsymbol{0} \\ -{}^{i}\boldsymbol{R}_j\,\boldsymbol{S}({}^{j}\boldsymbol{p}_i) & {}^{i}\boldsymbol{R}_j \end{bmatrix}. \tag{2.54}$$

Multiple of these transformations can be combined through multiplication such that

$$ {}^{k}\boldsymbol{X}_{i} = {}^{k}\boldsymbol{X}_{j}\, {}^{j}\boldsymbol{X}_{i} \,. \tag{2.55} $$

Through further differentiation, the corresponding linear and angular acceleration definitions are formulated similar to (2.44) and (2.45). These can also be expressed in different coordinate systems by

$$ {}^{j}\dot{\boldsymbol{v}} = {}^{j}\boldsymbol{R}_{i} \left({}^{i}\dot{\boldsymbol{v}} + {}^{i}\dot{\boldsymbol{\omega}} \times {}^{i}\boldsymbol{p}_{j} + 2\, {}^{i}\boldsymbol{\omega} \times {}^{i}\boldsymbol{v} + {}^{i}\boldsymbol{\omega} \times ({}^{i}\boldsymbol{\omega} \times {}^{i}\boldsymbol{p}_{j}) \right) \tag{2.56} $$

$$ {}^{j}\dot{\boldsymbol{\omega}} = {}^{j}\boldsymbol{R}_{i}\, {}^{i}\dot{\boldsymbol{\omega}} + {}^{j}\boldsymbol{R}_{i}\, {}^{i}\boldsymbol{\omega} \times {}^{j}\boldsymbol{\omega} \,. \tag{2.57} $$

A connection between velocities in operational space and the velocities in the joints of a robot system can be established through partial differentiation of the forward position kinematics equations with respect to time [38, 137, 30]. A Jacobian matrix $\boldsymbol{J}$ is a multidimensional representation of such a derivative that can transform velocities between the two domains relative to a chosen reference frame. For a serial chain with n degrees of freedom and one end effector with six combined linear and angular velocity terms this results in a $6 \times n$ matrix. A vector of joint velocities $\dot{\boldsymbol{q}}$ at a current joint configuration $\boldsymbol{q}$ can thus be mapped to a vector of combined linear and angular velocities $\dot{\boldsymbol{x}}$ relative to a coordinate system i by

$$ {}^{i}\dot{\boldsymbol{x}} = {}^{i}\boldsymbol{J}_{n}(\boldsymbol{q})\, \dot{\boldsymbol{q}} \,, \tag{2.58} $$

where n represents the reference frame associated with the end effector. This operation is commonly referred to as forward velocity kinematics. Changing the reference coordinate system in which a Jacobian is expressed can be achieved with a 6×6 matrix in the following way:

$$ {}^{j}\boldsymbol{J}_{n}(\boldsymbol{q}) = \begin{bmatrix} {}^{j}\boldsymbol{R}_{i} & \boldsymbol{0} \\ \boldsymbol{0} & {}^{j}\boldsymbol{R}_{i} \end{bmatrix} {}^{i}\boldsymbol{J}_{n}(\boldsymbol{q}) \,. \tag{2.59} $$

In a similar fashion, the Jacobian defining the velocity in a different frame can be calculated according to (2.52). By differentiation, a similar correlation between joint accelerations $\ddot{\boldsymbol{q}}$ and end effector accelerations $\ddot{\boldsymbol{x}}$ can be derived with

$$ \ddot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\, \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\, \dot{\boldsymbol{q}} \,, \tag{2.60} $$

also known as forward acceleration kinematics. Systems with multiple end effectors are described by additional rows in the Jacobian. Depending on the number of end effectors and their operational degrees of freedom this gives the general form of a $m \times n$ matrix, where m is usually a multiple of 6. The opposite velocity relationship—or inverse velocity kinematics—is described by the inverse of the Jacobian such that

$$ \dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}(\boldsymbol{q})\, \dot{\boldsymbol{x}} \,, \tag{2.61} $$

however this leads to problems in cases where the matrix is not invertible. These configurations are referred to as kinematic singularities and can appear at multiple

locations in a robot's workspace. At least one degree of freedom at the end effector is lost in positions that can occur either close to the boundaries of the workspace or whenever multiple joint axes line up. A rating of the closeness to a singularity is given in form of the manipulability measure

$$\mu(\boldsymbol{q}) = \sqrt{\det\bigl(\boldsymbol{J}(\boldsymbol{q})\,\boldsymbol{J}^{\mathrm{T}}(\boldsymbol{q})\bigr)}\,, \tag{2.62}$$

that describes the volume of a manipulability ellipsoid. A Jacobian can also be written in singular value decomposition (SVD) form as $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$ with a matrix $\boldsymbol{U}$ of m orthonormal output basis vectors $\boldsymbol{u}$, a matrix $\boldsymbol{\Sigma}$ of m singular values σ, and a matrix $\boldsymbol{V}$ of n orthonormal input vectors $\boldsymbol{v}$. The vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m$ also represent the eigenvectors of the product of this Jacobian with its transpose. A velocity ellipsoid's principal axes are therefore defined by the directions of these vectors and their length is given by the corresponding singular values. The manipulability measure can thus also be calculated by a product of all singular values. A traditional Moore-Penrose pseudoinverse $\boldsymbol{J}^{+}$ based on SVD has stability issues in the vicinity of singularities and should be replaced by other methods such as damped least squares. Together with a damping factor λ, they provide a numerically stable solution

$$\boldsymbol{J}^{\dagger}(\boldsymbol{q}) = \boldsymbol{J}^{\mathrm{T}}(\boldsymbol{q})\left(\boldsymbol{J}(\boldsymbol{q})\,\boldsymbol{J}^{\mathrm{T}}(\boldsymbol{q}) + \lambda^2\,\mathbf{1}\right)^{-1} \tag{2.63}$$

that becomes equal to (2.61) when this variable is set to 0. This factor can also be used in a matrix inversion based on SVD for an alternative representation

$$\boldsymbol{J}^{\dagger}(\boldsymbol{q}) = \sum_{i=1}^{r} \frac{\sigma_i}{\sigma_i^2 + \lambda^2}\,\boldsymbol{v}_i\,\boldsymbol{u}_i^{\mathrm{T}}\,, \tag{2.64}$$

where only the singular values greater zero up to rank r are included. For the non-singular case and a zero damping factor this method does not change the robot's behavior. Another difficulty remains in choosing an appropriate λ that is determined by the distance to a singularity, however a solution is naturally provided by the value of the smallest singular value $\sigma_{\min}$ compared to a specified threshold ε with

$$\lambda^2 = \begin{cases} 0 & \sigma_{\min} \geq \varepsilon \\ \left(1 - (\sigma_{\min}/\varepsilon)^2\right)\lambda_{\max}^2 & \sigma_{\min} < \varepsilon \end{cases} \tag{2.65}$$

and an upper bound $\lambda_{\max}$ occurring at a singularity. High values of the damping factor result in low tracking accuracy in all directions and additional filtering based on the responsible components may be applied.

Besides linear and angular velocities, static forces and moments as well as their relationship to joint torques also have to be considered. These properties can be described in vector notation with respect to a specified reference frame through definitions of a static force vector $\boldsymbol{f}$ and a moment vector $\boldsymbol{n}$. Analog to (2.48–2.49) and (2.50–2.51), these quantities can be described relative to a coordinate system with different origin and orientation by the formulas

$$^{j}\boldsymbol{f} = {}^{j}\boldsymbol{R}_i \, {}^{i}\boldsymbol{f} \tag{2.66}$$

$$^{j}\boldsymbol{n} = {}^{j}\boldsymbol{R}_i \, ({}^{i}\boldsymbol{n} + {}^{i}\boldsymbol{f} \times {}^{i}\boldsymbol{p}_j) \tag{2.67}$$

or the equivalent and convenient matrix transformation

$$\begin{bmatrix} {}^{j}\boldsymbol{n} \\ {}^{j}\boldsymbol{f} \end{bmatrix} = {}^{j}\boldsymbol{X}_i^{-\mathrm{T}} \begin{bmatrix} {}^{i}\boldsymbol{n} \\ {}^{i}\boldsymbol{f} \end{bmatrix} . \tag{2.68}$$

In accordance with (2.53–2.54), this presentation with its forward and inverse operations is defined as

$${}^{j}\boldsymbol{X}_i^{-\mathrm{T}} = \begin{bmatrix} {}^{j}\boldsymbol{R}_i & \boldsymbol{S}({}^{j}\boldsymbol{p}_i) \, {}^{j}\boldsymbol{R}_i \\ \boldsymbol{0} & {}^{j}\boldsymbol{R}_i \end{bmatrix} = {}^{i}\boldsymbol{X}_j^{\mathrm{T}} \tag{2.69}$$

and its matching counterpart

$${}^{j}\boldsymbol{X}_i^{\mathrm{T}} = {}^{i}\boldsymbol{X}_j^{-\mathrm{T}} = \begin{bmatrix} {}^{i}\boldsymbol{R}_j & -{}^{i}\boldsymbol{R}_j \, \boldsymbol{S}({}^{j}\boldsymbol{p}_i) \\ \boldsymbol{0} & {}^{i}\boldsymbol{R}_j \end{bmatrix} . \tag{2.70}$$

The connection between the combined static forces and moments in operational space $\vec{\boldsymbol{f}}$ to an n-dimensional vector of joint torques $\boldsymbol{\tau}$ is realized through the transpose of the Jacobian matrix:

$$\boldsymbol{\tau} = \boldsymbol{J}^{\mathrm{T}}(\boldsymbol{q}) \, \vec{\boldsymbol{f}} \tag{2.71}$$

$$\vec{\boldsymbol{f}} = \boldsymbol{J}^{-\mathrm{T}}(\boldsymbol{q}) \, \boldsymbol{\tau} . \tag{2.72}$$

Resembling the issues in the position domain, the inverted operation that maps joint torques to operational space forces also suffers from singularities. The force manipulability ellipsoid's principal axes are identical to the ones used in the velocity variant, the axes lengths however are defined by the reciprocals of the singular values instead.

Transformation of velocities and static forces between frames and their relation to joint velocities and torques is only part of the process of achieving motion in a robot system. In order to create appropriate joint torques for following a trajectory described by joint positions, velocities, and accelerations, a relation between the forces and moments exerted by a robot's links and the joints to which they are connected needs to be established. With this in mind, the force $\boldsymbol{f}$ exerted by the acceleration $\dot{\boldsymbol{v}}$ of a rigid body with mass m at its center of mass is given by Newton's equation as

$$\boldsymbol{f} = m\dot{\boldsymbol{v}} , \tag{2.73}$$

whereas the moment necessary for creating a rigid body's angular velocity $\boldsymbol{\omega}$ and acceleration $\dot{\boldsymbol{\omega}}$ is defined by Euler's equation

$$\boldsymbol{n} = \boldsymbol{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega} . \tag{2.74}$$

The 3×3 matrix $\boldsymbol{I}$ describes the inertia of the rigid body at its center of mass. It characterizes the rigid body's distribution of mass along the corresponding coordinate system's axes such that

$$\boldsymbol{I} = \begin{bmatrix} i_{\mathrm{xx}} & -i_{\mathrm{xy}} & -i_{\mathrm{xz}} \\ -i_{\mathrm{xy}} & i_{\mathrm{yy}} & -i_{\mathrm{yz}} \\ -i_{\mathrm{xz}} & -i_{\mathrm{yz}} & i_{\mathrm{zz}} \end{bmatrix} . \tag{2.75}$$

The general equation of inverse dynamics in which an n-dimensional vector of joint torques $\boldsymbol{\tau}$ is related to a trajectory's description in form of joint positions $\boldsymbol{q}$, velocities $\dot{\boldsymbol{q}}$, and accelerations $\ddot{\boldsymbol{q}}$ is given by

$$\boldsymbol{\tau} = \boldsymbol{H}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{b}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) , \tag{2.76}$$

where $\boldsymbol{H}$ represents the $n \times n$ mass matrix with respect to the current joint settings. The centrifugal and Coriolis terms at this position and velocity are written in form of a $n \times 1$ vector $\boldsymbol{b}$, while the n-dimensional vector $\boldsymbol{g}$ describes the influence of gravity on the current posture. This formulation however does not include any compensation for joint friction. The inverse relationship

$$\ddot{\boldsymbol{q}} = \boldsymbol{H}^{-1}(\boldsymbol{q})\left(\boldsymbol{\tau} - \boldsymbol{b}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{g}(\boldsymbol{q})\right) \tag{2.77}$$

can be used in order to simulate the motion that would result from applying specific joint torques at a current state. In combination with integration formulas such as the Runge-Kutta methods or DASSL (Differential Algebraic System Solver) [118], this forward dynamics equation can for instance be used in visualization applications. It is also useful to apply this principle to operational space in order to be able to describe paths and forces in this domain. In accordance with (2.76), the same connection between $m \times 1$ vectors of operational forces $\vec{\boldsymbol{f}}$ and acceleration $\ddot{\boldsymbol{x}}$ can be specified as

$$\vec{\boldsymbol{f}} = \boldsymbol{\Lambda}(\boldsymbol{q})\,\ddot{\boldsymbol{x}} + \boldsymbol{\mu}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{\rho}(\boldsymbol{q}) , \tag{2.78}$$

with an operational mass matrix $\boldsymbol{\Lambda}$ of size $m \times m$, an m-dimensional vector of operational centrifugal and Coriolis terms $\boldsymbol{\mu}$, and the operational gravity component $\boldsymbol{\rho}$ as a vector of dimension $m \times 1$. By applying (2.71) this can be rewritten in the form

$$\boldsymbol{\tau} = \boldsymbol{J}^{\mathrm{T}}(\boldsymbol{q})\left(\boldsymbol{\Lambda}(\boldsymbol{q})\,\ddot{\boldsymbol{x}} + \boldsymbol{\mu}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{\rho}(\boldsymbol{q})\right) , \tag{2.79}$$

and the various operational space components can be described in terms of their joint space counterparts when combining this property with (2.58) and (2.60), thereby resulting in the expressions

$$\boldsymbol{\Lambda}(\boldsymbol{q}) = \boldsymbol{J}^{-\mathrm{T}}(\boldsymbol{q})\,\boldsymbol{H}(\boldsymbol{q})\,\boldsymbol{J}^{-1}(\boldsymbol{q}) \tag{2.80}$$

$$\boldsymbol{\mu}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{J}^{-\mathrm{T}}(\boldsymbol{q})\left(\boldsymbol{b}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{H}(\boldsymbol{q})\,\boldsymbol{J}^{-1}(\boldsymbol{q})\,\dot{\boldsymbol{J}}(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}}\right) \tag{2.81}$$

$$\boldsymbol{\rho}(\boldsymbol{q}) = \boldsymbol{J}^{-\mathrm{T}}(\boldsymbol{q})\,\boldsymbol{g}(\boldsymbol{q}) . \tag{2.82}$$

The definition of the operational space mass matrix in (2.80) can be used in order to create an alternative formulation of the Jacobian inverse as

$$\boldsymbol{J}^{-1}(\boldsymbol{q}) = \boldsymbol{H}^{-1}(\boldsymbol{q})\,\boldsymbol{J}^{\mathrm{T}}(\boldsymbol{q})\,\boldsymbol{\Lambda}(\boldsymbol{q})\,, \tag{2.83}$$

which is also referred to as the dynamically consistent generalized inverse $\bar{\boldsymbol{J}}$. It can be rewritten solely in joint space terms by substitution such that

$$\bar{\boldsymbol{J}}(\boldsymbol{q}) = \boldsymbol{H}^{-1}(\boldsymbol{q})\,\boldsymbol{J}^{\mathrm{T}}(\boldsymbol{q})\left(\boldsymbol{J}(\boldsymbol{q})\,\boldsymbol{H}^{-1}(\boldsymbol{q})\,\boldsymbol{J}^{\mathrm{T}}(\boldsymbol{q})\right)^{-1}. \tag{2.84}$$

Compared to the previous definition that is only based on kinematic characteristics of the robot, it has the advantage of including inertia properties as well, thereby improving stability.

2.1.4 Denavit-Hartenberg Convention

One of the earliest conventions for efficiently describing robot systems composed of joints and links was introduced in [42]. It exploits the fact that only four parameters are required in order to locate a line in space compared to the six in a universal transformation. This can be achieved by economic placement of the reference frames attached to joints and links such that the axis $\hat{\boldsymbol{x}}$ of a frame is both intersecting and perpendicular to the axis $\hat{\boldsymbol{z}}$ of the previous frame. The length and twist of every link in the system is described by a translation a followed by a rotation α. The joint offset translation d and angle of rotation θ complete this convention. All a and α parameters are constants, while the other two may serve as joint variables. For instance, a revolute joint changes the rotation angle θ, while only the value of d is adjustable in the prismatic version and cylindrical joints rely on both parameters. A screw joint on the other hand controls θ and calculates d according to the joint variable's value and its pitch. Several variations for locating the reference frames have been introduced in the literature, with different mismatches in the indexes of joints, links, frames, and parameters [117, 90, 38]. This section uses the version introduced in [90], as it applies the same index i to the joint, its coordinate system, the link moved by it, and the corresponding parameters. The respective a_i and α_i values however characterize the length and twist of the previous link.

In this notation, the first parameter a_i describes the translation along the $\hat{\boldsymbol{x}}_{i-1}$ axis, followed by a rotation α_i about the $\hat{\boldsymbol{x}}_{i-1}$ axis. This is succeeded by a translation d_i along $\hat{\boldsymbol{z}}_i$ and finally a translation θ_i about $\hat{\boldsymbol{z}}_i$. The axis of joint i is thus always aligned with the $\hat{\boldsymbol{z}}_i$ axis (Fig. 2.3). These operations can be expressed in the form of homogeneous transformation matrices

$${}^{i-1}\boldsymbol{T}_i = \mathrm{Trans}(\hat{\boldsymbol{x}}_{i-1}, a_i)\ \mathrm{Rot}(\hat{\boldsymbol{x}}_{i-1}, \alpha_i)\ \mathrm{Trans}(\hat{\boldsymbol{z}}_i, d_i)\ \mathrm{Rot}(\hat{\boldsymbol{z}}_i, \theta_i)\,, \tag{2.85}$$

where Trans describes a pure translation along a specific axis and Rot a basic rotation as demonstrated in (2.29) and the four components can also be interpreted as a

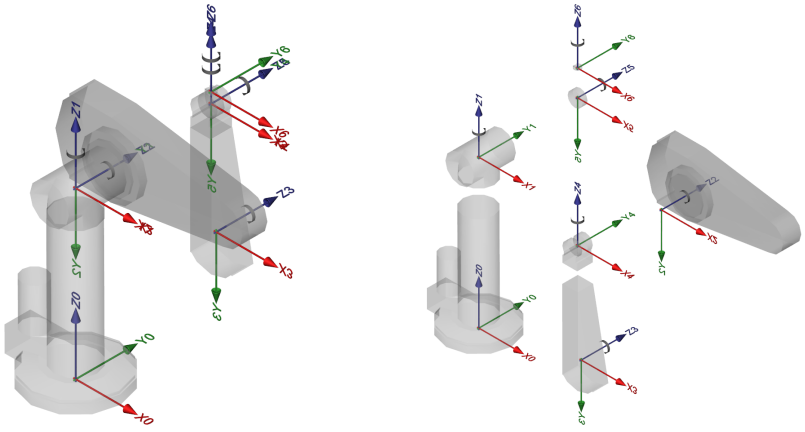


Fig. 2.3 Denavit-Hartenberg kinematics of a Puma robot with axes, rotation directions, and corresponding links. **a** combined view, **b** exploded view

combination of two axial screw transformations. Merged in a single transformation matrix, this is equal to

$$ {}^{i-1}\boldsymbol{T}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_i \\ \sin\theta_i \cos\alpha_i & \cos\theta_i \cos\alpha_i & -\sin\alpha_i & -\sin\alpha_i\, d_i \\ \sin\theta_i \sin\alpha_i & \cos\theta_i \sin\alpha_i & \cos\alpha_i & \cos\alpha_i\, d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.86} $$

Due to the restrictions on the placement of coordinate systems and the fact that joint axes are always aligned with the $\hat{\boldsymbol{z}}_i$ axis, the column vector of joint i in the Jacobian matrix that transforms velocities of the end effector in frame n to operational velocities relative to frame j can be specified as

$$ {}^{j}\boldsymbol{j}_{n|i}(\boldsymbol{q}) = \begin{bmatrix} {}^{j}\hat{\boldsymbol{z}}_i \times ({}^{j}\boldsymbol{p}_n - {}^{j}\boldsymbol{p}_i) \\ {}^{j}\hat{\boldsymbol{z}}_i \end{bmatrix} \tag{2.87} $$

in case of a revolute joint and

$$ {}^{j}\boldsymbol{j}_{n|i}(\boldsymbol{q}) = \begin{bmatrix} {}^{j}\hat{\boldsymbol{z}}_i \\ \boldsymbol{0} \end{bmatrix} \tag{2.88} $$

for a prismatic one.

Calculation of the inverse dynamics equation (2.76) is realized in two steps. First, the velocities and accelerations in each coordinate system are propagated from the base to the end effector, adding the velocities $\dot{q}_i$ acting in each joint. This is performed by adapting the equations for the transformation of linear (2.50) and angular (2.51) velocity as well as linear (2.56) and angular (2.57) acceleration for each

joint type. For a revolute joint in a serial chain this results in the formulas

$$\boldsymbol{\omega}_i = \hat{\boldsymbol{k}}\dot{q}_i + {}^{i}\boldsymbol{R}_{i-1}\,\boldsymbol{\omega}_{i-1} \tag{2.89}$$

$$\dot{\boldsymbol{\omega}}_i = \hat{\boldsymbol{k}}\ddot{q}_i + {}^{i}\boldsymbol{R}_{i-1}\,\dot{\boldsymbol{\omega}}_{i-1} + {}^{i}\boldsymbol{R}_{i-1}\,\boldsymbol{\omega}_{i-1} \times \hat{\boldsymbol{k}}\dot{q}_i \tag{2.90}$$

$$\boldsymbol{v}_i = {}^{i}\boldsymbol{R}_{i-1}\,(\boldsymbol{v}_{i-1} + \boldsymbol{\omega}_{i-1} \times {}^{i-1}\boldsymbol{p}_i) \tag{2.91}$$

$$\dot{\boldsymbol{v}}_i = {}^{i}\boldsymbol{R}_{i-1}\left(\dot{\boldsymbol{v}}_{i-1} + \dot{\boldsymbol{\omega}}_{i-1} \times {}^{i-1}\boldsymbol{p}_i + \boldsymbol{\omega}_{i-1} \times (\boldsymbol{\omega}_{i-1} \times {}^{i-1}\boldsymbol{p}_i)\right), \tag{2.92}$$

where velocities and angular accelerations are initialized with a value of zero. Gravitational influence can be accounted for with an appropriate linear acceleration at the base. During this first run, the forces $\boldsymbol{f}_i^{\mathrm{B}}$ and moments $\boldsymbol{n}_i^{\mathrm{B}}$ acting at the center of mass C_i of each body can also be determined. Their result is based on the mass m_i, the inertia $\boldsymbol{I}_i^{\mathrm{B}}$, and acceleration $\dot{\boldsymbol{v}}_i^{\mathrm{B}}$ at the center of mass such that

$$\dot{\boldsymbol{v}}_i^{\mathrm{B}} = \dot{\boldsymbol{v}}_i + \dot{\boldsymbol{\omega}}_i \times {}^{C_i}\boldsymbol{p}_i + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times {}^{C_i}\boldsymbol{p}_i) \tag{2.93}$$

$$\boldsymbol{f}_i^{\mathrm{B}} = m_i\,\dot{\boldsymbol{v}}_i^{\mathrm{B}} \tag{2.94}$$

$$\boldsymbol{n}_i^{\mathrm{B}} = \boldsymbol{I}_i^{\mathrm{B}}\,\dot{\boldsymbol{\omega}}_i + \boldsymbol{\omega}_i \times \boldsymbol{I}_i^{\mathrm{B}}\,\boldsymbol{\omega}_i\,. \tag{2.95}$$

In the second step of the recursive Newton-Euler algorithm, the effect of these forces and moments on the previous frames and joints is evaluated through

$$\boldsymbol{f}_i = \boldsymbol{f}_i^{\mathrm{B}} + {}^{i}\boldsymbol{R}_{i+1}\,\boldsymbol{f}_{i+1} \tag{2.96}$$

$$\boldsymbol{n}_i = \boldsymbol{n}_i^{\mathrm{B}} + \boldsymbol{f}_i^{\mathrm{B}} \times {}^{C_i}\boldsymbol{p}_i + {}^{i}\boldsymbol{R}_{i+1}\,(\boldsymbol{n}_{i+1} + \boldsymbol{f}_{i+1} \times {}^{i+1}\boldsymbol{p}_i) \tag{2.97}$$

$$\tau_i = \boldsymbol{n}_i^{\mathrm{T}}\,\hat{\boldsymbol{k}}\,. \tag{2.98}$$

In case of a revolute joint, the torque τ is a result of moments acting on the joint axis. The linear velocity terms are never used in this algorithm and their computation may thus be omitted. External forces acting on the system can optionally be integrated during the final step and the forces and moments of several subchains in a tree-like structure are combined through addition.

2.1.5 Spatial Vector Algebra

As seen in Sect. 2.1.4, the description of robot system using the Denavit-Hartenberg convention is fairly complex and many incompatible versions are currently in use. The placement of link frames has been restricted for performance reasons, kinematics and dynamics equations have to be specified individually for each joint type and it is rather difficult to integrate new joint classes. Especially dynamics algorithms tend to appear extremely complicated when formulated with three-dimensional vectors. The alternative representation based on six-dimensional vectors introduced in (2.52) and (2.68) can be used to create more compact descriptions without sacrificing performance. By directly integrating concepts for different joint models, such

an algorithm is valid for a broad range of robot systems and can also support kinematic trees and closed-loop constraints.

Spatial vectors are not part of Euclidean space and the notation [48, 137] distinguishes between members of either spatial motion or force vector spaces. Elements such as velocity or acceleration are classified as motion vectors, while force vectors include momentum, impulse, and other related quantities. The dot product is only defined between motion and force vectors, all other combinations are undefined:

$$\vec{\boldsymbol{v}} \cdot \vec{\boldsymbol{f}} = \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{n} \\ \boldsymbol{f} \end{bmatrix} = \boldsymbol{\omega} \cdot \boldsymbol{n} + \boldsymbol{v} \cdot \boldsymbol{f} \,. \tag{2.99}$$

As demonstrated in (2.46), the derivative of a constant vector rotating with an angular velocity is calculated using a skew-symmetric matrix or the identical cross product operation. This concept can be extended to the domain of spatial vectors with the analog definition of a matrix

$$\boldsymbol{S}(\vec{\boldsymbol{v}}) = \begin{bmatrix} \boldsymbol{S}(\boldsymbol{\omega}) & \boldsymbol{0} \\ \boldsymbol{S}(\boldsymbol{v}) & \boldsymbol{S}(\boldsymbol{\omega}) \end{bmatrix}, \tag{2.100}$$

such that a constant vector $\vec{\boldsymbol{v}}_2$ moving with a velocity $\vec{\boldsymbol{v}}_1$ is mapped to

$$\dot{\vec{\boldsymbol{v}}}_2 = \vec{\boldsymbol{v}}_1 \times \vec{\boldsymbol{v}}_2 = \boldsymbol{S}(\vec{\boldsymbol{v}}_1)\, \vec{\boldsymbol{v}}_2 = \begin{bmatrix} \boldsymbol{\omega}_1 \\ \boldsymbol{v}_1 \end{bmatrix} \times \begin{bmatrix} \boldsymbol{\omega}_2 \\ \boldsymbol{v}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\omega}_1 \times \boldsymbol{\omega}_2 \\ \boldsymbol{v}_1 \times \boldsymbol{\omega}_2 + \boldsymbol{\omega}_1 \times \boldsymbol{v}_2 \end{bmatrix}. \tag{2.101}$$

Similar to (2.52) and (2.68), there are different cross product operators for spatial motion and force vectors. A constant force $\vec{\boldsymbol{f}}$ subject to velocity $\vec{\boldsymbol{v}}$ is represented as

$$\dot{\vec{\boldsymbol{f}}} = \vec{\boldsymbol{v}} \times^{-\mathrm{T}} \vec{\boldsymbol{f}} = -\boldsymbol{S}^{\mathrm{T}}(\vec{\boldsymbol{v}})\, \vec{\boldsymbol{f}} = \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v} \end{bmatrix} \times \begin{bmatrix} \boldsymbol{n} \\ \boldsymbol{f} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\omega} \times \boldsymbol{n} + \boldsymbol{v} \times \boldsymbol{f} \\ \boldsymbol{\omega} \times \boldsymbol{f} \end{bmatrix}. \tag{2.102}$$

The exerted spatial force of a rigid body equals its change of momentum with respect to time. The spatial equation of motion can therefore be formulated as

$$\vec{\boldsymbol{f}} = \frac{\mathrm{d}(\vec{\boldsymbol{I}}\vec{\boldsymbol{v}})}{\mathrm{d}t} = \vec{\boldsymbol{I}}\vec{\boldsymbol{a}} + \dot{\vec{\boldsymbol{I}}}\vec{\boldsymbol{v}} = \vec{\boldsymbol{I}}\vec{\boldsymbol{a}} + \vec{\boldsymbol{v}} \times^{-\mathrm{T}} \vec{\boldsymbol{I}}\vec{\boldsymbol{v}} = \vec{\boldsymbol{I}}\vec{\boldsymbol{a}} + \vec{\boldsymbol{\beta}}\,, \tag{2.103}$$

where the spatial inertia matrix $\vec{\boldsymbol{I}}$ with respect to a frame j is defined by

$${}^{j}\vec{\boldsymbol{I}} = \begin{bmatrix} {}^{i}\boldsymbol{I} + m\boldsymbol{S}({}^{j}\boldsymbol{p}_i)\, \boldsymbol{S}({}^{j}\boldsymbol{p}_i)^{\mathrm{T}} & m\boldsymbol{S}({}^{j}\boldsymbol{p}_i) \\ m\boldsymbol{S}({}^{j}\boldsymbol{p}_i)^{\mathrm{T}} & m\boldsymbol{1} \end{bmatrix} \tag{2.104}$$

and $\vec{\boldsymbol{\beta}}$ can be interpreted as a vector of bias force. In this representation, the rigid body's center of mass is located in frame i and its origin relative to the spatial inertia one's is described by a displacement vector $\boldsymbol{p}$. The distribution of the body's mass m at this location is given in form of the inertia matrix $\boldsymbol{I}$ according to (2.75). In case of equal origins, the spatial equation of motion can be written as

$$\begin{aligned}\begin{bmatrix} \boldsymbol{n} \\ \boldsymbol{f} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & m\boldsymbol{1} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{\omega}} \\ \dot{\boldsymbol{v}} \end{bmatrix} - \begin{bmatrix} \boldsymbol{S}(\boldsymbol{\omega})^{\mathrm{T}} & \boldsymbol{S}(\boldsymbol{v})^{\mathrm{T}} \\ \boldsymbol{0} & \boldsymbol{S}(\boldsymbol{\omega})^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & m\boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{I}\,\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega} \\ m\dot{\boldsymbol{v}} \end{bmatrix}, \end{aligned} \tag{2.105}$$

thus resulting in the formulation given by Newton's (2.73) and Euler's (2.74) respective equation.

With these definitions, the two loops of the recursive Newton-Euler algorithm introduced in Sect. 2.1.4 can be formulated in spatial vector algebra. The propagation of velocity and acceleration in a serial chain is accomplished in the first step by

$$\vec{\boldsymbol{v}}_i = {}^i\boldsymbol{X}_{i-1}\,\vec{\boldsymbol{v}}_{i-1} + \boldsymbol{\Phi}_i\,\dot{\boldsymbol{q}}_i \tag{2.106}$$

$$\vec{\boldsymbol{a}}_i = {}^i\boldsymbol{X}_{i-1}\,\vec{\boldsymbol{a}}_{i-1} + \boldsymbol{\Phi}_i\,\ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{\Phi}}_i\,\dot{\boldsymbol{q}}_i\,, \tag{2.107}$$

with initial values of zero for velocity and gravity influence in case of acceleration. The spatial transformation $\boldsymbol{X}$ contains rotation and translation descriptions that depend on the current joint configuration. An n_i-dimensional joint's directions of free motion are modeled by a matrix $\boldsymbol{\Phi}_i$ of size $6 \times n_i$. Its derivative $\dot{\boldsymbol{\Phi}}_i$ can also be written in the form

$$\dot{\boldsymbol{\Phi}}_i = \mathring{\boldsymbol{\Phi}}_i\,\dot{\boldsymbol{q}}_i + \vec{\boldsymbol{v}}_i \times \boldsymbol{\Phi}_i\,\dot{\boldsymbol{q}}_i\,, \tag{2.108}$$

where $\mathring{\boldsymbol{\Phi}}_i$ is the apparent derivate of $\boldsymbol{\Phi}_i$ in body coordinates. For most common joint types including revolute and prismatic joints, its value is equal to zero. A revolute joint's rotation matrix can be generated using the axis-angle description and its translation component is equal to the null vector. If the joint is aligned with the $\hat{\boldsymbol{z}}_i$ axis as in the Denavit-Hartenberg norm, its free motion directions are given by

$$\boldsymbol{\Phi}_i = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}. \tag{2.109}$$

A prismatic joint on the other hand only produces translation but no rotation, therefore employing the identity matrix and a displacement vector in combination with

$$\boldsymbol{\Phi}_i = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^{\mathrm{T}}. \tag{2.110}$$

Joint descriptions with other axes or additional degrees of freedom can be integrated in a similar fashion. As demonstrated in Sect. 2.1.1 and 2.1.2, spherical and 6-DOF joints are of special interest for free-floating robots. They are modeled with three- or six-column motion matrices respectively and use (2.26) in order to convert quaternions to a rotation matrix. A mapping from four position to three velocity variables is given by the derivative

$$\begin{bmatrix} \dot{h}_0 \\ \dot{h}_1 \\ \dot{h}_2 \\ \dot{h}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -h_1 & -h_2 & -h_3 \\ h_0 & -h_3 & h_2 \\ h_3 & h_0 & -h_1 \\ -h_2 & h_1 & h_0 \end{bmatrix} \begin{bmatrix} \omega_{\mathrm{x}} \\ \omega_{\mathrm{y}} \\ \omega_{\mathrm{z}} \end{bmatrix}. \tag{2.111}$$

After calculating current velocities and accelerations in the body frames, the net force in body coordinates generated by the acceleration of each link can now be determined with the equation

$$\vec{f}_i^{\,\mathrm{B}} = \vec{I}_i\,\vec{a}_i + \vec{v}_i \times^{-\mathrm{T}} \vec{I}_i\,\vec{v}_i\,. \tag{2.112}$$

The second and final loop of the algorithm propagates the forces generated by the individual links from the frame at the end effector back to the base:

$$\vec{f}_i = \vec{f}_i^{\,\mathrm{B}} + {}^{i}\boldsymbol{X}_{i+1}^{-\mathrm{T}}\,\vec{f}_{i+1} \tag{2.113}$$

$$\boldsymbol{\tau}_i = \boldsymbol{\Phi}_i^{\mathrm{T}}\,\vec{f}_i\,. \tag{2.114}$$

The transpose of the matrix $\boldsymbol{\Phi}$ is employed in order to map these forces to the torques in the individual joints of a robot system. A graph structure can be used for a description of such a model, where edges represent joints and bodies are marked by vertices. Multiple forces acting at the forks of a tree can be combined through addition and external influences are incorporated in the same way, however these may have to include a proper spatial transformation if their description is not given in body coordinates.

Forward dynamics calculation (2.77) is used in order to calculate joint accelerations from a given set of joint positions, velocities, and torques. The articulated-body algorithm can solve this task by interpreting a subtree of several joints and rigid bodies as one articulated body with a combined articulated-body inertia matrix $\vec{I}^{\mathrm{A}}$ and bias force vector $\vec{\boldsymbol{\beta}}^{\mathrm{A}}$. The unknown force exerted on joint i by this subtree is described by the following equation:

$$\vec{f}_i = \vec{I}_i^{\mathrm{A}}\,\vec{a}_i + \vec{\boldsymbol{\beta}}_i^{\mathrm{A}}\,. \tag{2.115}$$

In combination with (2.114) and (2.107), this can be related to the generated torques and is solved for the unknown joint accelerations such that

$$\boldsymbol{\tau}_i = \boldsymbol{\Phi}_i^{\mathrm{T}}\left(\vec{I}_i^{\mathrm{A}}\,(\vec{a}_{i-1} + \boldsymbol{\Phi}_i\,\ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{\Phi}}_i\,\dot{\boldsymbol{q}}_i) + \vec{\boldsymbol{\beta}}_i^{\mathrm{A}}\right) \tag{2.116}$$

$$\ddot{\boldsymbol{q}}_i = (\boldsymbol{\Phi}_i^{\mathrm{T}}\,\vec{I}_i^{\mathrm{A}}\,\boldsymbol{\Phi}_i)^{-1}\left(\boldsymbol{\tau}_i - \boldsymbol{\Phi}_i^{\mathrm{T}}\left(\vec{I}_i^{\mathrm{A}}\,(\vec{a}_{i-1} + \dot{\boldsymbol{\Phi}}_i\,\dot{\boldsymbol{q}}_i) + \vec{\boldsymbol{\beta}}_i^{\mathrm{A}}\right)\right). \tag{2.117}$$

Two articulated bodies can be combined through the addition of their individual articulated-body inertias and bias forces. If the newly-created articulated body is connected to the remaining system through joint $i-1$, equation (2.107) can be used in order to define

$$\vec{f}_{i-1} = \vec{I}_{i-1}^{\mathrm{A}}\,\vec{a}_{i-1} + \vec{\boldsymbol{\beta}}_{i-1}^{\mathrm{A}} + \vec{I}_i^{\mathrm{A}}\,(\vec{a}_{i-1} + \boldsymbol{\Phi}_i\,\ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{\Phi}}_i\,\dot{\boldsymbol{q}}_i) + \vec{\boldsymbol{\beta}}_i^{\mathrm{A}}\,. \tag{2.118}$$

After substituting (2.117) for the joint accelerations, the new equation can be regrouped into two parts, where one is a coefficient of the spatial acceleration. This in correspondence with the definition in (2.115) gives

$$\vec{\boldsymbol{I}}_{i-1}^{\mathrm{A}} = \vec{\boldsymbol{I}}_{i-1}^{\mathrm{A}} + \vec{\boldsymbol{I}}_{i}^{\mathrm{A}} - \vec{\boldsymbol{I}}_{i}^{\mathrm{A}}\,\boldsymbol{\Phi}_i\,(\boldsymbol{\Phi}_i^{\mathrm{T}}\,\vec{\boldsymbol{I}}_{i}^{\mathrm{A}}\,\boldsymbol{\Phi}_i)^{-1}\,\boldsymbol{\Phi}_i^{\mathrm{T}}\,\vec{\boldsymbol{I}}_{i}^{\mathrm{A}} \tag{2.119}$$

$$\begin{aligned}\vec{\boldsymbol{\beta}}_{i-1}^{\mathrm{A}} = {} & \vec{\boldsymbol{\beta}}_{i-1}^{\mathrm{A}} + \vec{\boldsymbol{I}}_{i}^{\mathrm{A}}\,\boldsymbol{\Phi}_i\,(\boldsymbol{\Phi}_i^{\mathrm{T}}\,\vec{\boldsymbol{I}}_{i}^{\mathrm{A}}\,\boldsymbol{\Phi}_i)^{-1}\left(\boldsymbol{\tau}_i - \boldsymbol{\Phi}_i^{\mathrm{T}}\,(\vec{\boldsymbol{I}}_{i}^{\mathrm{A}}\,\dot{\boldsymbol{\Phi}}_i\,\dot{\boldsymbol{q}}_i + \vec{\boldsymbol{\beta}}_{i}^{\mathrm{A}})\right) \\ & + \vec{\boldsymbol{I}}_{i}^{\mathrm{A}}\,\dot{\boldsymbol{\Phi}}_i\,\dot{\boldsymbol{q}}_i + \vec{\boldsymbol{\beta}}_{i}^{\mathrm{A}}\,,\end{aligned} \tag{2.120}$$

with the two components defined independent of spatial acceleration. The articulated-body equations above are not given in link coordinates and therefore still need to add appropriate spatial transform matrices. In the first of three loops of a recursive implementation, spatial velocities are propagated from the base to the end effector, starting with an initial value of zero. During this run, the articulated-body inertias and bias forces are assigned with the respective spatial inertias of the links and their velocities. In the second loop, articulated-body inertias and bias forces are calculated backwards from the end effector to the base. After initializing the base according to gravity, the final loop then transforms the spatial accelerations toward the end effector and uses these values in order to generate appropriate joint accelerations. This result can now be leveraged in combination with integration methods for simulation purposes. The extension to kinematics trees is given by the addition of articulated-body inertias and bias forces at forked subtrees.

2.1.6 Task-Based Control

Highly-redundant manipulators possess more degrees of freedom than are necessary for a desired end effector task. An operational space activity including all six position and orientation axes requires a robot system with at least six degrees of freedom, a specification met by most standard industrial manipulators. For tasks with fewer constraints or systems with redundant joints, the additional degrees of freedom can be used in order to follow other operations while maintaining the constraints of the main task. This is especially interesting for systems with multiple end effectors and complex humanoid systems. Constraints can include various activities such as posture control, obstacle avoidance, gravity compensation, or joint limit avoidance [92, 134, 135, 137, 102].

Section 2.1.3 introduced different control schemes for end effector trajectories, including operational space control. In order to integrate an operational task with a non-interfering posture command, the latter control velocities or torques are projected into the dynamically consistent nullspace $\boldsymbol{N}$ of an operational space task given by

$$\boldsymbol{N}_{\mathrm{task}}(\boldsymbol{q}) = \mathbf{1} - \bar{\boldsymbol{J}}(\boldsymbol{q})\,\boldsymbol{J}(\boldsymbol{q})\,. \tag{2.121}$$

The combined control vector keeps the operational space task's constraints at all times, while using redundant joints in order to reach a desired posture:

$$\boldsymbol{\tau} = \boldsymbol{J}^{\mathrm{T}}(\boldsymbol{q})\,\vec{\boldsymbol{f}}_{\mathrm{task}} + \boldsymbol{N}_{\mathrm{task}}^{\mathrm{T}}(\boldsymbol{q})\,\boldsymbol{\tau}_{\mathrm{posture}} \tag{2.122}$$

$$\dot{\boldsymbol{q}} = \bar{\boldsymbol{J}}(\boldsymbol{q})\,\dot{\boldsymbol{x}}_{\mathrm{task}} + \boldsymbol{N}_{\mathrm{task}}(\boldsymbol{q})\,\dot{\boldsymbol{q}}_{\mathrm{posture}}\,. \tag{2.123}$$

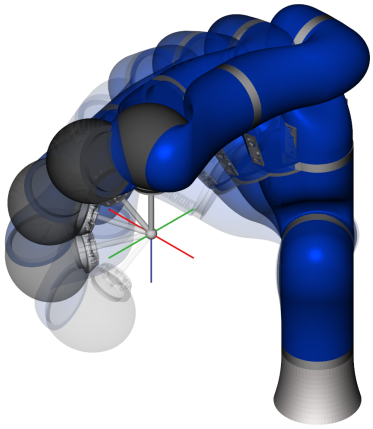


Fig. 2.4 Demonstration of task-based control: only three degrees of freedom are required in order to maintain a position in space, the remaining joints of a 7-DOF manipulator can be used for posture control

Specific decoupling of axes in the operational task can be introduced with an $m \times m$ selection matrix $\boldsymbol{\Upsilon}$. Its diagonal corresponds to the m rotational and translational degrees of freedom in operational space. This number is usually equal to six in case of a manipulator with one end effector. If initialized with the identity matrix, all axes are subject to decoupling and the position and orientation constraints of the task are observed. In the definition

$$\boldsymbol{N}_{\text{task}}(\boldsymbol{q}) = \mathbf{1} - \bar{\boldsymbol{J}}(\boldsymbol{q})\,\boldsymbol{\Upsilon}\,\boldsymbol{J}(\boldsymbol{q})\,, \tag{2.124}$$

selective axes can be omitted by setting the corresponding entries in the diagonal to zero, for instance in order to lock position of the end-effector (Fig. 2.4). Various control paradigms can be integrated as the different tasks are all integrated on the joint torque or velocity level. Integration of components is achieved by a generated control vector and a matching projection function.

This type of task definition has the additional advantage that control programs are independent from a specific robot system or model and can be ported between several kinematics and dynamics specifications. Multiple parallel tasks can be ordered according to priorities and projected into the nullspace of the superior task with the two equations

$$\boldsymbol{\tau} = \boldsymbol{\tau}_1 + \boldsymbol{N}_1^{\mathrm{T}}(\boldsymbol{q})\left(\boldsymbol{\tau}_2 + \boldsymbol{N}_2^{\mathrm{T}}(\boldsymbol{q})\,(\boldsymbol{\tau}_3 + \ldots)\right) \tag{2.125}$$

$$\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_1 + \boldsymbol{N}_1(\boldsymbol{q})\left(\dot{\boldsymbol{q}}_2 + \boldsymbol{N}_2(\boldsymbol{q})\,(\dot{\boldsymbol{q}}_3 + \ldots)\right). \tag{2.126}$$

Several different control strategies can be combined using this approach. For instance, a humanoid can keep its balance by including a center of gravity task while using force control in its right hand in order to keep in contact with a table. At the same time it can focus its vision on a moving object in space and avoid obstacles in its vicinity with a potential field approach. Hard constraints that may not be violated should be assigned the highest priorities (e.g., tasks associated with hardware limits or balance).

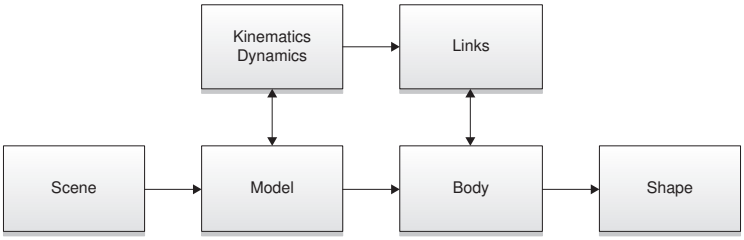


Fig. 2.5 Connection between data from kinematics and dynamics representations of a robot system to its geometric model. The coordinate systems in each link of the kinematics structure are used in order to position the corresponding bodies in the scene graph. Shapes are placed relative to a moving body and their frames are updated in the process as well

2.2 Geometric Representation

Kinematics and dynamics equations are only part of a robot system's description. Its geometry and shape specifications are also required in order to generate proper visualizations and simulations of the mechanism. For the system itself, this information is usually available in form of CAD data with varying accuracy and sometimes these details are also available for stationary or moving objects of the environment. Very often however these obstacles will rather be modeled and tracked through sensor data with even lower precision.

A description of the robot and other models in the scene is given in form of a collection of rigid bodies (Fig. 2.3). Each of these elements can be located relative to a world frame by the position of its origin and a corresponding orientation (Sect. 2.1.1). The forward position kinematics of a robot system can be used in order generate this data based on the current joint positions and their individual links can then be synchronized with the respective geometry parts (Fig. 2.5). This can include serial chains, trees, or even closed-loop structures. Moving obstacles are updated in a similar fashion. A single rigid body can be composed of various basic shapes such as boxes, spheres, cylinders, or general polygon meshes. Detailed visualizations are usually based on models with many shapes and a high polygon count, however this type of resolution can vastly decrease performance of collision detection and physics simulation algorithms. This is often prevented by including two geometry descriptions, where the second one has a much lower complexity and often falls back to basic shapes or convex hulls [8]. This simplification should however always avoid common approximations from computer graphics that remove important parts of the model, as this can lead to collisions on the actual hardware [32]. Due to this, the new shape should rather be slightly larger than smaller (Fig. 2.6). Concave parts can also be split into a number of convex objects in order to accomplish a better approximation and performance, finding a minimal representation for this however is NP-hard [6]. The extreme detail of CAD models typically also requires simplifications for visualization components. Material properties and lighting parameters need to be considered here as well.

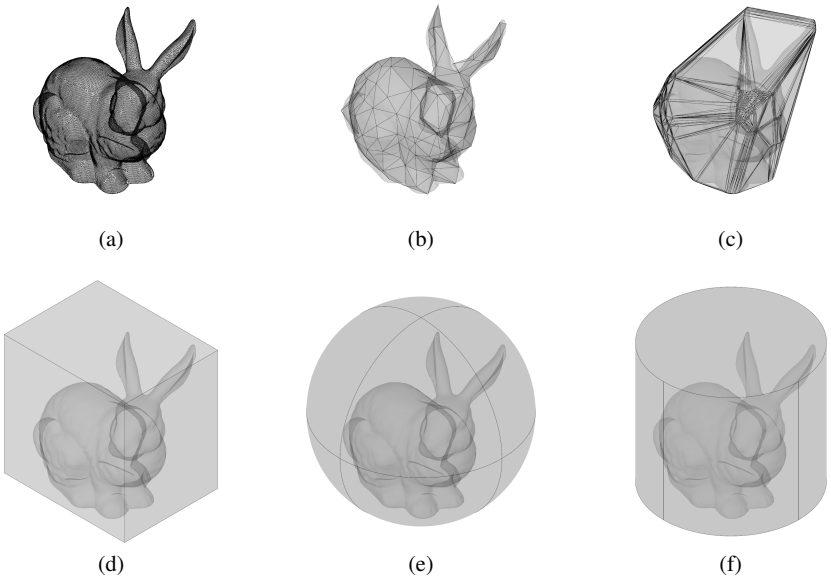


Fig. 2.6 Stanford bunny with 35 947 vertices and 69 451 triangles. **a** original, **b** critical reduction, **c** convex hull, **d** axis-aligned bounding box, **e** bounding sphere, **f** axis-aligned bounding cylinder

2.2.1 Collision Detection

Factory settings typically have robot manipulators working in areas with limited space and in close cooperation with other machines and mechatronic systems. Collisions are manually prevented by detailed inspection of the individual robot's trajectories, especially with regard to other moving objects or robots. Even self-collisions of the robot are only partially eliminated by restricted joint angles and different tool systems often require new verifications. In particular scenarios without extensively tested trajectories and programs that can be created on demand by less specialized personnel call for collision detection systems based on geometry data and sensor information. Robot systems in industrial scenarios and other applications with complex environment models operate with fast control cycles of up to 1000 Hz, thus relying on high performance algorithms that can provide direct feedback in case of online modifications. Three basic query types are generally available when dealing with this kind of problem. The most simple and fastest form of calculation is the intersection test. It is used in order to determine whether two objects in their current configuration are colliding with each other. Finding actual witness points to such a collision however is already much more complex. Imperfect geometries and sensor readings can sometimes be compensated for with enlarged models, but more detailed adjustments and informations are obtained through rather expensive distance

computations, that calculate the minimum clearance between two non-colliding objects. A similar concept is available for intersecting geometries in the form of penetration depth computation, which is often used in physics simulations.

Whether an object is contained within a different geometric part can only be answered for a closed, bounded, and nonempty set of points. This task is rather difficult for a number of unordered polygons with no information regarding their connectivity and relation toward each other [15, 44]. Several classes of objects are typically found in collision detection applications, with basic primitives in the form of boxes and spheres to more abstract definitions of solid objects such as convex and concave types. If all line segments between any pair of points are contained within the object itself, it is referred to as convex. This includes basic shapes such as boxes, cones, cylinders, and spheres. Much simpler and faster algorithms are usually available for these categories compared to concave objects of arbitrary polygon meshes. Even in case of a collection of convex objects, a precise test of all geometries is rather expensive in order to answer a simple intersection query. In addition to a narrow phase that handles these exact computations, a prior broad phase introduces different types of simplified bounding volume abstractions in order to only test likely collisions.

Spheres can be represented by a center point and a corresponding radius in a very inexpensive way. They are often used as bounding volumes for rigid bodies due to their efficient description and invariance under rotation. Intersection between two spheres occurs when the distance between their center points is less or equal to the sum of their radii. In a similar fashion, the space between them can be calculated by subtracting the sum of their radii from the distance of their origins. This value is naturally equal to zero in case of intersection, as is the penetration depth for non-colliding spheres. The latter is determined for intersecting spheres by the sum of their radii minus the distance between center points.

Another popular bounding volume representation are axis-aligned boxes. They can either be specified by using minimum and maximum points, a minimum point and widths along the axes, or a center point and corresponding half widths. The latter representation however is often chosen for efficiency reasons, as fewer parameters need to be updated and it can also be used to create a matching bounding sphere description. Intersection can be efficiently determined by comparison when two boxes are represented with respective minimum and maximum points. Similar tests can be performed with boxes specified by a single point and extents along axes.

Convex objects possess various properties that make them ideal candidates for collision detection purposes. Two non-intersecting convex shapes can always be separated by a plane and a local minimum distance between them is also always a global minimum. The latter property enables the use of simple hill climbing methods for distance computation, such as in the very efficient Gilbert-Johnson-Keerthi (GJK) algorithm [64] and its adapted version for general convex objects [63]. This approach calculates the distance between the Minkowski difference of two objects and an origin instead of working on the actual shape descriptions. Explicit calculation of this representation is however avoided by introducing functions that map a vector to a support point of a convex object. Different types of convex shapes can be integrated by supplying corresponding support mappings. This

method can also be adapted in order to determine the penetration depth of two intersecting objects.

2.2.2 Physics Simulation

Testing and verifying a new implementation or algorithm before actually running it on real hardware is generally advisable. Proper visualization of inverse kinematics solutions or joint trajectories alone can help identify errors in calculations as the mapping from joint to operational space is not always intuitive. This is even more important for architectures with direct torque control, where false values can lead to serious hardware damage. Forward dynamics solutions as introduced in Sect. 2.1.5 in combination with integration methods can be used in order to evaluate the underlying physics equations in such a scenario. Physics simulation does not end here however; other aspects include contact and penetration with other objects, deformation of soft bodies, friction models and air resistance, spring dynamics, or particle systems. A virtual robot model that approximates influences in the real world can assist in the development of new algorithms or in the training of machine learning systems before they are deployed onto the actual hardware.

Contact and penetration information from geometric models is an important factor in physics engines [45]. This data can be evaluated for a specific configuration of rigid bodies and its runtime is dependent on the complexity and amount of objects in the simulated world. Simplified models and bounding volume abstractions are typically introduced in order to increase performance (Sect. 2.2.1). Moving objects and fixed time step intervals however demand the availability of continuous detection methods in order to register all collisions. Geometric coherence also needs to be considered as these features often remain constant between consecutive steps.

Current approaches can be classified into three major categories: constraint-based, penalty-based, and impulse-based methods. Tight constraints as they occur in robotic structures with various joints can be efficiently simulated by the first of these paradigms. A large selection of algorithms are available in this area, including the Articulated-Body Algorithm or the Composite-Rigid-Body Algorithm [48]. Penalty-based engines [7] model systems as unconstrained equations, where deviations are compensated through the addition of corresponding penalty forces. Stiff springs try to restore two colliding objects to a non-penetrating state. This concept can also be used in the simulation of soft-body dynamics. The last paradigm uses a series of small impulses in order to prevent penetration between objects [107]. These are applied at the contact points of collisions and thus require a high frequency simulation loop. Highly unconstrained systems with many fast moving objects are ideal domains for this approach, whereas stable constraints such as joints require an excessive amount of collision handling. Hybrid solutions can combine the advantages of multiple methods and may provide more stable and efficient simulations.

Chapter 3
Obstacle Avoidance and Motion Planning

A human operator has to pay close attention to the environment of a robot when manually programming movements between two joint configurations. Obstacles within the reachable area of the system and the robot's structure itself confine the possible number of solutions and may result in collisions. Joint limits generally do not eliminate self-collisions, especially given additional end-effector tools. Furthermore, work areas of robot systems in close proximity may overlap and require exact synchronization. An operator has to consider all of this when creating a path between two configurations. This may result in a series of seemingly arbitrary movements in the program. Motion planning algorithms on the other hand can automatically generate collision free paths given a specified start and goal configuration. Adjustments in target positions do not require manual reprogramming of movements and thus enable easier maintenance and program readability. Complex scenarios however can provide difficult challenges for planning algorithms and randomly generated paths may prove inefficient. The removal of fences and the introduction of direct interaction with a robot system further intensifies the need for sensor integration and dynamic obstacle avoidance.

The classic problem of motion planning as it is described in [97, 99] comprises an articulated robot $\mathcal{A}$ as a single rigid object in a workspace $\mathcal{W}$ without dynamic properties or contact motions. All obstacles $\mathcal{O}_1, \ldots, \mathcal{O}_m$ are fixed rigid objects, the geometry and location of the robot and all obstacles are given and perfectly accurate, and there are no kinematic constraints that limit the motions of the robot. After providing a start and goal configuration ($\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$), the task is to find a path for the robot that does not collide with any obstacle or to report that no path could be found. A *complete* planner must be able to report the availability of a solution to a given query within a finite amount of time.

The workspace of a robot with n degrees of freedom is modeled as the Euclidean space $\mathbb{R}^2$ or $\mathbb{R}^3$. It can be mapped to a configuration space $\mathcal{C}$, which is a manifold of dimension n. Obstacles in $\mathcal{W}$ are mapped to obstacles in the configuration space and then referred to as $\mathcal{C}$-obstacles such that

$$\mathcal{C}\mathcal{O}_i = \left\{ \boldsymbol{q} \in \mathcal{C} : \mathcal{A}(\boldsymbol{q}) \cap \mathcal{O}_i \neq \emptyset \right\}. \tag{3.1}$$

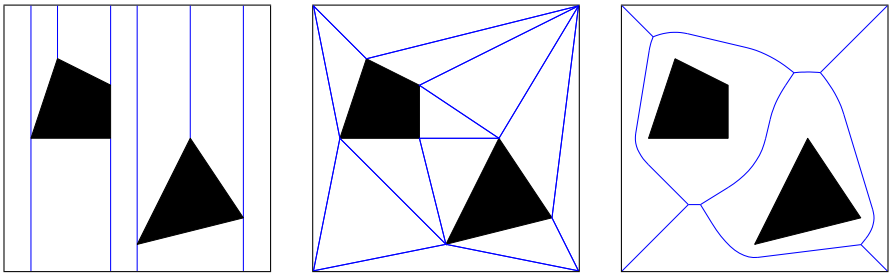


Fig. 3.1 Combinatorial approaches to a two-dimensional configuration space with polygonal obstacles. **a** vertical or trapezoidal cell decomposition, **b** Delaunay triangulation, **c** generalized Voronoi diagram

The robot collides with its environment for every configuration that is part of the obstacle region $\mathcal{C}_{\text{obst}}$ of the configuration space, represented by the union of all obstacles in $\mathcal{C}$:

$$\mathcal{C}_{\text{obst}} = \bigcup_{i=1}^{m} \mathcal{C}\mathcal{O}_i \,. \tag{3.2}$$

Every non-colliding configuration of the robot is called a free configuration and part of the free space

$$\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{obst}} = \left\{ \boldsymbol{q} \in \mathcal{C} : \mathcal{A}(\boldsymbol{q}) \cap \bigcup_{i=1}^{m} \mathcal{O}_i = \emptyset \right\}. \tag{3.3}$$

The general motion planning problem is PSPACE-hard [125]. Early solutions have worked on complete representations of the configuration space, for instance in polygonal form [103]. These combinatorial approaches include vertical cell-decomposition, triangulation, or Voronoi diagrams (Fig. 3.1). While all these methods are complete, they are typically not suitable for high dimensional problems.

3.1 Potential Fields

Potential functions [97, 31] view the robot as a particle moving through a gradient vector field (Fig. 3.2). The goal position acts as an attractive force on the robot, while it is being repelled by forces from obstacles in its environment:

$$U(\boldsymbol{q}) = U_{\text{att}}(\boldsymbol{q}) + U_{\text{rep}}(\boldsymbol{q}) \,. \tag{3.4}$$

The robot stops moving when the gradient equals zero. This can happen either at a local maximum, a saddle point, or a local minimum. Neither a local maximum nor

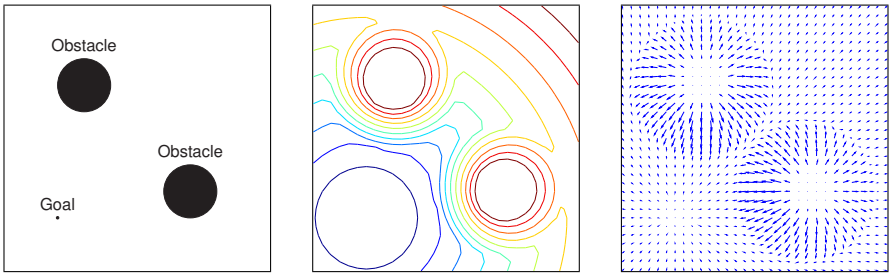


Fig. 3.2 Motion planning scenario with goal configuration in the lower left and two circular obstacles. **a** configuration space, **b** attractive and repulsive isolines, **c** attractive and repulsive vector field

a saddle point is stable and any movement of the robot will exit the critical point. However, if the robot reaches a local minima, further disturbances will generally return the robot to this critical point. If this happens at a position unequal to the goal configuration, the robot will not be able to reach its target. Due to this, a motion planner based on potential fields alone is not complete. One possible solution for this was introduced in the form of navigation functions [129]. They describe a set of potential functions with no local minima, but finding these requires full knowledge of the configuration space.

The attractive potential's task is to lead the robot to a specified goal configuration. It should depend on the distance $d(\boldsymbol{q}, \boldsymbol{q}_{\text{goal}})$ of the robot's current configuration compared to the goal configuration. A very basic definition can be achieved by combining this distance with a scaling factor ζ, resulting in a conic potential

$$U(\boldsymbol{q}) = \zeta \, d(\boldsymbol{q}, \boldsymbol{q}_{\text{goal}}) \,. \tag{3.5}$$

The corresponding attractive gradient however is undefined at the goal configuration as can be seen from

$$\nabla U(\boldsymbol{q}) = \frac{\zeta}{d(\boldsymbol{q}, \boldsymbol{q}_{\text{goal}})} \, (\boldsymbol{q} - \boldsymbol{q}_{\text{goal}}) \,. \tag{3.6}$$

A simple quadratic potential can therefore replace the conic potential within a certain distance d^*_{goal} to the goal configuration:

$$U(\boldsymbol{q}) = \tfrac{1}{2} \zeta \, d^2(\boldsymbol{q}, \boldsymbol{q}_{\text{goal}}) \,. \tag{3.7}$$

The combined attractive potential $U_{\text{att}}(\boldsymbol{q})$ (Fig. 3.3a) is well defined at the boundary between both potentials and also prevents overshooting of the robot at far distances:

$$U_{\text{att}}(\boldsymbol{q}) = \begin{cases} \frac{1}{2} \zeta \, d^2(\boldsymbol{q}, \boldsymbol{q}_{\text{goal}}) & d(\boldsymbol{q}, \boldsymbol{q}_{\text{goal}}) \leq d^*_{\text{goal}} \\ d^*_{\text{goal}} \, \zeta \, d(\boldsymbol{q}, \boldsymbol{q}_{\text{goal}}) - \frac{1}{2} \zeta \, (d^*_{\text{goal}})^2 & d(\boldsymbol{q}, \boldsymbol{q}_{\text{goal}}) > d^*_{\text{goal}} \,. \end{cases} \tag{3.8}$$

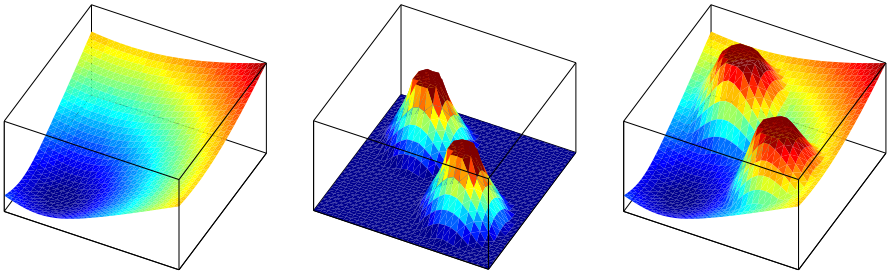


Fig. 3.3 Potential field with goal configuration in the lower left and two circular obstacles. **a** attractive potential, **b** repulsive potential, **c** combined attractive and repulsive potential

In order to prevent collisions with obstacles in its environment, the repulsive potential $U_{\text{rep}}(\boldsymbol{q})$ has to push the robot away from these objects. In contrast to the attractive potential, it should only be active within a limited region of influence D^* to the corresponding obstacles. Such a function is defined in [91], depending on the distance $D(\boldsymbol{q})$ to the closest obstacle and a scaling factor η, resulting in

$$U_{\text{rep}}(\boldsymbol{q}) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{D(\boldsymbol{q})} - \frac{1}{D^*}\right)^2 & D(\boldsymbol{q}) \leq D^* \\ 0 & D(\boldsymbol{q}) > D^* \,. \end{cases} \tag{3.9}$$

This potential increases toward infinity as the robot moves toward the obstacle, resulting in a nonlinear dependency between repulsive forces and distance to obstacles. A different repulsive potential function (Fig. 3.3b) with a linear dependency is presented in [23]:

$$U_{\text{rep}}(\boldsymbol{q}) = \begin{cases} \frac{1}{2}\eta\left(D^* - D(\boldsymbol{q})\right)^2 & D(\boldsymbol{q}) \leq D^* \\ 0 & D(\boldsymbol{q}) > D^* \,. \end{cases} \tag{3.10}$$

By only using the distance to the closest obstacle in the environment, the solution may become numerically unstable in regions where obstacles are close together and the distance function starts to oscillate between multiple obstacles. This can be avoided by introducing multiple repulsive potentials for the individual obstacles.

Constructing a complete representation of the configuration space is usually not possible for higher dimensions and even for trivial scenarios often quite expensive. Calculating forces in the workspace on the other hand is much simpler, be it in the form of sensors or calculations based on representations in three-dimensional space. Forces in the workspace can further be mapped to the configuration space using the robot's Jacobian (Sect. 2.1.3). While a mobile platform is often represented by a single rigid body, a robot manipulator consists of several articulated rigid bodies. Depending on the number of degrees of freedom of the robot, a set of control points on its rigid bodies are defined. The attractive and repulsive potentials are calculated

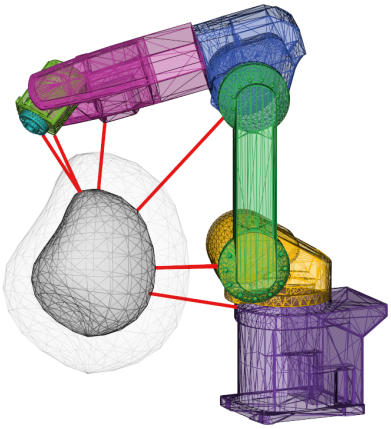


Fig. 3.4 Repulsive forces in workspace: The *red lines* illustrate the minimum distances of an obstacle to the body parts of the robot in a given joint configuration. If a distance is below a specified threshold (*transparent bubble*), the distance is used in order to compute virtual forces on the robot

and finally added in the configuration space after transformation with the Jacobian. Distance calculation between the robot and its obstacles can be used to add floating control points (Fig. 3.4). As only the current configuration of the robot and the environment affect its behavior, this method is also applicable for online control.

3.2 Sampling-Based Motion Planning

Especially with higher dimensional configuration spaces, creating a full representation becomes impractical. Instead, sampling-based motion planners try to avoid this issue by only sampling a big enough subset that will suffice in finding a solution path. The planners have limited access to the configuration space through a separate collision detection procedure. Various geometry models and spaces can be approximated by this, from simple two-dimensional ones up to three dimensions or even more abstract scenarios. Many different collision detection implementations are available, often being able to use CAD or simplified 3D model representations for the robot and its environment (Sect. 2.2). Sampling-based planners benefit from avoiding a complete representation of the configuration space, not from relying on random or grid-based sampling—although these are commonly used in many planners. A small subset of sample points can often be sufficient for solving every possible query for a given scenario, but finding them would require complete knowledge of the configuration space. A more intelligent selection of samples can help increase the planning performance [27].

If a solution to a planning query exists, a complete planner must be able to find a path within a finite amount of time. Implementing a complete planner however is impossible for higher dimensions. The advantage of sampling-based planners derives from only using an approximate representation of the whole configuration space; they therefore feature a different type of completeness. If a solution to a query exists, the planner will eventually find it as he continues to take more sam-

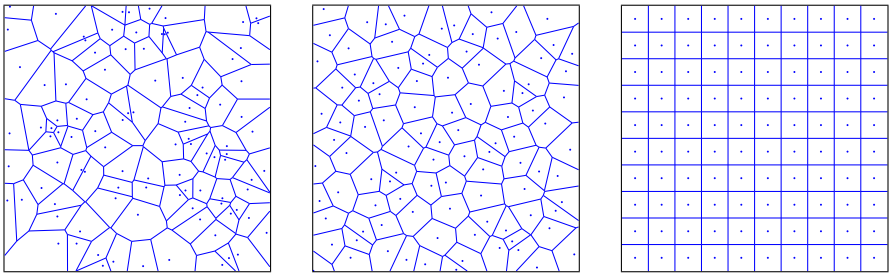


Fig. 3.5 Sampling strategies and their corresponding Voronoi diagrams (100 sample points each). **a** uniform distribution, **b** Halton sequence, **c** grid-based

ples of the configuration space. This is called *probabilistic completeness* [144] in the case of random sampling and *resolution completeness* for grid-based sampling schemes (Fig. 3.5).

Problems often arise in the form of so called narrow passages. Scenarios may have many large open spaces where many collision free samples are generated. These samples however will not help in solving queries that depend on placing vertices in very small areas surrounded by obstacles. The robot's movement is very restricted in these areas and the probability of successfully placing a sample there is very low. Due to the random nature of most planners, performance can differ greatly between repeated runs. However, practical real life scenarios (e.g., in industrial settings) can typically be solved easily by these planners.

As a sampling-based planner's result often does not represent the shortest, smoothest, or most efficient path, its quality can often be improved by different forms of optimization in a post processing step [58]. A suitable collision free trajectory can also be generated here. Especially for free floating robots, choosing good metrics (Sect. 2.1.2) can be very important for the planner's performance [93].

3.2.1 Probabilistic Roadmaps

Probabilistic Roadmaps (PRM) [84, 85] present a typical multi-query planning approach. They benefit from the fact that testing a single configuration for collision is comparatively inexpensive and try to create an overview approximation of the complete space rather than focusing on an actual path planning query. This is especially useful in static environments, where the roadmap can be reused for future queries.

The basic structure of a PRM is based on a graph representation. Building a roadmap for a given scenario starts with a construction phase (Algorithm 1). An empty graph is filled with a specified amount n of collision free samples. After adding all samples to the list of vertices, the nearest k neighbors of each vertex are

Algorithm 1: CONSTRUCT

Input: n, k, r
Output: $G = (V, E)$

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
while $|V| < n$ **do**
  repeat
    $\boldsymbol{q} \leftarrow \text{RAND}()$
  until $\boldsymbol{q} \in \mathcal{C}_{\text{free}}$
  $V \leftarrow V \cup \{\boldsymbol{q}\}$
forall the $q \in V$ **do**
  INSERT$(G, \boldsymbol{q}, k, r)$
return G

Algorithm 2: INSERT

Input: $G = (V, E), \boldsymbol{q}, k, r$
Output:

$N_{\boldsymbol{q}} \leftarrow \text{NEAREST}(G, \boldsymbol{q}, k, r)$
forall the $\boldsymbol{q}' \in N_{\boldsymbol{q}}$ **do**
  if $(\boldsymbol{q}, \boldsymbol{q}') \in \mathcal{C}_{\text{free}}$ **then**
    $E \leftarrow E \cup \{(\boldsymbol{q}, \boldsymbol{q}')\}$

Algorithm 3: SOLVE

Input: $G = (V, E), \boldsymbol{q}_{\text{start}}, \boldsymbol{q}_{\text{goal}}, k, r$
Output:

$V \leftarrow V \cup \{\boldsymbol{q}_{\text{start}}\}$
INSERT$(G, \boldsymbol{q}_{\text{start}}, k, r)$
$V \leftarrow V \cup \{\boldsymbol{q}_{\text{goal}}\}$
INSERT$(G, \boldsymbol{q}_{\text{goal}}, k, r)$
while $(\boldsymbol{q}_{\text{start}}, \ldots, \boldsymbol{q}_{\text{goal}}) \notin G$ **do**
  repeat
    $\boldsymbol{q} \leftarrow \text{RAND}()$
  until $\boldsymbol{q} \in \mathcal{C}_{\text{free}}$
  $V \leftarrow V \cup \{\boldsymbol{q}\}$
  INSERT$(G, \boldsymbol{q}, k, r)$

calculated. The selection of neighbors can optionally be restricted to fall within a given radius r of the distance metric in use. Every sample point is tested for connectivity with each of its neighbors. For this, a simple straight line edge verifier with a specified step size is commonly used. The intermediate collision checks between the two vertices can be performed in sequential order or by using subdivision (e.g., the van der Corput sequence [36]). The latter often requires fewer tests to report a collision. Only edges between vertices that are not already in the same connected

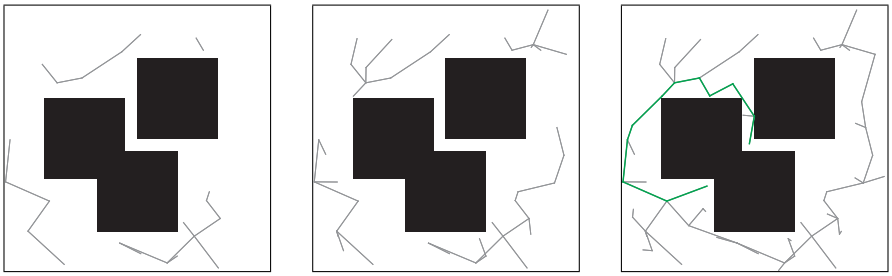


Fig. 3.6 Behavior of a Probabilistic Roadmap planner with uniform sampling in a two-dimensional scenario with three square-shaped obstacles. Several connected components are formed until a collision free path (*green*) is found using a graph search algorithm

component of the graph are usually added to the graph in order to avoid cycles in the graph. However, having a certain amount of redundancy in the form of cycles can be beneficial in dynamic environments, where edges might have to be removed again later due to collisions.

In order to solve a specific path planning query after the initial roadmap has been constructed, the given start and goal configurations have to be connected to the graph (Algorithm 3). The calculation of their nearest neighbors and the insertion into the graph are handled in the same way as during the construction phase (Algorithm 2). The path planning query is successful if the start and goal configurations are in the same connected component of the graph afterwards (Fig. 3.6). A search algorithm can then be used to extract the shortest collision free path (e.g., Dijkstra's algorithm). Otherwise, the roadmap has to be extended until a collision free path can be found. Although this is not its primary application, the PRM planner can be modified to perform single-query requests by omitting the initial construction phase.

Initially, uniform sampling was used for the creation of sample points. This has proven to be good for a variety of problems and ensures probabilistic completeness. For scenarios with narrow passages however, many free configurations are found and added in large open areas that do not help in improving the quality of the roadmap. The probability of placing a sample in a narrow passage that is required for solving a query however is very low. As the planner tries to create a representation of the whole configuration space, it also places samples in areas that are not even connected to later start and goal configurations. Large roadmaps also lead to slow performance when it comes to finding nearest neighbors. This can be partially circumvented through different connection strategies [61] and advanced methods such as k-dimensional trees [155]. An ideal roadmap would only include as few vertices as necessary for solving all possible path planning queries. Various sampling strategies have been introduced to improve the quality of the roadmap and to reduce the number of sample points and vertices.

In order to have more vertices in regions close to obstacles, Gaussian sampling [19] creates random samples until a collision is detected. A second sample point close to the previous one is then chosen based on a Gaussian distribution and

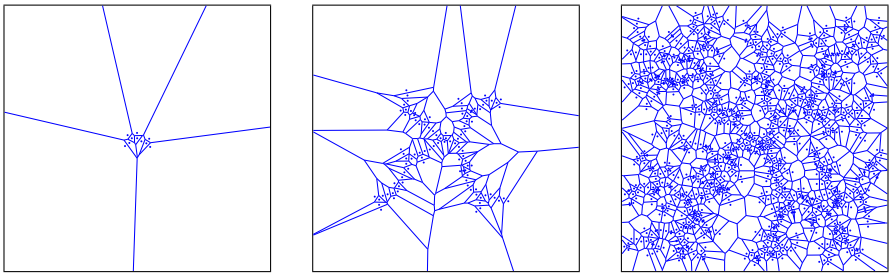


Fig. 3.7 Implicit Voronoi bias in a Rapidly-Exploring Random Tree. The series shows the state with 10, 100, and 1000 vertices. An expansion quickly covers the configuration space before increasing density in other areas

added to the roadmap if it is within $\mathcal{C}_{\text{free}}$. Choosing a standard deviation that is too large will result in an almost uniform distribution.

Similar to this, the bridge test [73] tries to increase the number of sample points in narrow passages by looking for a free configuration in the middle of a line between two collision samples. The endpoints of the *bridge* across the narrow passage are located within $\mathcal{C}_{\text{obst}}$, while the free sample point *hovers* in $\mathcal{C}_{\text{free}}$. The first endpoint is chosen uniformly at random, while a Gaussian distribution with an appropriately chosen standard deviation σ is usually used for all dimensions of the configuration space for finding the second one. Additional sample points with uniform distribution can be introduced depending on a specified probability during sampling.

Other strategies include visibility-based roadmaps [113, 139], where guard and connector nodes are introduced to prevent the creation of too many vertices in open free spaces, or lazy evaluation of roadmap vertices and edges [18]. Utility-guided [26] sampling uses information from colliding nodes instead of ignoring this valuable data. However, selecting good parameter values becomes increasingly difficult with the number of settings and choosing the best sampling technique for a specific scenario is equally important [60].

3.2.2 Rapidly-Exploring Random Trees

Single-query planners focus on solving one specific path planning query instead of creating an approximation of a current scenario for future problems. Optimally, the planner only has to cover areas of the configuration space relevant for the current request. Search trees with a random sequence that try to cover the configuration space with increasing resolution are called Rapidly-Exploring Random Trees (RRT) [98].

RRTs were designed to require a minimum number of parameters as opposed to other planners that require a large amount of fine tuning. The planner starts with the construction of a tree that uses the initial configuration as root vertex (Algorithm 4). A random sample configuration within the boundaries of $\mathcal{C}$ is constructed and its

Algorithm 4: RRT

Input: $\boldsymbol{q}_{\text{start}}, \boldsymbol{q}_{\text{goal}}$
Output: $T = (V, E)$

$V \leftarrow \{\boldsymbol{q}_{\text{start}}\}$
$E \leftarrow \emptyset$
repeat
  $\boldsymbol{q}_{\text{new}} \leftarrow \text{RAND}()$
  $\boldsymbol{q}_{\text{near}} \leftarrow \text{NEAREST}(T, \boldsymbol{q}_{\text{new}})$
  if $\boldsymbol{q}_{\text{new}} \leftarrow \text{EXTEND}(T, \boldsymbol{q}_{\text{near}})$ **then**
    $V \leftarrow V \cup \{\boldsymbol{q}_{\text{new}}\}$
    $E \leftarrow E \cup \{(\boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{new}})\}$
until $\boldsymbol{q}_{\text{new}} \equiv \boldsymbol{q}_{\text{goal}}$
return T

Algorithm 5: EXTEND

Input: $T, \boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{new}}$
Output: $\boldsymbol{q}_{\text{new}}$

$\Delta\boldsymbol{q} \leftarrow |\boldsymbol{q}_{\text{near}} - \boldsymbol{q}_{\text{new}}|$
$\boldsymbol{q}_{\text{new}} \leftarrow \boldsymbol{q}_{\text{near}} + \Delta\boldsymbol{q}$
if $\boldsymbol{q}_{\text{new}} \in \mathcal{C}_{\text{free}}$ **then**
  return $\boldsymbol{q}_{\text{new}}$
else
  return $\varnothing$

Algorithm 6: CONNECT

Input: $T, \boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{new}}$
Output: $\boldsymbol{q}_{\text{new}}$

while $\boldsymbol{q}_{\text{new}} \leftarrow \text{EXTEND}(T, \boldsymbol{q}_{\text{near}})$ **do**
  $\boldsymbol{q}_{\text{near}} \leftarrow \boldsymbol{q}_{\text{new}}$
return $\boldsymbol{q}_{\text{new}}$

nearest neighbor in the tree is identified. The planner then tries to extend the existing tree toward the new configuration from the point of this closest vertex in the tree (Algorithm 5). The configuration on the line between these two points and a specified distance (step size) away from the nearest neighbor is tested for collision. If the tested configuration is within $\mathcal{C}_{\text{free}}$, it is inserted into the tree and connected to the neighbor vertex. Afterwards, a new random sample is created and the planner continues until it can connect to the goal configuration (Fig. 3.8). In order to bias the planner more toward the target, it can be modified to choose the goal configuration instead of a random sample with a certain probability. Planning performance will begin to suffer as the cost of the nearest neighbor search increases rapidly with a growing number of vertices [155]. The planner was also designed with kinodynamic constraints in mind [100]. A state of the planner, represented by a vertex in

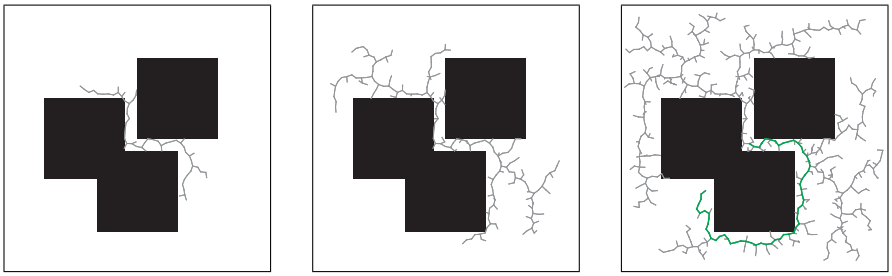


Fig. 3.8 Rapidly-Exploring Random Tree with extend step and a single tree. The planner explores the two-dimensional configuration space by trying to add random samples to the closest neighbor in the tree until start and goal configuration are connected (*green*)

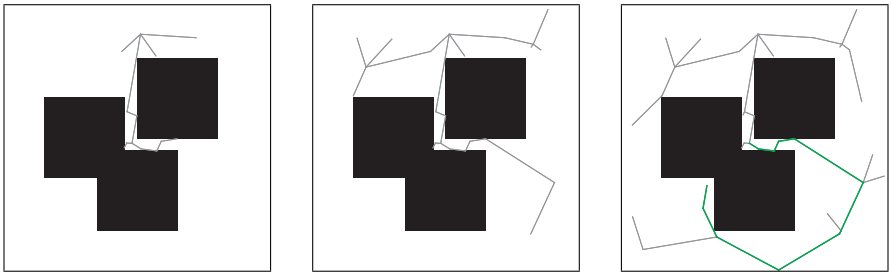


Fig. 3.9 Connect step and goal bias in a single-tree RRT planner. Extension of the tree is only stopped after reaching the sampled configuration or colliding with an obstacle. Additionally, the goal configuration is chosen instead of a random value for some connection attempts

the tree, usually consists of a position in the configuration space. It can however also be extended with velocity or other dynamic state informations. The extension step of the tree can then be replaced with a time step rather than a simple step size. Kinematic constraints can also be considered here.

During planning, an implicit Voronoi bias is shown. The tree expands rapidly in the beginning, as the probability of choosing a vertex as the nearest neighbor is proportional to the size of its Voronoi region. It continues to cover the whole configuration space with increasing density as the planning progresses (Fig. 3.7), leading to a probabilistic complete planner. This can however also be counterproductive. Depending on the obstacles in a scenario and the chosen sampling region, the tree can be biased to grow in regions close to obstacles. If the Voronoi region of vertices in these areas is very large as opposed to those next to openings, they have a higher probability of being selected for expansion. In order to overcome this, a limitation for the sampling domain of boundary points was introduced [154]. Vertices for which the expansion fails are assigned a maximum search radius that is used when finding the nearest neighbor. A planner with an unlimited maximum radius is equal to a standard RRT. An adaptive version of this method was presented in [82].

A greedier version of the planner was introduced in [94] (Fig. 3.9). Here, the planner does not stop extending the tree after only one step, but rather tries to continue toward the new sample until reaching an obstacle (Algorithm 6). In order to reduce the number of vertices and increase nearest neighbor performance, only the last collision free configuration on this line can be inserted into the tree. It also features the introduction of a second tree that is grown from the goal configuration. The planner switches between the two trees as one of them is expanded as usual while the other one tries to connect the two trees. The greedy or the traditional strategy can be selected for either one depending on the planning scenario.

RRT-based planners and PRMs can be combined in a roadmap of trees [120], where the RRT replaces the basic local planner in a PRM algorithm. Utility information has also been integrated with RRTs and applies to elements such as guided exploration, node expansion, expansion direction, or exploration distance [29]. Statistical evaluation in the form of Principal Component Analysis (PCA) may favor certain expansion directions based on tree growth [40].

3.3 Workspace Information

A mapping from high-dimensional solutions in $\mathcal{C}$ to the low-dimensional space $\mathcal{W}$ is given via the workspace volume swept by the robot along its trajectory defined in the configuration space. The workspace features a fixed dimension with lower complexity and provides an explicit geometric representation. The two spaces are connected through the Jacobian matrix, with the total swept-volume of a robot system defined by the collection of all possible motions, its geometry, and joint limits. Advanced collision detection calculations such as distance or penetration computation can be used in order to gather additional information for motion planning algorithms.

3.3.1 Decomposition-Based Motion Planning

This planning algorithm divides the path-finding problem into two parts [20, 21, 22]. First, a connected volume in the workspace is captured and then a navigation function is imposed onto it. Calculations in the low-dimensional workspace can be performed at a relatively low cost compared to a high-dimensional configuration space. This leads to solving a global problem of capturing sufficient workspace connectivity and a local problem of using this information in order to create a corresponding motion in the high-dimensional configuration space. Similar to the reduced completeness definition of sampling-based planners, this method results in a tradeoff between efficiency and completeness. Various common planning scenarios with large clearances between obstacles can be solved in real-time, whereas solving more complex problems may fail completely.

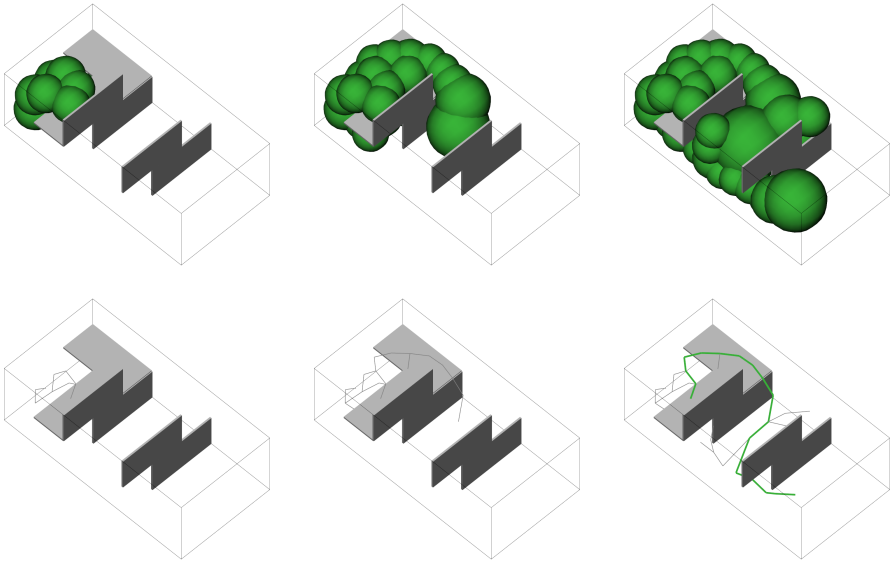


Fig. 3.10 Evolution of the wavefront expansion in three steps with start position in the upper left and goal position in the lower right section. Distance computation in three-dimensional workspace provides information on narrow sections and connectivity

Several approaches and algorithms can be used to handle the two subproblems. For determining a workspace connectivity volume that can be used as guide for the local planning part, a wavefront expansion algorithm is introduced. Distance computation in the workspace is used to determine a sequence of spheres that expand in an omnidirectional wave from the start position (Fig. 3.10). During this expansion, the connectivity information of the spheres is captured using a tree. Large areas of free space are represented by fewer spheres with a big radius, while narrow sections require a higher number of smaller spheres. Compared to other methods, such as trapezoidal decomposition, this approach only explores parts of the environment necessary for finding a solution. It can also be used with dynamic environments through the integration of sensor readings. As the expansion starts from the current position of the robot rather than the target, it can already begin to move without waiting for the planner to finish.

The wavefront expansion starts with an empty tree T and an empty priority queue Q (Algorithm 7). First, the distance of the start position to its closest obstacle in the workspace is calculated. The resulting sphere uses this distance as its radius and the start position as its center. It is then inserted into the priority queue with a priority of the distance of its center to the goal position minus its radius. The next steps are repeated until the queue contains no more spheres. The sphere in the queue with the highest priority is removed and added to the list of vertices. It consists of a center position, a radius, and a parent sphere to which it is now connected

Algorithm 7: WAVEFRONT

Input: $\boldsymbol{p}_{\text{start}}, \boldsymbol{p}_{\text{goal}}, n$
Output: $T = (V, E)$

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
$Q \leftarrow \emptyset$
$S = (\boldsymbol{p}_{\text{start}}, r, \varnothing) \leftarrow \text{DISTANCE}(\boldsymbol{p}_{\text{start}})$
$\text{INSERT}(Q, S, \|\boldsymbol{p}_{\text{goal}} - \boldsymbol{p}_{\text{start}}\| - r)$
repeat
  $S = (\boldsymbol{p}_{\text{center}}, r, S_{\text{parent}}) \leftarrow \text{POP}(Q)$
  $V \leftarrow V \cup \{S\}$
  $E \leftarrow E \cup \{(S_{\text{parent}}, S)\}$
  if $\|\boldsymbol{p}_{\text{goal}} - \boldsymbol{p}_{\text{center}}\| < r$ **then**
    return T
  $P \leftarrow \text{RAND}(S, n)$
  forall the $\boldsymbol{p}_i \in P$ **do**
    forall the $S_i = (\boldsymbol{p}_{\text{center}_i}, r_i, S_{\text{parent}_i}) \in V$ **do**
      if $\|\boldsymbol{p}_i - \boldsymbol{p}_{\text{center}_i}\| < r_i$ **then**
        $S' = (\boldsymbol{p}_i, r', S) \leftarrow \text{DISTANCE}(\boldsymbol{p}_i)$
        $\text{INSERT}(Q, S', \|\boldsymbol{p}_{\text{goal}} - \boldsymbol{p}_i\| - r')$
until $Q \equiv \emptyset$
return $\varnothing$

through an edge in the tree. If the distance of the center position to the goal is less than the radius of the sphere, the expansion was successful and the algorithm returns the tree with the path of spheres through the workspace. Otherwise, n random sample positions are created, that are uniformly distributed on the current sphere. All samples that are contained within any sphere in the current tree are discarded. For the remaining points, new spheres are created by calculating their minimum distances to obstacles in the workspace. They are added to the priority queue with the current sphere as parent and by using their distance to the goal minus the radius as priority. An alternative method can use the reciprocal of the radius as priority, resulting in a greedy optimization for space rather than distance. It is also possible to specify a minimum radius for new spheres. All spheres that fall below a specified threshold are discarded.

Assuming the whole robot can be contained within a tunnel of spheres toward the goal position, a valid path in configuration space for the robot exists. While computing a navigation function analytically is difficult, the wavefront expansion and its tree of spheres can be used to define a set of potential functions with no local minima. In a sequence of spheres, the center of a node has been sampled on the boundary of its parent during construction. As long as the robot is contained within the parent sphere, the robot's attractor potential is defined by its distance to the center of the next node in line. Upon reaching this sphere, the potential switches to the succeeding sphere in order to move through the tunnel until reaching the goal. By translating the resulting force $\boldsymbol{f}_{\text{task}}$ to joint torques $\boldsymbol{\tau}$ using the Jacobian

matrix $\boldsymbol{J}$, a collision free trajectory can be generated. In addition, reactive obstacle avoidance can be handled during execution through the incorporation of repulsive forces $\boldsymbol{f}_i$ on various points of the robot:

$$\boldsymbol{\tau} = \boldsymbol{J}^{\mathrm{T}} \left(\boldsymbol{f}_{\mathrm{task}} + \sum_i \boldsymbol{f}_i \right). \tag{3.11}$$

This can however lead to undesired changes in the task behavior. Manipulators with many degrees of freedom can compensate for this by using only task-independent joints for obstacle avoidance. Together with the Jacobian's dynamically consistent generalized inverse $\bar{\boldsymbol{J}}$, the repulsive forces can be projected into the nullspace and combined with the attractive force of the task with

$$\boldsymbol{\tau} = \boldsymbol{J}^{\mathrm{T}} \boldsymbol{f}_{\mathrm{task}} + (\mathbf{1} - \boldsymbol{J}^{\mathrm{T}} \bar{\boldsymbol{J}}^{\mathrm{T}}) \sum_i \boldsymbol{J}_i^{\mathrm{T}} \boldsymbol{f}_i \,. \tag{3.12}$$

Due to the tradeoff between completeness and efficiency, not all navigation functions that are generated through the wavefront expansion lead to a solution for the local planner (e.g., due to structural local minima of the robot or complicated workspaces). In these cases, a different planner has to be used.

3.3.2 Adaptive Sampling Density

Probabilistic Roadmaps tend to create more vertices in large regions of free space rather than placing samples in the narrow passages required for finding a valid path. Other than uniform sampling, several non-uniform sampling strategies in configuration space have been introduced in order to improve this [19, 113, 139, 73, 27, 59]. For free-flying robots, information gathered from the workspace can also be used to enhance planner performance. By sampling close to the medial axis of the workspace, a balance between open areas and narrow passages can be achieved. The probability of creating collision free samples increases with the distance to obstacle boundaries, especially for constrained areas.

In [50], a generalized Voronoi diagram (GVD) is created for the workspace. The corresponding graph representation is then used to find a preliminary path with maximum clearance for a point robot. In order to choose an orientation for each point on the workspace path, a major axis for the actual robot shape is calculated and fitted to the tangent vector of the path. The orientation about the axis can be chosen freely and is varied constantly across the path. A simple local planner tries to connect the configuration along the path using direct line connections. For segments that have been marked invalid afterwards, a single-query randomized planner with a restricted configuration space for the positional degrees of freedom is used [75, 76]. These limitations are removed if no solution is found after a specified number of iterations.

The framework in [70] also uses GVDs and defines handle points for a robot's geometry. The planner randomly samples a pose in the configuration space and then tries to match its points close to the medial axis using a physical model of springs. Afterwards, random walks [71] are used to plan local paths between the free configurations created from this step.

With an algorithm for discrete GVDs accelerated by graphics hardware [69], the method in [119] also places samples near the medial axis. A local planner similar to the one in [2] is used to generate connections; otherwise new nodes are added on the midpoints of Voronoi edges. The distance information in the diagram is used to identify and distinguish two types of narrow passages: those that can be passed by translational movement alone and those that also require changes in orientation. The type depends on the size of the largest dimension of the robot's oriented bounding box compared to those of the narrow passage. Different strategies are utilized when dealing with these areas. Uniform sampling increases the density of samples, while a Gaussian distribution for positional degrees of freedom and uniform sampling for the orientation results in a bias toward the medial axis. Finally, by adapting the orientation toward the tangent of the Voronoi graph with a Gaussian distribution, the robot's dimensions are oriented to fit narrow passages.

For dynamic environments, [57] shows an algorithm using a GVD and with various constraints on rigid bodies. These can be used to follow an estimated path, to link bodies together in the case of an articulated robot, to comply with joint limits, and for several other tasks. They are classified into two categories, with hard constraints that always have to be satisfied and optional soft constraints. The algorithm of [69] is used again for a fast approximation of the GVD and serves as global information for the planner. It also provides distance information that is used in a soft constraint to align the robot with the medial axis. The constraint solver uses an iterative relaxation method.

Inspired by the watershed transform algorithm from image processing [149], approximate cell decomposition with octrees is applied in [13, 14] in order to identify large collision free regions and label connecting space as narrow passages. Using watershed segmentation, cells are grouped into regions with a label and weight. All labeled regions are given the same weight in order to increase sampling in smaller regions, as a value corresponding to cell volume would result in uniform sampling.

A Delaunay tetrahedralization of the workspace is computed in [95] to locate narrow passages and guide PRM sampling in the configuration space. The tetrahedra are given a rating based on their size, with a higher likelihood of belonging to narrow passages for lower values. The number of samples per chosen tetrahedra depends on their rating. Also, tetrahedra in these areas are usually smaller, resulting in a higher probability of being selected with a uniform distribution compared to larger ones. The positional components of a rigid body are sampled uniformly within the chosen tetrahedra. A separate random orientation with uniform distribution is used for the rotational component. For articulated robots, the position and orientation of the wrist are used instead and a number of matching configurations are calculated using the robot's inverse kinematics.

The approximate medial axis algorithm presented in [151] is used in [150] in order to achieve a higher sampling density for PRMs in narrow passages. By using partially overlapping maximum spheres, calculating the medial axis is relatively inexpensive compared to other methods such as GVDs. A number of points close to the medial axis are identified and serve as an approximation to the medial axis. Multiple random configurations are generated in the vicinity of each point. The distance between points on the axis and handle points on the robot's configuration is minimized with the help of potential fields.

3.3.3 Disassembly

The method described in [151, 152] decomposes the path planning problem into three parts. After determining workspace connectivity, this information is used to locate restricted configurations in narrow passages. Traditional sampling strategies for PRMs usually fail to place collision free samples inside these regions. These configurations are finally disassembled to create a valid path. As good samples in constrained regions have already been placed in the previous step, it is much easier for the planner to find a solution.

The wavefront expansion algorithm used in [20, 21, 22] is applied to create connectivity information restricted to the actual planning query (Algorithm 7). A resulting tunnel of spheres (Fig. 3.10) represents a possible workspace volume for the robot to move through. The minimum radius of the spheres can be adjusted to fit the robot's size. Also, additional tunnels can be generated if no solution can be found for a specific one in the next steps of the planner, eventually resulting in a complete exploration of the workspace.

Unrestricted configurations in free space are rather easy to generate for a sampler. To find a solution in difficult scenarios, the hard task for the planner is to place the robot in a position relevant to the current query with little room for movement. Identifying the corresponding regions in workspace and creating this assembly pose is equal to finding a narrow passage in configuration space. For the workspace tunnel, narrow sections are represented by spheres that fall below a threshold based on the robot's geometry. These regions are then explored in more detail and the watershed segmentation from [149, 13, 14] is used to group the spheres and label the regions accordingly. In an assembly, a robot is either completely contained within a narrow passage or can reach across it. The latter is achieved by specifying handle points on the robot's geometry and generating a collision free configuration where they are contained in regions on different sides of a watershed sphere. The positional components of the free-flying robot are limited so that its chosen reference point coincides with the narrow passage.

In its last phase, the planner does not simply try to connect the set of restricted configurations, but further guides sampling using the gathered workspace information. The positional components are biased to move the robot into open regions defined by the workspace tunnel and only short incremental motions are used due

to constraints in the area surrounding narrow passages. The probability of finding collision free samples around assemblies increases with movement toward less constrained regions, creating a bias for disassembly. This procedure is complete once the robot has left the narrow sections of the tunnel and standard PRM sampling takes over. Connecting the separate components in the remaining open areas is of lower complexity.

3.3.4 Elastic Roadmaps

Potential functions are often used in order to combine robot control with reactive obstacle avoidance. They are however very susceptible to local minima. Navigation functions can overcome this issue with the help of global information. By combining a set of local minima free potential functions, a robot can safely reach its target without violating real-time feedback requirements. Calculating global navigation functions can be very challenging and they become invalid with dynamic changes in the environment. As long as a given global motion stays valid, approaches such as the Elastic Strips framework [23] can locally modify the path in order to react to external disturbances and comply with required task constraints. They are not able to create a new global solution that can overcome restrictions in local control or take advantage of new routes.

In contrast to traditional roadmaps in motion planning, where static vertices and edges capture global information in a graph, [153] presents the concept of Elastic Roadmaps with a set of moving collision free milestones that are able to satisfy task constraints. Milestones are placed in the vicinity of obstacles to provide better coverage of free space. Feedback controllers are used to navigate between milestones, thereby creating a hybrid system of potential functions. By using workspace information and trading completeness for efficiency in real-life scenarios, the planner complies with feedback requirements.

A small amount of vertices in a roadmap can be sufficient to provide good coverage of the configuration space. Large quantities of redundant samples in open areas of free space usually do not provide additional information. The quality of a vertex is mainly defined by the amount of free space visible to it [74]. Samples within narrow passages tend to have less visibility but are often essential in finding a valid path. A sample with good visibility can replace several others in its vicinity of lesser or similar quality [113, 139]. Also, a planner's task in an environment filled with objects is to find a way around these obstacles. If all obstacles and their boundaries are known to the planner, he can strategically place samples around them and look for a solution, as the remaining space between obstacles is free [2]. By following the movement of all obstacles and updating the corresponding milestones, the planner can react to dynamic changes in the environment. Determining either the visibility rating of a sample or the exact boundaries of obstacles in the configuration space however is not applicable to higher dimensions, but these properties can be approximated in the workspace at a much lower complexity. Milestones in

an Elastic Roadmap not only have to avoid collisions, they also have to satisfy the other constraints of the current task. Two different types of end effector movements are distinguished, where either the position of the end effector is only relevant at the goal position, or its movement is also restricted along the way. For the first kind, a configuration close to an obstacle is selected and the end effector is attracted to a feature on the object by using a combination of task-level controllers [134]. Bounding boxes around convex workspace obstacles are used to determine features that the end effector can move freely in-between. Other tasks involve posture potentials, obstacle avoidance, as well as imminent collision prevention and kinematic joint limit restrictions:

$$T_{\text{posture}} \lhd T_{\text{avoidance}} \lhd T_{\text{task}} \lhd T_{\text{feature}} \lhd T_{\text{collision}} \lhd T_{\text{kinematic}} \,. \tag{3.13}$$

Milestones are added to the roadmap and marked as valid if they do not collide with obstacles and adhere to kinematic constraints. They are updated constantly during changes to the environment and follow obstacles as they move. The second type of movement uses the point on the robot closest to a feature instead of the end effector and exchanges the order of T_{task} and T_{feature}.

Possible connections between valid milestones are estimated by testing straight line segments between a set of handle points on links of the robot. Graph search algorithms are then used to determine a path among the valid milestones. Actual movement between vertices is achieved through task-level control and may fail to reach the target node, but the planner can recover by invalidating the current edge hypothesis in the graph and choosing a new solution.

3.4 Exploration vs. Exploitation

Building a configuration space representation for higher dimensions is practically impossible. However, the configuration spaces of many practical planning problems contain considerable structure that may help in solving a task. Also, not the whole configuration space has to be explored in order to find a solution. Planners therefore have to gradually explore parts of the configuration space until they can construct a collision free path. Many planners employ sampling strategies with random distributions, but more sophisticated methods have also been introduced to solve difficult queries.

In reinforcement learning [147, 83, 143], an agent interacts with its environment in order to accomplish a certain goal. With information about the current state of the environment, the agent decides to pursue a certain action. Afterwards, it is presented with a new state and a reward value corresponding with this change. This procedure is repeated in every step (Fig. 3.11). In contrast to supervised and many other forms of machine learning, the agent is not told about the best possible action. The action with the most reward has to be identified using trial and error. However, the state of the environment has changed after an interaction, making comparisons

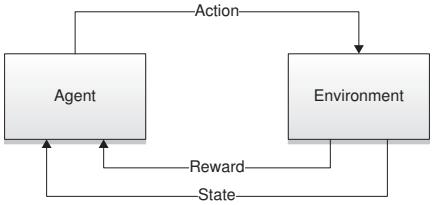


Fig. 3.11 Interaction of an agent with its environment in reinforcement learning. Future actions are influenced by the current state and a reward function. The latter provides feedback on a chosen action for a known state

more difficult. In addition, results of future interactions with the environment might be influenced by a decision as well.

The agent's policy determines which action to take given the state of the environment. It should try to maximize reward values with a certain goal in mind. This goal is defined through a reward function that gives the agent a feedback on the quality of the chosen action for a known state. The agent cannot change the reward function. In case of a low reward, the agent may however decide to modify its policy in order to favor a different action for this state the next time. The quality of actions in the long run is determined by the value function. While the reward function only assesses the change of the current state to the next one, the value function also takes into account all possible future states. An action resulting in a low immediate reward might still lead to a series of states with high rewards or vice versa. Determining a good value function is extremely difficult and involves all the information acquired by the agent throughout his runtime. To simulate the outcome of an action with a given state, a model of the environment can be used during planning. Instead of relying on trial and error, the predicted reward and next state can directly influence the decision making.

Finding an optimal policy requires selecting actions that produce the best sum of rewards. In the beginning, the agent starts with no knowledge and these actions have to be identified at first by trying new and unknown ones. Using existing information equals exploitation and tries to minimize costs, whereas acquiring new information equals exploration and tries to minimize learning time. Exploitation of known information is necessary in order to maximize reward, but only further exploration can lead to actions with even higher reward. The two states contradict each other, which leads to the dilemma of exploration and exploitation tradeoff—an agent cannot ignore either one if he wants to succeed. Questions on how to balance these opposing forces include what kind of information is available, what part of this information is already exploited, and how long this information can or should be exploited.

Exploration can be divided into guided and unguided categories. While the former employs exploration-specific knowledge, the latter usually resorts to a uniform random distribution. Some forms of unguided exploration modify the probability distribution in order to bias action selection, thereby decreasing the likelihood of unfavorable actions. Guided exploration on the other hand does not rely on randomness when selecting actions, but rather tries to select actions with the highest

exploration utility. Various heuristics need to be employed when making these decisions, as the actual information cannot be determined directly.

3.4.1 Exploration and Exploitation in Motion Planning

Similar to work in reinforcement learning, the motion planning problem can also be cast as a state search problem, together with its two competing goals of planning. The scenario here usually involves a robot in an environment with a kinematic, dynamic, and geometric description. The planner can examine the environment, move the robot, and receive feedback through various forms of collision detection and distance or penetration depth calculations between objects.

In motion planning, *exploration* seeks to understand the connectivity of configuration space, regardless of a specific goal or task. It tries to gain the maximum amount of knowledge possible by improving the planner's understanding of configuration space, but also wastes time exploring sections of the environment irrelevant to the actual task. The initial roadmap building phase of the basic PRM planner with uniform random sampling [84] is a typical example for this unguided form of exploration. Many sampling-based planners today employ *guided exploration* techniques. They improve on pure, unguided exploration by leveraging available information that is based on characteristics of the underlying state space. However, this guidance is still directed at achieving an efficient, complete understanding of the configuration space rather than accomplishing a specific task. *Exploitation* on the other hand tries to find a valid path for a particular task as efficiently as possible based on available information. It does not try to reach a full understanding of the whole space and just begins to act while assuming it has sufficient knowledge to solve the current task. Such pure exploitation can be observed in the artificial potential field approach [91].

Motion planning will be most efficient when exploration and exploitation are adequately balanced. Although a lot of sampling-based motion planners combine exploration and exploitation, all of these planners employ the two competing goals as distinct steps of the planning process (Table 3.1) rather than deliberately balancing them within a unified framework. Exploration is required to understand the connectivity of relevant configuration space regions whereas exploitation should be used whenever greedy actions can solve sub-problems quickly. Given the local and global properties of a particular configuration space and task, a planner can achieve computational efficiency by appropriately employing both strategies.

The original PRM planner with uniform random sampling performs pure exploration [84]. Its exploratory behavior is not affected by the task or by information obtained during the exploration. The refinement step of PRM planners however constitutes guided exploration. A large number of sampling-based multi-query motion planners perform guided exploration. They assess properties of regions of configuration space to guide exploration. These properties can depend on obstacles [2, 19], visibility [139], or narrow passages [73]. Other planners use workspace informa-

Table 3.1 Exploitation vs. exploration in a selection of motion planning algorithms. All of these examples employ the two opposing goals as distinct steps rather than actively balancing them during the planning process

Planner	Exploration			Exploitation
Potential Fields				*Potential Field*
RPP	*Brownian Motion*			*Potential Field*
PRM	*Construct*		*Refine*	
RRT		*Voronoi Bias*		*Extend*
RRT-Connect		*Voronoi Bias*		*Extend+Connect*
Lazy/Fuzzy PRM			*Resample*	*Graph*

tion to adapt exploration [50, 70, 14, 150]. In another planner, global information about the entire configuration space is used to guide exploration [27]. Artificial potential field approaches employ pure exploitation [91]. The complete elimination of exploration makes these methods computationally efficient but also susceptible to local minima. This is also true for potential field approaches that are applied to entire paths [5]. Fuzzy PRM [112] and Lazy PRM planners [18] initially perform exploitation in a random configuration space graph. When this exploitation fails, these planners perform guided exploration to augment the graph in difficult regions. A number of planners alternate between exploration and exploitation. The Randomized Path Planner (RPP) combines potential field-based exploitation with random exploratory moves [12]. RRT planners alternate exploration based on the Voronoi bias with exploitation in the extend step [98]. When RRT planners employ two trees, a second exploitative step is taken when the planner attempts to connect both trees [94]. Variants of RRT planners replace exploration based on the Voronoi bias with guided exploration [29, 82, 130]. Other planners combine exploration and exploitation in workspace and configuration space. Some planners initially perform efficient exploitation in the low-dimensional workspace and subsequently use the obtained information to perform exploitation [22, 153] or guided exploration [152] in configuration space. Another planner performs workspace exploration and uses the resulting information to perform guided configuration space exploration [122, 123].

3.4.2 Balancing Exploration and Exploitation

This chapter has already demonstrated a large variety of solutions for motion planning based on either exploration or exploitation; however no solution has yet been presented on how the opposing forces can actually be combined in order to address more challenging scenarios. Such a planner requires a mechanism that allows it to continuously shift between exploration and exploitation extremes based on some form of information and a way to gather this kind of data. It also raises the question what additional measurement values can be used in order to make these decisions

in a specific situation. Possible actions of the planner range from gathering additional information, directing exploration and exploitation, up to a full balance of these methods.

The computation of local geometry information ranges from a simple collision check of a robot configuration to the calculation of penetration depth, distance estimation, or other advanced functions. These operations can be used in order to gather connectivity information in workspace, rate configurations based on the information how links in the chain are colliding, or in an estimation of the proximity to obstacles and self-collision. While sophisticated methods provide more information about the local geometry, they also suffer from higher computational costs and should therefore only be used when additional information can actually improve the planning step. This also includes the differentiation and assessment between methods that approach complete solutions for all possible queries and those that focus on an actual task. Although the calculation of local free space information is more expensive than pure intersection computation, this data can also be used for obstacle avoidance in a refined step size and may then actually increase performance.

Other factors can include an adjusted rather than a fixed number of nearest neighbor nodes, distance metrics that observe the influence of joints on the overall movement, a disregard of exhausted nodes that have resulted in a high number of failed connections or an adaptive impact value, and sampling methods directed by utility information from the current node distribution and colliding configurations. Workspace connectivity data with an optional weighting factor can provide insights for difficult scenarios with narrow passages, as can alternative paths using reduced geometries.

3.4.3 Exploring/Exploiting Trees

This section introduces Exploring/Exploiting Trees (EET), a tree-based motion planning algorithm that performs exploitation whenever possible and gradually transitions to exploration when necessary [126]. The planner is based on tree expansion in configuration space, similar to RRT methods [98, 94] introduced in Sect. 3.2.2. Its main design objective is to carefully balance exploratory and exploitative behavior so as to leverage the structure inherent in the planning problem for rendering motion planning as efficient as possible. In order to accomplish this, the planner should behave like a potential field planner whenever possible and gradually turn into a complete motion planner when required.

The EET planner leverages several sources of information in order to perform exploitation and to balance between exploitation and exploration. Guided exploitation is performed by acquiring global connectivity information for relevant portions of the workspace. This is achieved with a sphere-based wavefront expansion in workspace, resulting in a tree of workspace spheres (Sect. 3.3.1). The branches of the tree capture the connectivity of the workspace and the size of the spheres along the paths in the tree captures the local free workspace (Fig. 3.10). These spheres

Algorithm 8: EET

Input: $\boldsymbol{q}_{\text{start}}, \boldsymbol{q}_{\text{goal}}, \alpha, \beta, \gamma$
Output: $T = (V, E)$

$V \leftarrow \{\boldsymbol{q}_{\text{start}}\}$
$E \leftarrow \emptyset$
$S \leftarrow \text{EXPLORE}(\boldsymbol{q}_{\text{start}}, \boldsymbol{q}_{\text{goal}})$
forall the $s = (\boldsymbol{p}_{\text{center}}, r, s_{\text{parent}})$ *on depth-first traversal of solution path in* S **do**
  $\sigma \leftarrow 1/\gamma$
  repeat
    $\boldsymbol{p}_{\text{new}} \leftarrow \text{GAUSS}(\boldsymbol{p}_{\text{center}}, \sigma\gamma r)$
    $\boldsymbol{R}_{\text{new}} \leftarrow \text{RAND}()$
    $\boldsymbol{q}_{\text{near}} \leftarrow \text{NEAREST}(T, \boldsymbol{p}_{\text{new}}, \boldsymbol{R}_{\text{new}})$
    if $\sigma < \beta$ **then**
      $\boldsymbol{R}_{\triangle} \leftarrow \text{GAUSS}(0, \sigma\pi)$
      $\boldsymbol{R}_{\text{new}} \leftarrow \boldsymbol{R}_{\text{near}} \boldsymbol{R}_{\triangle}$
    if $\boldsymbol{q}_{\text{new}} \leftarrow \text{CONNECT}(T, \boldsymbol{q}_{\text{near}}, \boldsymbol{p}_{\text{new}}, \boldsymbol{R}_{\text{new}})$ **then**
      $V \leftarrow V \cup \{\boldsymbol{q}_{\text{new}}\}$
      $E \leftarrow E \cup \{(\boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{new}})\}$
      $\sigma \leftarrow (1 - \alpha)\,\sigma$
    else
      $\sigma \leftarrow (1 + \alpha)\,\sigma$
      if $\sigma \geq 1$ **then**
        $s \leftarrow s_{\text{parent}}$
  until $\|\boldsymbol{p}_{\text{new}} - \boldsymbol{p}_{\text{center}}\| < r$
return T

and their connectivity define an approximate navigation function over parts of the workspace [20, 22]. The gradient of this navigation function defines an attractive force for a point on the robot, which *pulls* the robot toward the goal in the workspace. The workspace force is then projected into a direction in the robot's configuration space with the Jacobian matrix (Sect. 2.1.3) and used by exploitation in order to move in configuration space.

Exploitation is only likely to be successful when it is based on accurate information, therefore the information represented in the workspace spheres can be augmented with more accurate information about the best configuration space directions for exploitation. This source of information is derived from repulsive forces [91] exerted by obstacles onto the robot as demonstrated in Sect. 3.1. These repulsive forces together with the attractive force derived from the workspace navigation function can then be projected into a direction in configuration space using the Jacobian matrix.

In order to balance exploitation and exploration, the planner leverages an additional source of information, namely the data obtained during the tree expansion steps. When exploitation fails, tree expansion will become increasingly exploratory, while successful tree expansions will lead to increasingly exploitative behavior.

Algorithm 9: CONNECT

Input: $T, \boldsymbol{q}_{\text{near}}, \boldsymbol{p}_{\text{new}}, \boldsymbol{R}_{\text{new}}$
Output: $\boldsymbol{q}_{\text{new}}$

```
1 while q_new ← EXTEND(T, q_near, p_new, R_new) do
2     q_near ← q_new
3 return q_new
```

Algorithm 10: EXTEND

Input: $T, \boldsymbol{q}_{\text{near}}, \boldsymbol{p}_{\text{new}}, \boldsymbol{R}_{\text{new}}$
Output: $\boldsymbol{q}_{\text{new}}$

```
1 Δx ← |(p_near, R_near) − (p_new, R_new)|
2 Δq ← J†(q_near) Δx
3 q_new ← q_near + Δq
4 if q_new ∈ C_free then
5     return q_new
6 else
7     return ∅
```

The EET planner (numbers in parenthesis refer to lines in Algorithm 8) builds a configuration space tree, much like an RRT-based planner [94]. However, every vertex $\boldsymbol{q}$ in this tree T is associated with a corresponding workspace frame, consisting of the position and orientation of a control point on the robot. This point can be the end effector in case of articulated robots or an arbitrary point on the robot in case of rigid body robots. The planner requires this information in order to leverage workspace-based information for exploitation. Initially, the starting configuration $\boldsymbol{q}_{\text{start}}$ is added to the configuration space tree (1). The sphere-based workspace expansion then determines a tree S of workspace spheres in order to capture workspace connectivity (3). The tree of workspace spheres is processed in depth-first fashion, considering only paths through the tree that lead to the specified goal location (4). The sphere s captures the free workspace volume into which the robot is trying to move next. Backtracking can be performed in case the planner fails to find a solution based on this initial workspace tunnel.

The planning algorithm now attempts to expand the configuration space tree T while balancing exploitation and exploration. Similar to RRT methods, a configuration is created toward which the tree should expand. In contrast to RRTs however, this configuration is determined based on the workspace information contained in S. Either the entire robot (in case of rigid bodies) or the robot's end effector (in case of articulated robots) should be *pulled* into the direction indicated by the workspace connectivity information. This enables the planner to solve a task frame specification in workspace rather than a specific goal configuration [140].

The variable σ (5) in the algorithm balances exploration and exploitation. A value of zero indicates pure exploitation in which the behavior of the planner can be compared to a potential field planner based on an approximate global navigation func-

tion [22]. It is also used in order to scale the variances of Gaussian distributions for generating samples and is initialized to the reciprocal value of the parameter γ. This parameter indicates how closely the planner follows the workspace information contained in S when generating new samples.

Based on a normal distribution around the sphere's center with a variance that depends on the sphere's radius scaled by σ and γ, a point $\boldsymbol{p}_{\text{new}}$ in the workspace is sampled (7). The sampling function GAUSS takes two parameters: the mean of the Gaussian distribution and the width within which 99.7 percent of the samples should fall (three times the standard deviation); it returns a random sample from this distribution. Together with a sampled random orientation $\boldsymbol{R}_{\text{new}}$ (8), this defines a workspace frame. With attention to proper distance metrics (Sect. 2.1.2), a configuration $\boldsymbol{q}_{\text{near}}$ from the tree T is then selected based on the closest match between this frame and the one associated with a vertex's control point. If the planner is able to perform exploitation as indicated by a small value of σ (10), the uniformly sampled orientation $\boldsymbol{R}_{\text{rand}}$ is replaced with one drawn from a Gaussian distribution centered around the orientation $\boldsymbol{R}_{\text{near}}$ of $\boldsymbol{q}_{\text{near}}$ and a variance that is scaled by σ.

Following this, the algorithm attempts to connect the workspace frame defined by position $\boldsymbol{p}_{\text{new}}$ and orientation $\boldsymbol{R}_{\text{new}}$ to the tree (13). This connect step is described in Algorithm 9 and is identical to the one for RRT-based planners [94]. Upon success, the penultimate vertex (14) and the corresponding edge (15) are added to the tree and the value of σ (16) is reduced. This reduction causes the planner to shift its behavior toward exploitation. If the connection attempt fails, the value of σ is increased and the balance is shifted toward exploration (18). As this variable reaches the exploration limit (19), we backtrack to the previous sphere (20). The series of expansion steps ends when the boundary of the sphere s has been reached (21).

The description of the EET planner is completed with a discussion of the extend step (Algorithm 10) used by the connect algorithm. This step moves the workspace frame associated with the configuration $\boldsymbol{q}_{\text{near}}$ toward the frame with position $\boldsymbol{p}_{\text{new}}$ and orientation $\boldsymbol{R}_{\text{new}}$ and determines the corresponding new configuration $\boldsymbol{q}_{\text{new}}$. This is accomplished by first determining the vector $\Delta\boldsymbol{x}$ pointing from the existing frame toward the new frame (1). This displacement is translated into a displacement in configuration space using the pseudo-inverse of the Jacobian (2) as shown in Sect. 2.1.3. Based on the value of $\Delta\boldsymbol{q}$, a new configuration $\boldsymbol{q}_{\text{new}}$ is determined (3) and tested for collision. The function reports a failure in case the configuration is not part of the free space. Repulsive forces from obstacles can be translated into configuration space directions in a similar fashion.

The workspace bubbles and the underlying normal distribution do not necessarily correlate to the robot's geometry or kinematics, especially for complex free-flying objects and even more so for manipulators and platforms. A simple potential field will often not be able to solve these parts, even in combination with random movement [12]. If the EET planner reaches such a difficult passage, the σ value will continue to grow, resulting in an increasing search space around the current area. This space will soon also cover areas outside the workspace bubbles. However, the nodes closest to the narrow passage with a high probability to get selected might represent a dead end. An increased σ value might also have created new nodes away

Fig. 3.12 Influence of the σ value in the Exploring/Exploiting Tree planner. Pure exploitation in the form of potential fields and uniform exploration in configuration space are the two extreme conditions the algorithm can achieve with a smooth transition in-between

from the difficult area worth further investigation. At some point, the normal distribution will also suffer from a large σ value and a more uniform distribution will provide better results. While the labeling of exhausted nodes might improve planning in some of these cases, a different approach vector to the narrow passage also represents a valid method (similar to fitting a key through a keyhole).

A simple method that can be used is to switch the planner to a previous sphere in the current workspace tunnel when the σ value exceeds a certain threshold (20). The σ value is also reset in this case. The planner has already been able to create valid nodes for the previous sphere and will likely find new connections in the second run and continue to the next sphere with a new approach. The exploration here is still limited to the workspace. Depending on the Jacobian mapping between workspace and configuration space, even a uniform random distribution in the workspace cannot reach the full flexibility of a uniform search in configuration space and is not complete. An RRT planner in configuration space however uses a uniform random distribution in this space and is proven to be probabilistically complete. In order to balance the full range between exploration and exploitation, a planner therefore at some point has to adapt to a configuration space sampling and back again.

The EET planner maintains a complete configuration space tree during planning and is able to supply a configuration space RRT with this information. As mentioned above, a high σ value represents a trend to perform more exploration, whereas a low value will approach exploitation in the form of a potential field. So far it only performs exploration in the workspace with a mapping to configuration space using the Jacobian and a normal distribution. In between this and a full uniform exploration in the configuration space, there should also be a phase concentrating on local configuration space exploration with a normal distribution.

A more advanced method can be introduced that utilizes the full range of available behavior. As the σ value increases beyond a certain threshold (e.g., 0.5), the EET adapts to an RRT behavior with a connect step and a single tree based on the gathered information. Edges in the tree are marked accordingly in order to respect the different interpolation methods. New sample points are chosen based on a normal distribution around the configuration of the last selected node and the current σ value in order to focus on the current region. The same principles as before are used to decrease/increase the σ value based on expansion failure. As the σ value exceeds another threshold (e.g., 1.0), the behavior changes to a uniform random distribution in configuration space identical to traditional RRT planners (Fig. 3.12). For

each configuration added to the existing tree, the corresponding forward kinematics is calculated in order to be able to change back to a workspace-based sampling at some point. The planner compares the forward kinematics position with each sphere in the current workspace tunnel starting from the goal configuration in order to make this decision. If a new configuration is contained within a workspace sphere, the planner changes back to workspace-based sampling and resets the σ value. Often, this will be a sphere in between the start configuration sphere and the current one, replicating the behavior of the simple method presented above.

3.4.4 Evaluation

The following experiments demonstrate the importance of carefully balancing exploration and exploitation in motion planning. The proposed EET planner is validated by comparing its performance to that of a PRM planner with uniform sampling (referred to as PRM in the following sections) [84], PRM with Gaussian sampling (Gaussian PRM) [19], PRM with bridge test (Bridge PRM) [73], RRT-Connect with one tree (RRT-Connect1) and with two trees (RRT-Connect2) [94], as well as an adaptive dynamic-domain RRT (ADD-RRT) with two trees [82]. For these planners, a nearest neighbor search based on kd-trees [155] was used, with $k = 30$ for all PRM variants. As the EET planner uses a single tree in order to explore configuration space, the performance increase afforded by balancing exploration and exploitation can most accurately be assessed by comparing it to the RRT-Connect1 planner with one tree.

The experiments are performed in six scenarios with the results for each planner and query averaged over 20 trials. If a planner was not able to solve a problem within 20 min, the experiment was aborted and the number of vertices, edges, and collision detections up to that point are reported. For the EET planner, the computational cost of workspace information based on the sphere-based wavefront expansion is negligible (never exceeded 0.5 s) and is included in the total reported planning time. The three parameters of the planner were set to $\alpha = 0.01$, $\beta = 0.08$, and $\gamma = 18$.

In order to be able to compare the various planners to each other, their corresponding implementations in the framework of Sect. 4.4 are used during the benchmarks. For verification purposes, these implementations have been compared to the ones in the Motion Strategy Library[1] (MSL 2.0). Figure 3.13 shows the resulting paths of various planners in the included *2dpoint1* scenario with a seed value of 0 for the random number generator. The query consists of a 1 m × 1 m rectangular robot with two translational degrees of freedom traveling through a 100 m × 100 m large maze from the bottom right to the top right. For this comparison, both applications employ the Proximity Query Package[2] (PQP) for collision detection. During the

[1] http://msl.cs.uiuc.edu/msl/

[2] http://gamma.cs.unc.edu/SSV/

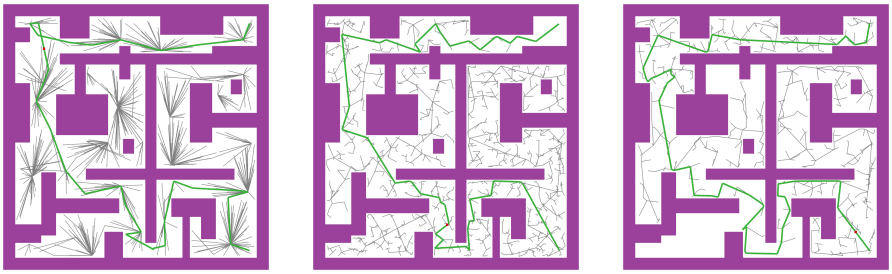


Fig. 3.13 Solution paths of the Motion Strategy Library's *2dpoint1* scenario with a common seed value of 0 for various planners. **a** PRM, **b** RRT-Connect1, **c** RRT-Connect2

benchmarks however, a faster implementation presented in [15] based on the GJK algorithm is used.

In this query, the PRM planner initializes the roadmap with 1000 vertices. The edge verification step size is set to 1 m, the search radius to 20 m, and the maximum degree per vertex to 2000, resulting in a roadmap with 653 vertices, 652 edges, and a total of 11419 collision checks (including start and goal configuration). The MSL implementation uses an unordered list for nearest neighbor selection and a sequential order during edge verification. The framework of Sect. 4.4 is able to replicate the above results with these adjustments, however the implementation used in the benchmarks reverts to a kd-tree implementation sorted by distance together with recursive edge verification [36, 99]. The roadmap is not initialized with any vertices and only one vertex is added in each step in order to be able to compare its results to those of the single-query planners.

The MSL library features several variants of RRT-based planners, including the traditional version (RRT) using only the extend step and one tree (5733 vertices, 5732 edges, 12153 collision checks) as well as its two tree (RRTExtExt) counterpart (2209, 2207, 10473). Also available are the two versions with connect step used in the benchmarks, the RRT-Connect1 (RRTCon) with one tree (1203, 1202, 6563) and the RRT-Connect2 (RRTConCon) with two trees (641, 639, 5901). The planner's step size and epsilon threshold in these queries are set to 1 m, with a 5 % probability of selecting the goal configuration for the RRT-Connect1 version. The MSL implementation always uses a fixed step size, even when the actual distance to a sampled configuration is shorter. This issue has also been addressed in the implementation used during the benchmarks.

3.4.4.1 Free-Flying Box in Maze

The first scenario (Fig. 3.14) consists of a 30 m × 30 m × 2 m large maze with walls that are 2 m apart from each other. The robot is a free-flying rigid box (six degrees of freedom, 3 m × 0.5 m × 0.5 m) moving from the lower left side of the maze to a

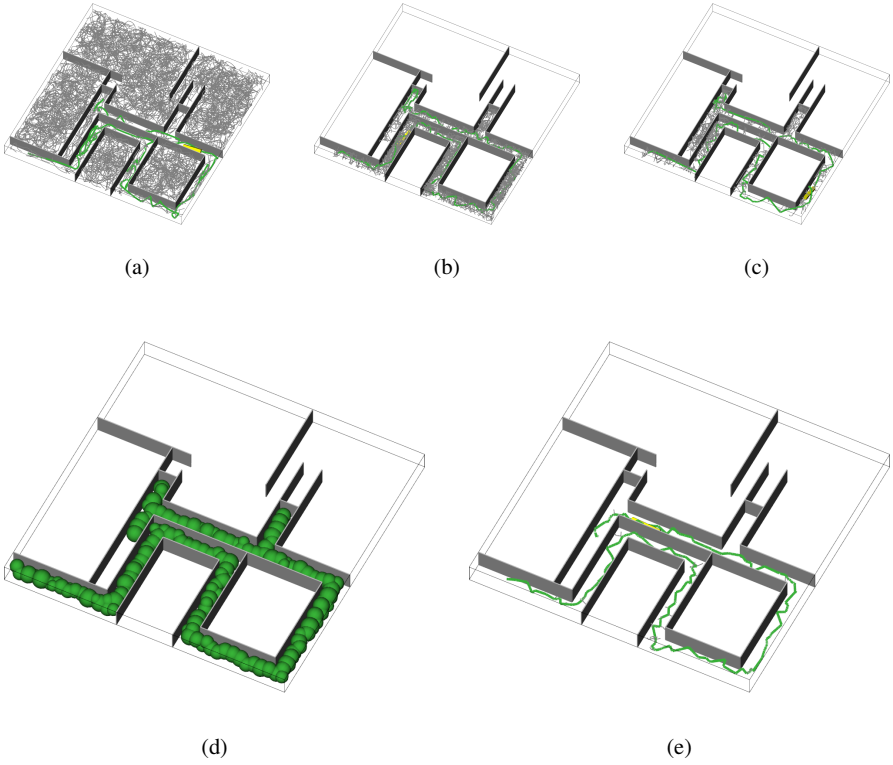
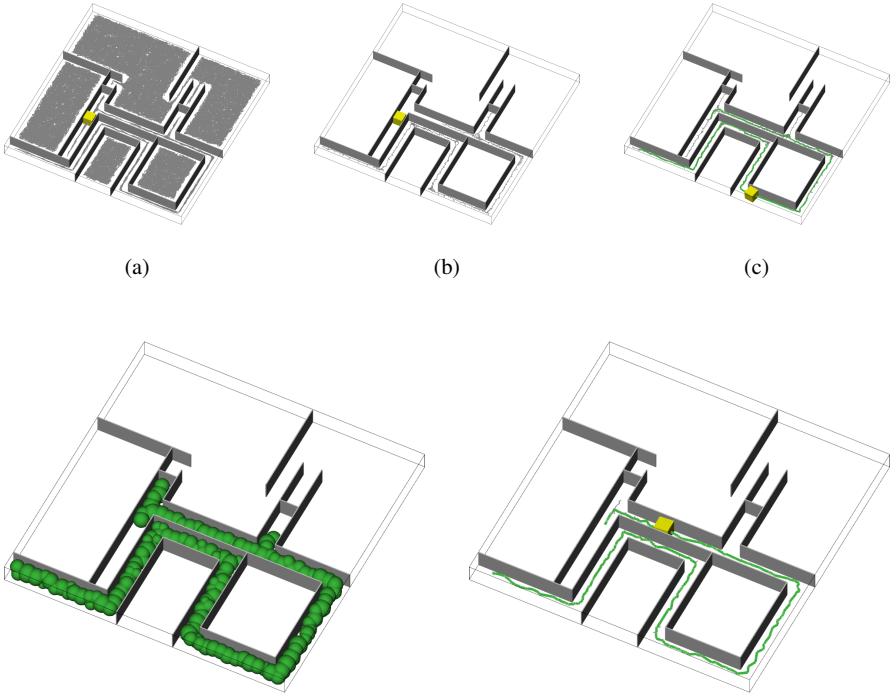


Fig. 3.14 Visualization of several planning algorithms in the first scenario with a free-flying box in a maze. **a** Bridge-PRM, **b** RRT-Connect1, **c** ADD-RRT, **d** workspace connectivity information captured by spheres, **e** EET

Table 3.2 Planner performances in the first scenario. All algorithms are able to find a collision free path. Multi-query planners waste computational resources by sampling in irrelevant configuration space regions. EET demonstrates the best performance

Planner	Vertices	Edges	Collision Checks	Time (s)	%
PRM	15 138±5 070	15 026±5 038	1 741 569±451 374	25.6± 8.1	100
Gaussian-PRM	13 276±4 690	13 168±4 659	1 658 665±461 204	23.2± 7.7	100
Bridge-PRM	11 445±3 621	9 571±3 037	1 966 568±533 011	25.0± 7.5	100
RRT-Connect1	14 859±5 800	14 858±5 800	596 121±239 843	63.6±36.6	100
RRT-Connect2	5 197±3 235	5 195±3 235	210 351±120 930	13.7±10.6	100
ADD-RRT	4 528±2 258	4 526±2 258	101 657± 47 528	9.0± 6.3	100
EET	288± 49	287± 49	9 563± 919	1.2± 0.1	100



(a) (b) (c)

(d) (e)

Fig. 3.15 Second scenario with a larger box and very small clearance. **a** Bridge-PRM, **b** RRT-Connect1, **c** ADD-RRT, **d** workspace connectivity information captured by spheres, **e** EET

Table 3.3 Comparison of algorithms in the second scenario with a larger free-flying box. PRM planners and the RRT-variant with a single tree are unable to solve this task within the given time frame. EET shows better performance than the RRT variants with two trees. Additional repulsive forces can reduce the number of vertices and edges even further

Planner	Vertices	Edges	Collision Checks	Time (s)	%
PRM	201 295±1 004	193 390± 996	61 302 764± 296 484	1 200.0± 0.0	0
Gaussian-PRM	251 940±1 251	243 755±1 233	67 062 589± 323 709	1 200.0± 0.0	0
Bridge-PRM	138 058± 572	110 065± 489	98 936 644± 306 197	1 200.0± 0.0	0
RRT-Connect1	12 908±5 500	12 907±5 500	12 691 936±6 510 831	1 195.6± 19.8	5
RRT-Connect2	6 851±3 895	6 849±3 895	3 032 624±4 880 296	256.6±424.2	85
ADD-RRT	6 114±2 131	6 112±2 131	369 815± 402 788	179.0±356.9	90
EET	942± 206	941± 206	18 874± 8 044	4.6± 3.4	100
EET-Repulsive	329± 263	328± 263	7 817± 2 420	1.9± 0.7	100

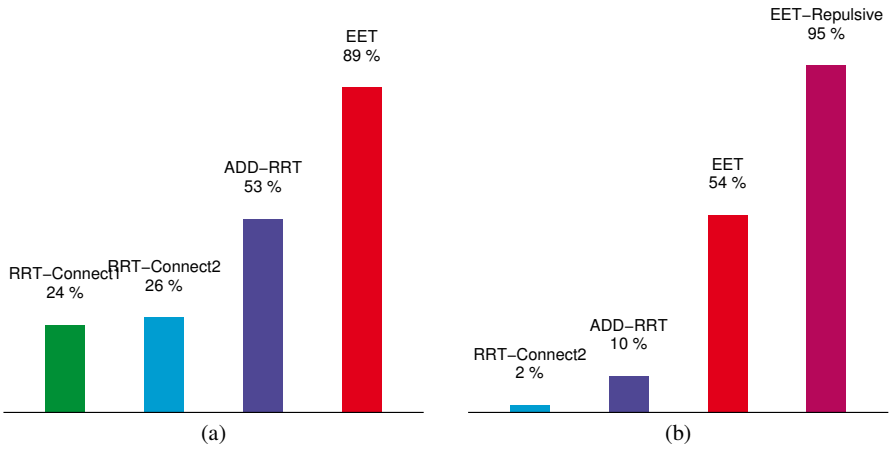


Fig. 3.16 Effectiveness of free sample placement in the scenarios with free-flying boxes traveling through a maze. **a** performance in the first scenario with a long and thin robot, **b** larger box with small clearance to the wall

position on the middle left. Due to the length of the robot and the distance to the walls, traveling through the corners of the maze is difficult and requires exploration. The straight corridors can be solved by exploitation, provided the robot is aligned with the walls. The maze also contains large regions irrelevant to the task that will distract some of the planners.

All planners solve this problem within the time limit of 20 min (Table 3.2). The single-query methods have the advantage of focusing on the correct area of the maze, but still spend considerable time exploring straight corridors instead of performing exploitation. The multi-query approaches waste computational resources exploring irrelevant configuration space regions compared to traditional single-query planners in this example. The EET planner is able to perform exploitation in the straight corridors. In the tighter turns, it shifts toward exploration. The resulting EET tree contains far fewer vertices than any other method and the computed path exhibits less of the zigzag behavior commonly observed in sampling-based motion planning. It shows a vast increase in performance compared to the RRT methods with one and two trees.

An interesting measure to assess the effectiveness of exploitation is the percentage of collision checks for which the tested configuration was in free space. For an ideal planner this number would be 100 %, indicating that available information is leveraged to effectively guide the planner through the free configuration space. The results of the PRM-based planners are however biased due to the large amount of free samples in the unconnected regions of the maze. In this scenario, this value was equal to 24 % for RRT-Connect1, 26 % for RRT-Connect2, and 53 % for ADD-RRT. In comparison, 89 % of the configurations checked by the EET planner were collision free (Fig. 3.16a).

3.4.4.2 Large Free-Flying Box in Maze

In the second query, a larger box (1.5 m × 1.5 m × 1.5 m) moves through the same maze (Fig. 3.15). The clearance to the walls is very small so that motion through the straight corridors becomes much more difficult.

The EET planner is the only planner capable of solving this task every time in less than 20 min (Table 3.3). The RRT-Connect variants with two trees achieve success rates between 85 % to 90 %. The reduced clearance around the robot makes it difficult for other tree-based methods to move through the straight, narrow corridors. The PRM planners have difficulties placing valid samples in the corridors. Again, the EET planner benefits from its ability to balance exploitation and exploration. In the straight corridors, it performs exploitation using the workspace information. In the more difficult turns, the planner increases exploratory behavior, as indicated by the increased number of vertices and edges in the resulting roadmap. In this scenario, only 2 % of the RRT-Connect2's and 10 % of the ADD-RRT's collision checks were placed in free configuration space, compared to a number of 54 % for the EET planner (Fig. 3.16b).

For this more difficult scenario, an additional source of information for exploitation was tested. As described in Sect. 3.1, the attractive force provided by the workspace spheres is combined with repulsive forces from obstacles. This additional information should further guide exploration toward the correct direction in configuration space. Only the distance to the closest object has been used. Indeed, as indicated by the results reported for EET-Repulsive, this additional information reduces the number of vertices and edges required to solve the planning problem, relative to EET. This demonstrates that the EET planner is able to balance exploration and exploitation adequately. When more accurate information is available for exploitation, the planner shifts further toward exploitative behavior. Collision checks performed by the EET-Repulsive planner are 95 % collision free, indicating that the planner leverages the available workspace information to perform highly effective configuration space search.

3.4.4.3 Free-Flying L-Shape in Maze

The previous scenarios have used a rather simple box shape for the free-flying robot with a rather good relationship between configuration space and workspace. As a consequence, the information gathered by the workspace spheres is extremely valuable. Additionally, the large open spaces and unconnected regions of the maze result in extra effort for the multi-query PRM planners. For this new query, the robot's geometry was replaced with a more complicated L-shape consisting of two boxes with dimension 1.75 m × 0.5 m × 0.5 m. In order to create more equal conditions for the PRM-based planners, only the lower half of the previous maze is used and the free unconnected regions are closed as well. The complexity of the scenario is increased by narrowing the width of the corridors to 1 m, therefore requiring a more advanced turning motion in the corners due to the robot's shape (Fig. 3.17).

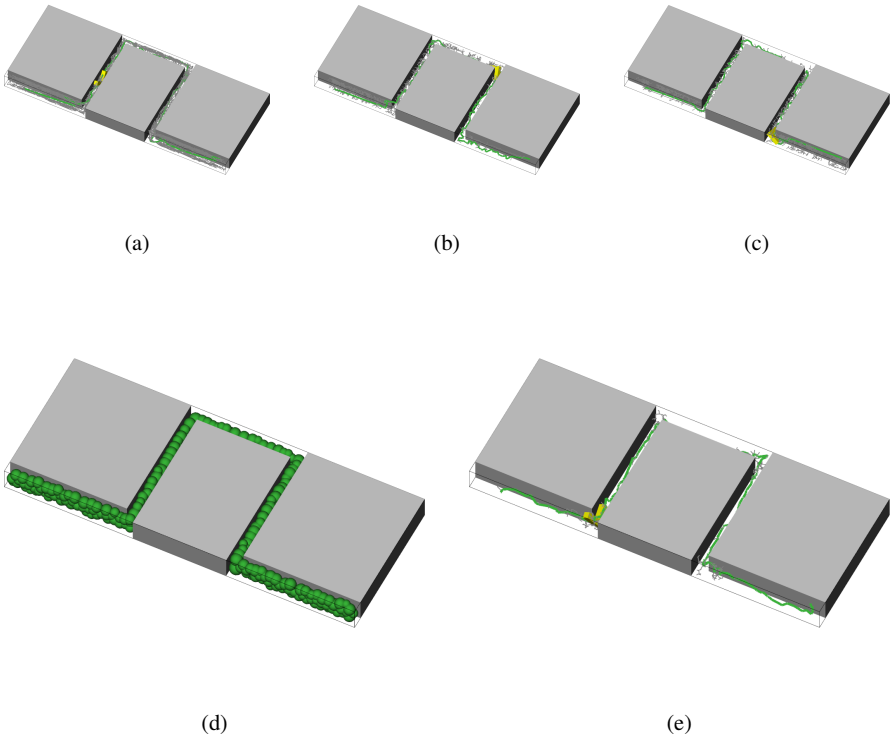


Fig. 3.17 Alternate maze scenario with free-flying L-shape robot. Only the lower half of the previous environment is used. Corridors are narrower and open unconnected regions have been closed. **a** Bridge-PRM, **b** RRT-Connect1, **c** ADD-RRT, **d** workspace connectivity information captured by spheres, **e** EET

Table 3.4 Only the EET planner is able to consistently solve the third scenario with a more complicated L-shaped geometry. PRM-based planners now generate fewer vertices and edges than the RRT variants. Repulsive forces in the EET-Repulsive planner result in a more effective sample placement and lower runtimes

Planner	Vertices	Edges	Collision Checks	Time (s)	%
PRM	3 352±1 331	3 210±1 288	42 615 866±16 974 549	707.0±281.9	90
Gaussian-PRM	3 974±1 545	3 832±1 500	46 611 215±18 145 569	672.0±261.9	90
Bridge-PRM	3 002± 491	2 385± 401	99 892 354±16 257 604	1 074.1±174.8	55
RRT-Connect1	13 165±3 707	13 164±3 707	5 625 968± 1 274 470	939.1±284.0	60
RRT-Connect2	12 312±3 046	12 310±3 046	5 202 913± 2 013 825	721.1±339.2	75
ADD-RRT	11 670±3 479	11 668±3 479	457 771± 136 762	748.6±357.1	80
EET	4 838±1 409	4 837±1 409	71 864± 24 145	55.4± 34.2	100
EET-Repulsive	4 157±1 832	4 156±1 832	27 456± 11 809	20.2± 19.0	100

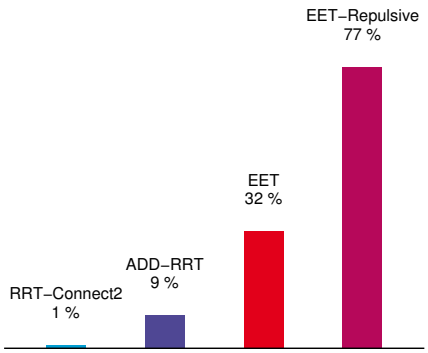


Fig. 3.18 Comparison of free sample placement percentage in the third scenario with an L-shape geometry. Only ADD-RRT and EET achieve values above 1 %, with EET-Repulsive demonstrating the most effective results

Table 3.4 shows that only the EET planner was able to generate a collision free path within the given time limit for all 20 passes. In contrast to the previous experiment, the PRM-based planners now produce roadmaps with fewer vertices and edges than the RRT-based algorithms, together with a higher success rate and lower runtime for the traditional PRM and the Gaussian-PRM. The number of collision checks performed by the planners however is much higher, especially for the version with bridge test sampling, where any advantages are outweighed by the related performance costs. With the information from the workspace spheres, the EET planner can focus on the difficult corner regions of the scenario. Due to the complex geometry of the robot, exploration in the configuration space is necessary in order to solve the query. This results in a much higher number of collision checks and vertices in the tree. However, while the percentage of collision tests performed in free configuration space is under 1 % for all planners (including the PRM versions this time) except the ADD-RRT with 9 %, the EET planner still manages to achieve a value of 32 %. This number can be increased even more with the help of additional workspace information in the EET-Repulsive variant, resulting in the shortest runtime and the most effective sample placement of 77 %.

3.4.4.4 Stationary Manipulator

In contrast to the free-flying robots used so far, this setup involves a stationary 6-DOF manipulator commonly found in industrial settings (Fig. 3.19). It is placed on the ground in front of a wall with several holes. The task consists of finding a way out of the bottom left hole and into the bottom right hole. Both openings represent narrow passages in the configuration space.

The PRMs with Gaussian and bridge test sampling are the only PRM variants able to solve this scenario reliably (Table 3.5). The roadmap mainly covers the open configuration space regions around the robot's base, as only few samples are actually placed in the vicinity of the narrow passages. The RRT variants with two trees on

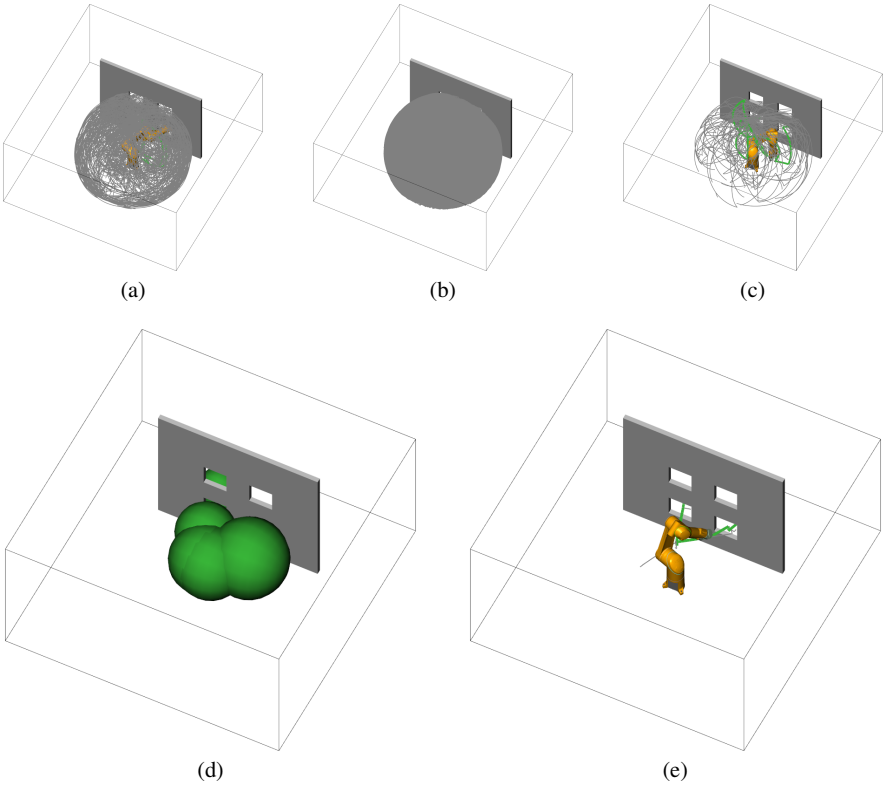


Fig. 3.19 First stationary manipulator reaching through openings in a wall. The robot has to move from the bottom left hole to the one on the bottom right. **a** Bridge-PRM, **b** RRT-Connect1, **c** ADD-RRT, **d** workspace connectivity information captured by spheres, **e** EET

Table 3.5 Benchmark results in the fourth scenario. The start and goal configuration of the stationary manipulator lie within narrow passages. The traditional PRM and single-tree RRT planners show the worst overall performance. Average path length of the EET planner is shorter compared to the other planners

Planner	Vertices	Edges	Collision Checks	Time (s)	%
PRM	156 478±43 496	156 354±43 467	9 563 761±2 392 399	1 066.8±277.9	20
Gaussian-PRM	2 311± 1 756	2 296± 1 749	390 405± 243 252	38.6± 23.9	100
Bridge-PRM	2 199± 1 418	2 097± 1 356	459 649± 258 523	44.3± 24.6	100
RRT-Connect1	332 644± 8 142	332 643± 8 142	11 661 402± 238 944	1 200.0± 0.0	0
RRT-Connect2	669± 353	667± 353	30 622± 22 116	2.6± 1.9	100
ADD-RRT	960± 331	958± 331	38 564± 25 039	3.4± 2.1	100
EET	46± 47	45± 47	2 957± 2 402	1.0± 0.7	100

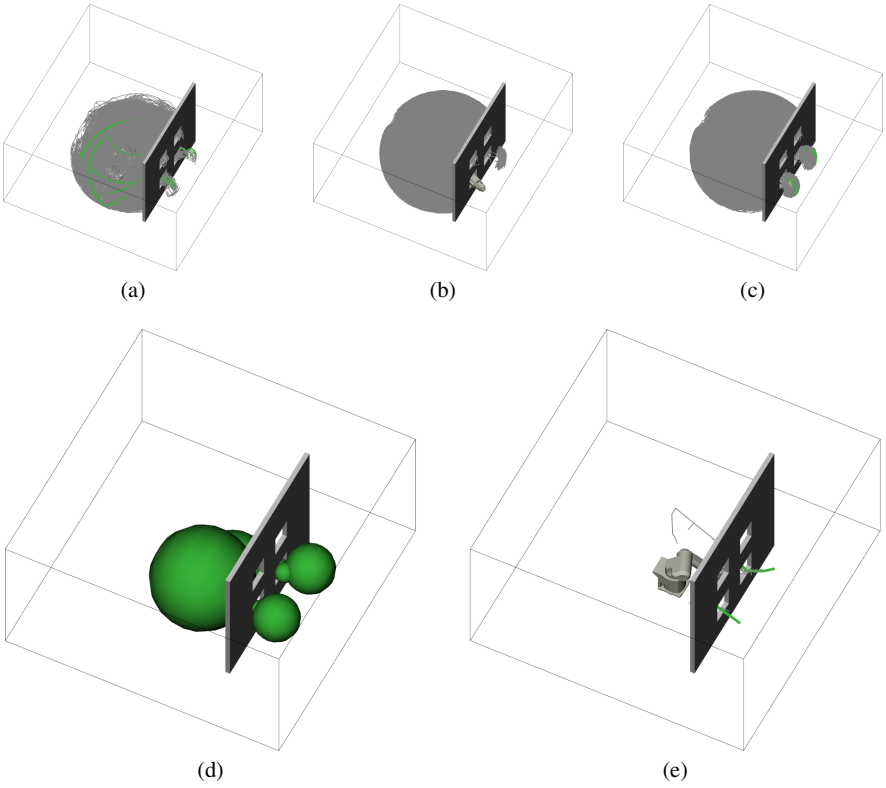


Fig. 3.20 Alternate manipulator with different kinematics and geometry but the same task as in the other setting. **a** Bridge-PRM, **b** RRT-Connect1, **c** ADD-RRT, **d** workspace connectivity information captured by spheres, **e** EET

Table 3.6 Comparison of planner performances in the fifth scenario. PRMs with Gaussian and bridge test sampling are the only planners beside the EET planner to solve all queries within the given time limit

Planner	Vertices	Edges	Collision Checks	Time (s)	%
PRM	183 329± 1 264	183 231± 1 263	10 179 372± 58 054	1 200.0± 0.0	0
Gaussian-PRM	11 799± 6 909	11 768± 6 904	1 369 712± 654 844	138.5± 68.6	100
Bridge-PRM	14 071± 7 188	13 450± 6 877	2 076 118± 914 161	210.1± 94.2	100
RRT-Connect1	292 030±52 674	292 029±52 674	10 194 891± 986 885	1 200.0± 0.0	0
RRT-Connect2	159 278±84 659	159 276±84 659	6 014 672±2 808 793	764.2±357.8	70
ADD-RRT	139 742±75 754	139 740±75 754	4 962 883±2 565 124	658.7±318.7	90
EET	145± 87	144± 87	7 809± 6 647	1.7± 1.3	100

the other hand are able to solve this scenario much more efficiently. The single tree RRT however is only able to navigate out of the first hole and fails to find a solution for the second narrow passage. Although the EET planner is also based on a single tree, it benefits from adequately balancing exploitation and exploration. Exploitation enables the robot to withdraw from the first narrow passage quickly. The same is true for the insertion into the second narrow passage.

The percentage of free samples in the number of collision tests used as a criterion in the earlier benchmarks is not significant in this scenario due to the large open space on the robot's side of the wall. Noticeable however is the difference in path quality obtained from the successful planners. While a number of optimization steps can be used in order to enhance a result [58], this procedure is expensive and also runs faster with an initial path of higher quality. As a measure of quality, the length of the individual path segments after a successful query is added and compared between a number of planners (Fig. 3.21a). Due to excessive sampling in the free space, the resulting paths of the PRM and RRT-based planners were often of poor quality with time-consuming detours. The EET planner's results on the other hand were usually more direct, indicating that little unnecessary exploration was performed.

3.4.4.5 Stationary Manipulator Variation

Basically identical to the previous query, this scenario only replaces the kinematic and geometric model of the manipulator with that of a different manufacturer. Both feature a similar construction type with six degrees of freedom and are intended for industrial applications. The two models only slightly differ in shape and length of several links (Fig. 3.20).

Table 3.6 demonstrates however that even these minor details can make a huge difference for motion planning algorithms. The results from the earlier query are practically reversed, with the Gaussian and bridge test PRM versions now showing fewer vertices, edges, collision queries, and runtimes than the RRT-based planners. Despite a huge time penalty, they are also the only ones besides the EET planner capable of solving the task in every pass within the given time frame. While showing a slight increase in the tree size and the number of collision checks, the EET planner's performance is very close to the one in the other scenario. This demonstrates an efficient use of both exploitation and exploration, especially compared to the performance of RRT-Connect1, the only other planner based on a single tree.

3.4.4.6 Mobile Manipulator

In this scenario, a holonomic 10-DOF mobile manipulator is placed in a room with dimensions $5\,\text{m} \times 5\,\text{m} \times 2.5\,\text{m}$ (Fig. 3.22). It has to perform the same task as the stationary manipulator, but the holes are further apart and the robot has to move its base in order to reach from one opening to the next.

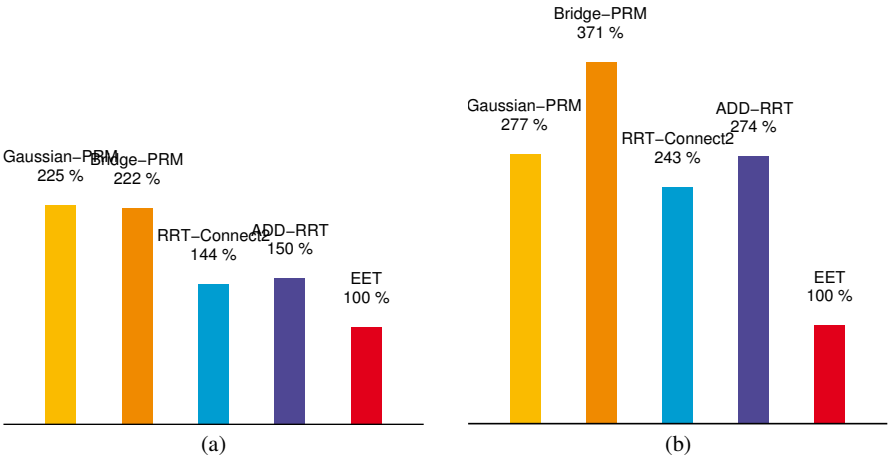


Fig. 3.21 Difference in length of collision free paths found by several planners. **a** comparison of fourth scenario with stationary manipulator, **b** mobile manipulator in experiment six

The PRM with uniform sampling succeeds in all trials. The RRT-Connect variants with two trees solve this task faster than the stationary scenario due to the fact that the robot has more degrees of freedom and a suitable direction for expansion is easier to find. The single tree variant is mostly unable to solve the narrow passage leading to the goal configuration. Again, the EET planner's ability to balance exploration and exploitation results in the best planning performance.

As with the stationary manipulator tasks, many of the PRM and RRT planner's sample points are placed in free space rather than showing a focus on the narrow passages. The length of the resulting paths can again be used as a quality measure for comparing the various planners. Figure 3.21b shows that the difference in length is even more visible than in the previous comparison. While the EET produces straightforward and short paths for this setting, the solution of the other planners often include indirect movements that appear unnecessary and unnatural to a human operator.

3.4.4.7 Mobile Manipulator with Attached L-Shape

While the connection between workspace and configuration space is not as distinctive as in the first two maze queries, the EET planner in the last three scenarios can still gather a lot of helpful data from the workspace connectivity information provided by the sphere exploration. As with the robot's more complex shape in Sect. 3.4.4.3, this task adds an object in the form of an L-shape to the tool of the mobile manipulator. Together with similarly reshaped holes, the robot is required to perform a more complex motion in order to escape and enter the narrow passages near the start and goal position (Fig. 3.23).

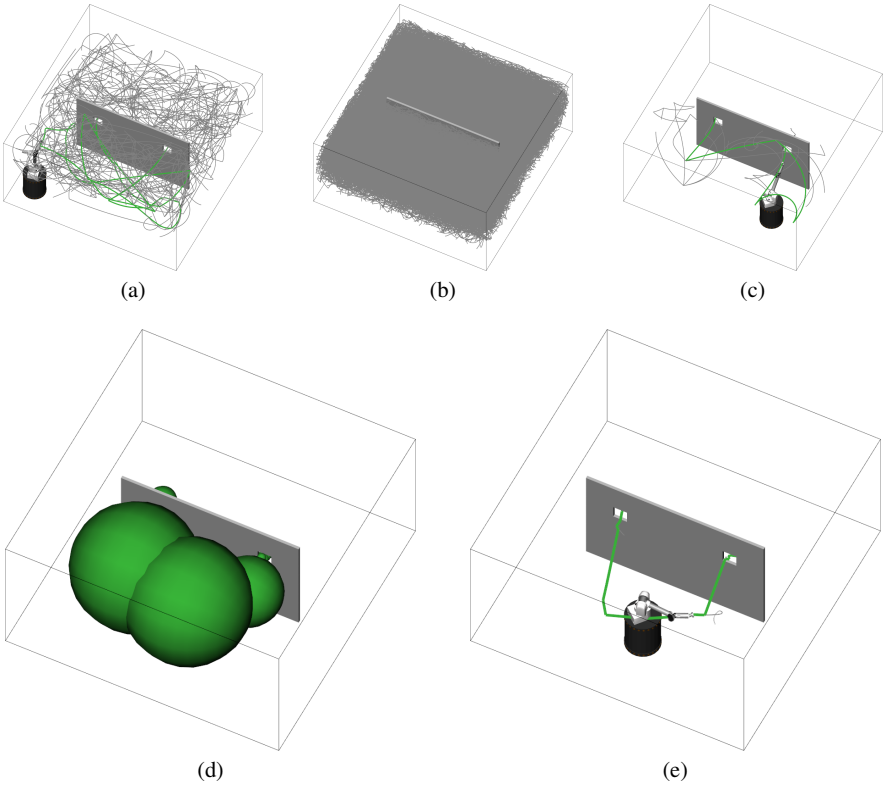


Fig. 3.22 Mobile manipulator in scenario six in front of a wall with two openings. The start configuration is reaching through the hole on the left. **a** Bridge-PRM, **b** RRT-Connect1, **c** ADD-RRT, **d** workspace connectivity information captured by spheres, **e** EET

Table 3.7 Planner performances in the sixth scenario. The additional degrees of freedom provide more suitable expansion directions. The EET's average path length compared to the other planners is even shorter than before

Planner	Vertices	Edges	Collision Checks	Time (s)	%
PRM	1 385± 1 704	1 384± 1 704	283 712± 300 731	67.7± 71.2	100
Gaussian-PRM	495± 570	493± 570	127 760± 131 718	31.9± 32.4	100
Bridge-PRM	642± 882	595± 821	179 770± 223 331	46.1± 56.1	100
RRT-Connect1	62 620±13 844	62 619±13 844	4 852 026±1 048 607	1 144.0±250.3	5
RRT-Connect2	144± 92	142± 92	7 910± 6 450	1.7± 1.3	100
ADD-RRT	164± 114	162± 114	9 467± 7 845	2.0± 1.6	100
EET	26± 12	25± 12	1 269± 215	1.2± 0.2	100

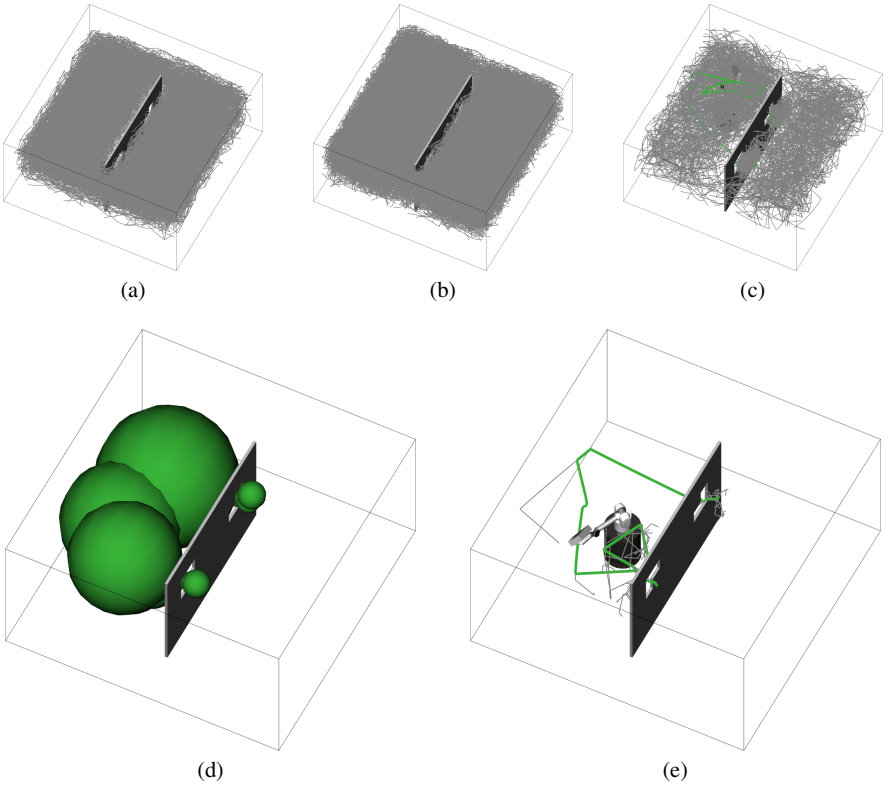


Fig. 3.23 Final scenario with an L-shape attached to the mobile manipulator and corresponding wall openings. **a** Bridge-PRM, **b** RRT-Connect1, **c** ADD-RRT, **d** workspace connectivity information captured by spheres, **e** EET

Table 3.8 Benchmarks for the mobile manipulator with L-shape. Both RRT-variants with two trees and the EET solve the given task within the given time frame. In 25 % of the queries, the EET planner uses a path around the wall due to the open space next to the wall

Planner	Vertices	Edges	Collision Checks	Time (s)	%
PRM	22 676± 1 604	22 654± 1 601	3 238 588±205 850	1 200.0± 0.0	0
Gaussian-PRM	21 210± 520	21 159± 519	3 344 724± 74 105	1 200.0± 0.0	0
Bridge-PRM	20 578± 312	18 637± 303	3 653 607± 37 713	1 200.0± 0.0	0
RRT-Connect1	59 418± 854	59 417± 854	4 009 823± 57 019	1 200.0± 0.0	0
RRT-Connect2	16 737±11 259	16 735±11 259	995 626±755 305	284.9±215.2	100
ADD-RRT	11 725±11 291	11 723±11 291	553 647±681 870	166.8±205.5	100
EET	1 837± 1 509	1 836± 1 509	27 021± 23 220	27.9± 26.4	100

Besides the EET planner, only the RRT-Connect variants based on two trees are able to solve the given task reliable for all repeated queries in the given time frame (Table 3.8). The PRM-based planners fail to place useful samples around the two narrow passages and the RRT-Connect1 version cannot find a proper way into the second one around the goal configuration. With more exploration around these two points, the EET planner manages to solve the task every time. However, due to the fact that exploitation guides the frame associated with the robot's end effector and the open space to the left and right of the wall, 25 % of the queries result in a path around the wall to the goal configuration's tool frame. The L-shape attached to the tool and the hole's geometry complicate the reach through the second opening and eventually exploration is able to find an easier path leading to the same goal frame.

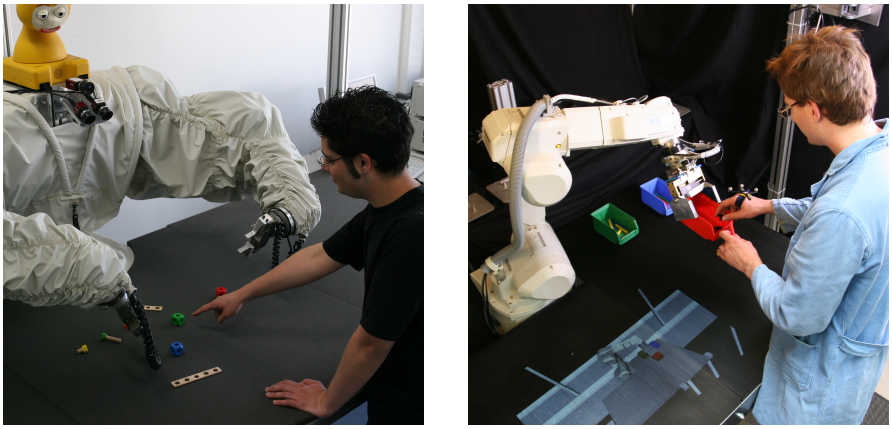
Chapter 4
Distributed System Architecture

An industrial robot in a traditional construction setting is usually running a fixed, deterministic program and is shielded from human operators through fences and other security devices. The manipulators in these scenarios are often designed to handle heavy loads and perform the same task efficiently over a long period of time. Communication or synchronization between different robots or other mechatronic equipment is normally handled via various input/output signals. The use of additional sensors is typically limited to force/torque devices or basic forms of image processing. Although industrial norms are changing and now allow direct cooperation of humans and robots to some degree (e.g., ISO 10218), the included limitations and uncovered aspects still often prevent true and efficient collaboration.

A robot system specifically designed for interaction with a human operator not only requires highly-skilled sensorimotor coordination, action planning and safety regulations on the part of the robot, but also the ability to understand and communicate with a human being in many modalities [131]. In a joint action environment (Fig. 4.1), partners working together on a common task have to communicate with the other person. This also includes monitoring the other's behavior and errors and modifying their own behavior accordingly [132, 39].

Indicating to the partner which object in the world to use is an important component of joint action. Situated references are often used depending on the location of parts and the current state of the dialog and the assembly plan. If only one instance of a type of object is available in the entire workspace, a complex reference can usually be avoided. The same is true if both partners know the state and the next part in the current assembly task. Dialog history also provides context and can be used to refer to objects with reduced expressions [11, 10].

Apart from a focus on speech and verbal communication, several non-verbal input channels such as nodding or pointing also need to be supported. During joint activities, humans tend to employ signals beyond language to indicate things [34]. Even people speaking entirely different languages are usually able to communicate with the help of actions, objects, or locations around them. A human can for example ask for a specific object either by using an elaborative verbal description or by solely relying on a combination of gestures such as pointing. He can also com-



(a) (b)

Fig. 4.1 Human-robot interaction scenarios addressed in this chapter. **a** Joint-Action Science and Technology (JAST), **b** Joint-Action for Humans and Industrial Robots (JAHIR)

bine both approaches and simply ask for the object he is currently pointing at. Most of these material signals can be categorized into two main types: *directing-to* and *placing-for* [33]. With the first technique, an action is used in order to direct an addressee's attention toward an object, place, or event. Pointing toward an object, a location, or oneself—using hands, fingers, thumbs, lips, eyes, or chin—is a typical example for this type of non-verbal communication. In contrast, *placing-for* is used in order to place an object or action in a location where an addressee should interpret this appropriately. During an assembly, this might include a part required in the next step of the current task plan.

The architecture of such a robot system needs to be able to handle all these various input types. It also has to process all available information and generate appropriate outputs. This all happens in an uncertain and dynamic environment, where sensors may not be able to precisely obtain each event and every action. Some parts of the system, such as hardware control, will depend on certain real-time requirements. Sensors and corresponding algorithms will have different time constraints and computational demands and high-level planning will be even more complex. In addition, several approaches exist for various aspects of the system, such as symbolic or sub-symbolic planning, different methods and levels for object and gesture detection, as well as multiple strategies for output control. Parts of the system will also have special requirements for operating systems, programming languages, and even load balancing across multiple machines.

A system architecture has to support all of the afore-mentioned aspects. Software modules for the various parts have to be able to communicate with each other across several boundaries such as operating systems, programming languages, and communication channels. It also has to be able to support changes and further devel-

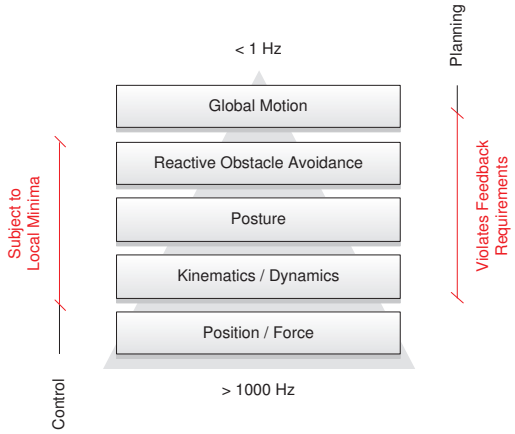


Fig. 4.2 Feedback constraints of various tasks. Global planning can address local minima issues, but cannot fulfill real-time requirements of local control

opment, such as different hardware components or algorithmic approaches, without the need of a total redesign.

4.1 Local Control vs. Global Planning

Robot programs generated in the teach-in method typically run in offline mode. The task is completely defined by a programmer beforehand and the trajectory of the whole movement can be precomputed. Internally, the controller calculates the motor torques from the desired joint positions using the kinematics and dynamics equations in the given time constraints. Additional direct input from a force/torque sensor can also be integrated here.

In contrast, an interactive robot system will not only run in online mode, but also feature several advanced control paradigms. Ideally, the system has direct access to the motor torques and can circumvent any position or velocity control loops. Only a few robot systems today grant this kind of access, often a position or velocity interface or no online access at all is provided. Torque control usually requires a control loop of 1000 Hz or higher, whereas access to other interfaces only reaches 100 Hz or even lower with no direct control over internal computations.

A controller written completely in software can directly implement various strategies such as position or force control (Fig. 4.2). Of course this also goes hand in hand with extreme real-time and fail-safe requirements. If no correct control torque can be provided at the necessary control cycle, the robot system will be damaged and likely also injure its human operator. The same is true for position control and sensor evaluation, such as force control with very high feedback constraints. Input

from other sensors with lower update rates, such as cameras for visual servoing, can also be integrated here.

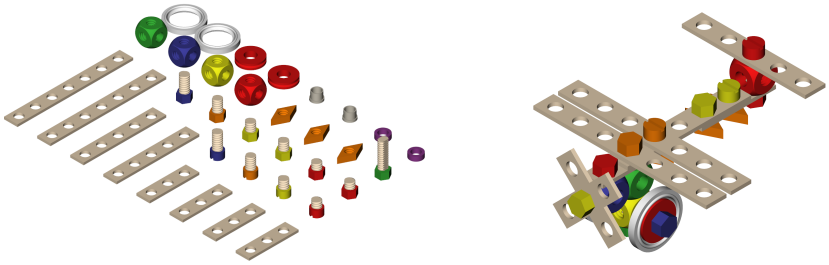
Whereas position and force control can be directly applied to local control laws, advanced calculations suffer from local minima. Inverse kinematics computations often suffer from singularities and optimizations based on dynamic parameters are even worse. The same is true for posture control, optimized trajectories, and especially for reactive obstacle avoidance. The latter also often has to integrate sensor input and distance computations from geometric models. Where simple collision detection performance can reach up to 1000 Hz on recent processing units, distance calculations are usually slower.

Global planning can generally solve most of the issues regarding local minima, but cannot meet the feedback requirements necessary for local control. Although many sampling-based methods are probabilistic complete, the computation of a collision free path might take several days or even longer. After planning, reactive obstacle avoidance or advanced strategies such as Elastic Strips [23] can ensure safe operation in a dynamic environment. Some approaches such as Elastic Roadmaps [153] can even achieve some form of global planning in a tradeoff with full completeness. Planning also includes several forms of high-level planning, such as interpretation of sensor data, multimodal fusion, a combined world model, assembly planning, dialog management, and decomposition of behaviors into actions, tasks, and events.

4.2 Integration of Input, Reasoning, and Output

In the JAST (Joint-Action Science and Technology) project, a wooden toy-model of an aircraft was chosen for a joint construction task (Fig. 4.3). The scenario consists of several parts, such as wooden slats, bolts, cubes, or nuts, and can be used to build a large variety of assemblies (e.g., a motorcycle). While the number of parts is limited, building plans for advanced assemblies can become highly complex. Due to variations in the production of these parts, their handling is sometimes challenging even for human operators. Actions performed during this joint construction task can be classified into a number of frequently occurring categories. Observation of human operators in this scenario showed operations such as *pick up*, *screw*, *point*, *put down*, and *plug* among those with the highest percentage [80]. Further actions include *give*, *receive*, *align*, *unplug*, *unscrew*, *stack*, and *unstack*.

The system is equipped with two industrial robot manipulators—mounted to resemble human arms—and an animatronic head, so that a human can interact with the system across a table (Fig. 4.1a). In order to ensure cooperation, the workspace is virtually divided into three areas. Human and robot each have sole access to a section in front of them, with a common area in between. During interaction, the parts required for a target assembly are distributed across these areas so that neither of the agents is able to reach all of the required components. Various cameras are installed for the recognition of objects on the table and for tracking the user. Force/torque



(a) (b)

Fig. 4.3 Baufix toy scenario used in the JAST project, including wooden cubes, slats, nuts, and bolts. **a** parts required for building an airplane, **b** fully assembled airplane

sensors and grippers (with position sensors) on each arm complete the installation. Due to hardware and software requirements, the system is distributed across several machines with different operating systems and programming languages. Modules communicate with each other using the Internet Communications Engine (Ice), an object-oriented middleware which supports distributed heterogeneous systems [67].

An important aspect in the development of this system was a focus on joint-action principles. Both verbal and non-verbal areas of communication have to be recognized and handled during interaction. The data sent between the different modules is given timestamps to enable multimodal integration. Because of the live interaction with the human instructor, the system needs to be able to abort or switch any behavior instantly and safely. Different approaches to input, reasoning and output handling may be implemented by different parties during the project. The behavioral layer of the system is therefore loosely categorized into three main areas: modules that mainly handle input channels and recognition, those processing obtained information and dealing with corresponding reasoning, and eventually the ones generating output of any form (Fig. 4.4). Modules can also be easily disabled or restarted at any time.

4.2.1 Recognition of Verbal and Non-Verbal Communication

Input modules have to deal with recognition in various areas that are part of interaction. Data from a top mounted camera is used to detect Baufix objects on the table as well as hand position and gestures. In order to be able to directly address the user when speaking to him, a camera in the animatronic head together with face

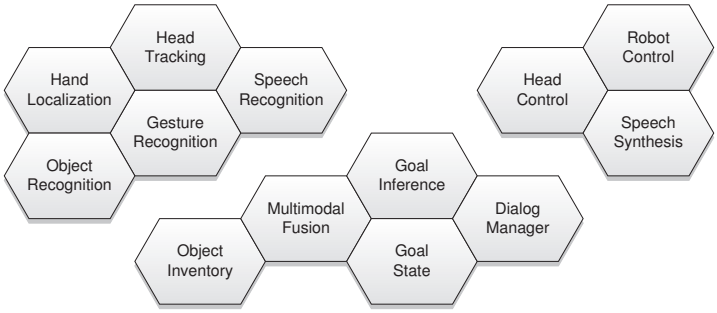
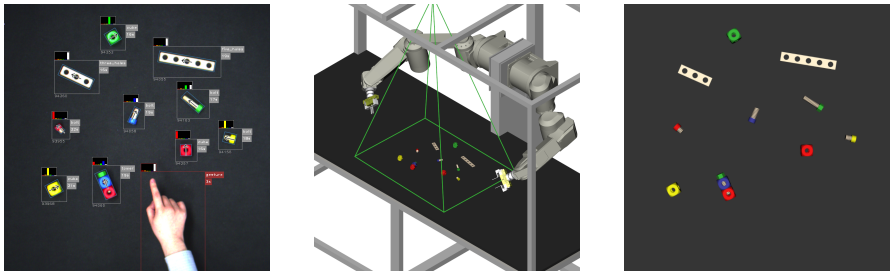


Fig. 4.4 Overview of modules in the JAST architecture. Elements are loosely categorized into input (*left*), reasoning (*bottom*), and output modules (*right*). Communication between modules is implemented using an object-oriented middleware with support for distributed systems

tracking is employed. Verbal input through a microphone is processed by a speech recognition engine.

The object recognition module uses a template matching approach in order to recognize the parts of the Baufix scenario [109]. Its task is to classify the various pieces into categories (e.g., slats, bolts, nuts, cubes) and to identify corresponding properties such as color (e.g., red, yellow, orange, blue, green) or size (e.g., small, medium, long slat) as well as the position and orientation of the object on the table (Fig. 4.5a). Segmentation is simplified through the use of a static, uniform background. Occluded objects can be detected up to a certain amount of overlapping. The camera is calibrated and the module transforms the coordinates to a common world frame used across all modules of the system. Recognized objects are assigned a unique identifier and a timestamp. Templates for the fixed set of known objects of the scenario are usually generated and stored beforehand, but the approach also allows for the addition and naming of new parts during runtime. This also applies to generating a template for an assembled object of several parts (Fig. 4.6). Performance of the recognition module is greatly improved through the use of multi-core architectures [110]. Visualization of the results can be provided to the user as a direct feedback and demonstration of possible recognition limitations and errors. Information such as the current position of the robot manipulators can be requested and included in order to exclude corresponding image regions in the detection using a virtual camera in a 3D representation. Position, orientation, and view frustum are determined by the intrinsic and extrinsic parameters of the camera calibration and lens distortion can be simulated by a fragment shader (Fig. 4.5b–c).

Data from the same camera is also used for hand localization and gesture recognition. This information is essential for the integration of non-verbal communication in the system. In combination with speech it can also be used to clarify ambiguous instructions. In this setup, three different types of gestures need to be detected: pointing, grasping, and holding-out. The first one is mostly used to highlight a specific object on the table, usually with the help of the index finger (Fig. 4.5a). Grasping (with index finger and thumb) is used to articulate the intention of taking an



(a) (b) (c)

Fig. 4.5 Object and gesture recognition example of top-mounted camera. **a** actual camera image with detected objects and pointing gesture, **b** synchronized 3D representation with current robot configuration, recognized objects, and distorted view frustum (*green lines*), **c** virtual camera image including lens distortion

object, whereas holding-out (open palm) symbolizes a request for an object. The approach used in this module computes the likelihood of different gesture types and assigns probabilities based on Bayesian inference rules [158, 159, 157]. As with the object recognition module, a static or dynamic threshold can be used to remove the table background during segmentation. Invariants include geometric ones (outer-contour length, X- and Y-gradients, gradients deviation, furthest distance) as well as the first six Hu moments. The gesture likelihoods are determined by a modified k-nearest neighbor classifier. The module reports on updates of the user's hand position (in world coordinates), combined with a timestamp. A list of gesture hypotheses and their corresponding probabilities is broadcast whenever the hand position remains stable for a specified amount of time. The pointing gesture also includes an approximated angle direction. New gestures can be added after a training session consisting of several samples of different users.

Verbal input is handled by a common speech recognition engine such as Dragon NaturallySpeaking. Multiple hypotheses sorted by confidence values are broadcast together with start and stop timestamps. A proper grammar for the current scenario or even a simple word list can vastly improve recognition results. This module can also be replaced by a simple keyboard input dialog for test scenarios or during evaluation.

Head tracking is used in order to locate the user when addressing him during conversation. The head's position in the camera frame is estimated with a Contracting Curve Density (CCD) algorithm implementation optimized for real-time applications [116]. The information is broadcast together with a timestamp and then used in the output modules for controlling the animatronic head. In the future, additional non-verbal information such as face position/orientation (e.g., nodding) and gaze direction (e.g., area of interest) could be included by using additional cameras and a more advanced detection algorithm [115]. Even further detection modules could enable full body tracking and three-dimensional gestures as well.

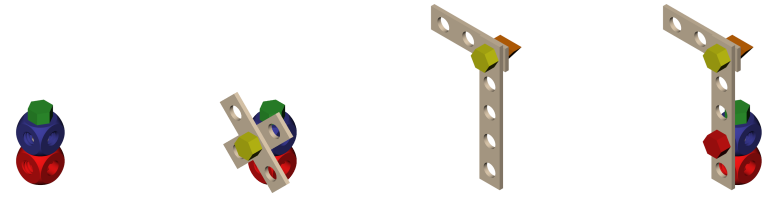


Fig. 4.6 Named Baufix assemblies used during the user evaluation. Some objects can be reused as subcomponents in other assemblies. **a** tower, **b** windmill, **c** L-shape, **d** railway signal

4.2.2 Internal Representation and Decision Making

Modules that belong to the reasoning area of the system cover all aspects related to higher level planning, such as updating an abstract world model representation, the combination and categorization of multiple related input streams, interpretation of non-verbal communication, task planning, as well as dialog management and error monitoring. Depending on the current context and situation as determined by the current state and input signals, proper actions are selected and eventually directed to the corresponding output modules. The JAST project combines two different approaches in order to provide better reasoning and perception capabilities: sub-symbolic processing in the form of dynamic neural fields is integrated with a more traditional symbolic method represented by a dialog management system. The former focuses on inferring the user's intended domain actions based on non-verbal behavior, while the latter concentrates on understanding and generating multimodal natural-language utterances.

The object inventory keeps track of the objects on the table as reported by the object recognition and goal state module. It also stores the location of every object, be it on the table, in the robot's hand, in the possession of the user, or in an assembled state. The representation is persistent throughout variations, for instance when the user covers objects during interaction. Modifications, such as new objects, changes in location or coordinates are broadcast together with timestamp information. Together with information from gesture recognition and robot movement, this data can be used for visualization purposes (Fig. 4.5b). Other modules can also request a list of objects matching a certain description, location, or world coordinates.

Multimodal fusion is responsible for combining input from several verbal and non-verbal modalities (e.g., speech, gesture, gaze) and processing this information in a way accessible to other components. These modalities can also be related to each other in time and may refer to objects in the environment [66]. A human could point to an object and afterwards to a location while asking to move the object to the indicated position. The module has to be able to handle raw text output from the speech recognition module and transfer it to a proper representation based on a grammar [65]. Accurate timestamp information from the speech and gesture recognition modules is essential in combining these modalities. Multiple hypotheses and

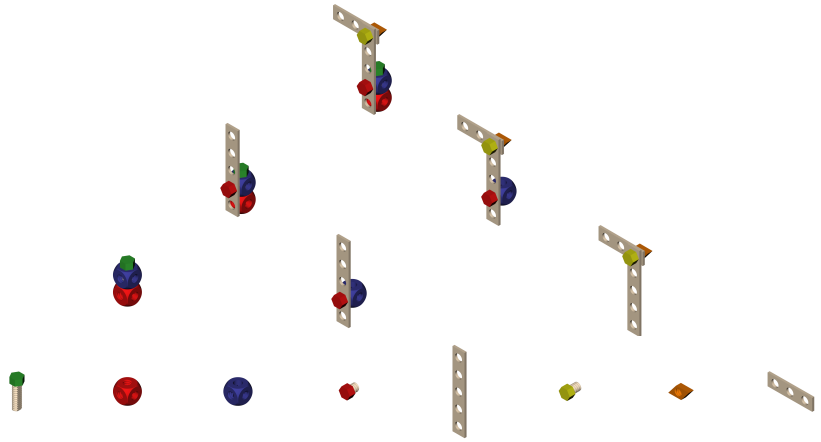


Fig. 4.7 Assembly plan of the railway signal in the evaluation scenario, that can be represented by an AND/OR graph. Different paths and combinations lead to the same goal state. Selected subcomponents present other named assemblies

corresponding confidence values from these modules are useful as well. Information about the environment is provided by the object inventory and helps in resolving pointing information to specific object instances. A single pointing gesture for example might highlight multiple objects and require additional verbal input for clarification.

A proper representation of assembly plans is essential in a joint construction scenario. The system needs to be able to discuss details of the current plan as well as monitor its execution. For this, parts and assemblies are usually given corresponding names and subcomponents can be combined to create new assemblies (Fig. 4.6). Even in the relatively simple Baufix scenario (Fig. 4.3a), multiple assembly sequences are available in order to reach a final goal state. During the construction phase, some pieces may be interchangeable and others can be attached in any order (Fig. 4.7). However, there are a still a number of conditions that have to be respected, such as geometric relationships among parts. An individual assembly step consists of a list of involved objects (typically a bolt, a number of unthreaded pieces and one threaded fastener), an action (place, insert, fasten, or find), a corresponding list of target holes (e.g., end or middle hole), and a unique identifier [56]. A complete plan then contains a sequence of assembly steps, represented in an AND/OR graph. Information on changes in the current plan, the execution of a given step, or completion of a plan is broadcast together with a matching timestamp.

During construction, it is important to select complimentary actions based on the user's immediate and final goal, common task knowledge, and context. Additionally, error detection is critical in case the user is confused about the current assembly state or its temporal order. The system's goal inference and error detection module uses an approach based on dynamic fields [46, 16]. Different pools of neurons in a distributed network of local, but connected neural populations encode task-relevant

information about action means, goals, and context, in the form of self-sustained activation patterns. The module receives data from the object and gesture recognition, multimodal fusion, and goal state modules. It broadcasts the user's final goal (i.e., the target object to be built), the immediate goal based on the current action, any detected conflict or error, and the next suggested action. The latter is based on inferred goals, errors, the distribution of pieces, as well as current and possible next assembly steps. This can include multimodal actions for manipulation and communication patterns for resolving conflicts or errors.

The dialog manager maintains the interaction history and selects an appropriate system output based on the current state of the interaction and of the plan, along with the user's assumed knowledge. It also updates the state of other components depending on current events. Based on the TrindiKit toolkit, it uses the information-state update approach. The information state in the JAST system includes data about the user's knowledge, the current step in the plan that is being executed, the history of steps that have been described to the user, and the history of the interaction. With information on the current goal state, the dialog manager must describe the correct assembly process to the user. It also maintains a list of objects already known to the user throughout the dialog. The plan and object inventory are updated accordingly after the completion of each assembly step [56]. Information from the task plan is also important when deciding on how to refer to objects in the world. An appropriate verbal or multimodal referring expression can be selected based on the context, such as when a part has been used in a preceding assembly step [52].

In the initial version of the system, the robot knows the target object and the corresponding building plan. It instructs the user on following the plan and hands over objects as they are needed. Humans are much quicker and better at dealing with complex construction tasks. Therefore, in this asymmetrical situation, the user follows the robot's instructions and performs all assembly actions while learning to make and reuse several subcomponents [53, 54]. This has been extended to a more symmetrical scenario, where both the robot and the user know the target object and the construction plan. They jointly execute the assembly and the system monitors the user's actions in order to anticipate which piece will be needed next and to detect and deal with any unexpected actions. In order to accomplish this, the goal inference system has been integrated in this version [55]. While the dialog system supports advanced linguistic interaction, it cannot infer the intentions solely based on actions in the world. The dynamic field system on the other hand can detect and reason about non-verbal actions, but uses dialog only for verbalizing activity in the field. The dialog manager incorporates the additional goal inference input alongside the other information from the multimodal fusion module and selects an appropriate action. Combined, the two reasoning approaches can support enhanced interaction patterns that would not be possible with either of the individual systems. This is demonstrated in the corresponding user evaluation in which the user was given a plan with a built-in error and the robot had to detect and explain the error in order for the human to assemble the piece correctly [9].

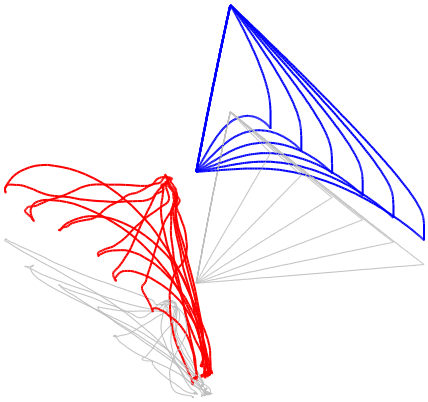


Fig. 4.8 Trajectories captured by an optical tracking system in a handover experiment with a sequence of six cubes. Interaction is shown between a robot giver (*blue lines*) and a human receiver (*red lines*) using decoupled minimum-jerk trajectories. *Gray lines* show projection onto the XY-plane

4.2.3 Multimodal Feedback and Action

In case of the JAST system, input from various sources is processed and analyzed by several reasoning modules until the dialog manager finally chooses an adequate response. Similar to the many available types of input channels used during interaction with human partners, the system should also employ a large variety of output modalities. This includes use of the animatronic head with its facial expressions and voice capabilities as well as both robotic manipulators and their demanding control strategies and real-time constraints.

Instead of a bare setup with only manipulators, cameras, and speakers, the animatronic head provides the user with a natural reference point during dialog and interaction. The robot's head is able to talk and move its lips accordingly for lip-synchronized speech. A number of facial expressions can be activated as well and extended if necessary. This can be utilized to smile if an object has been successfully built by the user or to look confused if the system did not understand the user. Other use cases can include nodding to indicate agreement, thinking expressions while processing information, or even the raising of eyebrows in order to emphasize words or phrases. The gaze behavior can be controlled as well in order to look at the user during dialog or to look at objects on the table when referring to them as a form of inaccurate, but fast pointing gesture. This can be realized with the help of the head's camera and face tracking as well as information from the object inventory [51]. Multimodal output is achieved in cooperation with the robotic arms. These can be moved in order to point at objects and to draw attention to them.

The system can execute a number of operations that have been identified as useful for this scenario, such as pointing at, picking up, putting down, and handing over objects. More advanced assembly actions such as bolting together a screw and a cube were also included in an earlier version. However, these operations have been removed for the evaluations in order to involve the human more in the construction task. The JAST scenario with its imperfect wooden parts is different compared

to optimized industrial settings and the human is much quicker at handling them anyway. Information such as the current joint configuration or tool position can be requested and used for visualization or other purposes. Additional data can also be used for error monitoring, such as feedback from the gripper sensor in order to validate the correct dimensions of objects that have been picked up or after an object has been given to the user. In a similar way, force/torque data is used when approaching parts on the table or during hand-over. This also includes an abstraction of inverse kinematics, where the reachability of an object on the table can be tested in advance. During interaction, the user needs to be able to barge in at any time and the system must permit adaptation and change of its behavior instantly. This can include changes in the current goal as well as interruptions from the user, such as entering the common work area of the robot manipulators. Especially in contrast to classic industrial settings, where there is no direct interaction between human and robot, appropriate safety regulations in the form of sensors and other resources are crucial. It is important to avoid offline trajectories calculated in advance and adhere to a strict online control strategy. Section 4.3 discusses this aspect in more detail. However, not only is it vital to ensure proper compliance with safety protocols, the robot's movements also have to be predictable to the user (Fig. 4.8). The trajectories of industrial robots in particular often appear extremely unnatural to users. Movements that are unexpected, unfamiliar, or too fast can interfere with efficient cooperation. In this setting, the handing over of objects between robot and human plays an important role during assembly. The overall acceptance and performance of this task can be improved by implementing advanced trajectory profiles that are similar to those observed in interactions among humans [79, 77, 78].

4.3 Distributed Task-Based Control and Sensor Integration

Apart from obvious safety aspects in the cooperation between human and robot, this kind of interaction also demands more advanced control concepts besides traditional trajectories in joint or operational space. This especially regards areas such as direct physical contact with proper force control or task concepts that require various forms of constraints. Ideally, this also includes a robot system with low-level torque control and its direct control possibilities. Interference between these tasks needs to be prevented in order to avoid uncontrollable or unwanted behavior. The fact that changes due to sensor data (e.g., force/torque control, obstacle avoidance) or modifications to the task itself (e.g., goal position, task constraints) can happen at any moment requires an online control system. This also disqualifies optimized or precomputed static control programs.

Based on concepts from task-based control (Sect. 2.1.6), a number of tasks T_k with corresponding constraints can be combined into a single action A. These tasks can range anywhere from position and force/torque control, to constraints on degrees of freedom in operational or joint space, visual servoing, and obstacle avoidance, or more advanced concepts such as proper balance in a humanoid. Addition-

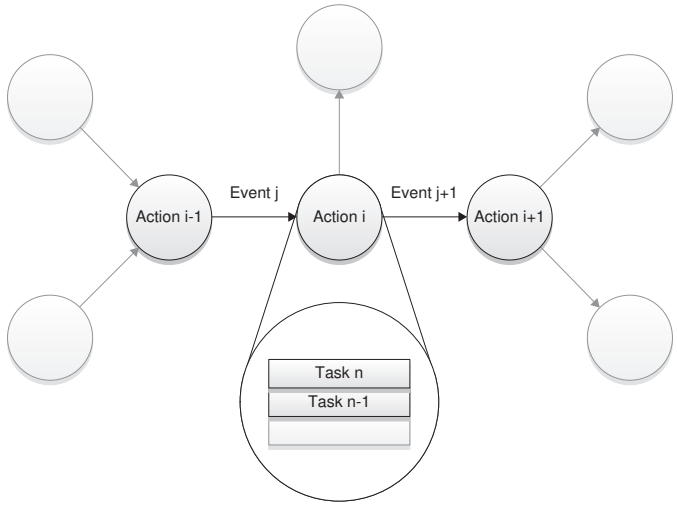


Fig. 4.9 Behavior description consisting of a series of actions and events. Actions combine a series of prioritized tasks with constraints in order to achieve a desired output. Events handle the control flow to the next state

ally, several of these actions can be combined through events in order to create advanced behaviors (Fig. 4.9). Such events can depend on task definitions (e.g., a goal position), time constraints, input from other modules, or even the user. A proper projection rule $\triangleleft_k$ for each task guarantees consistent behavior and provides a well-defined hierarchy. Depending on the actuator system, the torques or velocities from each task as well as their projection rule are calculated in each step and transformed into the superior subspace, ending with the highest ranking task T_n:

$$A = T_n \triangleleft_n T_{n-1} \triangleleft_{n-1} \ldots \triangleleft_1 T_0 \,. \tag{4.1}$$

These task-based control programs have to be able to embed dynamic sensor information (e.g., visual servoing) and also require the compliance with various safety aspects and real-time constraints. Sensor and actuator hardware in particular usually demand strict cycle times. Due to the complexity of various tasks it can be required to distribute their computation among several computer systems (e.g., advanced image processing). Obstacle avoidance typically uses an internal three-dimensional representation that can be static or even dynamic. Dynamic obstacles have to be detected and updated using a large selection of sensors and algorithms. These environment models may also have to be managed and updated among a distributed system with multiple processing units and their respective sensor devices.

This kind of control architecture can be divided into five layers (Fig. 4.10). In order to separate control programs from specific hardware implementations, a generic abstraction interface between various components is advisable. This also includes proper descriptions for kinematics, dynamics, and geometry data. Such common interfaces can facilitate easy integration of alternative or even new components in

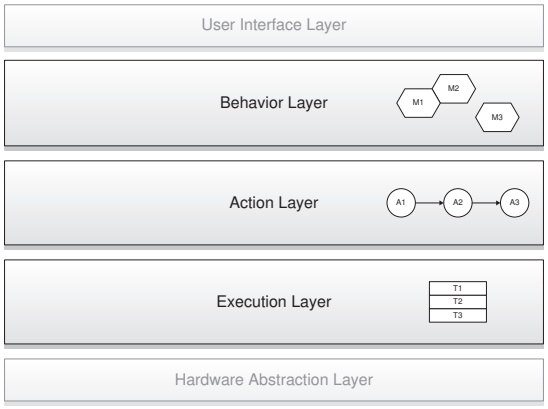


Fig. 4.10 Classification of a control architecture into behavior, action, and execution layers. Hardware abstraction and user interface layer complete the design

a working system. Communication on this hardware level is strictly synchronous. The execution layer is directly attached to this hardware abstraction layer and requires hard real-time constraints. The robot's kinematics and dynamics equations have to be evaluated here, as well as task constraints such as position and force control, visual servoing, or obstacle avoidance. This also includes reading and updating actuator and sensor information (e.g., force/torque, camera, or 6D mouse data). In addition, this layer is responsible for projecting the various tasks of a single action into the proper subspaces as defined in the corresponding hierarchy. Temporal sequences of these primitives are handled in the action layer. Events (e.g., a goal position, a force threshold, the press of a button) handle the change from one action primitive to the next and relay this information back to the execution layer. These sequences can be defined by a programmer, as well as be generated or changed by modules in higher layers, for example using a visual representation or a guided dialog. More advanced behavior patterns are established through the behavior layer. Here, the system can also communicate with other modules and is able to decide when to activate other action sequences (e.g., by the dialog manager in the JAST system). Modules can also update and inquire information from the same world model representation in order to implement a global strategy, such as when exploring or mapping an unknown environment and computing global navigation plans. Finally, input from the user cannot guarantee any kind of real-time constraints. The user can however modify and influence current control programs or even specify new tasks, actions, or behaviors.

Some of the tasks in a control program are critical and the system has to calculate proper responses for every control cycle (e.g., gravity compensation for the actuators). Depending on the hardware this results in requirements of 1000 Hz or higher. Other components however might not be as critical or simply impossible to calculate within the given time frame (e.g., advanced image processing). The update rates of these tasks can then be handled asynchronously to the rest of the core

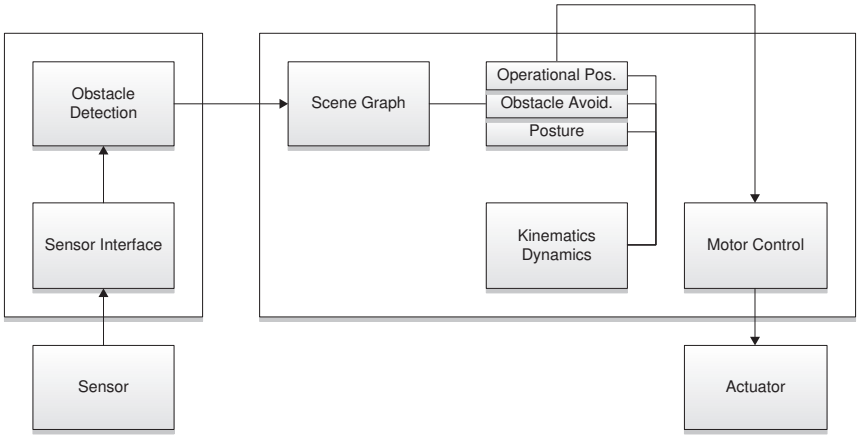


Fig. 4.11 Distributed control between two machines. Sensor data from a camera in combination with an obstacle detection is used in order to modify a scene graph representation. An obstacle avoidance task uses this information as input and generates a proper control output

system. Due to the combination of tasks through target vectors and projection rules, only these results have to be exchanged between different components and the main computation can be handled in separate threads, processes, or even machines. Some tasks can also handle other forms of input and only need to exchange partial data. The type of communication between components is flexible and only depends on corresponding real-time requirements. It can range from direct communication, to shared memory, or even network protocols. A visual servoing node can for instance be directly connected to a camera, use this information in order to compute a new target position, and then send the result as input to a goal position task running on the main control node rather than raw image data. It can also use other types of communication in order to gather images from multiple distributed cameras and combine the resulting data before generating and sending a target position. In another scenario, data of various obstacle detection modules on different nodes can be combined into a shared three-dimensional world representation. An obstacle avoidance task on the core node can then use this model in order to calculate distance information and combine the output with other tasks before sending it to the motor control (Fig. 4.11).

In the JAHIR (Joint-Action for Humans and Industrial Robots) project, this concept has been applied to an actual human-robot interaction scenario. Compared to the JAST evaluations, this setup focuses more on joint action in an industrial setting. Here, a human cooperates with an industrial robot across a working table that represents a common workspace. The robot is equipped with a force/torque sensor, as well as an automated tool changer with several different equipment options, and features a real-time interface with access to position-based control. Optical trackers on the user's upper and lower arm as well as the palm are used in order to update a dynamic world representation within a control architecture similar to the one

presented in Fig. 4.11. Nodes can dynamically add, update, and remove obstacles from the scene graph using an object-oriented middleware [67], where each obstacle is associated with a unique identifier and can consist of an arbitrary number of shapes. Many different sensors and algorithms for the detection of the human's position in the workcell and other objects can be integrated with this approach. Various other devices are installed in the setup, including top-mounted optical cameras, a projector, and a time-of-flight camera, as well as a laser range scanner and pressure-sensitive safety mats. The projector can be used in order to visualize the current model used for obstacle avoidance. This way, the human operator can verify that his movements in the vicinity of the manipulator are registered properly.

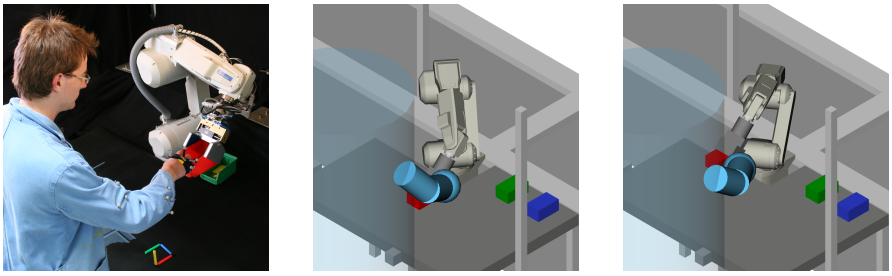
A series of experiments shows the capabilities of such architectures in joint action environments [102]. The first application scenario demonstrates the use of an industrial manipulator as a mobile storage box (Fig. 4.12). The robot's task is to provide access to different assembly parts depending on the context and the current production step. Live data from the tracker on the palm is used in order to place the box within reach of the user. This is implemented with a goal position task that only acts on positional data in operational space. Due to the loose content of the open box, it has to be kept upright at all times. This can be achieved by including a task with restrictions on the rotational degrees of freedom in operational space. For safety reasons, an obstacle avoidance task with access to the dynamic scene graph representation is included. If the distance to the user falls under a specified threshold, the system is instructed to perform a full stop. Additionally, a posture task tries to achieve a more accessible, upright joint configuration:

$$A_1 = T_{\text{orientation}} \lhd T_{\text{avoidance}} \lhd T_{\text{position}} \lhd T_{\text{posture}} . \tag{4.2}$$

In order to assist the user during complex assembly procedures, the second scenario features the manipulator equipped with a light source as its tool (Fig. 4.13). The main position task has its target at the object's position on the table in operational space, with a virtual tool corresponding to the focus length of the light. The focus of the spot light is directed at the target object as a result. A posture task is also included, as the ideal position for the light source is directly above the target object. However, the user might occlude the light during assembly and the direction has to be adjusted accordingly. Obstacle avoidance is therefore divided into two representations: the first one handles only the human and a model of the robot with a virtual light cylinder and has a lower priority. The second one omits the cylinder, but adds the full environment model in order to handle safety aspects during the interaction:

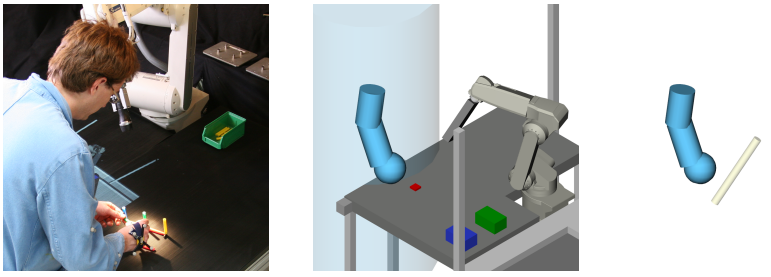
$$A_2 = T_{\text{position}} \lhd T_{\text{environment}} \lhd T_{\text{posture}} \lhd T_{\text{light}} . \tag{4.3}$$

A final example uses a top-mounted camera for object recognition (Fig. 4.14). The camera requires a direct line-of-sight to regions of interest on the table. Due to this, the robot can perform various tasks in cooperation with the human, but should avoid interfering with the image processing component if possible. This can be addressed by creating two obstacle avoidance tasks with different priorities as explained in the previous scenario. Virtual cylinders are used in order to guard the



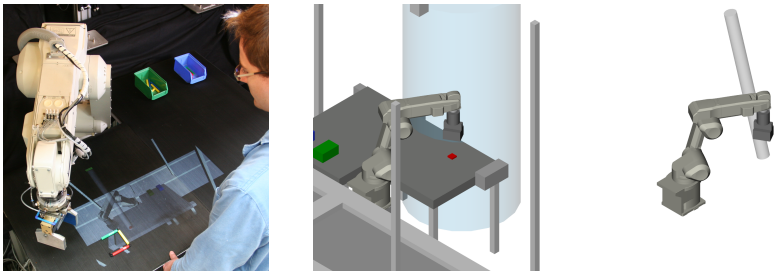
(a) (b) (c)

Fig. 4.12 First experiment in the JAHIR setting. Sensor data from an optical tracker is used for assistance and dynamic obstacle avoidance. Movement is completely stopped if the computed distance falls below a given safety threshold. **a** real-world scenario, **b** collision detection representation, **c** reaction to movement



(a) (b) (c)

Fig. 4.13 Support in form of an automated lighting system. The robot tries to avoid postures that conflict with the human's arm. **a** implementation on the robot, **b** first collision scene for obstacle avoidance, **c** second scene used for line-of-sight



(a) (b) (c)

Fig. 4.14 The robot system moves to a goal position without crossing the top-mounted camera's view on an object on the table. **a** actual robot system with projected visualization on the table, **b** collision scene representation used for obstacle avoidance, **c** second scene reflecting the camera's region-of-interest

camera's regions of interest. Other tasks can then be integrated into this hierarchy with respect to their priorities:

$$A_3 = T_{\text{environment}} \triangleleft T_{\text{orientation}} \triangleleft T_{\text{line-of-sight}} \triangleleft T_{\text{position}} \triangleleft T_{\text{posture}} . \tag{4.4}$$

Rather than requiring an expert with extensive training to develop individual robot movements using each manufacturer's programming language, a human operator can instruct the system with minimal training. Tasks can be used in order to create a set of basic manipulation or interaction skills. Analog to the JAST setup, common requirements include high-level behaviors, such as moving to a resting position, picking up objects from the table or conveyor belt, handing over objects, or changing end effector tools. As mentioned before, these skills consist of a series of actions and events that can involve any type of actuator or sensor device. Picking up an object for example will typically involve placing the end effector above an appropriate goal position (e.g., from an object recognition module), slowly moving downwards until either a certain distance or force threshold value is reached, closing the gripper, and advancing upwards. Using an appropriate user interface or even dialog management, the operator can combine several actions and events of various complexities to a new behavior. For this, he can choose from several building blocks, including sensor devices for identifying equipment in the tool changer, the status of signal lights in the vicinity of the worker, and interactive program control through virtual buttons (using object recognition and the top-mounted projector).

4.4 A Framework for Developing Robotics Applications

The previous sections focus on concepts and interface specifications of a distributed interaction system rather than new middleware implementations. Existing solutions are well suited for these applications, as long as all operating systems and programming languages are supported [68]. As discussed in Sect. 1.1, the main issue in developing these systems often arises from providing basic robotics functionalities in the individual components. Common tasks, such as mathematics, kinematics and dynamics calculations, or hardware drivers, usually have to be reimplemented from scratch and new algorithms typically depend on these basic functions. Not only does this apply to other parts of the development process, such as visualization or simulation environments, but also to stand-alone applications in research, education, and industry.

The framework presented in this section addresses many of these issues and has been used in the development of various projects, including the earlier presented JAST and JAHIR systems, as well as the motion planning benchmarks. Most parts of the C++ libraries have already been released as open source and are freely available[1]. In the development workflow, new robot types can be added by specifying proper kinematics, dynamics, and geometry definitions. Hardware components

[1] http://roblib.sourceforge.net/

are classified into several actuator and sensor categories and this selection can be extended by providing appropriate drivers. Other elements include compatible interfaces for several visualization, collision detection, and physics simulation libraries, as well as an implementation of motion planning and task-based control.

With reusability in mind, the components of the framework can be combined for building a large number of applications in the robotics domain. Kinematics, dynamics, and geometric representations provide a base for motion planning algorithms and task-based control. In combination with proper visualization, the simulation of robot movements can be tested before actual execution on the hardware. Collision detection and distance computation enable obstacle avoidance and other safety aspects in a real-time control architecture.

4.4.1 Mathematical Foundations

Good support for vector and matrix operations are among the most common requirements when building robotic applications. Most algorithms in kinematics, dynamics, and other areas are built around vector and matrix operations such as addition or multiplication, as well as advanced calculations including LU, QR, and SVD decomposition, eigenvalue and eigenvector computation, or linear equation solvers. While a large number of implementations and mostly incompatible data containers are available to developers today, none is usually able to natively achieve the complete functionality of BLAS (Basic Linear Algebra Subprograms) [101, 17], LAPACK (Linear Algebra PACKage) [3], and similar collections. A viable alternative can be found in the form of numeric bindings [105], where a generic layer between data containers and algorithm packages allows for interoperability between different implementations. This method uses the traits technique [111] in order to provide interfaces that are separate from any specific container package. Generic functions provide support for single and double precision as well as complex arguments and can determine elements such as size, stride, or matrix layout automatically. Optimized implementations for specific hardware architectures or vector and matrix dimensions can also be integrated in this fashion [145, 146]. Regardless of the specific implementation, the underlying data containers should provide proper access to subranges, rows, columns, and similar features that are often required when adding algorithms. Overloaded vector and matrix operations such as addition and multiplication are usually more convenient than generic function calls; however this often results in poor performance due to temporaries and has to be addressed with expression templates [148].

Besides these basic features, geometry computations in robotics often involve operations such as translation, rotation, or scaling. Section 2.1.1 shows how homogeneous transformation matrices are one form of combining these aspects, including optimized calculations for inverse operations or structures for Denavit-Hartenberg links. Orientation or rotations are traditionally specified using 3×3 matrices (2.6), Euler angles (2.16), quaternions (2.18), or axis/angle combinations (2.25), therefore

requiring conversion between these representations, as well as handling distance, interpolation, and vector rotation accordingly. Other frequent computations may include PID controllers, Kalman filters, or trajectory interpolation between configurations. While combining two rotations is more efficient in quaternion form rather than by matrix multiplication, the cost of using them for vector rotation is much more expensive, also when performing both translation and rotation. By exploiting the special orthonormal structure of rotation matrices during matrix multiplication, this cost can be reduced even further. Optimized data structures and operations however become much more important when performing spatial vector algebra. Multiplying two 6×6 spatial transformation matrices is six times more expensive than the relatively low cost of combining the involved rotation and translation components. This performance issue becomes even more apparent when transforming rigid body components across frames, an operation where three matrices of this size are involved. More efficient spatial data structures and formulas as proposed in [48] can be used in order to speed up calculations like the ones in Sect. 2.1.5. Spatial motion (2.52) and force (2.68) vector types $\vec{\boldsymbol{v}}$ and $\vec{\boldsymbol{f}}$ are each described with two 3×1 vectors for angular/linear velocity $\boldsymbol{v}$ and $\boldsymbol{\omega}$ or momentum/force $\boldsymbol{n}$ and $\boldsymbol{f}$ respectively. In combination with storing the rotation matrix $\boldsymbol{R}$ and translation vector $\boldsymbol{p}$ components of a spatial transformation $\boldsymbol{X}$ rather than the evaluated matrix, the forward and inverse transformation operations of both motion (2.53–2.54) and force (2.69–2.70) domain can be efficiently expressed. The spatial inertia matrix $\vec{\boldsymbol{I}}$ (2.104) for rigid bodies can be described by its mass m, center of gravity vector $\boldsymbol{c}$, and inertia matrix $\boldsymbol{I}$. For the latter, using a lower-triangular matrix is sufficient due to its symmetry (2.75). In a similar fashion, the articulated-body inertia matrix $\vec{\boldsymbol{I}}^{\mathrm{A}}$ requires three 3×3 matrices for its center of gravity, mass, and inertia representation, where only the latter two are of lower-triangular structure.

4.4.2 Kinematics, Dynamics, and Metrics

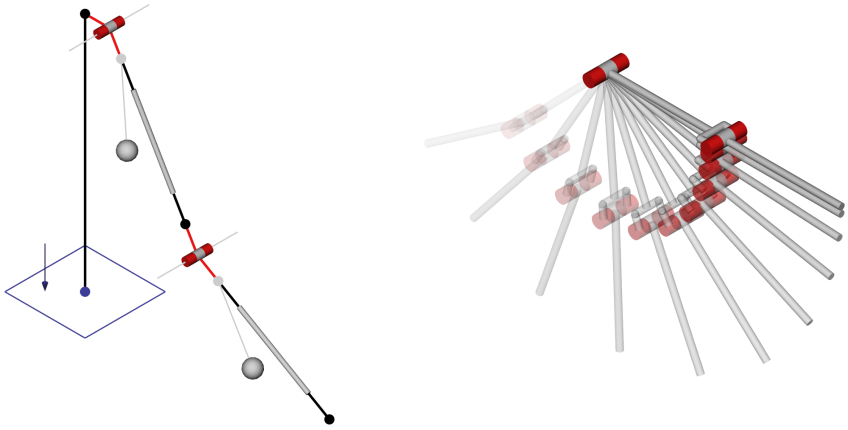
As outlined in Sect. 1.1.2 and 2.1, the number, type, and properties of joints and rigid bodies are part of the robot system's physical properties. A robot application will often involve calculations such as forward and inverse position kinematics, as well as velocity and force transformations. Angles and displacements of individual joint configurations can be transformed to a set of coordinate systems that specify the position and orientation of the robot's rigid bodies in three-dimensional space. Dynamics algorithms such as the recursive Newton-Euler formulation help determine matching joint torques for a given trajectory with desired velocity and acceleration profiles. The system's Jacobian matrix (Sect. 2.1.3) provides a connection between velocities and forces in configuration and operational space. A change in the kinematics and dynamics parameters of the system should however not result in a complete redesign of the application.

Parameters for joint and link configurations are often given in Denavit-Hartenberg notation (Sect. 2.1.4). Due to restrictions in the placement of reference frames, this

convention only requires four values in order to describe the length, twist, and offsets of each link. Joint movement is restricted to rotations and translations along the respective $\hat{\boldsymbol{z}}$ axis, resulting in very efficient formulations for kinematics and dynamics algorithms. Symbolic optimization techniques are another way of achieving high performance, while still allowing for elaborate and accessible descriptions [114]. This includes evaluation of constant expressions, elimination of redundant assignments and operations, as well as a number of other algebraic simplifications. As a consequence, restructuring of a system is usually limited to the adjustment of several parameters, since the addition, removal, or substitution of joints and rigid bodies would require a new optimization step. Using object-oriented design, the individual equations can be arranged into specific components such as joints and rigid bodies [86, 72]. These entities provide support for a number of operations on elements such as velocity, acceleration, and force. New entity types can be integrated in a similar fashion and each implementation can be optimized separately. This allows for a flexible composition of model descriptions even during runtime.

Figure 4.15a shows such a representation of a basic double pendulum model in graph form. This example consists of two cylindrical rods moved by two joints revolving around their $\hat{\boldsymbol{z}}$ axis. The system is translated and rotated with respect to the ground floor and is subject to gravitational force acting downwards. The propagation of velocity (2.106), acceleration (2.107), and force (2.113) in such a system can be expressed with spatial vector algebra (Sect. 2.1.5). Each fixed translation and rotation, joint transformation, and exerted rigid body force can be expressed with corresponding spatial transformation and inertia matrices as well as direction and free motion descriptions. Position, velocity, acceleration, and force propagation equations are reflected in the directed edges of the graph. The formulas in their individual complexity are implemented in the corresponding transmission object. An element representing a fixed translation for example does not have to consider joint velocity or rotation aspects. The input and output values of these equations are stored in the attached vertices. This also includes inertia matrices and bias forces for rigid and articulated bodies. Similar to transmission elements, the objects matching these vertices can be used in order to represent or calculate influences such as external or gravitational force (2.112). Models with Denavit-Hartenberg parameters can be integrated using a combination of basic components or by providing optimized implementations. This may also include an assembly in the form of a series of joints with additional functions such as analytical inverse kinematics.

The state of the model can be initialized by providing position, velocity, and acceleration data for the respective joint elements. Kinematics and dynamics algorithms can be integrated through the addition of related functions to the elements of the graph. Coordinate systems for the individual links in the current configuration are updated by applying the corresponding translations and rotations of each joint in the proper order. The recursive Newton-Euler method is represented by a forward and backward iteration through the model. The first pass is used in order to propagate the velocities and accelerations from the base frame to the end effector vertices. Additionally, the force exerted by all rigid bodies due to gravity is computed in this step. These forces are finally iterated in a reverse order toward the base frame and



(a) (b)

Fig. 4.15 Kinematics and dynamics of a double pendulum. **a** model description in graph form with world (*blue*), rigid body (*gray*), and frame vertices (*black*) as well as fixed translation/rotation (*black*) and revolute joint edges (*red*) **b** forward dynamics calculation and integration with initial position, velocity, acceleration, and torque set to zero

the resulting joint torques are calculated. In a similar fashion, the articulated-body algorithm for forward dynamics can be integrated with three iterations (Fig. 4.15b). Besides kinematics and dynamics formulations, a robot model also defines a corresponding metric (Sect. 2.1.2). Specific joint types can provide custom implementations for distance calculations and interpolation. This can also include decisions about Euclidean, Manhattan, or other metrics, depending on structure and target application.

4.4.3 Hardware and Operating System Abstraction

Finding a proper open source or even proprietary driver for a piece of robotic hardware is difficult enough in most cases. Basic access to a device is not always sufficient however, as this aspect also has to pay attention to modular systems and otherwise exchangeable hardware. The physical properties of a robot system (Sect. 1.1.2) include the selected components and their individual communication protocols. Replacing an actuator or sensor with a device of similar functionality should be simplified by providing compatible interfaces. This is also important for aspects regarding the underlying operating system, such as standardized access to mutexes, semaphores, threads, and high-precision timers or conversions regarding endianness.

Common devices in robotics include motors, range and force sensors, grippers, cameras, as well as digital or analog input/output modules. While many different

devices of these categories include capabilities that are specific to a certain manufacturer, the basic functionalities and features in the same hardware class are similar. A camera typically provides access to the captured image data and a number of adjustable parameters. LIDAR (Light Detection and Ranging) systems deliver an array of distance values together with information about angular resolution. Some units also include intensity data or the status of safety and warning fields. Depending on the manufacturer, robot manipulators offer either position, velocity, or torque control and feedback. This can include the ability to provide custom real-time trajectories in joint or operational space. All devices require different real-time constraints and communication protocols.

In order to provide common interfaces for devices that can be categorized into the same hardware class, an object-oriented hardware abstraction layer is introduced. Rather than implementing multiple instances of the same basic commodity, devices can use standardized interfaces for functionalities such as serial ports, sockets, threads, or timers. Basic device classes with features shared among specific manufacturers and units can be extended in order to integrate new hardware components, e.g. cameras, range sensors, or position actuators and sensors in joint space. Communication with devices is categorized into functions for establishing and closing communication channels (e.g., sockets, bus interfaces), starting and stopping operations (e.g., locking brakes), as well as periodical data transmission and acquisition. The latter is performed based on the individual update rate of the unit and performs all send and receive operations required for updating the current state. Input and output ports of the individual device class are used in order to convey desired actions and to listen to new responses. In case of a manipulator, inputs and outputs include target joint positions for the next iteration and the current joint configuration. A camera provides the captured image data, whereas a range sensor returns its measured distance values. Specific units and their implementations offer access to additional information apart from the common interfaces.

4.4.4 Scene Graph Abstraction

The definition of the individual shapes of the robot's links and its surroundings completes the model description. In combination with kinematics, dynamics, and metrics, the geometric representation is essential in various computations, including collision detection and physical simulation (Sect. 2.2). Kinematics and geometry are linked through the coordinate systems of the robot's links and their corresponding rigid bodies. Each link can be composed of several basic or advanced shapes, such as boxes, spheres, polygons, or convex hulls. Due to the high detail of CAD models and other aspects such as lighting and material properties, different models are typically used for visualization (Fig. 4.16b) and operations involving complex and time-critical calculations (Fig. 4.16c).

Ideally, a geometric model is specified in a common data structure with a large number of support functions for creating and manipulating shapes and related ob-

(a) (b) (c)

Fig. 4.16 Physics simulation and visualization of an omnidirectional mobile platform. **a** scenario with two platforms on a table, camera view frustums (*green* lines), and distance sensors (*orange* and *red* lines), **b** detailed model used for graphics, **c** reduced model used for physics

jects [47]. Implementations and algorithms for graphical output, collision detection, penetration depth calculation, and similar computations can access this data and provide advanced functionality. The available libraries in this area however usually provide their own data structures and interfaces due to a lack of standardization and individual implementation aspects. Most libraries either do not support loading geometry descriptions from disk or introduce proprietary file formats. In order to provide a common level of abstraction for similar algorithms and use cases, the various objects involved in an application can be structured into a hierarchy of scenes, models, bodies, and shapes. Scene instances in different implementations are synchronized through the frames of the corresponding bodies. A common format such as VRML[2] (Virtual Reality Modeling Language) is used in order to introduce global file support. Scene descriptions can be created through a combination of different models from a component library and multiple instances of the same geometry can be used in order to increase performance and reduce memory costs. While this geometric description is sufficient for visualization and collision detection implementations, physics simulations require additional properties, such as mass, inertia, constraints, or friction. The COLLADA[3] (COLLAborative Design Activity) file format on the other hand provides support for exchanging both visual and physical attributes of a system in the same document. Besides reflecting the aforementioned scene structure, instances of geometry shapes and inlining of existing files together with growing usage across the field facilitates integration.

Collision detection queries comprise tests regarding the complete scene and between individual models, bodies, and shapes. Broad and narrow phases can take advantage of this structure to increase performance. Intersection between selected bodies or shapes can be disabled on request in order to prevent overlapping geometries in adjacent links from interfering with the result or to suppress irrelevant queries. Similar interfaces are available for distance and penetration depth compu-

[2] http://www.web3d.org/x3d/vrml/

[3] http://www.khronos.org/collada/

tation. The former includes calls that return the distance from a custom point to the nearest object. Physics simulation calculates the influences of forces, contact responses, and rigid constraints in each step and updates the current world state accordingly. External forces and torques can be added by the user and an array of contact points is provided on demand. Figure 4.16a demonstrates how all of these aspects can be combined in order to create a simulation system for a mobile robot. This platform is equipped with three omnidirectional wheels, that are represented by spheres with different friction values for each direction. Raycasts provide distance information for the range sensors in the lower half of the base and collision response from a specific shape around this area serves as input for the system's bumper feedback. The visual scene is synchronized to the coordinate systems of the rigid bodies and can show the virtual sensor data. The robot's camera output is generated based on the graphics in this scenario and can optionally simulate effects like distortion using fragment shaders. Additional hardware components such as grippers or LIDARs are supported in the same way. An industrial application based on these principles supports the same communication protocol as the actual hardware of the mobile platform and can function as a replacement during development, educational training, or machine learning tasks.

4.4.5 Motion Planning

Comparing modern path planning algorithms is already difficult due to influences of random sampling. Different underlying implementations of basic mathematics, kinematics algorithms, geometry descriptions, and collision detection libraries only add to these issues. Many planners share a number of basic principles or are even based on each other. In addition, all planners typically provide the same basic output in the form of collision free paths or trajectories. These algorithms may have different use cases or perform better than their counterparts in certain scenarios. Being able to choose between a number of several options is therefore highly desirable and has to be manageable with little modifications in the target application.

The central element in all planning algorithms is a model of the robot system and its environment, including kinematics, dynamics, geometry, and metrics. The previous sections demonstrate how each of these aspects can be defined in an abstract way that allows for changes in structure and setting. Based on the number of joints and the connected links, a configuration generated by the planner is translated into coordinate frames for the system's rigid bodies. Collision detection, distance and penetration depth computation are performed based on this information and the results directly influence the planner's decisions. Metrics and interpolation procedures are also included in this model and can be accessed during runtime.

Rapidly-Exploring Random Trees (RRT) and Probabilistic Roadmaps (PRM) are amongst the most commonly used algorithms in motion planning and a large number of implementations are based on these two principles (Sect. 3.2). Variants based on single or multiple trees and different extension strategies are found

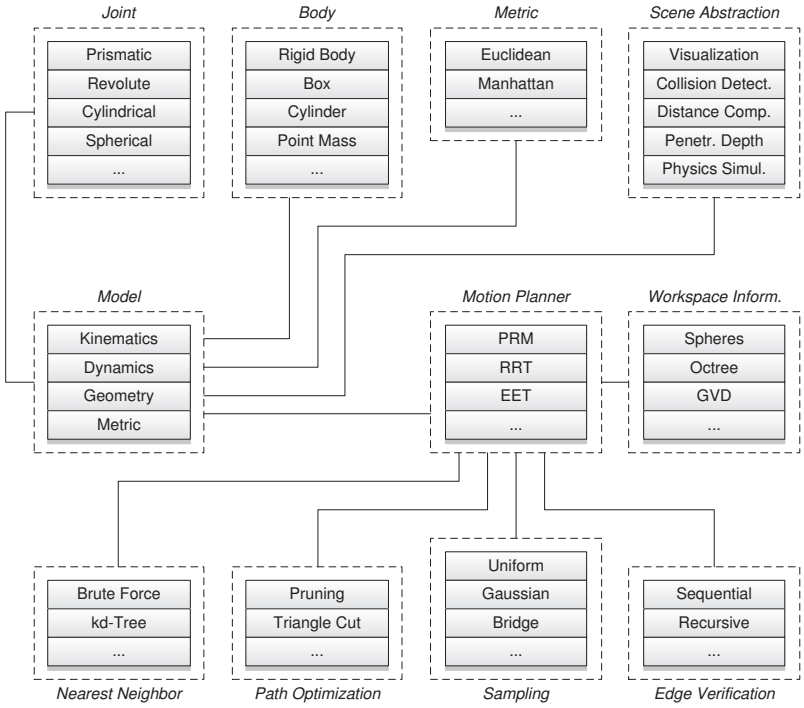


Fig. 4.17 Overview of the elements involved in a motion planner specification. A robot model is defined by its kinematics, dynamics, geometry, and metric. Different sampling, edge verification, and nearest neighbor strategies, as well as sources for workspace information can be selected. Further optimization techniques can be applied to a collision free path

in RRT-based planners. Many different sampling strategies have been introduced especially for PRM-based implementations. Both rely on versatile graph and tree structures [138] as well as good random sampling methods [104]. The cost of adding and removing vertices and edges as well as increasing complexity of nearest-neighbor calculations demands special attention.

The same amount of abstraction that has been applied to the robot's model and the planner design is also important for the various components of a motion planner (Fig. 4.17). A PRM-based planner can choose between uniform, Gaussian, bridge test, and several other sampling techniques without a change in its algorithm. The same uniform sampling component can be used for other sampling-based implementations. All of these samplers require access to the robot's kinematics model, metrics, and the selected collision detection representation. With the same interfaces, edge verification in a roadmap can be performed in a sequential or recursive fashion. Brute force computation or more efficient structures such as k-dimensional trees can be employed for nearest-neighbor selection. As outlined in Sect. 3.3, there are several sources of workspace information that can be integrated into motion planning algorithms. In case of workspace spheres, this is achieved in combination

with distance computation and the geometric model of the environment. Finally, the resulting collision free movement of a query can be further optimized in order to create shorter and smoother paths or trajectories.

4.4.6 Task-Based Control

In addition to the kinematics and dynamics of a robot system, the control architecture requires access to all hardware components. The abstraction layer introduced in Sect. 4.4.3 simplifies the integration or substitution of devices from different manufacturers and the adaptation of new real-time operating systems. New functions and algorithms are integrated through the definition of corresponding tasks and events (Sect. 4.3). Based on its calculations, a task generates a velocity or torque signal and passes this output on to the next object with higher priority as defined by its parent action. This task can project this input into a nullspace that does not interfere with its own output signal. The actions in the behavior are connected through selected event objects that switch to the next step in the program when triggered.

Task implementations include basic operations such as moving to an operational position or a joint posture. The former's target frame can be selected from the number of coordinate systems in the robot's kinematic model and individual degrees of freedom in operational or joint space can be disabled on request. The target position of these tasks can either be static or updated based on data from user input or sensor devices. Other tasks comprise the integration of external forces or torques in both spaces and advanced calculations such as obstacle avoidance. As demonstrated in the previous sections, the latter uses distance information from a geometric representation that can be dynamically updated. An action can also simply execute a specific function, such as opening a pneumatic gripper or performing a delay for a specified time frame.

A behavior switches to the next action in its program upon the completion of certain events. This can include crossing a specified threshold of a linked sensor value or reaching a target position. Events can also be coupled with Boolean expressions in order to create a desired reaction. Global events can implement safety aspects such as an emergency stop upon falling below a safety margin in distance computations.

Chapter 5
Conclusion and Future Work

Classic methods of programming robot systems are rapidly approaching limits where solutions to complex applications are no longer feasible. Advanced integration of sensor devices for handling new object classes and especially safety issues in human-robot interaction scenarios demand corresponding engineering standards. Where fences have been traditionally used in industrial settings, these aspects will enable new possibilities for production efficiency and small batch series. This includes the introduction of new programming paradigms that enable personnel with no expert training to instruct these systems.

Instead of manually optimizing tasks such as pick-and-place operations, motion planning algorithms can be used in order generate collision free paths and a corresponding instruction in the program is less likely to be misinterpreted by an operator than a series of arbitrary joint movements. Provided the geometric representation is updated accordingly, this also considers changes such as rearranged target positions or obstacles in the robot's vicinity. Resulting paths of the planners have to be available within a short amount in order to respect the operator and the production time-frame. Ideally, this runtime is fast enough to react to dynamic changes. The generated movement also has to appear reasonable or the operator will fall back to manual optimization. The planning algorithm presented in this work addresses these issues by introducing the concept of exploration and exploitation. By deliberately balancing the two opposing forces, the amount of configuration space exploration can be reduced to a minimum. Workspace information is used in order to gather global connectivity data and to increase efficiency. The balance between exploration and exploitation is adjusted according to the difficulty of the local planning problem. Experimental results demonstrate the improvements in performance in a series of real-world scenarios.

User-friendly interfaces and programming paradigms can be realized by a combination of multimodal instructions with task-based control algorithms. Complex procedures can be described through a series of actions and events. Prioritized tasks can perform required robot movements while respecting additional constraints and integrating online sensor information. Descriptions are not hard-coded for a specific kinematic structure or hardware devices and can be transferred to other se-

tups. The feasibility of such an approach and the associated possibilities have been demonstrated in a series of example applications. No expert knowledge is required in order to interact with such a robot system. The incorporation of intelligent behavior through the combination of symbolic processing and dialog management with sub-symbolic reasoning, goal inference, and error monitoring has been evaluated together with the influence of specific robot trajectories on human-robot interaction.

In order to refine software engineering aspects when building applications in this domain, a framework has been developed that integrates basic functionalities such as kinematics and dynamics foundations, hardware abstraction, visualization and collision detection methods, as well as motion planning and task-based control algorithms. This software has been used in the development of several projects in research, industry, and education. The need for reengineering of common features is avoided and new applications can be created faster and more efficient. Most parts of the framework have been released as open source and are freely available.

5.1 Further Balancing of Exploration and Exploitation

The evaluations of Sect. 3.4.4 have demonstrated the effectiveness of actively balancing exploration and exploitation in motion planning. Incorporation of additional information such as repulsive forces can increase efficiency even further in some settings, even though the actual cost of computing this data is more expensive. Several other methods of gathering valuable input are available, for instance using utility information [26, 29] or statistical analysis [40]. Integrating and exploiting these alternative data sources is an interesting objective. Rather than manually estimating the usefulness of extra information however, an algorithm should automatically decide which exploitations are worth the additional computation cost.

Elastic roadmap planners (Sect. 3.3.4) can integrate online task-control with global navigation. Multiple constraints with different priorities can be maintained, including dynamic obstacle avoidance. By trading completeness for efficiency, this method supports a number of real-life scenarios. The introduction of additional exploration techniques and proper balancing while still maintaining online capabilities could enable the use in more challenging settings. This also would provide a natural integration of global planners with the proposed task-based control paradigms in environments with direct human-robot interaction.

Usually a planner is given a workspace model and has to explore the corresponding configuration space. Unknown or uncertain environments however require the integration of workspace exploration with active sensor data acquisition into the planning concept [28]. Such an algorithm has to consider which parts of the environment model are important for the current task and what kind of resolution is required. Provided kinematics and geometry models of a robot may also be inaccurate or incomplete, thus resulting in additional errors in movement execution and acquired sensor data. Dynamic obstacles further demand the monitoring and update of existing environment estimations.

Interaction with movable obstacles may create simpler and more direct solutions to a planning query [37]. While a chair or closed door might be blocking the way, a task might still be solvable if the robot actively moves the obstacle. Distance computation or contact information can be used in order to determine repulsive forces that influence passive objects in the environment. A planner has to decide when and where to move these obstacles and if the involved effort is lower than that of executing a path without manipulation. Branched kinematic chains with multiple end-effectors are another aspect in this regard.

5.2 Task-Based Control and Natural Language Descriptions

The systems presented in Chap. 4 have introduced modern instruction paradigms without textual programming languages and attachment to specific robot systems. Task specifications on a more abstract level and with automated interpretation of general requirements can be realized through the addition of explicit domain knowledge [128]. That the transportation of a cup with liquid content implies a fixed orientation appears natural to a human worker, however this information is not easily available for a robot system. This expertise can be independent from specific problems and may be organized and managed in ontologies or databases. Object affordances may imply certain interactions (e.g., the ability of a cup to hold liquids or proper grasp points) and can be used in the automated creation of subtasks. A human operator would most likely not give detailed specifications when instructing a system to pick up an object and place it somewhere else.

After natural language input is processed and transformed into a textual representation, a syntactic and semantic analysis follows. Associations with grammatical structures result in rule-based instructions that reflect commands with side effects. Data from other components such as object recognition is used in order to identify and locate referenced objects. A matching behavior is created based on a set of actions and events, for instance approaching a cup, grasping and lifting the object followed by transportation to a target position, and finally placing it on a surface. Goals and subgoals in a database are defined as hierarchic relationships and linked in a graph with post and preconditions. A dialog manager can be employed in order to resolve conflicts in case no trivial path exists.

5.3 Model-Driven Development

Abstract descriptions for kinematics, dynamics, geometry, and hardware are just a first step toward model-driven engineering [127]. Physical and functional properties in logical form can facilitate code generation and verification. Algorithms and expert knowledge from different domains can be integrated not just for simulation purposes, but also for control architectures [1, 43]. Hierarchical and detailed models

can be used in order to describe robot systems from electronic components to gear mechanics and dynamic properties. Full access to all aspects of a model—including details such as real-time constraints or specific communication channels—opens many possibilities for verification, optimization, and deployment.

Applications can take advantage of new code generators that support parallelization of existing algorithms without difficult reengineering. A system can be changed from deployment to a single or multiple machines by exchanging shared memory communication with network protocols that conform to real-time requirements. Manufacturers can either provide optimized parameters for generic components or custom models that resemble actual hardware details more closely. The same models can be directly modified and used for both simulation and control purposes.

References

1. Åkesson, J., Elmqvist, H., Nordström, U.: Dymola and Modelica_EmbeddedSystems in teaching – experiences from a project course. In: Proceedings of the International Modelica Conference (2009) 5, 113
2. Amato, N.M., Bayazit, O.B., Dale, L.K., Jones, C., Vallejo, D.: OBPRM: An obstacle-based PRM for 3D workspaces. In: Proceedings of the International Workshop on the Algorithmic Foundations of Robotics, pp. 155–168. Houston, TX, USA (1998) 56, 58, 61
3. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide, third edn. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1999) 101
4. Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T., Yoon, W.K.: RT-middleware: Distributed component middleware for RT (robot technology). In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3933–3938. Edmonton, AB, Canada (2005) 5
5. Baginski, B.: Local motion planning for manipulators based on shrinking and growing geometry models. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3303–3308. Minneapolis, MN, USA (1996) 62
6. Bajaj, C.L., Dey, T.K.: Convex decomposition of polyhedra and robustness. SIAM Journal on Computing **21**(2), 339–364 (1992) 36
7. Baraff, D.: Dynamic simulation of non-penetrating rigid bodies. Ph.D. thesis, Cornell University, Ithaca, NY, USA (1992) 39
8. Barber, C.B., Dobkin, D.P., Huhdanpaa, H.: The Quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software **22**(4), 469–483 (1996) 36
9. Bard, E.G., Bekkering, H., Bicho, E., de Bruijn, E.R.A., Cuijpers, R.H., Erlhagen, W., Foster, M.E., Giuliani, M., Isard, A., Hipólito, N., Hulstijn, M., Knoll, A., Louro, L., Maas, Y., Matheson, C., Meulenbroek, R.G.J., Müller, T., Newman-Norlund, R., Oberlander, J., Rickert, M., de Ruiter, J.P., van Schie, H.T., Silva, R., Sousa, M.: Goal inferencing and situated referencing in human-robot interaction: A user-evaluation study. Deliverable D5.13, EU FP6 IST Cognitive Systems Integrated Project JAST (FP6-003747-IP) (2009) 92
10. Bard, E.G., Hill, R., Foster, M.E.: What tunes accessibility of referring expressions in task-related dialogue? In: Proceedings of the Annual Meeting of the Cognitive Science Society. Chicago, IL, USA (2008) 83
11. Bard, E.G., Hill, R., Foster, M.E.: Who tunes accessibility of referring expressions in task-related dialogue? In: Proceedings of the Workshop on the Semantics and Pragmatics of Dialogue. London, UK (2008) 83
12. Barraquand, J., Latombe, J.C.: A Monte-Carlo algorithm for path planning with many degrees of freedom. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1712–1717. Cincinnati, OH, USA (1990) 62, 66

13. van den Berg, J.P., Overmars, M.H.: Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 453–460 (2004) 56, 57
14. van den Berg, J.P., Overmars, M.H.: Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. International Journal of Robotics Research **24**(12), 1055–1071 (2005) 56, 57, 62
15. van den Bergen, G.: Collision Detection in Interactive 3D Environments. The Morgan Kaufmann Series in Interactive 3D Technology. Morgan Kaufmann Publishers, San Francisco, CA, USA (2004) 38, 69
16. Bicho, E., Louro, L., Hipólito, N., Erlhagen, W.: A dynamic field approach to goal inference and error monitoring for human-robot interaction. In: Proceedings of the Symposium on New Frontiers in Human-Robot Interaction, Adaptive and Emergent Behaviour and Complex Systems Convention, pp. 31–37. Edinburgh, Scotland (2009) 91
17. Blackford, L.S., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., Heroux, M., Kaufman, L., Lumsdaine, A., Petitet, A., Pozo, R., Remington, K., Whaley, R.C.: An updated set of basic linear algebra subprograms (BLAS). ACM Transactions on Mathematical Software **28**(2), 135–151 (2002) 101
18. Bohlin, R., Kavraki, L.E.: Path planning using lazy PRM. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 521–528. San Francisco, CA, USA (2000) 49, 62
19. Boor, V., Overmars, M.H., van der Stappen, A.F.: The Gaussian sampling strategy for probabilistic roadmap planners. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1018–1023. Detroit, MI, USA (1999) 48, 55, 61, 68
20. Brock, O., Kavraki, L.E.: Decomposition-based motion planning: Towards real-time planning for robots with many degrees of freedom. Tech. Rep. TR00-367, Rice University, Houston, TX, USA (2000) 52, 57, 64
21. Brock, O., Kavraki, L.E.: Towards real-time motion planning in high-dimensional spaces. In: Proceedings of the International Symposium on Robotics and Automation. Monterey, Mexico (2000) 52, 57
22. Brock, O., Kavraki, L.E.: Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1469–1474 (2001) 52, 57, 62, 64, 66
23. Brock, O., Khatib, O.: Elastic strips: A framework for motion generation in human environments. International Journal of Robotics Research **21**(12), 1031–1052 (2002) 44, 58, 86
24. Brooks, A., Kaupp, T., Makarenko, A., Williams, S., Orebäck, A.: Orca: A component model and repository. In: Software Engineering for Experimental Robotics, *Springer Tracts in Advanced Robotics*, vol. 30, pp. 231–251. Springer (2007) 4
25. Bruyninckx, H.: Open robot control software: The OROCOS project. In: Proceedings of the IEEE International Conference on Robotics and Automation, vol. 3, pp. 2523–2528. Seoul, Korea (2001) 4
26. Burns, B., Brock, O.: Sampling-based motion planning using predictive models. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3120–3125. Barcelona, Spain (2005) 49, 112
27. Burns, B., Brock, O.: Toward optimal configuration space sampling. In: Proceedings of the Robotics Science and Systems Conference. Cambridge, MA, USA (2005) 45, 55, 62
28. Burns, B., Brock, O.: Sampling-based motion planning with sensing uncertainty. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3313–3318. Rome, Italy (2007) 112
29. Burns, B., Brock, O.: Single-query motion planning with utility-guided random trees. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3307–3312. Rome, Italy (2007) 52, 62, 112
30. Buss, S.R.: Introduction to inverse kinematics with Jacobian transpose, pseudoinverse and damped least squares methods (2009) 24

31. Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L.E., Thrun, S.: Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press (2005) 42
32. Ciesla, C.: Entwicklung eines Verfahrens zur Vereinfachung von 3D-Polygonmodellen für die Verwendung in der Robotik. Diplomarbeit, Technische Universität München, Munich, Germany (2007) 36
33. Clark, H.H.: Pointing and placing. In: S. Kita (ed.) Pointing: Where Language, Culture, and Cognition Meet. Lawrence Erlbaum Associates (2003) 84
34. Clark, H.H.: Coordinating with each other in a material world. Discourse Studies **7**(4–5), 507–525 (2005) 83
35. Collett, T.H.J., MacDonald, B.A., Gerkey, B.P.: Player 2.0: Toward a practical robot programming framework. In: Proceedings of the Australasian Conference on Robotics and Automation. Sydney, Australia (2005) 4
36. van der Corput, J.G.: Verteilungsfunktionen. In: Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen, vol. 38, pp. 813–821 (1935) 47, 69
37. Cortés, J., Jaillet, L., Siméon, T.: Disassembly path planning for complex articulated objects. IEEE Transactions on Robotics **24**(2), 475–481 (2008) 113
38. Craig, J.J.: Introduction to Robotics: Mechanics and Control, third edn. Pearson Prentice Hall, Upper Saddle River, NJ, USA (2005) 2, 13, 14, 22, 24, 28
39. Cuijpers, R.H., van Schie, H.T., Koppen, M., Erlhagen, W., Bekkering, H.: Goals and means in action observation: A computational approach. Neural Networks **19**(3), 311–322 (2006) 83
40. Dalibard, S., Laumond, J.P.: Control of probabilistic diffusion in motion planning. In: Proceedings of the International Workshop on the Algorithmic Foundations of Robotics. Guanajuato, Mexico (2008) 52, 112
41. Dam, E.B., Koch, M., Lillholm, M.: Quaternions, interpolation and animation. Tech. Rep. DIKU-TR-98/5, University of Copenhagen, Copenhagen, Denmark (1998) 17, 21
42. Denavit, J., Hartenberg, R.S.: A kinematic notation for lower-pair mechanisms based on matrices. Journal of Applied Mechanics **22**, 215–221 (1955) 28
43. Elmqvist, H., Otter, M., Henriksson, D., Thiele, B., Mattsson, S.E.: Modelica for embedded systems. In: Proceedings of the International Modelica Conference, pp. 354–363. Como, Italy (2009) 5, 113
44. Ericson, C.: Real-Time Collision Detection. The Morgan Kaufmann Series in Interactive 3D Technology. Morgan Kaufmann Publishers, San Francisco, CA, USA (2005) 38
45. Erleben, K.: Stable, robust, and versatile multibody dynamics animation. Ph.D. thesis, University of Copenhagen, Copenhagen, Denmark (2004) 39
46. Erlhagen, W., Mukovskiya, A., Bicho, E.: A dynamic model for action understanding and goal-directed imitation. Brain Research **1083**(1), 174–188 (2006) 91
47. Fabri, A., Giezeman, G.J., Kettner, L., Schirra, S., Schönherr, S.: On the design of CGAL, a computational geometry algorithms library. Software: Practice and Experience **30**(11), 1167–1202 (2000) 106
48. Featherstone, R.: Rigid Body Dynamics Algorithms. Springer, New York, NY, USA (2008) 14, 31, 39, 102
49. Finkemeyer, B., Kröger, T., Kubus, D., Olschewski, M., Wahl, F.M.: MiRPA: Middleware for robotic and process control applications. In: Proceedings of the Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware, IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 76–90. San Diego, CA, USA (2007) 4
50. Foskey, M., Garber, M., Lin, M.C., Manocha, D.: A Voronoi-based hybrid motion planner. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 55–60. Maui, HI, USA (2001) 55, 62
51. Foster, M.E.: Roles of a talking head in a cooperative human-robot dialogue system. In: Proceedings of the International Conference on Intelligent Virtual Agents, *Lecture Notes in Computer Science*, vol. 4722, pp. 375–376. Springer, Paris, France (2007) 93

52. Foster, M.E., Bard, E.G., Hill, R.L., Guhe, M., Oberlander, J., Knoll, A.: The roles of haptic-ostensive referring expressions in cooperative, task-based human-robot dialogue. In: Proceedings of the ACM/IEEE International Conference on Human Robot Interaction, pp. 295–302. Amsterdam, Netherlands (2008) 92
53. Foster, M.E., Giuliani, M., Isard, A., Matheson, C., Oberlander, J., Knoll, A.: Evaluating description and reference strategies in a cooperative human-robot dialogue system. In: Proceedings of the International Joint Conferences on Artificial Intelligence, pp. 1818–1823. Pasadena, CA, USA (2009) 92
54. Foster, M.E., Giuliani, M., Knoll, A.: Comparing objective and subjective measures of usability in a human-robot dialogue system. In: Proceedings of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing. Singapore (2009) 92
55. Foster, M.E., Giuliani, M., Müller, T., Rickert, M., Knoll, A., Erlhagen, W., Bicho, E., Hipólito, N., Louro, L.: Combining goal inference and natural-language dialogue for human-robot joint action. In: Proceedings of the 1st International Workshop on Combinations of Intelligent Methods and Applications, 18th European Conference on Artificial Intelligence. Patras, Greece (2008) 92
56. Foster, M.E., Matheson, C.: Following assembly plans in cooperative, task-based human-robot dialogue. In: Proceedings of the Workshop on the Semantics and Pragmatics of Dialogue. London, UK (2008) 91, 92
57. Garber, M., Lin, M.C.: Constraint-based motion planning using Voronoi diagrams. In: Proceedings of the International Workshop on the Algorithmic Foundations of Robotics (2002) 56
58. Geraerts, R., Overmars, M.: Creating high-quality paths for motion planning. International Journal of Robotics Research **26**(8), 845–863 (2007) 46, 78
59. Geraerts, R., Overmars, M.H.: Reachability analysis of sampling based planners. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 404–410 (2005) 55
60. Geraerts, R., Overmars, M.H.: Sampling and node adding in probabilistic roadmap planners. Robotics and Autonomous Systems **54**(2), 165–173 (2006) 49
61. Geraerts, R., Overmars, M.H.: Reachability-based analysis for probabilistic roadmap planners. Robotics and Autonomous Systems **55**(11), 824–836 (2007) 48
62. Gibson, J.J.: The Theory of Affordances. Erlbaum, Hillsdale, NJ, USA (1977) 10
63. Gilbert, E.G., Foo, C.P.: Computing the distance between general convex objects in three-dimensional space. IEEE Transactions on Robotics and Automation **6**(1), 53–61 (1990) 38
64. Gilbert, E.G., Johnson, D.W., Keerthi, S.S.: A fast procedure for computing the distance between complex objects in three-dimensional space. IEEE Journal of Robotics and Automation **4**(2), 193–203 (1988) 38
65. Giuliani, M.: A basic system for interpretation of utterances in natural language, based on the combinatory categorial grammar. Master's thesis, Technische Universität München, Munich, Germany (2006) 90
66. Giuliani, M., Knoll, A.: MultiML: A general purpose representation language for multimodal human utterances. In: Proceedings of the ACM International Conference on Multimodal Interfaces, pp. 165–172. Chania, Greece (2008) 90
67. Henning, M.: A new approach to object-oriented middleware. IEEE Internet Computing **8**(1), 66–75 (2004) 87, 98
68. Henning, M.: The rise and fall of CORBA. Component Technologies **4**(5), 28–34 (2006) 100
69. Hoff III, K.E., Culver, T., Keyser, J., Lin, M.C., Manocha, D.: Fast computation of generalized voronoi diagrams using graphics hardware. In: Proceedings of the International Conference on Computer Graphics and Interactive Techniques, pp. 277–286 (1999) 56
70. Holleman, C., Kavraki, L.E.: A framework for using the workspace medial axis in PRM planners. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1408–1413. San Francisco, CA, USA (2000) 56, 62

71. Holleman, C., Kavraki, L.E., Warren, J.: Planning paths for a flexible surface patch. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 21–26. Leuven, Belgium (1998) 56
72. Höpler, R., Otter, M.: A versatile C++ toolbox for model based, real time control systems of robotic manipulators. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 4, pp. 2208–2214. Maui, HI, USA (2001) 103
73. Hsu, D., Jiang, T., Reif, J., Sun, Z.: The bridge test for sampling narrow passages with probabilistic roadmap planners. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 4420–4426 (2003) 49, 55, 61, 68
74. Hsu, D., Latombe, J.C., Kurniawati, H.: On the probabilistic foundations of probabilistic roadmap planning. The International Journal of Robotics Research **25**(7), 627–643 (2006) 58
75. Hsu, D., Latombe, J.C., Motwani, R.: Path planning in expansive configuration spaces. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 2719–2726. Albuquerque, NM, USA (1997) 55
76. Hsu, D., Latombe, J.C., Motwani, R.: Path planning in expansive configuration spaces. International Journal of Computational Geometry and Application **9**(4&5), 495–512 (1999) 55
77. Huber, M., Lenz, C., Rickert, M., Knoll, A., Brandt, T., Glasauer, S.: Human preferences in industrial human-robot interactions. In: Proceedings of the International Workshop on Cognition for Technical Systems. Munich, Germany (2008) 94
78. Huber, M., Radrich, H., Wendt, C., Rickert, M., Knoll, A., Brandt, T., Glasauer, S.: Evaluation of a novel biologically inspired trajectory generator in human-robot interaction. In: Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication, pp. 639–644. Toyama, Japan (2009) 94
79. Huber, M., Rickert, M., Knoll, A., Brandt, T., Glasauer, S.: Human-robot interaction in handing-over tasks. In: Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication, pp. 107–112. Munich, Germany (2008) 94
80. Hulstijn, M., Meulenbroek, R., Wijers, M., de Ruiter, J.P.: A frequency analysis of joint-action primitives. JAST Project Deliverable D2.3, Nijmegen Institute for Cognition and Information, Nijmegen, The Netherlands (2005) 86
81. Jackson, J.: Microsoft robotics studio: A technical introduction. IEEE Robotics & Automation Magazine **14**(4), 1070–9932 (2007) 5
82. Jaillet, L., Yershova, A., LaValle, S.M., Siméon, T.: Adaptive tuning of the sampling domain for dynamic-domain RRTs. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2851–2856 (2005) 51, 62, 68
83. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research **4**, 237–285 (1996) 59
84. Kavraki, L., Latombe, J.C.: Randomized preprocessing of configuration space for path planning: Articulated robots. In: Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems, pp. 1764–1771. Munich, Germany (1994) 46, 61, 68
85. Kavraki, L.E., Švestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation **12**(4), 566–580 (1996) 46
86. Kecskeméthy, A., Hiller, M.: An object-oriented approach for an effective formulation of multibody dynamics. Computer Methods in Applied Mechanics and Engineering **115**(3–4), 287–314 (1994) 103
87. Kelley, T.D.: Symbolic and sub-symbolic representations in computational models of human cognition: What can be learned from biology? Theory & Psychology **13**(6), 847–860 (2003) 10
88. Kelley, T.D., Long, L.N.: Deep Blue cannot play checkers: The need for generalized intelligence for mobile robots. Journal of Robotics **2010**, 523757 (2010) 10
89. Khalil, W., Dombre, E.: Modeling, Identification and Control of Robots. Hermes Penton Ltd., London, GB (2002) 2, 13, 14, 22

90. Khalil, W., Kleinfinger, J.F.: A new geometric notation for open and closed loop robots. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1174–1180. San Francisco, CA, USA (1986) 28
91. Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. International Journal of Robotics Research **5**(1), 90–98 (1986) 44, 61, 62, 64
92. Khatib, O.: A unified approach for motion and force control of robot manipulators: The operational space formulation. IEEE Journal of Robotics and Automation **3**(1), 43–53 (1987) 34
93. Kuffner Jr., J.J.: Effective sampling and distance metrics for 3D rigid body path planning. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3993–3998 (2004) 20, 46
94. Kuffner Jr., J.J., LaValle, S.M.: RRT-connect: An efficient approach to single-query path planning. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 995–1001 (2000) 52, 62, 63, 65, 66, 68
95. Kurniawati, H., Hsu, D.: Workspace importance sampling for probabilistic roadmap planning. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1618–1623 (2004) 56
96. Langley, P., Laird, J.E., Rogers, S.: Cognitive architectures: Research issues and challenges. Cognitive Systems Research **10**(2), 141–160 (2009) 10
97. Latombe, J.C.: Robot Motion Planning. Kluwer Academic Publishers, Boston, MA, USA (1991) 17, 19, 20, 41, 42
98. LaValle, S.M.: Rapidly-exploring random trees: A new tool for path planning. Tech. Rep. TR 98-11, Iowa State University, Ames, IA, USA (1998) 49, 62, 63
99. LaValle, S.M.: Planning Algorithms. Cambridge University Press (2006) 19, 20, 41, 69
100. LaValle, S.M., Kuffner Jr., J.J.: Randomized kinodynamic planning. The International Journal of Robotics Research **20**(5), 378–400 (2001) 50
101. Lawson, C.L., Hanson, R.J., Kincaid, D.R., Krogh, F.T.: Basic linear algebra subprograms for Fortran usage. ACM Transactions on Mathematical Software **5**(3), 308–323 (1979) 101
102. Lenz, C., Rickert, M., Panin, G., Knoll, A.: Constraint task-based control in industrial settings. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3058–3063. St. Louis, MO, USA (2009) 34, 98
103. Lozano-Pérez, T., Wesley, M.A.: An algorithm for planning collision-free paths among polyhedral obstacles. Communications of the ACM **22**(10), 560–570 (1979) 42
104. Matsumoto, M., Nishimura, T.: Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation **8**(1), 3–30 (1998) 108
105. Meerbergen, K., Fresl, K., Knapen, T.: C++ bindings to external software libraries with examples from BLAS, LAPACK, UMFPACK, and MUMPS. ACM Transactions on Mathematical Software **36**(4), 1–23 (2009) 101
106. Metta, G., Fitzpatrick, P., Natale, L.: YARP: Yet another robot platform. International Journal of Advanced Robotic Systems **3**(1), 43–48 (2006) 4
107. Mirtich, B.V.: Impuse-based dynamic simulation of rigid body systems. Ph.D. thesis, University of California, Berkeley, CA, USA (1996) 39
108. Montemerlo, M., Roy, N., Thrun, S.: Perspectives on standardization in mobile robot programming: the Carnegie Mellon navigation (CARMEN) toolkit. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 3, pp. 2436–2441. Pittsburgh, PA, USA (2003) 4
109. Müller, T.: Konzeption und Implementierung eines adaptiven, robusten und effizienten Verfahrens zur Klassifikation von Objekten mittels Template Matching. Diplomarbeit, Technische Universität München, Munich, Germany (2007) 88
110. Müller, T., Ziaie, P., Knoll, A.: A wait-free realtime system for optimal distribution of vision tasks on multicore architectures. In: Proceedings of the International Conference on Informatics in Control, Automation and Robotics, pp. 301–306. Funchal, Portugal (2008) 88
111. Myers, N.C.: Traits: A new and useful template technique. C++ Report **7**(5), 32–35 (1995) 101

112. Nielsen, C.L., Kavraki, L.E.: A two level fuzzy PRM for manipulation planning. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1716–1721. Takamatsu, Japan (2000) 62
113. Nissoux, C., Siméon, T., Laumond, J.P.: Visibility based probabilistic roadmaps. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1316–1321. Kyongju, South Korea (1999) 49, 55, 58
114. Otter, M., Elmqvist, H., Cellier, F.E.: Modeling of multibody systems with the object-oriented modeling language Dymola. Nonlinear Dynamics **9**, 91–112 (1996) 5, 103
115. Panin, G., Knoll, A.: Real-time 3D face tracking with mutual information and active contours. In: Proceedings of the International Symposium on Advances in Visual Computing, *Lecture Notes in Computer Science*, vol. 4841, pp. 1–12. Springer, Lake Tahoe, NV, USA (2007) 89
116. Panin, G., Ladikos, A., Knoll, A.: An efficient and robust real-time contour tracking system. In: Proceedings of the IEEE International Conference on Computer Vision Systems, pp. 44–51. New York, NY, USA (2006) 89
117. Paul, R.P.: Robot Manipulators: Mathematics, Programming and Control. MIT Press, Cambridge, MA, USA (1981) 2, 13, 14, 22, 28
118. Petzold, L.R.: A description of DASSL: A differential/algebraic system solver. Tech. Rep. SAND82-8637, Sandia National Laboratories, Livermore, CA, USA (1982) 27
119. Pisula, C., Hoff III, K.E., Lin, M.C., Manocha, D.: Randomized path planning for a rigid body based on hardware accelerated Voronoi sampling. In: Proceedings of the International Workshop on the Algorithmic Foundations of Robotics (2000) 56
120. Plaku, E., Bekris, K.E., Chen, B.Y., Ladd, A.M., Kavraki, L.E.: Sampling-based roadmap of trees for parallel motion planning. IEEE Transactions on Robotics **21**(4), 597–608 (2005) 52
121. Plaku, E., Bekris, K.E., Kavraki, L.E.: OOPS for motion planning: An online open-source programming system. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3711–3716. Rome, Italy (2007) 4
122. Plaku, E., Kavraki, L.E., Vardi, M.Y.: Discrete search leading continuous exploration for kinodynamic motion planning. In: Proceedings of the Robotics: Science and Systems Conference, pp. 326–333. Atlanta, GA, USA (2007) 62
123. Plaku, E., Kavraki, L.E., Vardi, M.Y.: Impact of workspace decompositions on discrete search leading continuous exploration (DSLX) motion planning. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3751–3756. Pasadena, CA, USA (2008) 62
124. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.Y.: ROS: An open-source robot operating system. In: Proceedings of the Workshop on Open-Source Software in Robotics, IEEE International Conference on Robotics and Automation. Kobe, Japan (2009) 4
125. Reif, J.H.: Complexity of the mover's problem and generalizations. In: Proceedings of the IEEE Symposium on Foundations of Computer Science, pp. 421–427. San Juan, Puerto Rico (1979) 42
126. Rickert, M., Brock, O., Knoll, A.: Balancing exploration and exploitation in motion planning. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 2812–2817. Pasadena, CA, USA (2008) 63
127. Rickert, M., Geisinger, M., Barner, S., Knoll, A.: Software development workflow in robotics. In: Proceedings of the Workshop on Open Source Software in Robotics, IEEE International Conference on Robotics and Automation. Kobe, Japan (2009) 113
128. Rickert, M., Kaßecker, M., Knoll, A.: Aufgabenbeschreibung mit verhaltensbasierter Robotersteuerung und natürlicher Kommunikation. Tech. Rep. TUM-I0914, Technische Universität München, Munich, Germany (2009) 113
129. Rimon, E., Koditschek, D.E.: Exact robot navigation using artificial potential functions. IEEE Transactions on Robotics and Automation **8**(5), 501–518 (1992) 43
130. Rodríguez, S., Tang, X., Lien, J.M., Amato, N.M.: An obstacle-based rapidly-exploring random tree. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 895–900. Orlando, FL, USA (2006) 62

131. de Ruiter, J.P.: Some multimodal signals in humans. In: Proceedings of the Workshop on Multimodal Output Generation, pp. 141–148. Aberdeen, Scotland (2007) 83
132. Sebanz, N., Bekkering, H., Knoblich, G.: Joint action: Bodies and minds moving together. Trends in Cognitive Sciences **10**(2), 70–76 (2006) 83
133. Sentis, L.: Synthesis and control of whole-body behaviors in humanoid systems. Ph.D. thesis, Stanford University, Stanford, CA, USA (2007) 10
134. Sentis, L., Khatib, O.: Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. International Journal of Humanoid Robotics **2**(4), 505–518 (2005) 34, 59
135. Sentis, L., Khatib, O.: A whole-body control framework for humanoids operating in human environments. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 2641–2648. Orlando, FL, USA (2006) 34
136. Shoemake, K.: Animating rotation with quaternion curves. ACM SIGGRAPH Computer Graphics **19**(3), 245–254 (1985) 17, 21
137. Siciliano, B., Khatib, O. (eds.): Springer Handbook of Robotics. Springer, Heidelberg, Germany (2008) 2, 13, 14, 22, 24, 31, 34
138. Siek, J.G., Lee, L.Q., Lumsdaine, A.: The Boost Graph Library: User Guide and Reference Manual. C++ In-Depth Series. Addison-Wesley Professional (2001) 108
139. Siméon, T., Laumond, J.P., Nissoux, C.: Visibility-based probabilistic roadmaps for motion planning. Advanced Robotics **14**(6), 477–493 (2000) 49, 55, 58, 61
140. Stilman, M.: Task constrained motion planning in robot joint space. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3074–3081. San Diego, CA, USA (2007) 65
141. Sun, R.: Artificial intelligence: Connectionist and symbolic approaches. In: International Encyclopedia of the Social & Behavioral Sciences, pp. 783–789. Pergamon, Oxford, UK (2001) 10
142. Sun, R.: The importance of cognitive architectures: An analysis based on CLARION. Journal of Experimental & Theoretical Artificial Intelligence **19**(2), 159–193 (2007) 10
143. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, USA (1998) 59
144. Švestka, P.: On probabilistic completeness and expected complexity for probabilistic path planning. Tech. Rep. UU-CS-1996-20, Universiteit Utrecht, Utrecht, Netherlands (1996) 46
145. Taylor, S.: Intel Integrated Performance Primitives: How to Optimize Software Applications Using Intel IPP. Intel Press, Hillsboro, OR, USA (2004) 101
146. Taylor, S.: Optimizing Applications for Multi-Core Processors: Using the Intel Integrated Performance Primitives, second edn. Intel Press, Hillsboro, OR, USA (2007) 101
147. Thrun, S.: The role of exploration in learning control. In: Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches. Van Nostrand Reinhold (1992) 59
148. Veldhuizen, T.: Expression templates. C++ Report **7**(5), 36–43 (1995) 101
149. Vincent, L., Soille, P.: Watersheds in digital spaces: An efficient algorithm based on immersion simulations. IEEE Transactions on Pattern Analysis and Machine Intelligence **13**(6), 583–598 (1991) 56, 57
150. Yang, Y., Brock, O.: Adapting the sampling distribution in PRM planners based on an approximated medial axis. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 4405–4410. New Orleans, LA, USA (2004) 57, 62
151. Yang, Y., Brock, O.: Viewing motion planning as disassembly: A decomposition-based approach for non-stationary robots. Tech. Rep. 04-108, University of Massachusetts Amherst, Amherst, MA, USA (2004) 57
152. Yang, Y., Brock, O.: Efficient motion planning based on disassembly. In: Proceedings of the Robotics: Science and Systems Conference, pp. 97–104. Cambridge, MA, USA (2005) 57, 62
153. Yang, Y., Brock, O.: Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation in dynamic environments. In: Proceedings of the Robotics Science and Systems Conference (2006) 58, 62, 86

154. Yershova, A., Jaillet, L., Siméon, T., LaValle, S.M.: Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3867–3872 (2005) 51
155. Yershova, A., LaValle, S.M.: Improving motion planning algorithms by efficient nearest-neighbor searching. IEEE Transactions on Robotics **23**(1), 151–157 (2007) 48, 50, 68
156. Zäh, M.F., Munzert, U., Oefele, F.: Offline concept for optimization of robot paths for remote laser welding without scanners. In: Proceedings of the international Conference on Trends in the Development of Machinery and Associated Technology, pp. 11–15. Barcelona, Spain (2006) 9
157. Ziaie, P.: Implementing and evaluating hand-gesture recognition for human-robot joint-action. Master's thesis, Technische Universität München, Munich, Germany (2008) 89
158. Ziaie, P., Müller, T., Foster, M.E., Knoll, A.: A naïve Bayes classifier with distance weighting for hand-gesture recognition. In: Proceedings of the International CSI Computer Conference. Kish Island, Iran (2008) 89
159. Ziaie, P., Müller, T., Knoll, A.: A novel approach to hand-gesture recognition in a human-robot dialog system. In: Proceedings of the International Workshops on Image Processing Theory, Tools and Applications, pp. 1–8. Sousse, Tunesia (2008) 89